Laboratori d’Estructura d’Ordinadors

Material de laboratori per a les assignatures de la ETSE-UV

• 34837 Estructura de Computadors, de segon curs del Grau en Enginyeria Multimèdia.

• 34657 Estructura de Computadors, de segon curs del Grau en Enginyeria Informàtica.

El material està dividit en 8 pràctiques, a desenvolupar en sessions de 2 hores i 30 minuts.

La primera pràctica serveix com a introducció al MipsIt, software educacional de distribució gratuïta. MipsIt simula un ordinador bàsic amb la possibilitat de configurar la memòria cau. Addicionalment el programa disposa d'espai d'entrada/eixida (E/E) i gestió d'interrupcions, de manera que és possible provar tècniques bàsiques d'E/E.

La segona sessió proposa una sèrie de programes en C que accedeixen a dades. Es proposa que s'analitze el rendiment dels programes en funció de diversos tipus de memòria cau de dades. En la tercera sessió es proposen una sèrie de tècniques que optimitzen el rendiment de la memòria cau, arreglant els problemes detectats en la sessió anterior.

A la quarta i cinquena sessió es proposa la implementació de programes que fan ús de l'espai d'E/E. Inicialment mitjançant prova d’estat i posteriorment mitjançant interrupcions.

En les tres últimes sessions, s'usa una màquina Linux, per poder així accedir a baix nivell al control dels perifèrics. La sisena sessió proposa l'accés a la controladora de teclat. La setena sessió, dedicada a discos magnètics, analitza les prestacions dels RAIDs: matriu redundant de discos independents, de l'anglès *redundant array of independent disks*. Finalment la vuitena sessió se centra en l'accés a baix nivell a la memòria de vídeo.

Paraules clau:

Estructura d’ordinadors. Jerarquia de memòria. Sistema d'entrada/eixida. Perifèrics.

Lab 1

Laboratori 1
Introducció al MipsIt

(Actualitzat el dia 16/09/2021)

Temps de treball a casa (previst): 1 hora i 30 minuts
Temps de laboratori: 2 hores i 30 minuts

1. Introducció

En aquesta primera sessió de laboratori es busca que l'alumnat es familiaritze amb el programari de simulació i assemblatge dels programes per als microprocessadors MIPS denominat MipsIt, compost per un entorn de programació i un simulador. Aquest simulador permet compilar i simular programes escrits tant en assemblador com en llenguatge C. L'entorn de simulació disposa d'una jerarquia de memòria configurable per l'usuari, a més d'un sistema d'entrada/eixida amb interrupcions i una consola de visualització. Aquesta plataforma s'usarà en les cinc primeres sessions de l'assignatura.

Per tal d'iniciar-se en l'ús d'aquesta plataforma, es comença descrivint-ne els components i els passos que cal seguir per a usar-la. A continuació es mostren diversos exemples de programa per tal de veure de forma completa tot el procés de compilació i simulació d'un programa escrit per a aquests microprocessadors. Això permet revisar i consolidar alguns conceptes del curs anterior i veure la relació del llenguatge de programació amb els elements del maquinari de l'ordinador que s'estudien en aquesta assignatura.

En les sessions de laboratori de l'assignatura es tracta de consolidar els coneixements vistos a les classes teòriques. Per això, abans de la sessió de laboratori es demana als membres de cada grup de treball que lligen la documentació relacionada amb el fonament teòric del treball per desenvolupar i amb les eines que s'empraran en el procés. La lectura prèvia d'aquests documents permet usar les aplicacions i l'equipament del laboratori i, alhora, respondre a preguntes relacionades sobre la funcionalitat i l'ús.

En les sessions de laboratori és important establir una dinàmica de treball en grup que permeta traure el màxim profit de l'estada al laboratori. Perquè això siga possible es requereix puntualitat i que es pare atenció a les indicacions del professorat al laboratori. En la primera sessió de laboratori això és especialment important atès que es presentaran les normes de funcionament necessàries per a un desenvolupament correcte de les pràctiques.

Objectius

En acabar aquesta pràctica, l'estudiant és capaç de:

- Conèixer els components bàsics i la potencialitat del MipsIt.
- Configurar el simulador MipsIt.
- Crear un projecte amb el MipsIt, tant en assemblador com en llenguatge C.
- Compilar, assemblar i simular un programa amb el MipsIt.

- Conèixer diversos aspectes d'interacció entre els programes d'alt nivell i els components del microprocessador: emmagatzematge de dades, alineament de variables, operacions de lectura i escriptura de dispositius d'entrada/eixida, etc.

Recursos

Per al desenvolupament d'aquestes sessions de laboratori es disposa d'un ordinador personal PC amb el sistema operatiu Windows i el simulador MipsIt. El simulador MipsIt per al sistema operatiu Windows es pot descarregar de l'Aula Virtual. El fitxer s'ha de descomprimir en qualsevol directori que no continga espais ni titles al nom. El directori `bin` és el que conté els programes executables. En aquestes sessions de pràctiques s'usen dos d'aquests programes:

- `MipsIt.exe`: MipsIt Studio 2000 (entorn de programació per MIPS).
- `Mips.exe`: MipsSim (simulador de MIPS).

L'annex I conté la resta de la informació sobre aquesta plataforma.

Aprenentatge i avaluació del laboratori

Treball previ. El treball previ a la pràctica consisteix en un conjunt d'activitats que preparen per a obtenir el màxim aprofitament de la sessió de laboratori. El treball previ es comprova mitjançant una breu entrevista amb el professor o la professora a l'inici de la sessió. Qualsevol estudiant o estudianta que encara no haja format un grup de treball ha de completar aquesta fase en solitari. Després al laboratori es formen els grups.

Treball al laboratori. Durant la pràctica s'avalua el treball completat pel grup en la sessió presencial al laboratori. Hi ha diverses etapes, formades per activitats d'aprenentatge, que tenen una puntuació segons la complexitat. A mesura que es vaja completant cada activitat proposada, l'estudiantat ha de cridar el professorat perquè comprove quines parts ha acabat fins a aquest moment. El professor o la professora pot preguntar qualsevol cosa relacionada amb l'apartat per tal de comprovar que el problema s'ha entès i que s'ha resolt de manera original. A més, al final de la sessió s'ha de lliurar un full amb la resposta a les preguntes formulades en la sessió.

Treball posterior. Pot ser que en alguna pràctica se sol·licite un petit informe consistent en la presentació dels resultats de les mesures realitzades al laboratori, en forma de taules o gràfics, justificació teòrica d'aquestes, extensió o generalització d'alguns dels treballs realitzats o qüestions diverses relacionades amb el treball fet a la sessió de laboratori. Aquest informe s'ha de lliurar abans de la següent sessió de laboratori, o a l'inici de la sessió, al professorat de pràctiques.

La nota de cada sessió està formada per la suma de l'avaluació del treball previ, els punts de control del treball de laboratori (normalment entre 2 i 3) i el treball posterior (si n'hi ha). Cada part s'avalua amb una A (bé), B (regular) o C (malament). El valor numèric de cada lletra depèn del nombre d'etapes, però fer-les totes bé (totes amb qualificació A) equival a un 9. Es reserva el 10 per als grups que hagen resolt la pràctica en primer lloc o que aporten solucions d'excepcional valor i originalitat.

2. Treball previ al laboratori

Per a poder traure el màxim partit a la sessió de laboratori és molt important que es facen les següents activitats que es proposen i que s'hi dedique el temps indicat. Perquè aquest temps no supere el temps màxim estimat, els membres de l'equip de treball han de distribuir-se les activitats proposades. Una mala distribució de les activitats o intentar fer-les totes un sol membre comporta no poder fer-les totes o una dedicació de temps que es resta d'altres activitats.

En aquest treball previ l'alumnat ha de llegir l'annex I d'aquest document on es descriuen els components i la forma de treballar amb l'entorn MipsIt. Aquesta lectura és important perquè és una primera aproximació a l'entorn amb el qual es treballarà durant les primeres cinc sessions de laboratori. Al mateix temps, s'han de revisar els conceptes sobre la memòria de l'ordinador vistos al tema 1. Per a certes qüestions finals cal consultar alguna de les fonts bibliogràfiques indicades a l'assignatura.

Una vegada llegit l'annex I, s'ha de respondre a les següents qüestions (cal presentar-les per escrit al començar la sessió):

Q1. S'han de descriure breument els elements que formen l'ordinador implementat al simulador.

Q2. S'ha d'indicar el nombre de línies del bus d'adreces que posseeix el microprocessador i quina quantitat màxima de memòria es pot adreçar.

Q3. S'han de descriure les posicions de la memòria principal de l'ordinador i indicar quines tecnologies de memòria RAM s'han fet servir en la implementació. Quina diferència hi ha entre les dues tecnologies?

Q4. Quin tipus d'arquitectura presenta l'ordinador implementat al simulador pel que fa a memòria cau, Von Neumann o Harvard?

Q5. Quines són les posicions de memòria que ocupen tots els ports d'entrada/eixida de l'ordinador simulat?

Q6. Quin volum de dades es pot llegir o escriure als ports d'entrada/eixida?

Q7. Es demana indicar a partir de quines posicions s'ha d'emmagatzemar una dada sencera perquè estiga alineada en la memòria del simulador.

Q8. Indica quins tipus de dades en C estan sempre alineades. S'ha de raonar la resposta.

Primer punt de control: les preguntes Q1-Q8 s'han de lliurar per escrit a l'inici de la sessió i s'ha de respondre a les preguntes del professorat de pràctiques.

3. Treball al laboratori

En aquesta sessió s'utilitza de forma pràctica l'aplicació MipsIt i es comprova com funciona amb diversos programes que accediran als seus components. Les qüestions s'han de respondre per escrit i lliurar-les en acabar la sessió.

3.1. A continuació, es presenta un programa exemple en llenguatge C per a introduir l'alumne en l'ús del compilador i el simulador. Per a fer-ho, s'ha d'executar l'aplicació MipsIt (`MipsIt.exe`) que és a `C:\Mipsit\bin`. Pot ser que el programa done un missatge d'error d'obertura de ports, però s'executarà amb normalitat. S'ha de crear un projecte nou (menú *Project*) amb l'opció de codi C mínim/assemblador (*C minimal/assembler)*. Com que els fitxers no s'han de desar a l'escriptori ni en cap directori que incloga espais, noms llargs o lletres amb titles, es recomana treballar en `C:\tmp`. En aquesta carpeta es pot crear un subdirectori amb un nom adequat, per exemple, `P1-1` (pràctica 1, part 1). A més, s'ha de crear un nou fitxer associat al projecte amb codi C, amb extensió `.c`, que ha de tenir el mateix nom que el projecte, en aquest cas `P1-1.c`. El codi del programa llegeix el valor de les entrades, llegint la posició d'entrada/eixida `0xbf900000`, escriu el valor llegit sobre la mateixa adreça i mostra el valor a la barra de LED.

```
#define TRUE  1
unsigned char *commutadors = (char*) 0xbf900000;
unsigned char *leds     = (char*) 0xbf900000;

int main ()
{
  while (TRUE)
  {
      *leds=*commutadors;
      printf("El valor d'entrada és: %X \n",*commutadors);
  }
}
```

- Les opcions per defecte del compilador s'han d'editar si es vol accedir a l'espai d'entrada/eixida. Aquestes opcions queden lligades a cada projecte i es desen amb el projecte, però no resten per defecte per al projecte següent, de manera que s'han de revisar cada vegada que s'haja de compilar un projecte nou no desat. Una bona idea és tenir un fitxer de text obert amb el `Notepad`, del qual anem copiant les dues caselles que caldrà editar (i que apareixen en roig a continuació). Les opcions per editar són en les opcions del projecte (*Project> Settings)*, dins de la pestanya d'enllaç (*Link)*, i són:

```
Base address: 0x80020000
Entry-point symbol: start
Libraries: kil,c,m,lnk,gcc,soft-float
Modules:crt0.o
Linker options: -Ttext 0x80020000 -e start -lkil -lc -lm -llnk -lgcc
-lsoft-float -N -nostdlib
```

- El projecte s'ha de construir en l'entorn de compilació (F7). El resultat ha de ser correcte i cal mostrar el missatge que indica que s'ha enllaçat correctament i que s'ha completat la construcció de l'executable (*`Linking... Post build... Done`*).
- Ara s'ha d'obrir el simulador (`c:\Mipsit\bin\Mips.exe`) i el codi s'ha de carregar al simulador mitjançant l'opció adequada (*`Build>Upload>To simulator`*) o, de manera equivalent, prement F5. Una vegada carregat el codi al simulador, el nom `P1-1.out` apareix a la part superior de la finestra.
- S'ha d'executar amb l'opció del simulador (*`Cpu>Run`*) o bé prement el botó amb el símbol del triangle de reproducció. S'obri la consola que mostra el missatge de l'ordre `printf`. Per comprovar-ne el funcionament s'ha d'escollir veure el mòdul d'entrada/eixida (`View>I/O`). Ara s'han d'anar canviant les entrades associades al port d'entrada mitjançant els interruptors. Els leds han d'activar-se amb els valors premuts. A més, s'ha de comprovar el valor d'eixida que es té a la consola i que aquest coincideix amb el valor introduït amb els interruptors.

  Q9. Per què el punter al port és de tipus caràcter sense signe?

3.2. En aquest nou programa s'accedeix a dues dades de tipus enter (32 bits) situades a les posicions de memòria `0x80020000` i `0xA0020000`. S'ha de crear un nou projecte (`P1-2`) amb la metodologia ja explicada i amb un codi C nou (`P1-2.c`). El codi del programa és:

```
#define TRUE  1
int *pos1 = (int*) 0x80020000;
```

```
int *pos2 = (int*) 0xA0020000;
int main ()
{
  *pos2=0x00000000;
  *pos1=0xf0f0f0f0;
  printf("El valor de memòria 1 és: %X \n",*pos1);
  printf("El valor de memòria 2 és: %X \n",*pos2);
  while (TRUE);
}
```

- El projecte s'ha de construir en l'entorn de compilació i s'ha de simular.

Q10. S'ha d'analitzar el codi assemblador que es genera als dos exemples anteriors i comprovar exactament quan s'efectua l'accés a les posicions d'entrada/eixida o a posicions de memòria. Quina instrucció s'usa per a accedir a les posicions d'entrada/eixida? Quina instrucció s'usa per a accedir a memòria utilitzant el MIPS? Hi ha, per tant, alguna diferència entre els dos tipus d'accessos?

Q11. Quants *bytes* té una instrucció? Quantes posicions de memòria ocupa? En quin tipus de posicions comença una instrucció? Quin tipus d'ordenació de la informació es fa?, és a dir, el *byte* més significatiu s'emmagatzema en l'adreça més menuda o més gran?

3.3. En aquest cas, s'accedeix a dues variables de tipus enter situades a la memòria a les posicions `0x80020000` i `0x80020001`. El codi del programa és:

```
#define TRUE  1
int *pos1 = (int*) 0x80020001;
int *pos2 = (int*) 0x80020000;
int main ()
{
  *pos2=0x00000000;
  *pos1=0xf0f0f0f0;
  printf("El valor de memòria 1 és: %X \n",*pos1);
  printf("El valor de memòria 2 és: %X \n",*pos2);
  while (TRUE);
}
```

- S'ha de construir el projecte en l'entorn de compilació i s'ha de simular.

- El programa funciona correctament? Quin problema sembla que té?

Segon punt de control: s'ha de cridar el professor o la professora i contestar les preguntes de les seccions 3.2 i 3.3 o preguntes relacionades.

3.4. En aquest cas s'accedeix a la posició d'entrada de les interrupcions i se'n mostra el valor.

```
#define TRUE  1
unsigned char *interr=(char*) 0xbfa00000;
int main ()
{
     while (TRUE){
          printf("El valor d'entrada és: %X \n",*interr);
     }
}
```

El projecte s'ha de construir en l'entorn de compilació i s'ha de simular.

Prem les tecles d'interrupció K1 i K2 alternativament. Comprova i anota el valor obtingut per la posició d'entrada de les interrupcions en cada cas. S'ha de comentar el resultat obtingut quan es prem unes quantes vegades el mateix interruptor.

3.5. S'ha de crear un programa en codi C que llija les entrades del mòdul d'entrada/eixida situat a la posició `0xbf900000` associada als interruptors i que, tenint en compte el valor corresponent, active els leds d'eixida com una barra de nivell. Els intervals de valors d'entrada amb els valors corresponents d'eixida desitjats són els següents:

Entrada	Eixida
0 - 31	1 (només activa el 1r led)
32 - 63	2 (només activa el 2n led)
64 - 95	4 (només activa el 3r led)
96 -127	8 (només activa el 4t led)
128 - 159	16 (només activa el 5è led)
160 - 191	32 (només activa el 6è led)
192 - 223	64 (només activa el 7è led)
224 - 255	128 (només activa el 8è led)

S'ha de crear el programa i simular-ne el funcionament. Una vegada finalitzat, cal mostrar el codi del programa.

Tercer punt de control: s'ha de cridar el professor o la professora i contestar les preguntes de l'apartat 3.4 i 3.5 o preguntes relacionades.

Les respostes a les preguntes Q1-Q11 s'han de lliurar per escrit.

Annex I. MipsIt Studio 2000

Introducció

El MipsIt és l'entorn de programació en el qual s'escriuran els programes de les primeres cinc sessions de pràctiques. El concepte fonamental per a treballar amb el MipsIt Studio 2000 és el de projecte. Aquest concepte, ja vist en altres assignatures anteriors, tant de programació com de disseny de maquinari, es pot entendre com un conjunt de fitxers font interrelacionats que es compilen i s'enllacen per a crear un fitxer executable que es porta al simulador del processador MIPS. Un projecte de MipsIt pot contenir programes escrits en assemblador del MIPS, programes escrits en llenguatge C i fitxers de text. En l'entorn, que es mostra a la figura 1, es poden observar les finestres següents:

- Finestra de programes: mostra el contingut dels fitxers.
- Finestra de l'espai de treball: conté una llista de tots els fitxers inclosos al projecte.
- Finestra d'eixida: proporciona informació durant les fases de compilació i assemblatge.

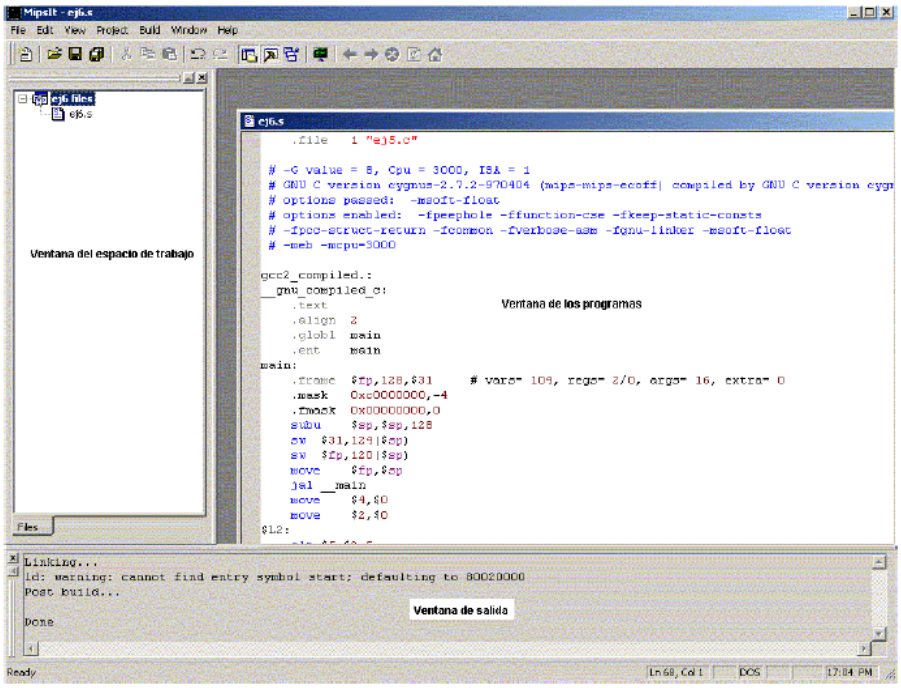


Figura 1. Entorn de finestres del MipsIt.

Configuració

En el menú del fitxer, submenú d'opcions, es pot configurar el MipsIt Studio 2000. En concret, es poden especificar els directoris on hi ha els fitxers executables (opció *bin*), els fitxers de llibreria (opció *lib*) i els de capçalera (opció *include*) de l'aplicació. També es pot indicar la localització del compilador (*bin/xgcc.exe*).

Creació d'un projecte

Els passos que cal seguir per a crear un projecte amb el MipsIt són els següents:

1. Seleccionar en el menú la creació d'un nou fitxer (*File -> New*). S'ha de seleccionar l'opció projecte (*Project*) si no està seleccionada per defecte.
2. Escollir el tipus de projecte que es vol crear. Hi ha tres possibilitats:
   a) Codi assemblador *(assembler)*: el projecte només conté fitxers en assemblador.
   b) Codi mixt C i assemblador (*C/assembler*): hi haurà només fitxers en llenguatge C o fitxers en C i en assemblador. Els programes amb aquesta opció no es poden simular amb Mips.
   c) *C(minimal)/assembler*: Igual que en el cas anterior, però es fa ús només dels conceptes bàsics de llibreries imprescindibles. Els programes amb aquesta opció sí que es poden simular amb Mips.
3. S'ha d'indicar el nom del projecte (*Project name*) i s'ha d'especificar la ubicació del projecte en l'arbre de directoris (*Location*).
4. Finalment, es prem *Aceptar* a la finestra que apareix a la figura 2.

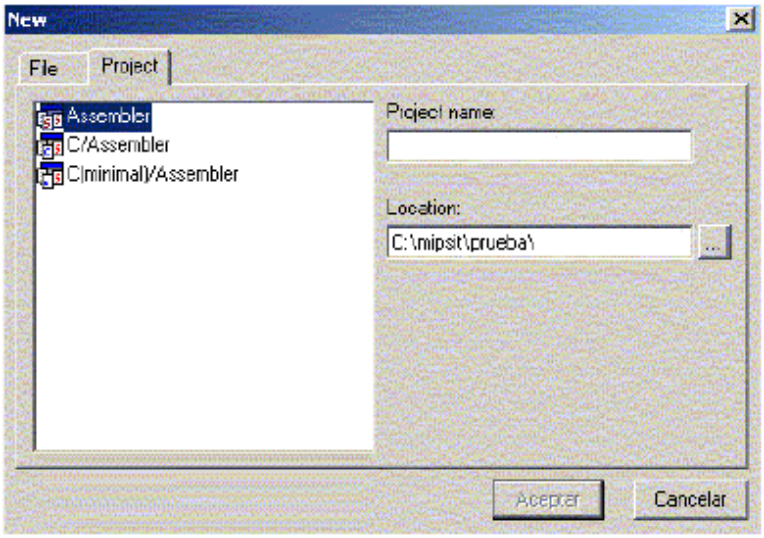


Figura 2. Menú per a crear un nou projecte.

Afegir fitxers nous a un projecte

Amb els passos anteriors tenim un projecte buit, sense cap fitxer. Per a crear fitxers a fi d'afegir-los al projecte, se segueixen els mateixos passos que abans però, en el primer pas, s'ha de triar l'opció fitxer.

Afegir fitxers existents a un projecte

En aquest cas, dins el menú projecte s'ha d'escollir l'opció d'afegir un fitxer (*Add file*). Apareix una finestra de diàleg en la qual es tria el fitxer que es vol afegir al projecte, com es mostra a la figura 3.

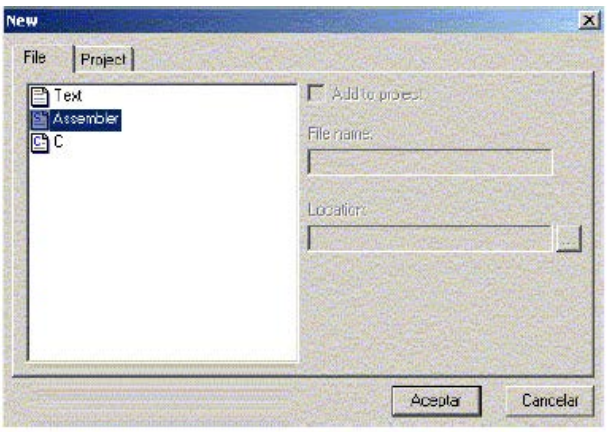


Figura 3. Menú per a afegir un nou fitxer a un projecte.

Compilació i enllaçament

Si es tria el menú de construcció de l'executable (*Build->Build*), es compilen tots els fitxers en llenguatge C i s'enllacen tots els fitxers del projecte. En la finestra d'eixida apareixeran missatges d'informació sobre l'èxit del procés, si hi ha errors, etc. Si es vol compilar tot el projecte, s'ha de triar l'opció que indica aquesta acció del menú principal de construcció (*Rebuild all*).

Assemblador generat per als fitxers en llenguatge C

Si es tenen programes escrits en llenguatge C, es pot veure quin és el programa equivalent en assemblador generat per MipsIt. Per això, una vegada compilat el programa, s'ha d'obrir el fitxer escrit en C i seleccionar l'opció de veure l'assemblador (*Build-> View assembler*).

Simulador de MIPS (Mips.exe)

Introducció

Si no es disposa d'una targeta programable amb el processador MIPS, es necessita un simulador de MIPS per a comprovar el funcionament dels programes escrits. El `Mips.exe` és un entorn gràfic que permet veure en tot moment l'estat de la memòria, els registres del processador, la consola d'eixida, etc. a mesura que aquest programa es va executant. Si seleccionem la icona del fitxer `Mips.exe`, apareix la finestra amb els mòduls que es mostra a la figura 4.

- Microprocessador MIPS R2000 (CPU).
- Memòria principal (RAM).
- Consola d'entrada i eixida estàndard (Console)
- Port d'entrada/eixida de 8 bits (I/O)
- Memòria cau separada de dades i d'instruccions (*D-cache* i *I-cache*)

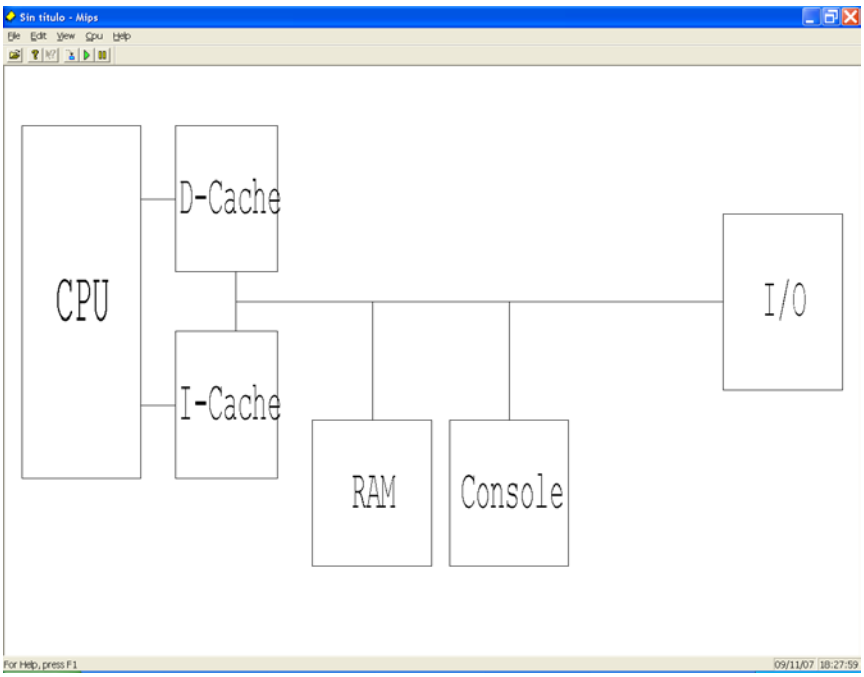


Figura 4. Esquema dels components del simulador del MipsIt.

Es pot obrir cadascun d'aquests mòduls fent-hi clic. A continuació es descriuen amb més detall els que s'usen en les sessions de laboratori.

Microprocessador MIPS R2000

Permet veure i modificar el contingut dels registres del processador, incloent-hi el comptador de programa (PC), els registres de multiplicació i divisió, que reben els noms de *mdhi* i *mdlo* respectivament, com també els registres del coprocessador 0 (`bad va`, `status`, `cause` i `epc`), encarregat de controlar les excepcions i la memòria virtual. L'aparença gràfica es mostra a la figura 5:

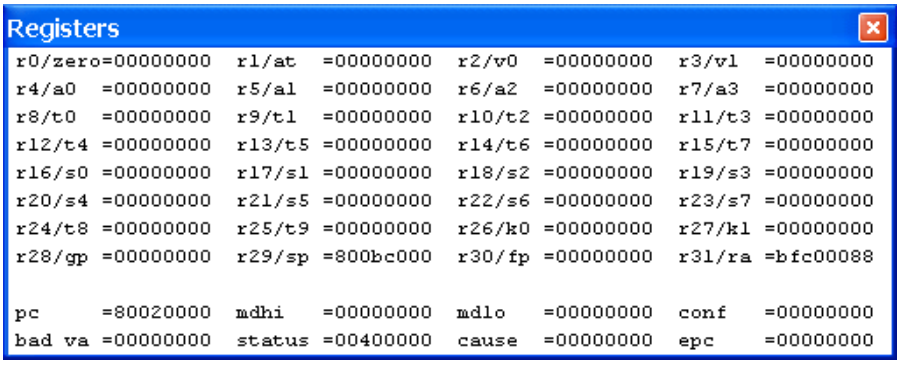


Figura 5. Registres del processador MIPS.

Memòria principal (RAM)

Si fem clic sobre la memòria apareix una finestra amb el contingut de la memòria per files. Cadascuna de les files porta especificada una adreça (*address*), al costat de la qual apareix el

contingut d'aquesta (*content*). Els dos valors apareixen en hexadecimal. Si hi ha una etiqueta que identifica aquesta instrucció, apareix a la columna etiqueta (*label*). En l'última columna apareix l'equivalent en llenguatge assemblador d'aquesta instrucció. No és obligat que les instruccions en assemblador que apareixen a la memòria coincidisquen exactament amb les que s'han introduït al MipsIt, a causa de l'ús de pseudoinstruccions o si l'assemblador reordena les instruccions amb finalitats d'optimització. El contingut de les posicions de memòria que no tenen dades assignades es mostren mitjançant interrogants.

La figura 6 mostra l'estructura de la memòria RAM:

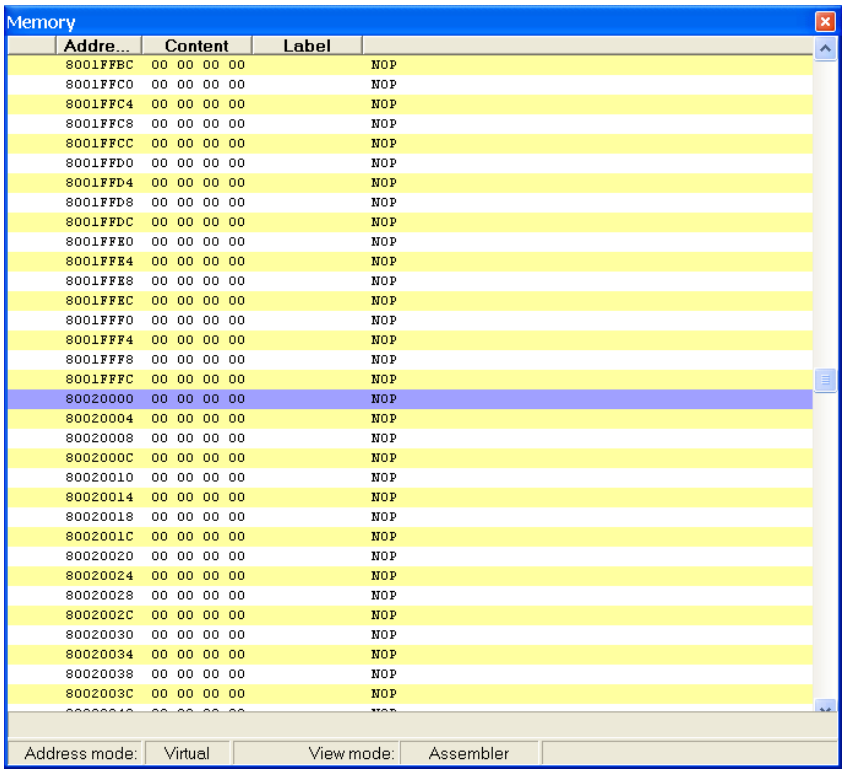


Figura 6. Contingut de les posicions de memòria que pot adreçar el MIPS.

Com que la memòria del MIPS és de 4 GiB, moure's per la memòria amb les fletxes de la finestra o la barra d'espaiat és molest i lent. Per això és millor usar el menú contextual que apareix prement el botó dret del ratolí.

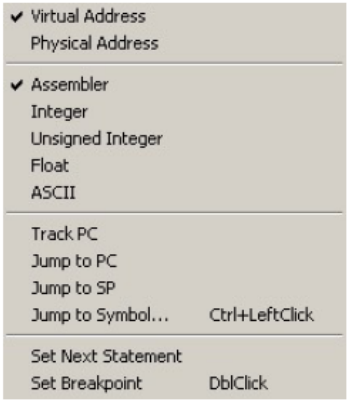


Figura 7. Opcions per a facilitar la visualització de la memòria.

El contingut d'aquest menú es divideix en quatre seccions:

1. La primera permet seleccionar entre mostrar les adreces de la memòria virtual (que pot adreçar el MIPS) o físiques (implementades realment).
2. La segona permet triar la manera de traducció del contingut de la memòria de cada instrucció. Per defecte, s'empra l'opció d'assemblador que presenta el contingut de la memòria en llenguatge assemblador. També es pot triar interpretar els nombres com si foren enters, enters sense signe, en coma flotant o en ASCII.
3. La tercera secció permet moure's fàcilment per la memòria. Les opcions són les següents:
    - Segueix el comptador de programa (*track PC*).
    - Salta a la posició de memòria apuntada per PC (*jump to PC*).
    - Salta a la posició de memòria indicada pel punter de pila (*jump to SP*).
    - Salta a la posició de memòria indicada per una etiqueta (*jump to symbol*).
4. La quarta secció permet modificar el valor del comptador de programa en especificar quina serà la pròxima instrucció que s'executarà (*set next statement*) i permet establir un punt de ruptura per a tasques de depuració del programa (*set breakpoint*). També es poden establir punts de ruptura directament clicant sobre la instrucció que interessa. El punt de ruptura s'identifica mitjançant un cercle a l'esquerra de la instrucció.

Consola d'entrada i eixida estàndard (*console*)

Proporciona una entrada/eixida estàndard per als programes. Ací es poden visualitzar resultats del programa mitjançant la funció *printf*.

Figura 8. Consola del simulador del MipsIt.

Port d'entrada/eixida de 8 bits (*I/O*)

Simula una unitat d'entrada/eixida de vuit bits. Inclou vuit interruptors (entrades) i vuit leds (eixides). El valor dels interruptors es pot llegir en accedir a la posició `0xBF900000;` i el valor d'eixida dels leds es pot modificar escrivint també sobre aquesta posició.

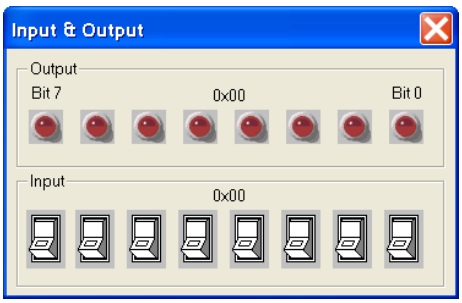


Figura 9. Entrada/eixida de vuit bits.

Interrupcions

Simula la unitat d'interrupcions, amb dos botons, K1 i K2 i amb dos temporitzadors. Habilitant el temporitzador extern es pot simular l'entrada d'un senyal periòdic. La freqüència es pot ajustar amb la barra de desplaçament que apareix més avall.

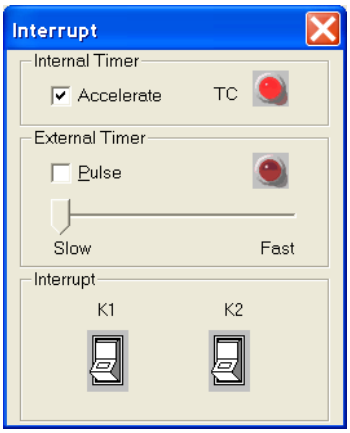


Figura 10. Menú de les interrupcions del MIPS.

Càrrega d'un programa en assemblador al simulador

Si es tenen oberts alhora els programes MipsIt.exe i Mips.exe, es compila un programa i, si no hi ha errors, es pot transferir el programa en assemblador del MipsIt al simulador sense més que escollint l'opció adequada al menú de construcció: pujar al simulador (*Build>Upload>To Simulator*). Si el simulador no està obert, l'operació per a pujar-lo al simulador falla (*Failed to upload to simulator*).

Una altra forma és prémer directament `F5` per a pujar el fitxer al simulador.

Finalment, també es pot seleccionar al menú del simulador, el fitxer amb extensió `.out` que MipsIt ha d'haver creat.

Simulació de programes

Si el programa ja està carregat a la memòria RAM, se'n pot simular el funcionament. Hi ha tres botons que permeten controlar l'execució del programa:

- Execució (*run)*: permet que el programa se simule de principi a fi (o fins al primer punt de ruptura). En el cas d'haver-se detingut el programa amb el botó de pausa o mitjançant un punt de ruptura, permet reprendre'n l'execució.
- Parada (*stop)*: permet fer una pausa en l'execució.
- Pas (*step)*: permet executar les instruccions del programa en codi màquina, pas a pas.

Mapa de memòria del sistema

Cal destacar que la memòria del sistema i els dispositius d'entrada/eixida comparteixen el mateix espai d'adreçament. En aquest tipus d'arquitectura, les instruccions per a llegir i escriure en les posicions de memòria són les mateixes que les que s'usen per a llegir i escriure en els dispositius d'entrada/eixida.

Les posicions assignades a cada component de la memòria són:

0x80000000-0x800FFFFF	1 MiB SRAM (memòria RAM estàtica).
0xA0000000-0xA00FFFFF	1 MiB SRAM sense memòria cau,[i] mateixa memòria física que 0x80000000-0x800FFFFF.
0x80400000-0x807FFFFF	4 MiB DRAM (memòria RAM dinàmica).
0xA0400000-0xA07FFFFF	4 MiB DRAM sense memòria cau, mateixa memòria física que 0x80400000-0x807FFFFF.
0x80020000	Adreça d'inici per defecte (on comencen per defecte els programes).
0xBF900000	Interruptors/led (vuit bits).
0xBFA00000	Interrupcions entrada/eixida (vuit bits). Bit: 0 – K2 (entrada): valor instantani de l'interruptor K2. 1 – K1 (entrada): valor instantani de l'interruptor K1. 2 – Temporitzador (entrada): valor instantani d'activació del temporitzador. 3 – N/A (no definit). 4 – Registre de K2: valor capturat de K2 en produir-se la interrupció. L'usuari ha de posar-lo a zero. 5 – Registre de K1: valor capturat de K2 en produir-se la interrupció. L'usuari ha de posar-lo a zero. 6 – Registre del temporitzador: valor capturat d'activació del temporitzador. L'usuari ha de posar-lo a zero. 7 – N/A (no definit).
0xBFB00000	16-bit entrada/eixida (no implementat al simulador Mips).

[i] Les posicions de memòria *sense memòria cau* coincideixen amb les posicions de memòria de referència i són, per tant, la mateixa memòria física. La diferència és que quan s'accedeix a aquestes posicions, l'accés es fa directament a la memòria i no es modifica el contingut de la memòria cau del sistema (les dades no són portades a la memòria cau).

Lab 2

Laboratori 2
Memòria cau (part I)

(Actualitzat el dia 16/09/2021)

Temps de treball a casa (previst): 1 hora i 30 minuts
Temps de laboratori: 2 hores i 30 minuts

1. Introducció

En aquesta segona pràctica s'estudia el sistema de memòria cau i com funciona quan executa programes senzills amb patrons d'accés a memòria característics. S'hi estudia el comportament de la memòria cau segons l'algorisme de mapatge emprat. A més a més, proposem en aquesta pràctica certes tècniques que poden ajudar a fer programes més eficients, des del punt de vista de la taxa de fallades, quan s'executen en un sistema amb memòria cau.

Es comença amb una descripció de la memòria cau del MipsIt i les opcions de configuració que permet el simulador. A continuació s'usa un programa senzill per a veure el procés de compilació i adequació del codi perquè els experiments realitzats siguen correctes. A partir d'aquest moment ja s'està en disposició de fer diversos experiments modificant les característiques de la memòria cau.

Objectius

L'estudiantat, en acabar aquesta pràctica, ha de ser capaç de:

- Conèixer les possibles configuracions de la memòria cau del MipsIt.
- Configurar el mòdul de la memòria cau del simulador MipsIt.
- Adaptar un programa en C per a avaluar l'ús de la memòria cau en el MipsIt.
- Avaluar l'ús de la memòria cau per a un programa senzill.
- Determinar les causes del funcionament de la memòria cau a partir dels patrons d'accés.

2. Treball previ al laboratori

En aquest treball previ, tots els membres de l'equip de treball han de llegir l'annex I d'aquest document, on es descriu el mòdul de la memòria cau del simulador MipsIt, i l'annex II, on es descriuen els passos que cal seguir per a adaptar el programa exemple que s'empra en la pràctica per a comprovar el funcionament de la memòria cau. S'han de revisar també els conceptes sobre la memòria de l'ordinador vistos fins ara en el tema 1. Per a certes qüestions cal consultar alguna de les fonts bibliogràfiques indicades al temari.

Una vegada llegits els annexos I i II, cal respondre a les qüestions següents:

1. Indica l'arquitectura (Von Neumann o Harvard) que té l'ordinador simulat tant en la memòria cau com en la memòria principal.

2. Explica el significat dels paràmetres de configuració de la memòria cau següents: grandària (*size*), grandària de bloc (*block size*) i nombre de vies en cada conjunt (*block in sets*).
3. En el programa en llenguatge C descrit a l'annex II s'ha d'indicar el nombre de posicions de memòria que ocupa cada element de les matrius. Com ha de ser la posició d'inici de cada element de les matrius? Quantes posicions separen l'inici de dos elements contigus de les matrius?
4. En el programa exemple, indica si els elements contigus (que ocupen posicions consecutives) en una matriu són els elements de la mateixa fila o de la mateixa columna.
5. Quina és la mida total (en *bytes*) de les matrius en la memòria?
6. En el programa exemple, els bucles serveixen per a recórrer les matrius `m1` i `m2`. Indica si amb els índexs utilitzats el bucle intern recorre la matriu per files o per columnes. També s'ha d'explicar si en cada iteració del bucle intern, els elements següents en les matrius estan situats en posicions de memòria contigües o no.
7. En el programa assemblador de l'annex II obtingut com a resultat de la compilació del codi C s'ha d'esbrinar quines instruccions del codi assemblador són les que efectuen l'accés a les posicions de les matrius (lectura de `m1[i][j]` i escriptura de `m2[i][j]`).

Primer punt de control: s'han de lliurar per escrit, a l'inici de la sessió, les preguntes del treball previ i respondre a les preguntes del professor o de la professora.

3. Treball al laboratori

En aquesta sessió es comprova de forma pràctica el comportament de la memòria cau del MipsIt per a diversos programes i configuracions.

3.1. Utilitzant el programa exemple de l'annex II, tal com s'ha modificat, i configurant la memòria cau de dades (*D-cache*) amb les característiques següents:

- Grandària: 16
- Grandària de bloc: 4
- Amb escriptura diferida (*write-back*)
- Algorisme de reemplaçament: LRU

- S'ha d'indicar la seqüència d'accessos (adreces de memòria expressades en hexadecimal) que realitza el programa sobre la memòria quan accedeix a les variables `m1` i `m2`. Per a fer-ho, s'ha d'emplenar la taula completa adjunta. S'ha d'enumerar també el rang de posicions de memòria que ocupa cadascuna d'aquestes variables expressades en hexadecimal.

Per a saber quines adreces ocupen aquestes variables, cal executar el programa pas a pas o posar un punt de parada. A continuació s'han de comprovar les adreces sobre la memòria de dades, a les quals accedeixen les instruccions que llegeixen la variable `m1` i que escriuen sobre `m2`, respectivament. (L'adreça apareix a la finestra de la memòria cau de dades dos cicles després que la instrucció s'haja executat).

Núm. d'accés	*i*	*j*	Adreça de `m1`	Adreça de `m2`
1				
2				
…	…	…	…	…

15				
16				

- Indiqueu per a cada algorisme de mapatge quin és el camp d'etiqueta, quins bits s'usen com a índex, els bits de selecció de paraula en una línia i els bits de selecció de *byte*.

Algorisme de mapatge (variable *blocks in set*)	**Bits d'etiqueta**	**Bits d'índex**	**Bits de paraula**	**Bits de *byte***
Directe (1)				
Associatiu per conjunts 2 vies (2)				
Totalment associatiu (4)				

- S'han de fer simulacions amb els algorismes de mapatge que s'indiquen a continuació i, també, completar la taula següent amb el nombre total de fallades i la taxa de fallades obtinguda de les simulacions:

Algorisme de mapatge	**Nombre total de fallades**	**% de fallades**
Directe (1)		
Associatiu per conjunts: 2 vies (2)		
Totalment associatiu (4)		

Segon punt de control: s'ha de cridar el professor o la professora a fi de mostrar-li les taules i ser capaç de respondre a les preguntes sobre el seu contingut.

3.2. En aquest segon apartat es modifica el programa per a millorar-ne el rendiment. En aquest cas, el programa intercanvia els bucles que s'usen per a recórrer les matrius. L'objectiu és que la matriu es recórrega en el mateix ordre en el qual està emmagatzemada a la memòria, la qual cosa augmentarà la proximitat de referències espacials del programa. Per a fer-ho es modifiquen els bucles de la manera següent:

```
#define NUM1 4
int main()
{
    register int i,j;
    int m1[NUM1][NUM1];
    int m2[NUM1][NUM1];
     for(i=0;i<NUM1;i++)
        for(j=0;j<NUM1;j++)
         m2[i][j]=m1[i][j];
}
```

El programa s'ha de compilar per a obtenir-ne el codi assemblador, eliminar les línies sobrants i fer les mateixes modificacions que en l'apartat anterior. Una vegada obtingut el codi final

executable, es carrega al simulador i es repeteixen les simulacions per als mateixos tipus de memòria cau de dades que en el cas anterior.

- S'ha d'indicar la seqüència d'accessos a la memòria que es realitza en aquest cas quan s'accedeix a les variables `m1` i `m2`. S'ha d'emplenar una taula com la d'abans.
- Com abans, s'ha de completar la taula següent amb el nombre total de fallades i la taxa de fallades:

Algorisme de mapatge	Nombre total de fallades	% de fallades
Directe (1)		
Associatiu per conjunts: 2 vies (2)		
Totalment associatiu (4)		

- Per a quins algorismes de mapatge funciona millor aquest algorisme?

3.3. Una vegada obtingudes les dades experimentals en la sessió de laboratori, responeu de forma raonada a les preguntes següents:

- S'ha d'explicar la raó per la qual la taxa de fallades per al segon algorisme disminueix en comparació amb el primer algorisme. (Ajuda: per a esbrinar la raó per la qual aquesta tècnica funciona millor, s'ha de veure la seqüència d'accessos i l'estat de la memòria cau després de cada iteració dels bucles)
- Existeix algun algorisme de mapatge que no millore la seua taxa de fallades per al segon algorisme? Se n'ha d'explicar la raó. (Ajuda: s'ha de comprovar quina línia de la memòria cau tenen assignada els blocs de diferents matrius en una mateixa iteració del bucle).
- S'ha de repetir la simulació realitzada amb mapatge associatiu per conjunts de l'apartat 3.2 amb el programa següent, que suma tres matrius:

```
#define NUM1 4
int main()
{
    register int i,j;
    int m1[NUM1][NUM1];
    int m2[NUM1][NUM1];
    int m3[NUM1][NUM1];
int m4[NUM1][NUM1];
     for(i=0;i<NUM1;i++)
        for(j=0;j<NUM1;j++)
             m4[i][j]=m1[i][j]+m2[i][j]+m3[i][j];
}
```

S'han de donar els resultats del procés, comparar-los respecte a l'algorisme original i proposar una explicació per a aquest comportament. Finalment, s'ha de repetir la simulació del programa de la suma de tres matrius amb l'algorisme de reemplaçament aleatori. Hi ha cap canvi? Per què?

Tercer punt de control: s'ha de cridar el professor o la professora, mostrar-li els resultats dels punts 3.2 i 3.3 i ser capaç de respondre a les preguntes sobre aquests resultats.

Annex I. Mòdul cau del MipsIt

Introducció

La memòria cau del microprocessador MIPS posseeix una arquitectura Harvard, amb memòries d'instruccions (*I-Cache*) i dades (*D-Cache*) independents, encara que la memòria principal segueix una arquitectura Von Neumann.

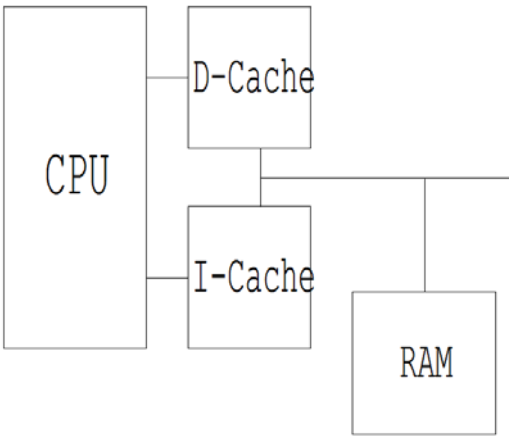


Figura 1. Esquema de la jerarquia de memòria del MIPS.

La figura 2 mostra el contingut de la memòria cau de dades quan es fa clic sobre el mòdul en el simulador del MipsIt. Aquesta informació és similar per a la memòria cau d'instruccions.

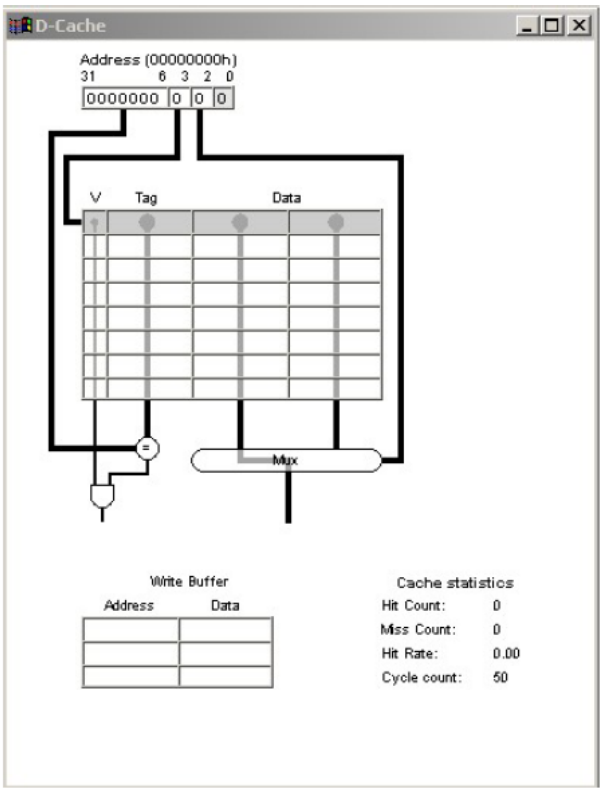


Figura 2. Contingut de la memòria de dades del simulador.

Aquest esquema conté quatre parts:

- Adreça (*address*). Dividida en els camps següents: etiqueta (*tag*), línia o conjunt i paraula. L'estructura de l'adreça canvia segons la configuració de la memòria cau. En l'esquema de la figura 2:
  - Etiqueta (*tag*): bits del 6 al 31.
  - Línia: bits del 3 al 5.
  - Paraula: correspon als dos últims camps (bits del 0 al 2), ja que a la memòria cau s'emmagatzemen paraules de quatre *bytes* i el processador adreça *byte* a *byte*. Els dos bits menys significatius (0 i 1) identifiquen un *byte* dins de la paraula, però són irrellevants per a l'accés a la memòria cau. Els altres bits del camp paraula fan referència a les paraules de quatre *bytes* que formen cada línia. En aquest cas, el bit 2 identifica les dues paraules de quatre *bytes* que formen cada línia.
- Esquema de la memòria cau: la línia seleccionada (sobre la base del camp línia o conjunt) apareix en color grisós. La paraula seleccionada mitjançant el camp paraula apareix en color blau. La columna V indica si aquesta línia és vàlida o no. Per a una memòria cau amb l'escriptura retardada (*WB*), apareix una columna extra que indica si la línia coincideix o no amb el valor emmagatzemat a la memòria principal. Aquesta columna apareixeria marcada amb la lletra D per a indicar que la línia està bruta (*dirty*).
- Memòria intermèdia d'escriptura (*write buffer*): apareix en el cas que la mida siga diferent de zero. Es pot configurar tant per a escriptura directa (*WT*) com retardada.
- Estadístiques de la memòria cau: mostra el nombre d'encerts de memòria cau (*hit count*), el nombre de fallades de memòria cau (*miss count*) i el percentatge d'encerts (*hit rate*).

Configuració de la memòria cau

Per a configurar la memòria cau s'ha de triar l'opció d'edició, configuració de memòria/cau (*Edit>Cache/Mem config*), i el programa mostra la finestra següent:

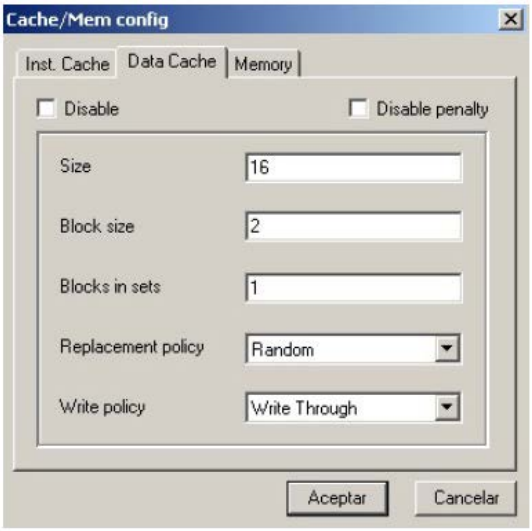


Figura 3. Menú de configuració de la memòria cau.

Es pot triar la configuració de la memòria cau de dades i d'instruccions. La memòria cau de dades té les opcions següents:

- Grandària de la memòria cau en paraules (*size*).
- Grandària del bloc (*block size*): nombre de paraules que formen cada bloc.
- Grau d'associativitat (*block in sets*).

  Cal tenir en compte:

  - La paraula sempre és de quatre *bytes.*
  - El nombre de línies de la memòria cau es pot calcular fent la divisió següent:

    Grandària de la memòria cau / Grandària del bloc
  - Aquestes línies es poden agrupar en conjunts.
  - El grau d'associativitat identifica el nombre de línies (o vies) per conjunt.
- Política de reemplaçament *(replacement policy)*. Es pot escollir entre tres possibles tècniques de reemplaçament de les línies: aleatòria; primer d'entrar, primer d'eixir (FIFO); o el que fa més temps que no s'usa (LRU).
- Política d'escriptura (*write policy)*. S'escull entre dues possibles tècniques d'escriptura: escriptura retardada (WB) o escriptura directa (WT).

La memòria cau es pot desactivar simplement seleccionant aquesta opció en la casella que ho indica. De manera anàloga, també es pot desactivar la penalització que comporta una fallada en la memòria cau. La configuració de la memòria cau d'instruccions és similar excepte que no hi ha l'opció de política d'escriptura. La configuració de la memòria és la que apareix en la figura 4:

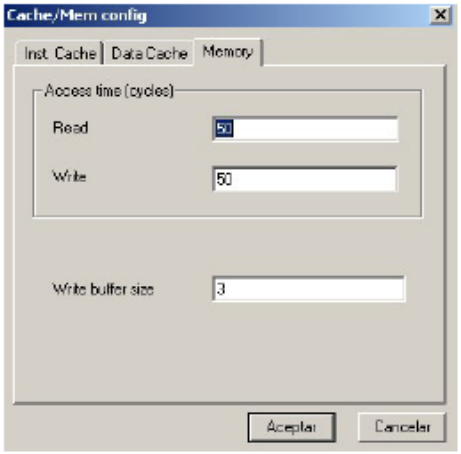


Figura 4. Configuració de la penalització d'accés a la memòria principal.

En aquesta finestra s'especifiquen els cicles que penalitzen les fallades de memòria cau en les operacions de lectura i escriptura, com també la grandària de la memòria intermèdia d'escriptura (*write buffer size*). També es poden visualitzar les estadístiques de la memòria cau d'instruccions i de dades respectivament. A continuació es presenten algunes configuracions de la memòria cau a manera d'exemple.

Exemple de configuració 1

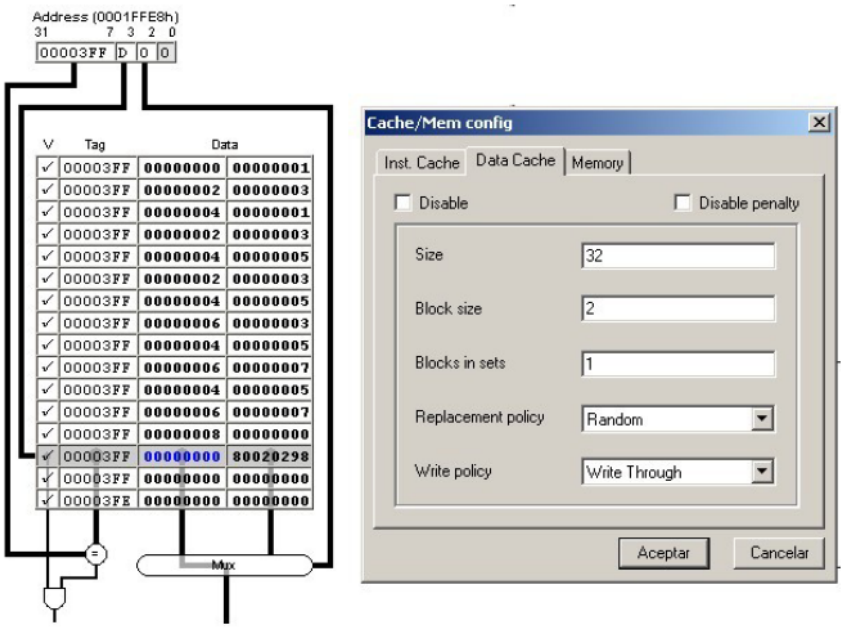


Figura 5. Exemple de configuració de memòria cau de dades (mapatge directe).

En aquest primer exemple es representa una memòria cau de dades amb les característiques següents:

- Grandària de la memòria cau: 32 paraules de quatre *bytes*, en total 128 *bytes*.
- Grandària de bloc: dues paraules en cada línia. En total, vuit *bytes* en cada línia.
- Funció de correspondència: mapatge directe o, de manera equivalent, cada conjunt té una línia (associativitat d'1).

D'acord amb la configuració de la memòria cau, l'adreça de memòria principal consta dels camps següents:

- Etiqueta: bits del 7 al 31.
- Línia: bits del 3 al 6 (16 línies = 128/8).
- Paraula: dues paraules (bit 2) de quatre *bytes* cadascuna (bits 0 i 1). Adreçable *byte* a *byte*.

A més, la tècnica de reemplaçament és aleatòria, encara que el fet d'usar mapatge directe no influeix en el funcionament. Finalment, la política d'escriptura és d'escriptura directa (WT).

Exemple de configuració 2

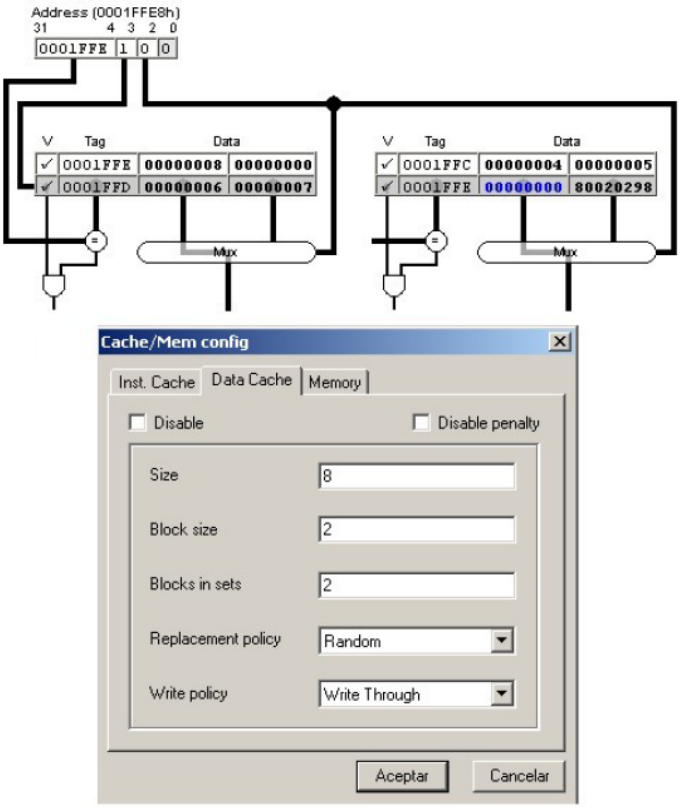


Figura 6. Exemple de configuració de memòria cau de dades (mapatge associatiu per conjunts).

En aquest exemple es representa una memòria cau de dades amb les característiques següents:

- Grandària de la memòria cau: vuit paraules de quatre *bytes*; en total 32 *bytes*.
- Grandària de bloc: dues paraules en cada línia; en total vuit *bytes*.
- Funció de correspondència: mapatge associatiu per conjunts (dos blocs per conjunt).

D'acord amb la configuració de la memòria cau, l'adreça de memòria principal consta dels camps següents:

- Etiqueta: bits del 4 al 31.
- Conjunt: bit 3 (dos conjunts).
- Paraula: dues paraules (bit 2) de quatre *bytes* cadascuna (bits 0 i 1). Adreçable *byte* a *byte*.

A més, la tècnica de reemplaçament és aleatòria i la política d'escriptura és d'escriptura directa (WT).

Annex II. Codi d'exemple

Introducció

En aquest annex es descriuen les modificacions que s'han de fer sobre els programes emprats, en aquesta pràctica i en la següent, per a poder comprovar de forma adequada el comportament de la memòria cau. En primer lloc, els programes s'elaboren en llenguatge C, posteriorment es compilen i després es modifica l'assemblador creat pel compilador. Així s'eliminen instruccions addicionals que provoquen accessos a la memòria fora de les matrius. En conseqüència, es pot executar sols el codi pur de l'algorisme que volem estudiar.

Programa per executar

En aquesta sessió es proposa estudiar el programa exemple següent (`p3.c`) escrit en llenguatge C:

```
#define NUM1 4
int main()
{
    register int i,j;
    int m1[NUM1][NUM1];
    int m2[NUM1][NUM1];
     for(j=0;j<NUM1;j++)
        for(i=0;i<NUM1;i++)
  m2[i][j]=m1[i][j];
}
```

En la pràctica, es vol obtenir la taxa de fallades de la memòria cau de dades en executar el programa anterior. Aquest programa efectua l'assignació de dues matrius: `m2` = `m1`. El programa exemple (`p3.c`) s'ha de compilar en l'entorn MipsIt. El resultat de la compilació és un fitxer assemblador que es pot visualitzar amb l'opció de veure l'assemblador en el menú de construcció (*Build →View assembler*) i que conté el codi següent:

```
.file   1 "p3.c"

 # -G value = 8, Cpu = 3000, ISA = 1
 # GNU C version cygnus-2.7.2-970404 (mips-mips-ecoff) compiled by GNU C version cygnus-2.7.2-
970404.
 # options passed:  -msoft-float
 # options enabled:  -fpeephole -ffunction-cse -fkeep-static-consts
 # -fpcc-struct-return -fcommon -fverbose-asm -fgnu-linker -msoft-float
 # -meb -mcpu=3000

gcc2_compiled.:
__gnu_compiled_c:
        .text
        .align  2
        .globl  main                    #Canviar l'etiqueta main per l'etiqueta start
        .ent    main                    #Canviar l'etiqueta main per l'etiqueta start
main:                                   #Canviar l'etiqueta main per l'etiqueta start
        .frame  $fp,216,$31             # Eliminar aquestes línies
        .mask   0xc0000000,-4
        .fmask  0x00000000,0
        subu    $sp,$sp,216
        sw      $31,212($sp)
```

```
        sw      $fp,208($sp)
        move    $fp,$sp
        jal     __main                  #=================
        move    $3,$0
$L2:
        slt     $4,$3,4
        bne     $4,$0,$L5
        j       $L3
$L5:
        .set    noreorder
        nop
        .set    reorder
        move    $2,$0
$L6:
        slt     $4,$2,4
        bne     $4,$0,$L9
        j       $L4
$L9:
        move    $4,$2
        sll     $5,$4,4
        addu    $4,$fp,16
        addu    $5,$5,$4
        addu    $4,$5,128
        move    $5,$3
        sll     $6,$5,2
        addu    $4,$6,$4
        move    $5,$2
        sll     $6,$5,4
        addu    $7,$fp,16
        addu    $5,$6,$7
        move    $6,$3
        sll     $7,$6,2
        addu    $5,$7,$5
        move    $6,$2
        sll     $7,$6,4
        addu    $6,$fp,16
        addu    $7,$7,$6
        addu    $6,$7,64
        move    $7,$3
        sll     $8,$7,2
        addu    $6,$8,$6
        lw      $5,0($5)
        lw      $6,0($6)
        addu    $5,$5,$6
        sw      $5,0($4)
$L8:
        addu    $2,$2,1
        j       $L6
$L7:
$L4:
        addu    $3,$3,1
        j       $L2
$L3:
$L1:
        move    $sp,$fp                 # Eliminar aquestes línies
        lw      $31,212($sp)
        lw      $fp,208($sp)
        addu    $sp,$sp,216             #=================
        j       $31                     # aquesta línia no s'elimina però
                                        # s'ha de canviar $31 per $L1
        .end    main                    #Canviar l'etiqueta main per l'etiqueta start
```

A fi de simplificar la simulació de l'execució del programa, s'han d'eliminar les línies del codi assemblador que hi ha en negreta i cursiva, així s'aconsegueix executar només el codi resultant dels dos bucles i eliminar accessos extra a la memòria que no es volen comptabilitzar. Les línies eliminades estan pensades per a la crida a la funció `main ()` de C, i comporten emmagatzemar i recuperar informació de la pila de dades del MIPS, que està situada en la memòria. Per tant, aquests accessos comportarien modificacions tant de la memòria cau de dades com de la d'instruccions.

També s'han de canviar les referències a l'etiqueta **`main`** per l'etiqueta **`start`**. Els programes són llançats des de l'adreça etiquetada com `start` que executa per defecte un salt a l'etiqueta `main`, de manera que perquè s'execute el codi en assemblador sense problemes, cal fer-hi les substitucions indicades.

El fitxer resultant quedaria de la manera següent:

```
.file   1 "p3.c"
 # -G value = 8, Cpu = 3000, ISA = 1
 # GNU C version cygnus-2.7.2-970404 (mips-mips-ecoff) compiled by GNU C version cygnus-2.7.2-
970404.
 # options passed:  -msoft-float
 # options enabled:  -fpeephole -ffunction-cse -fkeep-static-consts
 # -fpcc-struct-return -fcommon -fverbose-asm -fgnu-linker -msoft-float
 # -meb -mcpu=3000

gcc2_compiled.:
__gnu_compiled_c:
        .text
        .align  2
        .globl  start
        .ent    start
start:
        move    $3,$0
        $L2:                    #      for(j=0;j<NUM1;j++)
        slt     $4,$3,4
        bne     $4,$0,$L5
        j       $L3
$L5:
        .set    noreorder
        nop
        .set    reorder
        move    $2,$0
        $L6:                    #         for(i=0;i<NUM1;i++)
        slt     $4,$2,4
        bne     $4,$0,$L9
        j       $L4
$L9:
        move    $4,$2           #
        sll     $5,$4,4
        addu    $4,$fp,16
        addu    $5,$5,$4
        addu    $4,$5,128
        move    $5,$3
        sll     $6,$5,2
        addu    $4,$6,$4
        move    $5,$2
        sll     $6,$5,4
        addu    $7,$fp,16
        addu    $5,$6,$7
        move    $6,$3
        sll     $7,$6,2
```

```
        addu    $5,$7,$5
        move    $6,$2
        sll     $7,$6,4
        addu    $6,$fp,16
        addu    $7,$7,$6
        addu    $6,$7,64
        move    $7,$3
        sll     $8,$7,2
        addu    $6,$8,$6
        lw      $5,0($5)
        sw      $5,0($4)
$L8:
        addu    $2,$2,1
        j       $L6
$L7:
$L4:
        addu    $3,$3,1
        j       $L2
$L3:
$L1:    j       $L1
        .end    start
```

A continuació s'ha de desar el fitxer, per exemple amb el nom `p3.s`, i tornar a crear un nou projecte de **tipus assemblador** que continga sols aquest fitxer. El nou projecte s'ha de compilar amb el MipsIt i carregar-lo al simulador. En aquest moment ja es pot simular el comportament del codi per la configuració de la memòria cau escollida.

Lab 3

Laboratori 3
Memòria cau (part II)
Optimitzacions del programari

(Actualitzat el dia 16/9/2021)

Temps de treball a casa (previst): 1 hora i 30 minuts
Temps de laboratori: 2 hores i 30 minuts

1. Introducció

En aquesta pràctica s'estudia el sistema de memòria cau i com es comporta en optimitzar algorismes senzills però que estan relacionats amb les estructures de dades característiques en moltes aplicacions. S'hi estudia el comportament de la memòria cau segons l'algorisme de mapatge emprat i la resposta segons les diverses optimitzacions aplicades.

En la pràctica anterior ja s'ha vist un exemple d'optimització del programari sobre un algorisme que feia la suma de dues matrius. En aquest exemple s'ha vist l'efecte sobre la memòria cau de l'intercanvi dels bucles que s'usen per a recórrer les matrius.

Objectius

L'estudiant, en acabar aquesta pràctica, ha de ser capaç de:

- Conèixer les tècniques d'optimització del codi per a obtenir un millor rendiment de la memòria cau.
- Avaluar el comportament de les tècniques d'optimització més comunes per a diverses configuracions de la memòria cau.
- Aplicar aquestes tècniques d'optimització a diversos algorismes.
- Determinar les causes del funcionament de la memòria cau amb diverses optimitzacions dels algorismes.

Recursos

Els mateixos que els emprats en les dues sessions anteriors (vegeu les memòries P1 i P2).

2. Treball previ al laboratori

Per a poder aprofitar al màxim la sessió de laboratori, és molt important que es facen les activitats que es proposen a continuació. En aquest treball previ tots els membres de l'equip de treball han de llegir l'annex I d'aquest document on es descriuen les tècniques d'optimització del programari més usuals a fi de traure un millor rendiment de la memòria cau. Al mateix temps, s'han de revisar els conceptes sobre la memòria de l'ordinador vistos al tema 1.

Una vegada llegit l'annex I i, si cal, la bibliografia recomanada, s'ha de respondre a les qüestions següents:

Q1. S'han de consultar les tècniques d'optimització de l'annex I. Què intenten millorar les tècniques que modifiquen l'accés a les dades?

Q2. Què intenten millorar les tècniques que modifiquen l'emmagatzematge de les dades?

Q3. S'ha d'explicar quina tècnica d'optimització s'ha emprat en l'algorisme de l'exemple següent:

```
// Algorisme original
#define N 16
int main()
{
  register int i,j,suma;
  int m[N][N];
  for(j=0;j<N;j++)
    for(i=0;i<N;i++)
     suma+=m[i][j];
}
```

```
//Algorisme modificat
#define N 16
int main()
{
  register int k,l,suma;
  int m[N][N];
  for(k=0;k<N;k++)
    for(l=0;l<N;l++)
     suma= suma + m[k][l];
}
```

Q4. S'ha d'explicar en quines situacions l'algorisme modificat no milloraria la taxa de fallades si es té una memòria cau amb mapatge directe.

Q5. Es requereix que s'explique quina tècnica d'optimització s'ha emprat en l'algorisme de l'exemple següent:

```
// Algorisme original
#define N 16
int main()
{
  register int i,j,suma;
  int m1[N][N];
  int m2[N][N];
  for(j=0;j<N;j++)
    for(i=0;i<N;i++)
     suma=m1[j][i]+m2[j][i];
}
```

```
//Algorisme modificat
#define N1 16
#define N2 17
int main()
{
  register int i,j,suma;
  int m1[N1][N2];
  int m2[N1][N2];
  for(j=0;j<N1;j++){
    for(i=0;i<N1;i++)
     suma= m1[j][i]+ m2[j][i];
  }
}
```

Q6. S'ha d'explicar quin tipus de memòria cau es beneficia més d'aquest últim tipus d'optimització.

Q7. Finalment, s'ha d'explicar quins tipus de memòria cau es beneficien més de l'optimització per combinació de matrius.

Primer punt de control: les preguntes del treball previ s'han de lliurar per escrit a l'inici de la sessió i respondre a les preguntes del professor o professora.

3. Treball al laboratori

En aquesta sessió es comprova de forma pràctica, utilitzant el programa que s'ha usat en la sessió anterior (suma de matrius), el comportament de la memòria cau amb diverses optimitzacions. Per això s'han de portar a aquesta sessió els resultats obtinguts en la sessió anterior: taules de fallades per diversos mapatges, etc.

3.1. Usant com a base l'algorisme de suma de matrius que es mostra a continuació, s'ha de modificar mitjançant la tècnica de l'**emplenament de matrius** (vegeu l'annex I).

```
#define NUM 4

int main()
{
    register int i,j; /*les variables s'emmagatzemen en registres*/
int m1[NUM][NUM];
    int m2[NUM][NUM];
int m3[NUM][NUM];
    int m4[NUM][NUM];

for(i=0;i<NUM;i++)
        for(j=0;j<NUM;j++)
m4[i][j]=m1[i][j]+m2[i][j]+m3[i][j];
}
```

S'ha de compilar el programa, obtenir-ne el codi assemblador i eliminar les línies sobrants per a evitar d'aquesta manera accessos de memòria fora de la suma de matrius, com s'ha fet en la pràctica anterior. Una vegada obtingut el codi final, s'ha de carregar al simulador després de configurar la memòria cau de dades amb els paràmetres següents:

- Grandària completa: 16
- Grandària de bloc: 4
- Amb escriptura diferida (WB)
- Algorisme de reemplaçament: l'element que fa més temps que no s'usa (LRU).

- Indicar la seqüència d'accessos, les adreces de memòria s'escriuen en hexadecimal, que realitza el programa sobre la memòria quan accedeix a les variables `m1`, `m2`, `m3` i `m4`. Per a fer-ho, s'ha d'emplenar la taula adjunta. S'ha d'indicar també el rang de posicions de memòria que ocupa cadascuna d'aquestes variables expressades en hexadecimal.

Núm. d'accés	i	j	Adreça de `m1`	Adreça de `m2`	Adreça de `m3`	Adreça de `m4`
1						
2						
3						
…	…	…	…	…	…	
14						
15						
16						

- S'han de fer simulacions de memòries cau amb diversos algorismes de mapatge i s'ha de completar la taula següent amb el nombre total de fallades i la taxa de fallades obtinguda. Tot açò s'ha de fer com a resultat de les simulacions per als dos algorismes (l'original i el millorat) amb l'emplenament de matrius:

Algorisme de mapatge (variable *blocks in set*)	**Nombre total de fallades**	**% de fallades**
Directe (1)		
Associatiu per conjunts: 2 vies (2)		
Totalment associatiu (4)		

- Per a cada algorisme de mapatge, s'han de comparar i interpretar els resultats obtinguts entre l'algorisme original i el millorat amb la tècnica d'emplenament de matrius.

S'ha de canviar, per al codi original, l'algorisme de reemplaçament de la memòria cau per l'aleatori. S'han de fer les simulacions i completar la taula següent amb el nombre total de fallades i la taxa de fallades:

Algorisme de mapatge	**Nombre total de fallades**	**Taxa de fallades**
Associatiu per conjunts: 2 vies (2)		
Totalment associatiu (4)		

Q8. Quines són les diferències que s'observen entre l'algorisme de reemplaçament LRU (el que fa més temps que no s'usa) i l'aleatori?

Segon punt de control: s'ha de cridar el professorat del laboratori, mostrar-li les taules i respondre a les preguntes sobre el seu contingut.

3.2. En aquest segon apartat s'ha de modificar el programa original usant la tècnica de **combinació de matrius**. L'execució s'ha de simular seguint els mateixos passos que en l'apartat 3.1.

- S'ha d'indicar la seqüència d'accessos (adreces de memòria expressades en hexadecimal) que realitza el programa sobre la memòria quan accedeix a les variables `m[][].m1`, `m[][].m2`, `m[][].m3` i `m[][].m4`. Per a fer-ho, s'ha d'emplenar la taula adjunta per a tots els valors. S'ha d'indicar també el rang de posicions de memòria que ocupa cadascuna d'aquestes variables expressades en hexadecimal.

Núm. d'accés	*i*	*j*	Adreça de `m1`	Adreça de `m2`	Adreça de `m3`	Adreça de `m4`
1						
2						
3						
…	…	…	…	…	…	
14						
15						
16						

- Utilitzant la configuració de la memòria cau proposada en l'apartat anterior, s'han de fer simulacions amb memòries cau amb diversos algorismes de mapatge i s'ha de completar la taula següent amb el nombre total de fallades i la taxa de fallades obtinguda com a resultat de les simulacions:

Algorisme de mapatge	**Nombre total de fallades**	**% de fallades**
Directe (1)		
Associatiu per conjunts: 2 vies (2)		
Totalment associatiu (4)		

Q9. Per a cada algorisme de mapatge s'han de comparar i interpretar els resultats obtinguts amb l'algorisme de combinació de matrius i l'algorisme original.

Q10. S'ha d'indicar si té alguna influència en aquest cas que l'algorisme de reemplaçament siga LRU o aleatori. Per a fer això, s'han de fer simulacions amb memòries cau amb aquest algorisme de mapatge i explicar els resultats obtinguts.

Tercer punt de control: s'ha de cridar el professorat del laboratori, mostrar-li les taules amb els resultats i respondre a les preguntes sobre el contingut.

S'ha de lliurar per escrit la resposta a les qüestions del treball previ (Q1 fins a Q7), i també les qüestions addicionals Q8, Q9 i Q10.

Annex I. Tècniques bàsiques d'optimització del programari

Introducció

La jerarquia de memòria permet ocultar tant la baixa amplada de banda com la latència de la memòria principal. Els dos paràmetres són petits en comparació amb les prestacions dels processadors actuals. Per això s'usen les memòries cau, situades entre la memòria principal i el processador. Aquestes memòries tenen una grandària limitada per raons de cost, de manera que només s'hi emmagatzemen còpies de les dades utilitzades recentment. Normalment, quan noves dades han de carregar-se a la memòria cau, altres dades han de ser reemplaçades.

Les memòries cau redueixen el temps d'execució dels programes perquè aquests compleixen el principi de localitat. Per això els algorismes s'han de dissenyar perquè exhibisquen localitat de referència, tant espacial com temporal. En aquest document es presenten algunes tècniques bàsiques que ajudaran a dissenyar algorismes que milloren l'ús eficient de la memòria cau.

Aquestes tècniques es basen en la transformació de l'algorisme original per a modificar l'ordre d'accés a les dades o l'emmagatzematge de manera que es millore la localitat de referència temporal o espacial. A continuació es presenten algunes de les tècniques més habituals.

Optimitzacions d'accés a les dades

En les optimitzacions d'accés a les dades, l'objectiu és millorar la localitat de les dades. Per això es fan modificacions als bucles dels algorismes, en molts casos bucles imbricats, perquè en cada iteració es reutilitzen les dades de les iteracions precedents. També es pot accedir a dades pròximes a les utilitzades que s'han copiat a la memòria cau en iteracions immediatament anteriors.

A continuació es presenta un conjunt de transformacions que milloren la localitat espacial per a un primer nivell de memòria cau en la jerarquia de memòria.

Intercanvi de bucles

En aquest cas el programa intercanvia els bucles que s'usen per a recórrer les matrius. L'objectiu és que la matriu es recórrega en l'ordre en la qual està emmagatzemada en la memòria, a fi d'augmentar la proximitat espacial de les referències al programa. Aquesta tècnica només pot aplicar-se si l'ordre dels bucles és indiferent.

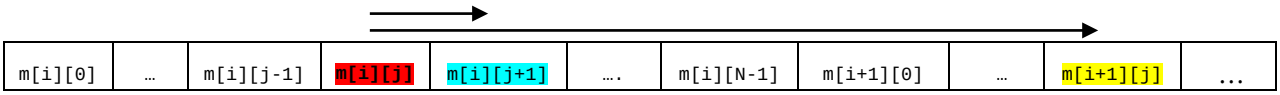
La tècnica de l'intercanvi de bucles pot generalitzar-se a més de dos bucles, llavors pren el nom de **permutació de bucles**. En aquest cas, dos bucles de l'algorisme poden intercanviar-se sense que siguen adjacents.

Per a poder il·lustrar aquesta tècnica s'usa l'algorisme de suma de matrius. En aquest algorisme es recorren tots els elements de dues matrius (`m1` i `m2`), se sumen i el resultat obtingut s'assigna a una tercera matriu (`m3`). La sentència que es repeteix és la següent:

```
m3[i][j]=m1[i][j]+m2[i][j];
```

Per a accedir als elements de les matrius, s'han de recórrer les files i les columnes de les matrius i accedir a cadascuna de les dades. Però els elements d'una matriu s'emmagatzemen en la

memòria seguint un ordre per files (cas del llenguatge C) o per columnes (cas del llenguatge Fortran). D'aquesta manera, en el cas del llenguatge C, l'element `m[i][j+1]` d'una matriu és al costat de l'element `m[i][j]`, mentre que l'element `m[i+1][j]` és a la distància d'una fila del `m[i][j]` (és a dir, a *N* posicions, en què *N* és el nombre d'elements d'una fila).



A continuació es mostra l'exemple de l'algorisme anterior en la versió original, en què el bucle interior fa que cada vegada s'accedisca a elements de files diferents, la qual cosa comporta una baixa localitat de referència.

Intercanvi de bucles

```
// Imbricació de bucles original
#define N 4
int main()
{
  register int i,j;
  int m1[N][N];
  int m2[N][N];
  int m3[N][N];
  for(j=0;j<N;j++)
  for(i=0;i<N;i++)
     m3[i][j]=m1[i][j]+m2[i][j];
}
```

```
//Imbricació de bucles intercanviats
#define N 4
int main()
{
  register int i,j;
  int m1[N][N];
  int m2[N][N];
  int m3[N][N];
for(i=0;i<N;i++)
  for(j=0;j<N;j++)
     m3[i][j]=m1[i][j]+m2[i][j];
}
```

No obstant això, en intercanviar els bucles en la implementació de la dreta, els accessos a cada iteració s'efectuen en elements de la mateixa fila fins que s'acaba la fila. Això incrementa la localitat espacial i, per tant, l'aprofitament de la memòria cau.

Fusió de bucles

La fusió de bucles comporta una transformació que pren dos bucles adjacents, els quals tenen el mateix espai d'iteracions transversal, i combina els seus cossos en un únic bucle. La fusió es pot fer si no hi ha dependències entre els bucles, és a dir, si les instruccions del primer bucle no afecten les instruccions del segon bucle. En fusionar dos bucles es redueix el sobrecost de les instruccions que implementen i controlen els bucles. També s'augmenta la localitat temporal i el rendiment del processador, amb un nombre més gran d'instruccions per iteració, i es redueix el nombre d'accessos a la memòria.

Per a il·lustrar aquesta tècnica usarem un algorisme de suma de dos vectors amb escalat. En aquest algorisme es recorren en primer lloc tots els elements d'un vector (`v1`) i s'escalen amb un valor (*a*). Posteriorment se suma cada element dels vectors (`v1` i `v2`) i el resultat s'assigna a un tercer vector (`v3`). Les sentències que es repeteixen són les següents:

```
v1[i]= v1[i]*a;
v3[i]= v1[i]+v2[i];
```

Si s'usen dos bucles, de primer cal recórrer tot el vector `v1` per a escalar-lo, de manera que quan es comença el segon bucle cal tornar a accedir als elements de `v1`. Després de la fusió de bucles, només cal accedir una vegada als elements de `v1`.

A continuació es mostra l'exemple de l'algorisme anterior en la versió original, en què s'usen dos bucles, l'un per a escalar el vector `v1` i l'altre per a sumar els vectors, la qual cosa comporta una baixa localitat de referència.

Fusió de bucles

```
// Codi original
#define N 4
int main()
{
register int i, a;
int v1[N],v2[N]v3[N];
  for(i=0;i<N;i++)
    v1[i]= v1[i]*a;
  for(i=0;i<N;i++)
 v3[i]= v1[i]+v2[i];
}
```

```
//Després de la fusió de bucles
#define N 4
int main()
{
  register int i,a;
  int v1[N],v2[N]v3[N];
  for(i=0;i<N;i++){
    v1[i]= v1[i]*a;
    v3[i]= v1[i]+v2[i];
   }
}
```

No obstant això, en fusionar els bucles en la implementació de la dreta, l'element calculat de `v1` es reaprofita immediatament a la suma amb el vector `v2`. Això incrementa la localitat temporal i, per tant, l'aprofitament de la memòria cau.

Optimitzacions d'emmagatzematge de dades

En l'apartat anterior s'ha vist com es millora la localitat de les dades en les aplicacions reordenant l'accés a les dades. No obstant això, en moltes aplicacions, la transformació dels bucles no és suficient per a obtenir una bona localitat als accessos, sobretot quan hi ha un alt grau de fallades per conflicte.

Les optimitzacions d'emmagatzematge de les dades modifiquen com s'organitzen les estructures de dades i les variables en la memòria. Aquestes transformacions tenen per objectiu eliminar els efectes de fallada per conflicte a fi de millorar la localitat espacial de les dades.

Emplenament de matrius

Quan s'accedeix de forma seqüencial als continguts de diverses matrius, es pot donar el cas que, a causa de les dimensions de les matrius, els accessos generen fallades per conflicte de forma sistemàtica. Un exemple característic d'aquesta situació es produïa en els exemples de la pràctica anterior. En aquests casos, els mapatges directes i associatius per conjunts produïen errors per conflicte de forma sistemàtica a causa que la grandària de les matrius coincidia amb la mida de la memòria cau. De forma genèrica, aquesta situació es produeix quan la posició dels elements als quals s'accedeix disten entre si un múltiple de la mida de la memòria cau.

Així, en l'exemple de la suma de matrius, tots els accessos a les matrius s'efectuen sobre blocs que tenen assignades les mateixes línies o conjunts de la memòria cau. Açò produeix fallades per conflicte i fa impossible la reutilització dels blocs entre diverses iteracions del bucle intern (accés dins de la mateixa fila).

Aquest problema es pot resoldre amb la tècnica d'emplenament de matrius. Amb aquesta tècnica s'augmenta la mida de la matriu, per exemple augmentant el nombre de columnes o files i, per tant, el nombre d'elements a la matriu. En realitat, aquests nous elements no s'usen en les iteracions de l'algorisme perquè no són necessaris. Sols s'hi introdueixen a fi d'evitar les fallades per conflicte en els accessos entre diverses matrius.

A continuació es mostra l'exemple de l'algorisme de la suma de matrius en la versió original. Si els elements `m1[i][j]`, `m2[i][j]` i `m3[i][j]` estan separats un múltiple de la mida de la memòria cau, es poden produir conflictes en l'ocupació de la memòria cau i, per tant, fallades per aquest motiu, la qual cosa comporta una baixa localitat de referència.

Emplenament de matrius

```
// Codi original
#define N 4
int main()
{
  register int i,j;
  int m1[N][N];
  int m2[N][N];
  int m3[N][N];
  for(i=0;i<N;i++)
  for(j=0;j<N;j++)
     m3[i][j]=m1[i][j]+m2[i][j];
}
```

```
//Codi amb emplenament de matrius
#define N1 4
#define N2 5
int main()
{
  register int i,j;
  int m1[N1][N2];
  int m2[N1][N2];
  int m3[N1][N2];
  for(i=0;i<N1;i++)
    for(j=0;j<N1;j++)
     m3[i][j]=m1[i][j]+m2[i][j];
}
```

No obstant això, en la implementació de la dreta, en completar les matrius `m1[i][j]`, `m2[i][j]` i `m3[i][j]`mitjançant un element de columna addicional, s'evita la coincidència d'aquests elements al mateix bloc de la memòria cau.

Combinació de matrius

Aquesta tècnica d'optimització es pot usar quan al programa es fa referència a diverses matrius de la mateixa dimensió, utilitzant els mateixos índexs. Aquesta tècnica tracta d'evitar el mateix problema que en el cas anterior (reduir el nombre de fallades per conflicte) però sense augmentar innecessàriament la mida de les dades. La tècnica de combinació consisteix a combinar aquestes matrius independents en una estructura composta.

Aquesta tècnica pot usar-se per a millorar la localitat espacial i s'aplica millor si els elements de les diverses matrius estan localitzats en posicions llunyanes de la memòria però s'accedeix a aquests elements junts. Així, la disposició dels elements de la suma de tres matrius en memòria originalment és:

m1[i][0]	…	m1[i][j]	….	m2[i][0]	…	m2[i][j]	….	m3[i][0]	…	m3[i][j]	….

Després de la combinació de matrius, la situació dels elements seria la següent:

m.m1[i][0]	m.m2[i][0]	m.m3[i][0]	…	m.m1[i][j]	m.m2[i][j]	m.m3[i][j]	….

A continuació es mostra el codi original de l'algorisme de la suma de matrius i l'aplicació de la combinació de matrius.

Combinació de matrius

`// Codi original`	`//Codi amb combinació de matrius`

```
#define N 4
int main()
{
  register int i,j;
  int m1[N][N];
  int m2[N][N];
  int m3[N][N];
  for(i=0;i<N;i++)
  for(j=0;j<N;j++)
     m3[i][j]=m1[i][j]+m2[i][j];
}
```

```
#define N 4
int main()
{
 register int i,j;
 struct mescla{int m1;
               int m2;
               int m3;
              };
 struct mescla m[N][N];
 for(i=0;i<N;i++)
   for(j=0;j<N;j++)
m[i][j].m3=m[i][j].m1+m[i][j].m2;
}
```

Bibliografia

M. Kowarschik i C. Weib (2003): "An Overview of Cache Optimization Techniques and Cache-Aware Numerical Algorithms", ***Algorithms for Memory Hierarchies***. Lecture Notes in Computer Science (Springer Berlin/Heidelberg), vol. 2625, p. 213-232. Disponible en línia ací: <http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.8.7904>.

Lab 4

Laboratori 4
Sistema d'entrada i eixida (part I)
Prova d'estat

(Actualitzat el dia 16/09/2021)

Temps de treball a casa (previst): 1 hora i 30 min
Temps de laboratori: 2 hores i 30 min

1. Introducció

En aquesta pràctica s'estudia el sistema d'entrada i eixida d'un ordinador i la tècnica de sincronització basada en la **prova d'estat**. S'hi estudien els algorismes per a l'accés a la informació d'estat dels dispositius d'entrada i eixida.

Objectius

L'estudiant, en acabar aquesta pràctica, ha de ser capaç de:

- Conèixer les tècniques per a consultar i modificar la informació d'estat d'un perifèric.
- Dissenyar algorismes que consulten el valor dels bits d'estat d'un perifèric.
- Usar operadors bit a bit en C per a consultar i modificar els bits d'estat, també denominats indicadors, d'un registre.
- Implementar programes en C que controlen l'entrada i l'eixida dels mòduls del MipsIt mitjançant la prova d'estat.

Recursos

Els mateixos que en les sessions anteriors: entorn de programació MipsIt.

2. Treball previ al laboratori

En aquest treball previ, tots els membres de l'equip de treball han de llegir l'annex I d'aquest document, on es descriu l'accés a les posicions d'entrada i eixida dins el MipsIt. També cal revisar l'annex II, en què es presenta una descripció dels operadors bit a bit en C, que són de molta utilitat en aquesta pràctica. Finalment, cal revisar els conceptes sobre el sistema d'entrada i eixida de l'ordinador vistos fins ara en el tema 2.

Una vegada feta la lectura dels annexos, s'ha de respondre a les qüestions següents:

Q1. En què consisteix la sincronització de la transferència de dades mitjançant prova d'estat?

Q2. S'ha de suposar que es té un mòdul d'entrada de dades que posseeix un registre d'estat, denominat `estat`, que té un valor `1` quan hi ha una dada disponible per a ser llegida i 0 en cas contrari. També hi ha un registre de dades, del qual es pot llegir la dada del mòdul d'entrada i eixida. S'ha d'escriure un procediment en llenguatge pseudoalgorítmic que llija les dades del mòdul esmentat i les escriga en una variable.

Q3. Quines fonts comparteixen la línia d'interrupció externa 3 en el simulador MipsIt?

Q4. Quin registre s'ha de llegir per a comprovar l'estat dels polsadors K1 i K2? I, anàlogament, quin registre s'ha de llegir per a comprovar el senyal del temporitzador extern associat al mòdul d'interrupcions del MipsIt?

Q5. S'ha de descriure com es pot conèixer, a partir d'una lectura de l'estat del mòdul d'interrupcions (mòdul 2 de l'annex), si en un moment donat està sent premut el polsador K2.

Q6. S'ha de descriure com s'ha d'esbrinar, a partir d'una lectura de l'estat del mòdul d'interrupcions, si el polsador K2 s'ha premut. S'ha d'indicar què s'hauria de fer per a evitar falses deteccions en el futur.

Q7. S'ha de proposar una sentència o sentències en llenguatge C per a detectar la pulsació de K2 i que, quan això passe, s'assigne a una variable booleana el valor cert (*true* en llenguatge C).

Q8. Es proposa dissenyar (en pseudocodi o amb un diagrama de flux) un programa que implemente el desplaçament d'un led encès al llarg del banc de leds del mòdul 1. En primer lloc, s'ha d'esperar que s'inicie el programa prement el botó K1 del mòdul 2. Llavors, s'encén el `led0`, després passa a `led1` i així successivament fins que s'arribe al `led7`. Arribat a aquest punt, el led retrocedeix fent el moviment invers fins a arribar al `led0` (tipus moviment del led del cotxe fantàstic) i de nou torna a començar. El programa finalitza només en la posició `led0` si s'ha premut el botó K2 en algun instant. S'ha d'usar el temporitzador per a controlar el temps d'encesa del led.

Primer punt de control: s'han de lliurar les preguntes Q1-Q8 per escrit a l'inici de la sessió i respondre a les preguntes del professorat de pràctiques.

3. Treball al laboratori

En aquesta sessió es veurà de forma pràctica la programació i gestió dels mòduls d'entrada i eixida. És per això que es proposa dissenyar diversos programes que inclouen l'ús d'aquests mòduls.

Segon punt de control: el programa comptador proposat en la Q8 del treball previ s'ha d'implementar en llenguatge C i, també, comprovar en el simulador del MipsIt. S'ha de mostrar al professorat del laboratori el funcionament correcte d'aquest programa i el codi corresponent.

Cal recordar que en estar accedint a l'espai d'entrada i eixida del processador, s'han d'editar les opcions per defecte del compilador, tal com s'ha fet en la primera sessió (secció 3.1).

3.1. Es proposa fer un programa en C de l'entorn de simulació MipsIt per al control d'un sistema de semàfors de vehicles i vianants. S'ha de suposar que el semàfor té:

- Els tres colors per al semàfor dels vehicles: roig, groc i verd.
- També els tres colors per al semàfors del vianants roig, groc i verd.

Per a simular l'encesa i l'apagada dels llums s'usen els leds associats al mòdul 1 d'entrada i eixida, dividint-los en dos bancs de quatre leds i fent que el roig siguen els dos leds més significatius de cada banc, situats a l'esquerra. A continuació hi ha el led groc i després el verd.

El comportament del sistema de semàfors ha de ser el següent:

1. Inicialment el sistema de semàfors està sempre donant pas als vehicles. Per tant, està en verd per als cotxes i en roig per als vianants.
2. Quan un vianant prem l'interruptor de petició de pas (el polsador K1 del mòdul 2), el semàfor dels vehicles canvia i s'activa el llum groc. El semàfor de vianants no canvia i continua en roig.
3. Després d'un cert interval d'espera, el pas per als vehicles es posa en roig i el semàfor de vianants es posa en verd.
4. Quan acaba el període assignat al pas de vianants, el semàfor de vianants passa a groc i el de vehicles es manté en roig. Després es torna a l'estat inicial descrit al punt 1.

Per a determinar els períodes en cada estat s'ha d'usar el temporitzador extern situat al mòdul 2. Els períodes en cada cas són:

- El groc per a vehicles i vianants ha de mantenir-se durant 5 impulsos del temporitzador extern.
- El verd per als vianants ha de mantenir-se durant 10 impulsos del temporitzador extern.
- El verd per als vehicles ha de mantenir-se durant 20 impulsos del temporitzador extern.

Cal recordar que en estar accedint a l'espai d'entrada i eixida del processador, s'han d'editar les opcions per defecte del compilador.

Tercer punt de control: el programa s'ha d'elaborar mitjançant el mètode de sincronització de prova d'estat i simular-ne el funcionament amb el programa MipsIt. En aquest cas, les interrupcions estan inhabilitades i s'ha de mostrejar contínuament el mòdul 2 d'entrada i eixida per a determinar quan canvien les entrades. S'ha de mostrar el correcte funcionament del programa i el codi al professorat del laboratori.

Annex I. Posicions de memòria d'entrada i eixida en el MipsIt

Introducció

El sistema d'entrada i eixida d'un ordinador està constituït pels elements (busos, registres, protocols, etc.) que permeten connectar l'ordinador amb els dispositius perifèrics.

Una operació d'entrada i eixida es pot dividir en tres fases:

1. **Adreçament**: l'accés al mòdul a través de la seua adreça a la memòria per a programar-lo o consultar-ne l'estat.
2. **Sincronització i control de transferència**: mecanisme que serveix per a saber si el perifèric està preparat per a transferir (enviar o rebre) dades, controlar el ritme d'aquesta transferència i comprovar si pot començar-ne una altra.
3. **Transferència de dades**: es tracta de la part final, on es fa realment la transferència de dades, procés que pot requerir diverses conversions de les dades enviades pels perifèrics.

La sincronització és una fase crítica en les operacions d'entrada i eixida i és fonamental per a la transferència correcta de les dades. A més a més, té molta influència en el rendiment final del sistema. Hi ha tres tècniques que s'usen habitualment: prova d'estat, interrupcions i accés directe a la memòria.

Sincronització per prova d'estat o interrupcions

Durant la transferència normal de dades entre l'ordinador i el mòdul o registre d'entrada i eixida, el processador és el responsable de transferir les dades de la memòria i enviar-les al mòdul d'entrada i eixida (durant una escriptura), i d'emmagatzemar les dades a la memòria procedents del mòdul (durant una lectura).

L'arribada de dades des del món exterior normalment és impredictible i el programa no pot saber exactament quan un mòdul d'entrada i eixida té disponibles dades que s'han de llegir. Per tant, cal un mètode per a sincronitzar l'execució del programa al processador amb els mòduls d'entrada i eixida. Mitjançant la sincronització, el mòdul notifica al processador quan hi ha alguna dada en espera de ser llegida o quan la cua de dades està buida.

Sincronització per prova d'estat

Amb aquest mètode, el processador executa un programa que controla directament l'operació d'entrada i eixida. El programa és responsable d'enviar l'ordre i esperar fins que la transferència de les dades acabe. El programa revisa contínuament l'estat del mòdul d'entrada i eixida fins que la transferència s'efectua i conclou. Normalment, la comprovació de l'estat es fa mitjançant un bucle de control que no finalitza fins que les dades es transfereixen.

Sincronització per interrupcions

Mitjançant sincronització per interrupcions, el mòdul d'entrada i eixida comunica al processador que està disponible per a fer una determinada transferència i efectua una petició d'interrupció. Tot i això, el processador és responsable d'extraure cadascuna de les dades del mòdul, en el cas d'una lectura, o de posar les dades al mòdul, si és una escriptura. Per tant, cadascuna de les dades transferides passa, igual que en el cas anterior, pel processador.

Posicions d'entrada i eixida al MipsIt (recordatori)

Elements en la posició `0xBF900000` (mòdul 1)

És un mòdul d'entrada i eixida de 8 bits que està constituït per 8 botons (entrades) i 8 leds (eixides). El valor dels polsadors es pot llegir en accedir a la posició `0xBF900000`; i el valor d'eixida dels leds es pot ajustar escrivint també sobre aquesta posició. Aquests elements no poden produir interrupció, és a dir, en modificar un valor d'entrada als polsadors no es produeix cap petició d'interrupció perquè aquest mòdul no té cap línia d'interrupció associada.

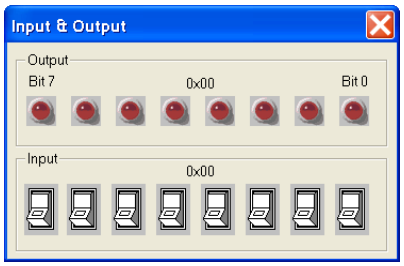


Figura 1. Finestra del mòdul 1.

Elements en la posició `0xBFA00000` (mòdul 2)

Cal recordar que en la posició de memòria `0xBFA00000` hi ha un mòdul d'entrada i eixida amb els elements següents connectats:

1. Polsador K2
2. Polsador K1
3. Temporitzador

La finestra a través de la qual es poden modificar les entrades és la següent:

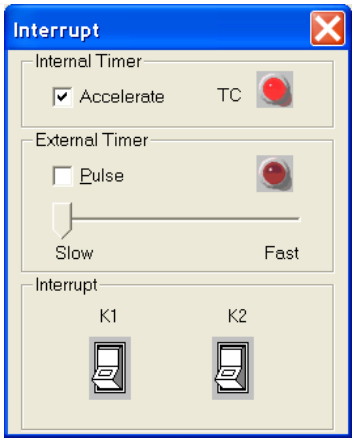


Figura 2. Finestra del mòdul 2.

Es pot comprovar l'estat de cada entrada llegint 8 bits de la posició `0xBFA00000`. El significat de cada bit és:

- `0xBFA00000` Interrupcions d'entrada i eixida (vuit bits)
  - Bit:
  - 0 – K2 (entrada)
  - 1 – K1 (entrada)
  - 2 – Temporitzador (entrada)
  - 3 – N/A (no definit)
  - 4 – Registre de K2: valor capturat de K2 en produir-se la interrupció. L'usuari l'ha de posar a zero.
  - 5 – Registre de K1: valor capturat de K1 en produir-se la interrupció. L'usuari l'ha de posar a zero.
  - 6 – Registre del temporitzador: valor capturat d'activació del temporitzador. L'usuari l'ha de posar a zero.
  - 7 – N/A (no definit)

Quan s'activa un polsador o s'activa el temporitzador extern, es fa una petició d'interrupció. Aquesta petició va associada a la línia d'interrupció externa 3 connectada al microprocessador. El succés que l'ha provocada es queda emmagatzemat al registre corresponent. El valor del registre no es desactiva fins que es faça una escriptura sobre aquesta mateixa posició `0xBFA00000` i el bit es pose a zero.

Les tres fonts d'interrupció comparteixen la mateixa línia d'interrupció externa del microprocessador (línia 3 dins del camp de línies de petició IP6-2). Per tant, totes tres produeixen la mateixa interrupció del maquinari, que es pot habilitar o no. Per a habilitar aquesta interrupció dins de la màscara d'interrupcions del registre d'estat (*status*) s'ha de posar a 1 el bit número 13, corresponent a la línia d'interrupció IP5. En cas contrari, la interrupció no es produeix però s'activen els bits del registre d'estat de la posició `0xBFA00000`.

Annex II. Operacions bit a bit en C

Introducció

En determinades ocasions ens pot interessar manipular dades bit a bit; per exemple, activar o desactivar **indicadors d'un bit** (*flags* en la bibliografia en anglès). Un indicador és una variable que pot prendre dos valors, de manera que se sol representar amb un bit. A causa que en C (i en la majoria de llenguatges de programació) no hi ha tipus de dades predefinides d'un bit, el que se sol fer és agrupar diversos indicadors en una variable de tipus enter o caràcter (`short int`, `int`, `unsigned char`).

Operadors bit a bit

Per a accedir a aquests indicadors, o simplement per a activar-los o desactivar-los, cal usar **operadors bit a bit** (*bitwise* en la bibliografia en anglès). Encara que normalment no se solen usar, es poden fer servir en la consulta i modificació de registres dels mòduls d'entrada i eixida i, per tant, en el disseny dels programes que els controlen.

A continuació es mostren els operadors més habituals.

Operadors bit a bit

Operador	*Acció*
&	Operació *AND* bit a bit
|	Operació *OR* bit a bit
^	Operació *XOR* bit a bit
~	Complement
<<	Desplaçament a l'esquerra
>>	Desplaçament a la dreta

Els operadors bit a bit funcionen en C amb variables de tipus caràcter o enter (`char`, `short`, `int` i `long`) però no amb variables de tipus flotant, com ara `float` o `double`. A continuació es descriu breument cadascun d'aquests operadors.

Operador AND (&)

L'operador AND compara dos bits. Si tots dos són 1, el resultat és 1; en qualsevol altre cas, el resultat és 0. Exemple:

```
c1 = 0x45       --> 01000101
c2 = 0x71       --> 01110001
---------------------------
c1 & c2 = 0x41 --> 01000001
```

Operador OR (|)

L'operador OR compara dos bits. Si qualsevol dels dos bits és 1, el resultat és 1. Només si els dos bits són 0, el resultat és 0. Exemple:

```
i1 = 0x47      --> 01000111
i2 = 0x53      --> 01010011
---------------------------
i1 | i2 = 0x57 --> 01010111
```

Operador XOR (^)

L'operador OR exclusiu o XOR dona com a resultat un 1 si qualsevol dels dos operands és 1, però no els dos simultàniament. Exemple:

```
i1 = 0x47      --> 01000111
i2 = 0x53      --> 01010011
---------------------------
i1 ^ i2 = 0x14 --> 00010100
```

Operador de complement (~)

Aquest operador torna com a resultat el complement a 1 del valor d'entrada. Exemple:

```
c = 0x45  --> 01000101
----------------------
~c = 0xBA --> 10111010
```

Operadors de desplaçament bit a bit (<<i>>):

Desplacen a l'esquerra o a la dreta un nombre especificat de bits. En un desplaçament a l'esquerra, els bits que sobren per la banda esquerra es descarten i els nous espais s'emplenen amb zeros. De manera anàloga passa amb els desplaçaments a la dreta. Exemple:

	c = 0x1C	00011100
c <<1	c = 0x38	00111000
c >>2	c = 0x07	00000111

És una bona idea revisar els conceptes de lògica digital i l'ús d'aquestes operacions en C/C++ estudiades el curs passat.

Lab 5

Laboratori 5
Sistema d'entrada i eixida (part II)
Interrupcions

(Actualitzat el dia 16/09/2021)

Temps de treball a casa (previst): 1 hora i 30 minuts
Temps de laboratori: 2 hora i 30 minuts

1. Introducció

En aquesta pràctica s'estudia el sistema d'entrada i eixida d'un ordinador i la tècnica de sincronització d'aquests dispositius basada en interrupcions. S'hi estudia el procés d'activació i gestió de les interrupcions en l'ordinador i la sincronització amb els dispositius d'entrada i eixida.

Objectius

L'estudiant, en acabar aquesta pràctica, ha de ser capaç de:

- Dissenyar algorismes que consulten o modifiquen el valor dels bits d'activació, bits d'estat i màscara del sistema d'interrupcions d'un processador.
- Usar operadors bit a bit en C per a consultar i modificar els bits associats a una interrupció.
- Implementar programes en C que controlen l'entrada i l'eixida dels mòduls del MipsIt mitjançant interrupcions.

2. Treball previ al laboratori

Tothom ha de llegir els annexos inclosos al final de la memòria. L'annex I descriu els registres involucrats en el sistema d'interrupcions del processador MIPS. L'annex II presenta dos exemples de com es gestiona el tractament de les interrupcions, l'un en assemblador i l'altre en llenguatge C. Finalment, l'annex III és un recordatori de les posicions de l'entrada i eixida descrites en la sessió anterior i que s'ha tornat a incloure en aquesta memòria per comoditat. Així mateix, es recomana revisar els conceptes sobre el sistema d'entrada i eixida de l'ordinador vistos fins ara en el tema 2.

Una vegada feta la lectura dels annexos i la bibliografia recomanada, s'ha de respondre a les qüestions següents:

Q1. S'ha d'explicar, breument, què és una interrupció.

Q2. Quines són les fonts d'interrupció del MipsIt?

Q3. Com podem saber el tipus d'interrupció o excepció que s'ha sol·licitat en el simulador MipsIt?

Q4. Indiqueu els bits que caldria activar en els registres de l'annex I per a habilitar les fonts d'interrupció externes del simulador (explicades en l'annex III).

Q5. S'ha d'executar el codi del programa exemple en C de l'annex II. A continuació s'han d'explicar els valors dels registres *Cause* i del registre d'entrada i eixida situat en la posició `0xbfa00000`. Aquest registre mostra el gestor d'interrupcions després de produir-se una interrupció provinent del polsador K1.

Q6. S'ha d'explicar quines accions efectuen les funcions `init_ext_int()`, `enable_int (EXT_INT3)` i `install_normal_int (InterruptHandler)` del programa exemple en C de l'annex II. Quin efecte tenen sobre els registres de configuració de les interrupcions?

Q7. Cal dissenyar (en pseudocodi o en un diagrama de flux) un programa basat en interrupcions que implemente un comptador cíclic basat en el senyal del temporitzador extern del mòdul d'interrupcions del MipsIt. Així, cada vegada que el temporitzador produeix un flanc de pujada, el programa ha d'incrementar una variable interna anomenada `comptador` i mostrar el valor que té a través dels leds del mòdul 1 d'entrada i eixida. (S'ha d'usar com a programa base l'exemple del programa en C de l'apèndix).

Q8. Quina tècnica seria més adequada per a gestionar la detecció dels polsadors K1 i K2 del Mipsit, la prova d'estat o interrupcions? S'ha d'explicar si aquesta tècnica seria també la més adequada per al temporitzador extern o si dependria de la freqüència d'aquest temporitzador.

Primer punt de control: s'han de lliurar les qüestions Q1-Q8 i respondre a les preguntes del professorat de laboratori sobre aquestes qüestions.

3. Treball al laboratori

En aquesta sessió es tracta de forma pràctica la programació i gestió dels mòduls d'entrada i eixida. I per a fer-ho es proposa dissenyar diversos programes que inclouen l'ús d'aquests mòduls.

Cal recordar que en estar accedint a l'espai d'entrada i eixida del processador, s'han d'editar les opcions per defecte del compilador.

3.1. Es proposa implementar en llenguatge C, i comprovar en el simulador del MipsIt, el programa *comptador* proposat en el punt 7 del treball previ, usant les interrupcions del temporitzador.

3.2. S'ha de fer un segon programa, similar a l'anterior, que incremente un comptador usant les interrupcions del temporitzador, però que, a més a més, faça servir el polsador K1 com a senyal d'habilitació. D'aquesta manera, només quan el polsador K1 estiga actiu, és a dir a nivell alt, el comptador estarà habilitat i s'incrementarà amb els impulsos del temporitzador. Però quan el polsador estiga inactiu, el comptador no s'incrementarà, encara que es produïsquen impulsos del temporitzador extern.

Segon punt de control: cal mostrar el funcionament dels dos programes al professorat de pràctiques i respondre a preguntes sobre el disseny i funcionament.

3.3. Es proposa fer un programa en C de l'entorn de simulació MipsIt per al control d'un sistema de semàfors de vehicles i vianants. Atenció, els requisits de funcionament són idèntics als de la

sessió anterior, però en aquest cas s'han d'usar interrupcions, no la prova d'estat, de manera que el codi serà diferent.

S'ha de suposar que el semàfor té:

- Els tres colors per al semàfor dels vehicles: roig, groc i verd.
- També els tres colors per al semàfor dels vianants roig, groc i verd.

Per a simular l'encesa i l'apagada dels llums s'usen els leds associats al mòdul 1 d'entrada/eixida, dividint-los en dos bancs de quatre leds i fent que el roig siguen els dos leds més significatius de cada banc, situats a l'esquerra. Després va el led groc i després el verd.

El comportament del sistema de semàfors és el següent:

1. Inicialment el sistema de semàfors ha d'estar sempre donant pas als vehicles. Per tant, està en verd per als cotxes i en roig per als vianants.
2. Quan un vianant prem l'interruptor de petició de pas (el polsador K1 del mòdul 2), el semàfor dels vehicles canvia i s'activa el llum groc. El semàfor de vianants continua en roig.
3. Després d'un interval d'espera, es posa en roig el pas per als vehicles i en verd el semàfor de vianants.
4. Quan acaba el període assignat al pas de vianants, el semàfor de vianants passa a groc i el de vehicles es manté en roig. Després es torna a l'estat inicial descrit al punt 1.

Per a determinar els períodes en cada estat s'usa el temporitzador extern situat al mòdul 2. Els períodes en cada cas són:

- El groc per a vehicles i vianants ha de mantenir-se durant 5 impulsos del temporitzador extern.
- El verd per als vianants ha de mantenir-se durant 10 impulsos del temporitzador extern.
- El verd per als vehicles ha de mantenir-se durant 20 impulsos del temporitzador extern.

Tercer punt de control: s'ha de mostrar el funcionament anterior i respondre a preguntes sobre el disseny i funcionament.

Annex I. Interrupcions en el MIPS

Introducció

Una excepció/interrupció és qualsevol succés o esdeveniment que trenca la seqüència normal d'execució d'un programa. Es pot distingir entre aquests dos tipus de successos:

- **Excepció**: quan el succés que es processa és causat per un error durant l'execució d'una instrucció (desbordament en una operació aritmètica, divisió per zero, etc.).
- **Interrupcions**: són successos externs al microprocessador generats per algun dispositiu d'entrada i eixida.

Aquesta secció descriu la implementació de les interrupcions dins de l'arquitectura MIPS. En l'arquitectura MIPS, juntament amb el processador, s'integren una sèrie de coprocessadors; en concret el coprocessador 0 és l'encarregat de gestionar la informació referent al funcionament de les excepcions i interrupcions. Els registres següents pertanyen al coprocessador 0:

- Adreça amb valor incorrecte *(BadVAddr)*: emmagatzema l'adreça que ha provocat un accés a memòria que ha causat l'excepció.
- Registre d'estat *(Status)*: màscara d'interrupcions i bits d'autorització.
- Registre de causa *(Cause)*: tipus d'excepció i bits d'interrupcions pendents.
- Comptador de programa de l'excepció *(EPC)*: emmagatzema l'adreça de la instrucció que s'estava executant en el moment de produir-se l'excepció/interrupció.

Registre d'estat

Els bits del registre d'estat controlen les excepcions/interrupcions que es poden activar. Hi ha un bit per cadascuna de les cinc línies d'interrupció de maquinari i tres per les interrupcions del programari definides. El camp màscara d'interrupcions conté un bit per cadascuna de les cinc entrades externes d'interrupció. El bit 10 correspon a l'entrada de petició d'interrupció externa 0, el bit 11 correspon a l'entrada 1, el bit 12 correspon a l'entrada 2, el bit 13 correspon a l'entrada 3 i el bit 14 correspon a l'entrada 4:

- Bit 10 => entrada 0
- Bit 11 => entrada 1
- Bit 12 => entrada 2
- Bit 13 => entrada 3
- Bit 14 => entrada 4

Per a cadascuna d'aquestes fonts d'interrupció, si el seu bit d'habilitació està a 1, la interrupció està habilitada. Per contra, si el bit d'habilitació està a 0, la interrupció està desactivada.

El camp de bits més baix del registre d'estat controla l'habilitació global de les interrupcions. Així, el bit menys significatiu (bit 0) és el bit de permís d'interrupció: si està a 1, les interrupcions estan habilitades; i si val 0, estan emmascarades totes de forma global. El bit nucli/usuari és un bit d'informació que val 0 si el programa estava en mode nucli del sistema operatiu quan es produeix l'excepció. El bit nucli/usuari val 1 si el programa estava en mode usuari. Quan es produeix una excepció/interrupció, aquests 6 bits es desplacen dues posicions a

l'esquerra. Els bits previs passen a ser els antics, els actuals passen a ser els previs i els actuals es posen a 0.

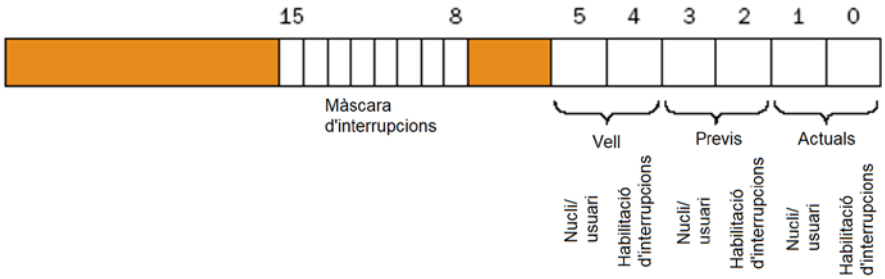


Figura 1. Registre d'estat (Status).

La màscara d'interrupció es defineix de la manera següent:

- Bit 15: IP7 interrupció del temporitzador.
- Bit 14-10: IP6:2 són les línies de petició d'interrupció externes.
- Bit 8-9: IP1:0 són interrupcions del programari.

Quan el bit corresponent de la màscara està a 1, aquesta interrupció està habilitada.

Registre de causa

El registre de causa té vuit bits de petició d'interrupció pendents que corresponen als bits d'habilitació d'interrupció del registre d'estat. El bit corresponent està a 1 quan s'ha sol·licitat una interrupció en aquesta línia, però encara no s'ha processat.

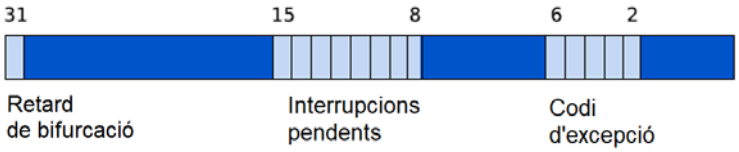


Figura 2. Registre de causa (Cause).

El codi d'excepció descriu la causa d'una excepció amb els valors següents:

Número	Nom	Descripció
0	INT	Interrupció externa
4	ADDRL	Excepció d'error en l'adreça (càrrega o cerca d'instrucció)
5	ADDRS	Excepció d'error en l'adreça (emmagatzematge)
6	IBUS	Error del bus en la cerca d'instruccions
7	DBUS	Error del bus en la càrrega o emmagatzematge de dades
8	SYSCALL	Excepció SYSCALL
9	BKPT	Excepció de punt de ruptura
10	RI	Excepció d'instrucció reservada

12	OVF	Excepció de desbordament aritmètic

Els números de l'1 al 3 van associats a la TLB de la memòria virtual, que no està implementada en el simulador.

Comptador de programa de l'excepció (*EPC*)

Emmagatzema l'adreça de la instrucció que s'estava executant en el moment de produir l'excepció/interrupció (instrucció que estava sent executada pel processador).

Registre adreça amb valor incorrecte (*BadVaddr*)

Emmagatzema l'adreça que ha provocat un accés a memòria que ha causat l'excepció.

Accions per al tractament d'interrupcions i excepcions

Quan s'activa una excepció/interrupció, els passos que se segueixen per a processar aquest succés són:

1. L'adreça de la instrucció que s'estava executant quan es produeix l'excepció/interrupció es desa a l'EPC. Si l'excepció la provoca l'accés a una posició de memòria (per exemple una fallada de pàgina en la memòria virtual), aquesta adreça s'emmagatzema al registre d'adreça amb valor incorrecte *BadVaddr* (*Bad Virtual Address*). Aquesta adreça és necessària per a repetir l'accés després de portar la pàgina corresponent a la memòria física de l'ordinador.
2. El codi de l'excepció s'emmagatzema al registre de causa i s'activa el bit que indica el nivell de la petició.
3. En el registre d'estat, els sis bits de menys pes es desplacen dues posicions a l'esquerra, mentre que els bits actuals es posen a 0.
4. S'atura l'execució del programa actual i se salta al gestor d'interrupcions. Per a fer això, es carrega al comptador de programa l'adreça de memòria fixa `0x80000080` (adreça de la rutina de tractament de l'excepció).
5. Una vegada executada la subrutina, s'ha de tornar al programa principal executant la instrucció de retorn des de l'excepció (`RFE` o *ReturnFromException*), que restaura l'estat de les interrupcions, abans de la instrucció de tornada de subrutina (`jr R31`). En el registre d'estat, els sis bits de menys pes es desplacen dues posicions a la dreta i es restaura la situació anterior en els quatre bits de menys pes.

Annex II. Programes d'exemple

Programa exemple 1 (llenguatge assemblador)

A continuació es mostra un programa exemple en assemblador que mostra el tractament de la interrupció associada a la línia 3. Aquest programa mostra per consola el contingut dels registres de causa (*Cause*) i comptador de programa de l'excepció (*EPC*) en cas d'interrupció. Per a comprovar-ho, s'ha de copiar el programa, compilar-lo i executar-lo, pas a pas, en el simulador. Aquest projecte s'ha de crear amb l'opció de codi assemblador.

```
#include<iregdef.h>
#include<idtcpu.h>
#include<excepthdr.h>

      .data
save0:      .word 0
save1:      .word 0
save2:      .word 0
save3:      .word 0
save4:      .word 0

Format: .asciiz "Cause= 0x%x, EPC= 0x%x, Interrupt I/O= 0x%x\n"

      .set noreorder
      .text
      .globl start       # 'Start' és l'etiqueta de la funció principal
      .ent start
start:
      lui   t0, 0xbfa0  # En t0 posa l'adreça del port d'interrupció
      sb    zero,0x0(t0)      # Posada a zero dels latchs
      la    t0, intstub # Instal·la el gestor d'interrupcions
      la    t1, 0x80000080 # en la posició 0x80000080
      lw    t2, 0(t0)     # Primera instrucció per a instal·lar el gestor
      lw  t3, 4(t0)     # Segona instrucció per a instal·lar el gestor
      sw    t2, 0(t1)      # Emmagatzema la primera instrucció
      sw    t3, 4(t1)      # Emmagatzema la segona instrucció

      andi  v0, v0, 0              # registre v0 a 0
      ori   v0, v0, 1              # habilita les interrupcions
      ori   v0, v0,0x00002000      # de la línia 3 (K1, K2, temporitzador)
      mtc0 v0, C0_SR               # actualitza el registre d'estat (Status)

Loop:
      j     Loop# bucle d'espera fins a rebre una petició
      nop

      .end  start

intstub:
      j     interrup    #Instruccions que es copien en les posicions
      nop               # 0x80000080 per a la instal·lació del gestor

      .globl      introutine  #gestor d'interrupcions
      .ent  interrup
interrup:
      sw    a0, save0        # Desa tots els registres que es
      sw    a1, save1        # modifiquen en posicions de memòria
      sw    a2, save2        # No s'usa la pila perquè la
      sw    a3, save3        # pila pot produir excepcions.
      sw    s0, save4        # No es necessita desar $k0/$k1
```

```
                              # Aquest codi no és reentrant.

      mfc0  k0, C0_CAUSE      # Mou Cause a $k0
      mfc0  k1, C0_EPC        # Mou EPC a $k1
      lui   s0, 0xbfa0        # Llegim el port de les interrupcions per a
                              # saber quina és la que s'ha activat

      la    a0, Format        # Impressió de la cadena. En a0 hi ha l'adreça
      move  a1, k0            # primer paràmetre de la cadena
      move  a2, k1            # segon paràmetre de la cadena
      lbu   a3, 0x0(s0)       # tercer paràmetre de la cadena

      jal   printf            # Imprimeix el missatge d'error d'excepció
      sb    zero,0x0(s0)      # Es posen a zero els latchs del port
                              # de petició d'interrupcions

      lw    a0, save0   # Restaura els registres que s'han modificat dins
      lw    a1, save1   # del gestor
      lw    a2, save2
      lw    a3, save3
      lw    s0, save4

      rfe               # Restaura l'estat de les interrupcions
      jr    k1          # Torna al programa principal amb el valor de l'EPC

      .end  interrup
```

Programa exemple 2 (llenguatge C)

Aquest programa exemple és similar a l'anterior però escrit en llenguatge C. Aquest exemple inclou les funcions de llibreria següents:

- *init_ext_int*: inicialitza el port d'interrupcions.
- *install_normal_int*: instal·la el gestor d'interrupcions.
- *enable_int*: habilita les interrupcions en el registre d'estat.
- *get_CAUSE*: retorna el contingut del registre de causa *Cause*.

Per a poder compilar-lo incloent-hi aquestes funcions, cal afegir el fitxer *interrup.s* al directori que conté el codi font i incloure en el codi la instrucció següent: `include<excepthdr.h>`

```
#define FALSE 0
#define TRUE  1
#include <excepthdr.h>

unsignedchar* Port    = (char*) 0xbfa00000;

void InterruptHandler () //Funció que s'executa en produir-se
                         //la interrupció
{
      printf ("Cause= 0x%x, Interrupt I/O= 0x%x\n", get_CAUSE(), *Port);
      *Port = 0; // posa a 0 el port d'estat de les interrupcions.
}

int main ()
{
      init_ext_int(); //Activació general de les interrupcions externes
      install_normal_int(InterruptHandler); //Instal·la l'interrupthandler
      enable_int(EXT_INT3); //habilita interrupcions externes de la línia 3
      while (TRUE) {};
}
```

Annex III. Posicions de memòria d'entrada i eixida en el MipsIt (recordatori)

Introducció

El sistema d'entrada/eixida d'un ordinador està constituït pels elements (busos, registres, protocols, etc.) que permeten connectar l'ordinador amb els dispositius perifèrics.

Una operació d'entrada/eixida es pot dividir en tres fases:

1. **Adreçament**: l'accés al mòdul a través de la seua adreça en memòria per a programar-lo o consultar-ne l'estat.
2. **Sincronització i control de transferència**: mecanisme que serveix per a saber si el perifèric està preparat per a transferir (enviar o rebre) dades, controlar el ritme d'aquesta transferència i comprovar si pot començar-ne una altra.
3. **Transferència de dades**: es tracta de la part final, on es fa realment la transferència de les dades, procés que pot requerir diverses conversions de les dades enviades pels perifèrics.

La sincronització és una fase crítica en les operacions d'entrada i eixida i és fonamental per a la transferència correcta de les dades. A més a més, té molta influència en el rendiment final del sistema. Hi ha tres tècniques que s'usen habitualment: prova d'estat, interrupcions i accés directe a la memòria.

Sincronització per prova d'estat o interrupcions

Durant la transferència normal de dades entre l'ordinador i el mòdul o registre d'entrada i eixida, el processador és el responsable de transferir les dades de la memòria i enviar-les al mòdul d'entrada i eixida (durant una escriptura), i d'emmagatzemar les dades a la memòria procedents del mòdul (durant una lectura).

L'arribada de dades del món exterior normalment és impredictible i el programa no pot saber exactament quan un mòdul d'entrada i eixida té disponibles dades que s'han de llegir. Per tant, cal un mètode per a sincronitzar l'execució del programa al processador amb els mòduls d'entrada i eixida. Mitjançant la sincronització, el mòdul notifica al processador quan hi ha alguna dada en espera de ser llegida o quan la cua de dades està buida.

Sincronització per prova d'estat

Amb aquest mètode, el processador executa un programa que controla directament l'operació d'entrada i eixida. El programa és responsable d'enviar l'ordre i esperar fins que la transferència de les dades acabe. El programa revisa contínuament l'estat del mòdul d'entrada i eixida fins que la transferència s'efectua i conclou. Normalment, la comprovació de l'estat es fa mitjançant un bucle de control que no finalitza fins que les dades es transfereixen.

Sincronització per interrupcions

Mitjançant sincronització per interrupcions, el mòdul d'entrada i eixida comunica al processador que està disponible per a fer una determinada transferència i efectua una petició d'interrupció. Tot i això, el processador és responsable d'extraure cadascuna de les dades del mòdul, en el cas d'una lectura, o de posar les dades al mòdul, si és una escriptura. Per tant, cadascuna de les dades transferides passa, igual que en el cas anterior, pel processador.

Posicions d'entrada i eixida en el MipsIt (recordatori)

Elements en la posició `0xBF900000` (mòdul 1)

És un mòdul d'entrada i eixida de 8 bits que està constituït per vuit botons (entrades) i vuit leds (eixides). El valor dels polsadors es pot llegir en accedir a la posició `0xBF900000`; i el valor d'eixida dels leds es pot ajustar escrivint també sobre aquesta posició. Aquests elements no poden produir interrupció, és a dir, en modificar un valor d'entrada als polsadors no es produeix cap petició d'interrupció perquè aquest mòdul no té cap línia d'interrupció associada.

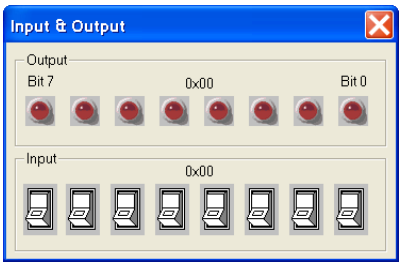


Figura 3. Finestra del mòdul 1.

Elements en la posició `0xBFA00000` (mòdul 2)

Cal recordar que en la posició de memòria `0xBFA00000` hi ha un mòdul d'entrada i eixida amb els elements següents connectats:

1. Polsador K2
2. Polsador K1
3. Temporitzador

La finestra a través de la qual es poden modificar les entrades és la següent:

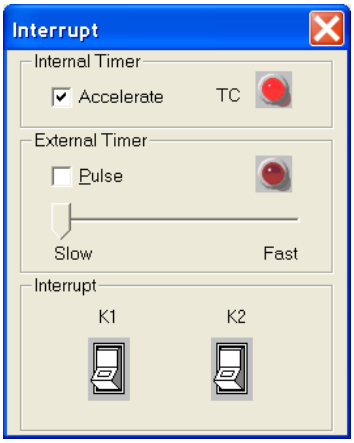


Figura 4. Finestra del mòdul 2.

Es pot comprovar l'estat de cada entrada llegint vuit bits de la posició `0xBFA00000`. El significat de cada bit és:

- `0xBFA00000` Interrupcions d'entrada i eixida (vuit bits)
  - Bit:
  - 0 – K2 (entrada)
  - 1 – K1 (entrada)
  - 2 – Temporitzador (entrada)
  - 3 – N/A (no definit)
  - 4 – Registre de K2: valor capturat de K2 en produir-se la interrupció. L'usuari l'ha de posar a zero.
  - 5 – Registre de K1: valor capturat de K1 en produir-se la interrupció. L'usuari l'ha de posar a zero.
  - 6 – Registre del temporitzador: valor capturat d'activació del temporitzador. L'usuari l'ha de posar a zero.
  - 7 – N/A (no definit)

Quan s'activa un polsador o s'activa el temporitzador extern, es fa una petició d'interrupció. Aquesta petició va associada a la línia d'interrupció externa 3 connectada al microprocessador. El succés que l'ha provocada es queda emmagatzemat al registre corresponent. El valor del registre no es desactiva fins que es faça una escriptura sobre aquesta mateixa posició `0xBFA00000` i el bit es pose a zero.

Les tres fonts d'interrupció comparteixen la mateixa línia d'interrupció externa del microprocessador (línia 3 dins del camp de línies de petició IP6-2). Per tant, totes tres produeixen la mateixa interrupció del maquinari, que es pot habilitar o no. Per a habilitar aquesta interrupció dins de la màscara d'interrupcions del registre d'estat (*status*) s'ha de posar a 1 el bit número 13, corresponent a la línia d'interrupció IP5. En cas contrari, la interrupció no es produeix però s'activen els bits del registre d'estat de la posició `0xBFA00000`.

Lab 6

Laboratori 6
Perifèrics
Teclat

(Actualitzat el dia 16/09/2021)

Temps de treball a casa (previst): 1 hora i 30 minuts
Temps de laboratori: 2 hora i 30 minuts

1. Introducció

En aquesta pràctica s'estudia el funcionament del teclat d'un ordinador i l'accés a baix nivell als mòduls d'entrada i eixida que controlen el funcionament d'aquest perifèric. També s'hi estudia la forma en què s'envien les ordres de configuració del teclat i s'implementen diversos programes en llenguatge C per a modificar-ne el funcionament en Linux.

Objectius

L'estudiant, en acabar aquesta pràctica, ha de ser capaç de:

- Conèixer les possibilitats de configuració del teclat d'un PC.
- Conèixer l'estructura i els registres interns d'un controlador de teclat, en particular del 8742 d'Intel.
- Usar les ordres bàsiques per a configurar el teclat que estan disponibles en el maquinari del mateix teclat.
- Usar funcions d'entrada i eixida a baix nivell en C, en el sistema operatiu Linux, per a consultar i modificar els bits dels registres integrats en els controladors del teclat.
- Dissenyar i implementar programes en C que canvien la configuració del teclat i recullen els codis d'escaneig de les pulsacions de l'usuari.

Recursos

Per a la realització d'aquestes tres últimes sessions de laboratori es disposa d'un ordinador personal PC i d'una màquina virtual amb el sistema operatiu Linux. Per a executar la màquina virtual cal tenir instal·lat el programari VMware Player que és gratuït i es pot descarregar de la pàgina web de la companyia VMware. Els fitxers amb la màquina virtual de Linux que cal executar són a l'Aula Virtual. Una vegada descomprimit el fitxer amb la màquina virtual, es pot executar tal com es mostra en l'annex III. La màquina virtual de Linux té instal·lat tot el programari necessari per a compilar i executar els programes que s'han de fer durant la pràctica.

2. Treball previ al laboratori

Per a poder traure el màxim partit a la sessió de laboratori és molt important fer les activitats que es proposen a continuació i que s'hi dedique el temps indicat. En aquest treball previ, tots els membres de l'equip de treball han de llegir els annexos I, II i III d'aquest document, on es descriu la configuració maquinari del controlador del teclat, un exemple d'ús de les funcions d'entrada i eixida necessàries per a implementar els programes i, també, la forma com s'usa la màquina virtual de Linux per a compilar i executar els programes.

Una vegada efectuada la lectura dels annexos, s'han de contestar les qüestions següents:

Q1. S'ha d'indicar a través de quina adreça o port es reben les dades provinents del teclat.

Q2. Com es pot saber si hi ha dades provinents del teclat en espera de ser llegides?

Q3. S'ha d'explicar a través de quina adreça o port s'envien les ordres al teclat.

Q4. S'ha de descriure breument què són els codis d'escaneig associats a una tecla: el codi en prémer una tecla i el codi en alliberar-la.

Q5. Quina utilitat tenen les funcions *iopl*, *outb* i *inb*?

Q6. S'ha d'explicar de forma justificada què fa el codi de l'annex II.

Q7. Quin tipus de sincronització s'ha emprat en el programa de l'annex II per a saber quan les dades que provenen del teclat estan disponibles al controlador: prova d'estat o interrupcions?

Primer punt de control: s'han de lliurar les qüestions Q1-Q7 i respondre a les preguntes del professorat sobre aquestes qüestions.

3. Treball al laboratori

En aquesta sessió tractem de forma pràctica la programació i gestió del teclat en el sistema operatiu Linux. Amb aquesta finalitat, proposem dissenyar diversos programes que inclouen l'ús d'aquest perifèric.

3.1. Es proposa compilar i executar el programa de l'annex II. Una vegada el codi s'ha editat i desat en un fitxer amb l'extensió `.c`, es pot fer la compilació i l'execució del programa des de la línia d'ordres d'un terminal de la màquina virtual, tal com es mostra en l'annex III. S'ha de comprovar el funcionament del programa i els valors retornats pel teclat.

3.2. S'ha d'implementar en llenguatge C un programa de control del teclat que canvie l'estat dels leds del teclat. Primerament, el programa ha d'encendre els tres leds; després ha d'esperar un segon, a continuació els apaga tots i finalment ha d'esperar un altre segon. Es demana implementar un programa que repetisca aquest parpelleig cinc vegades.

3.3. Es proposa fer un tercer programa en C que desactive el teclat durant deu segons i que el torne a habilitar després dels deu segons d'espera. Per a fer-ho, s'han d'usar les ordres que accepta el teclat explicades al final de l'annex I. S'ha de comprovar que durant els deu segons d'espera el teclat no respon, encara que es pressionen les tecles. El programa s'ha d'editar, desar, compilar i executar tal com s'ha fet en l'exemple anterior.

Segon punt de control: s'ha de mostrar el funcionament dels programes i les respostes de les qüestions al professor o professora de pràctiques i, també, respondre a preguntes sobre el disseny i funcionament.

3.4. S'ha d'implementar un quart programa en C, prenent com a base els anteriors, que reba contínuament l'enviament dels codis de les tecles premudes des del teclat i que imprimisca a la

pantalla els codis d'escaneig rebuts. Els valors llegits han de mostrar-se en format hexadecimal. La sincronització amb el teclat s'efectua per prova d'estat, és a dir, revisant contínuament si hi ha dades disponibles i accedint a aquestes dades tan prompte com se'n detecte la presència.

Aquest procés de revisió es repeteix indefinidament fins que es detecte alguna tecla en particular, per exemple la tecla `q`, que té associat el codi de pulsació `0x10`. D'aquesta manera, prémer la tecla farà que acabe el bucle i finalitze el programa. Amb la intenció que el procés de revisió no interferisca amb els caràcters que mostra el terminal, és recomanable desactivar la resposta de caràcters que genera el terminal (eco). El codi C que caldria afegir al principi i al final del programa per a evitar els ecos del terminal es mostra a continuació:

Les funcions que desactiven l'eco del terminal són:

```
// cal incloure les directives següents per a les llibreries que contenen les
// funcions que s'utilitzaran a continuació
#include <unistd.h>
#include <termios.h>

// cal declarar aquestes dues variables per a llegir i emmagatzemar l'estat actual de la
// configuració del terminal
        struct termios attr;
        struct termios orig_attr;

//la funció tcgetattr llegeix els atributs de tractament del teclat al terminal
        tcgetattr(STDIN_FILENO, &orig_attr);
        attr = orig_attr;

//desactivem el flag que produeix eco
        attr.c_lflag &= ~(ECHO) ;

//la funció tcsetattr escriu els nous atributs
        tcsetattr(STDIN_FILENO, TCSAFLUSH, &attr);
```

Com que, abans d'acabar el programa, la configuració original s'ha de restaurar, cal incloure-hi el codi següent:

```
//aquestes funcions són per a buidar la memòria intermèdia del teclat de possibles caràcters que
// estiguen en espera de ser llegits ja que s'ha desactivat l'eco anteriorment, on
// chac és una variable de tipus: unsigned char chac[256];
        printf("\r\n Per a eixir, introdueix qualsevol tecla + enter\r\n");
        scanf("%s",chac);

// Aquesta funció recupera els atributs originals del terminal
        tcsetattr(STDIN_FILENO, TCIFLUSH, &orig_attr);
```

Una vegada comprovat el funcionament del programa 3.4, s'ha de respondre a les preguntes següents:

3.4.1. Quins codis de pulsació i alliberament tenen les tecles següents?: `a`, `k`, `Alt` i `Enter`.

3.4.2. Què passa quan es manté premuda una tecla molt de temps?

3.4.3. Què passa quan el programa s'està executant i es mou el ratolí?

Tercer punt de control: s'ha de mostrar el funcionament dels programes i les respostes a les qüestions al professorat de pràctiques, i respondre a preguntes sobre el disseny i funcionament.

Annex I. Interrupcions al PC i control del teclat

El teclat es connecta a l'ordinador per mitjà d'un cable que conté quatre fils hàbils: dos que condueixen el corrent, l'un per a les dades i l'altre per al rellotge. El teclat és en realitat un petit microordinador que s'encarrega de detectar la pulsació i l'alliberament de les tecles, acció que genera uns **codis d'escaneig** que les identifiquen i, a continuació, els envia pel cable a través d'un protocol de comunicació en sèrie. Una tecla pot generar dos tipus de codi diferents. En **prémer** una tecla es genera un codi. En **alliberar** una tecla es genera el mateix codi que en prémer-la però amb el bit 7 actiu. El teclat s'encarrega de repetir els codis d'una tecla quan es manté polsada durant un cert temps, en el conegut mecanisme d'autorepetició de la majoria dels teclats.

Les dades, quan arriben a l'ordinador, reben un tractament diferent que depèn de si l'ordinador és un XT o un AT, molt més senzill en el primer. En els XT, els bits que arriben a l'ordinador es van col·locant en un simple registre de desplaçament connectat al port `60h`; en completar els vuit bits es produeix una interrupció de tipus IRQ 1 (INT 9), la segona de més prioritat després de la del temporitzador. No obstant això, el teclat és capaç de memoritzar fins a vuit pulsacions quan el processador no té temps per a atendre'l. Després de llegir el codi de la tecla, el programa que la gestiona ha d'enviar un senyal de reconeixement als circuits de l'ordinador per tal de permetre que continue la recepció de dades.

En els AT i les versions posteriors de PC hi ha un circuit integrat encarregat d'interpretar les dades procedents del teclat, després de traduir-les adequadament. Un exemple d'aquesta classe de controlador és el 8742 d'Intel. També serveix d'intermediari a les transmissions de dades del processador al teclat, que en l'AT és un perifèric bidireccional que pot rebre ordres per a configurar els leds, entre altres tasques. El controlador del teclat fa el paper de mòdul d'entrada i eixida que connecta el processador amb el teclat. El controlador recolza en tres registres bàsics: un d'estat, un de control i el d'ordres; més dos ports d'entrada i eixida: l'un d'eixida de dades i l'altre d'entrada de dades cap al teclat.

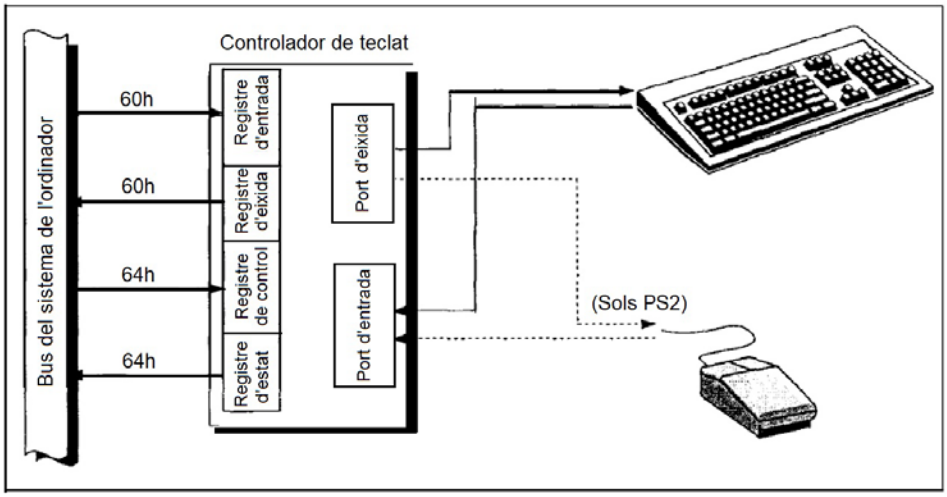


Figura 1. Esquema dels ports del controlador del teclat/ratolí 8742.

- El **registre d'eixida de dades** està situat al **port `60h`** i és només de lectura; el 8742 el fa servir per a enviar els codis de les tecles al processador. Hauria de llegir-se només quan el bit 0 del registre d'estat està a 1 (és a dir, hi ha una tecla pendent de ser processada).
- El **registre d'entrada de dades** del 8742 és només d'escriptura i s'hi pot accedir a través del **port 60h**. Les dades es reenvien pel 8742 cap al teclat. Les dades han de ser escrites en aquest registre només quan el bit 1 del registre d'estat estiga inactiu. Cal destacar que s'accedeix als registres d'eixida i entrada de dades pel mateix port (`60h`), i que és la lectura i l'escriptura les que seleccionen l'accés a l'un o l'altre respectivament.
- Es pot accedir al **registre d'estat** llegint del **port d'entrada i eixida `64h`** i pot ser llegit en qualsevol moment. El significat dels seus bits és el següent:
  - Bit 0. **Port d'eixida de dades ple**. Un 1 indica que el 8742 ha col·locat una dada al port d'eixida (port `60h`) i el processador encara no l'ha llegida. Aquest bit es posa a 0 quan el processador llegeix el port `60h`.
  - Bit 1. **Port d'entrada de dades ple**. Un 1 assenyala que s'ha col·locat una dada en el port d'entrada (port `60h`) i que el 8742 encara no l'ha llegida.
  - Bit 2. **Autotest correcte**. Val 1 si el resultat del test és positiu i 0 en cas contrari.
  - Bit 3. **Ordre/dada**. Té el valor 1 o 0 en enviar qualsevol *byte* al port `60h` o al `64h` respectivament, d'aquesta manera, el 8742 sap si el que s'envia són ordres o dades (ordres = 1, dades = 0).
  - Bit 4. **Bit d'inhibició**. Aquest bit s'actualitza sempre que es col·loca una dada al port d'eixida de dades (port `60h`).
  - Bit 5. **Transmissió fora de temps**. Indica que la transmissió d'una dada cap al teclat no ha sigut resposta dins del temps adequat.
  - Bit 6. **Recepció fora de temps**. Indica que el teclat no ha enviat una dada dins del temps adequat.
  - Bit 7. **Error de paritat**. Indica si hi ha hagut un error en la paritat: 0 és el valor correcte.

Ordres que accepta el teclat

Hi ha certes ordres que el teclat accepta. Aquestes ordres es poden enviar en qualsevol moment al teclat a través del **port `60h`** del controlador 8742, i aquest les encamina cap al teclat. El teclat respon en menys de 20 mil·lisegons retornant un senyal de reconeixement per mitjà del *byte* `0FAh`. Les principals ordres (diferenciades de les dades per a tenir el bit 7 actiu) són:

- **Reiniciar (`0FFh`)**: reinicialitza el teclat i fa una autoverificació. Com a resultat, el teclat contesta: `0xAA` (test passat), `0xFC` o `0xFD` (fallada en el test).
- **Reenviament (`0FEh`)**: el sistema pot enviar aquesta ordre al teclat quan detecta una fallada en la recepció des del teclat. Aquesta ordre només pot enviar-se després d'una transmissió del teclat i abans d'habilitar la comunicació per a la recepció següent. El teclat respon enviant de nou la dada anterior.
- **Establir valors per defecte (`0F6h`)**: torna l'autorepetició als valors per defecte, neteja el registre d'eixida i continua rastrejant les tecles si no estava inhibit; és una espècie de reinici en calent.
- **Inhabilitar (`0F5h`)**: deixa de rastrejar les tecles i resta inhibit fins a rebre una ordre d'habilitació.

- **Habilitar (`0F4`)**: reprèn el funcionament interromput per l'ordre anterior o per una altra ordre.
- **Establir una ràtio i un retard d'autorepetició (`0F3h`)**: després d'aquesta ordre s'ha d'enviar un altre *byte* a continuació, interpretat com una dada, que estableix els valors d'autorepetició. En aquest segon *byte*, el bit 7 està sempre a zero i els altres bits indiquen la velocitat. El valor `1Fh` fixa el factor de repetició al mínim possible i `00h` el fixa al màxim possible.
- **Establir/identificar els codis d'escaneig (`0F0h`)**: aquesta ordre selecciona un dels tres conjunts de codis d'escaneig alternatius del teclat MF II: els valors 1, 2 i 3 són vàlids. La configuració estàndard és el conjunt de codis 2. Després de l'ordre, el teclat respon amb un codi de reconeixement (valor `0xFA`) i espera la transferència del segon *byte* amb l'opció del repertori de codis d'escaneig seleccionat: 1, 2 o 3.
- **No operació (de `0F7h` a `0FDh` i de `0EFh` a `0F2h`)**: són codis reservats. En rebre'ls, el teclat envia el senyal de reconeixement habitual i no fa cap acció.
- **Eco (`0EEh`)**: si el teclat rep aquesta ordre, la reenvia a continuació cap al PC. És una ajuda útil per a fer el diagnòstic del dispositiu.
- **Encendre/apagar els leds (`0EDh`)**: després d'aquesta ordre s'ha d'enviar un altre *byte* de dades. Els bits 0, 1 i 2 estan lligats a l'estat dels leds de bloqueig de desplaçament, de bloqueig del teclat numèric i de bloqueig de majúscules, respectivament; els altres estan reservats.

Annex II. Programa exemple

```
#include <stdio.h>
#include <sys/io.h>
#include <unistd.h>

int main(int argc, char **argv)
{
        unsigned int result=0;
        unsigned char scan;
        result = iopl(3);                       // Obté privilegis d'administrador
        if (result < 0) {
                perror("error en obrir els ports");
                return 1;
        }
        fflush(stdin);                          // Buida el buffer d'entrada (stdin)
        sleep (1);                              // espera 1 segon

        while (inb(0x64) & 0x02);               // comprova si es pot enviar una ordre
        outb(0xEE,0x60);                        // Envia l'ordre 0xEE al port0x60

        while(!(inb(0x64)&0x01));               // Verifica si el buffer de recepció no està buit

        scan=inb(0x60);                         // Llegeix el buffer
        sleep (1);                              // espera un segon (no és necessari)
        printf("\t %x \n", scan);               // i imprimeix

        iopl(0);                                // torna el nivell de privilegi a fil normal
        return 0;
}
```

A continuació es descriuen les funcions en C emprades:

- **iopl**. La funció *iopl* serveix per a poder accedir als ports d'entrada i eixida des de l'aplicació passant-li un 3 com a únic paràmetre (es pot utilitzar l'ordre `man iopl` a l'intèrpret d'ordres per a obtenir més informació). La funció *iopl* està inclosa a la llibreria *sys/io.h*, i torna un `0` si l'acció ha reeixit i `-1` si aquesta ha fallat. L'aplicació (programa executable) s'ha d'executar amb permisos d'administrador. El prototip de la funció és:

  ```
  int iopl(int level);
  ```

- **inb** i **outb**. Per a accedir als ports s'usen les funcions *inb* i *outb*. Per a poder usar les dues funcions, i les bàsiques d'entrada i eixida, s'han d'incloure en el programa les llibreries *sys/io.h* i *stdio.h* respectivament. Els prototips d'aquestes funcions es mostren a continuació:

  ```
  unsigned char inb(unsigned short port);
  /* Llegeix un valor de 8 bits del port especificat */
  void outb (unsigned char valor, unsigned short port);
  /* Escriu un valor de 8 bits al port especificat */
  ```

- **sleep**. La funció *sleep* és una funció blocadora que implementa un retard igual al valor del paràmetre que es passa, mesurat en segons. Per a utilitzar-la, cal incloure-hi la llibreria `unistd.h`. El prototip de la funció és:

  ```
  unsigned int sleep(unsigned int seconds);
  ```

Annex III. Ús de VMware i Linux

Tot el programari ha d'estar instal·lat amb antelació al laboratori. No obstant això, per a poder preparar la sessió a casa, s'ha de descarregar la màquina Linux i el seu reproductor.

La màquina Linux NETinVM es pot descarregar de l'adreça següent:

`http://www.netinvm.org`

El reproductor VMware es pot descarregar d'ací:

`https://www.vmware.com/go/downloadplayer`

Una vegada instal·lat VMware i descomprimit l'arxiu, la màquina virtual s'ha d'obrir amb el reproductor VMware amb l'opció del menú: ***File> open a virtual machine***.

L'arxiu que s'obri té l'extensió `vmx`. Per exemple: `NETinVM KVM 2020-07-15.vmx`, però pot ser-ne una versió més recent.

A continuació s'ha d'executar la màquina amb l'opció: ***virtual machine-> power> play virtual machine***.

La primera vegada que s'executa la màquina virtual, el programa pregunta si la màquina s'ha mogut o si s'ha copiat. La resposta ha de ser que s'ha copiat. La màquina s'inicia sense demanar cap contrasenya i té l'aspecte següent:

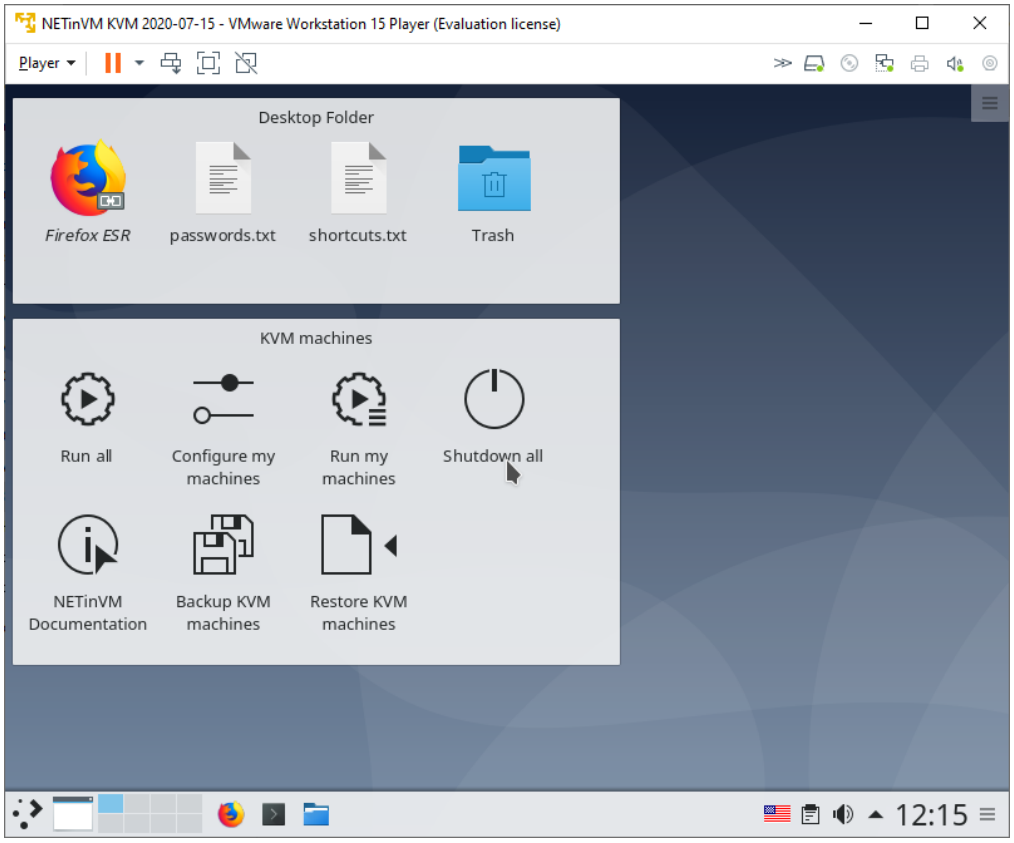


Figura 2. Entorn gràfic de NETinVM.

Tasques inicials:

- **La finestra es pot maximitzar** perquè ocupe tota la pantalla, com qualsevol altra finestra de Windows.
- **Canviar el teclat a espanyol**: a la part inferior dreta de la finestra apareix una bandera dels EUA, cosa que indica que el teclat és compatible amb teclats d'aquest país. Per a canviar-lo a espanyol, simplement s'ha de fer clic a la bandera. Llavors hauria d'aparèixer una bandera espanyola.

A la part inferior esquerra hi ha alguns enllaços directes interessants que es poden veure en aquesta figura:

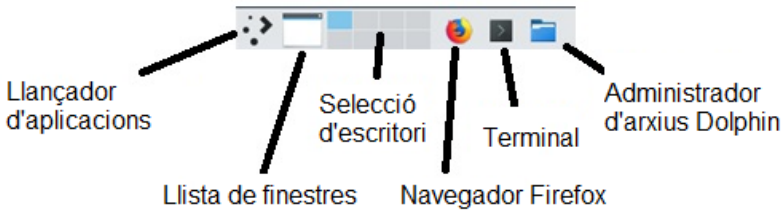


Figura 3. Enllaços directes en la part inferior esquerra.

Proposem crear un directori específic per a la sessió de laboratori. La forma més senzilla de fer-ho és obrir l'administrador d'arxius *Dolphin* i, dins de la carpeta *Documents*, crear una nova carpeta amb el nom P6 (clic amb el botó dret i opció crear carpeta).

La carpeta i tots els elements creats dins seu són accessibles a través del gestor de fitxers, però també es pot accedir al directori i a tots els seus elements mitjançant el terminal. Per a fer-ho, simplement s'ha de fer clic en la icona del terminal (vegeu la figura 3). Apareix un terminal/consola (anomenada *Konsole*):

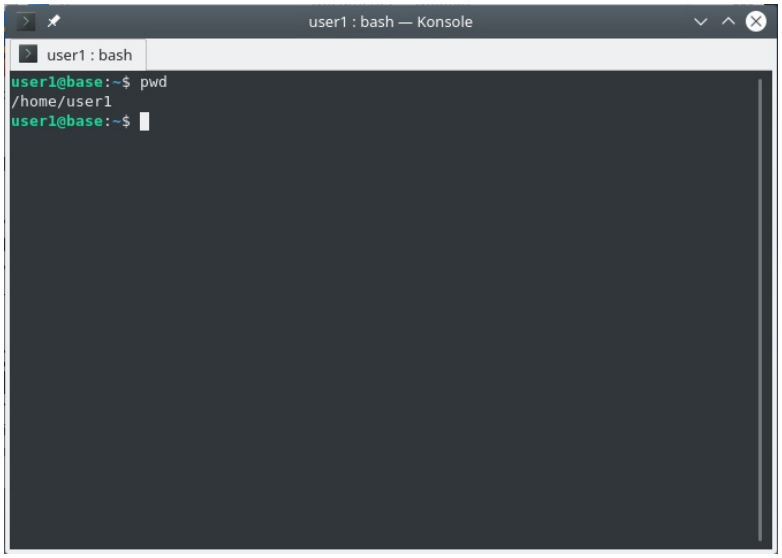


Figura 4. El terminal després d'executar l'ordre `pwd`.

Podem saber quin és el directori actual simplement amb l'ordre `pwd`. En el cas de la imatge, el terminal respon amb la ruta: `/home/user1`.

Per a anar al directori P6 acabat de crear, cal canviar el directori mitjançant l'ordre `cd` o *canviar directori* més la ruta relativa a la posició actual. En aquest cas, l'ordre seria: `cd`

`Documents/P6/` perquè P6 és dins de la carpeta `Documents`. La figura 5 mostra els missatges que apareixen després d'executar l'ordre `cd` i una ordre `pwd` addicional, per a mostrar novament la ruta absoluta.

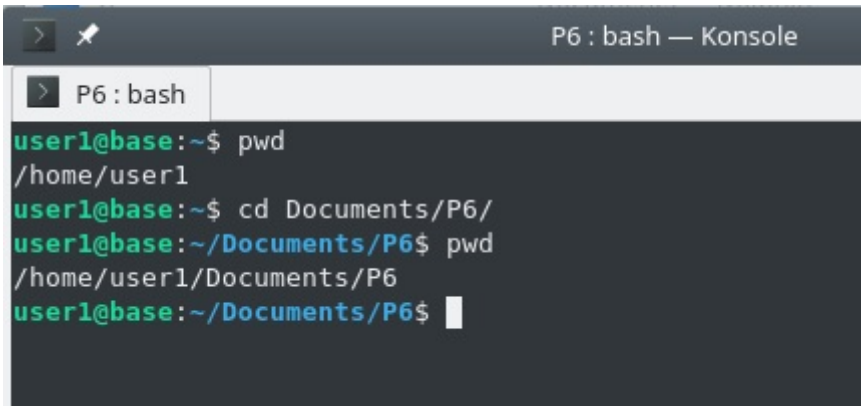


Figura 5. El terminal després d'executar les ordres `cd` i `pwd`.

Per a editar les fonts en NETinVM, és una bona idea usar l'editor KWrite, que està disponible al menú:

```
Application Launcher> Text Editor
```

Una vegada que el codi s'ha desat amb l'extensió `.c`, l'editor usa automàticament el color de fonts a partir de la sintaxi, la qual cosa és extremadament útil. La figura 6 mostra el codi de l'annex II copiat i enganxat al *KWrite*, després de ser desat amb extensió `.c`.

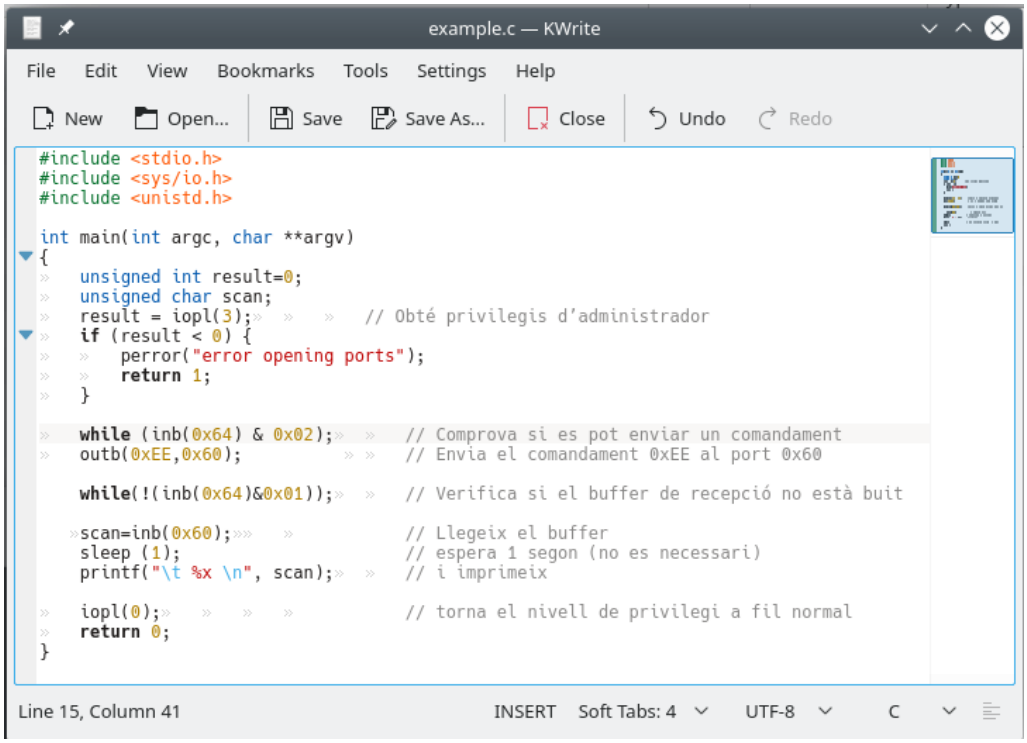


Figura 6. Exemple de codi font escrit amb Kwrite.

Per a compilar els programes, s'ha d'executar l'ordre següent en el terminal:

```
gcc arxiu-font.c -o arxiu-executable
```

L'arxiu d'eixida és l'arxiu executable. Aquest fitxer no necessita extensió. Per a executar l'arxiu, cal tenir privilegis d'administrador. Per a fer-ho, simplement s'ha d'executar:

```
su
```

Aquesta ordre sol·licita la contrasenya de l'administrador, que s'emmagatzema en l'arxiu `passwords.txt`, situat a la carpeta de l'escriptori (vegeu la figura 2). La contrasenya és la mateixa per a *user1* (usuari predeterminat) i *root* (administrador), i s'ha d'introduir sense les cometes dobles. Es pot copiar de l'arxiu i enganxar directament (clic dret a la finestra del terminal).

> Atenció: en el cas de ser administradors, cal tenir molta cura perquè qualsevol arxiu es pot modificar o esborrar, de manera que el sistema de fitxers pot ser destruït.

En el cas de l'administrador, per a executar el codi s'ha d'usar l'ordre següent en el mateix directori on és l'arxiu executable:

```
./executable-file
```

En el cas del codi d'exemple `exemple.c`, tota la seqüència d'accions: compilar, obtenir privilegis d'administrador i executar el codi es pot veure a la figura 7 (la contrasenya no apareix quan s'enganxa al terminal):

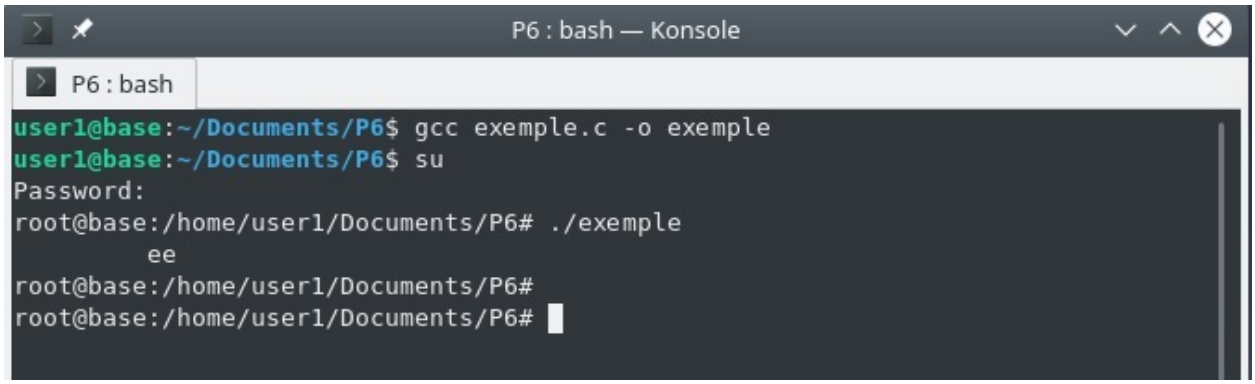


Figura 7. Compilació, obtenció de privilegis d'administrador i execució del programa exemple.

Després de l'execució del programa, el mòdul d'entrada i eixida retorna `0xEE`, que és el retorn esperat en l'ordre `eco`.

Lab 7

Laboratori 7
Perifèrics
Matriu redundant de discos independents (RAID)

(Actualitzat el dia 16/9/2021)

Temps de treball a casa (previst): 1 hora i 30 minuts
Temps de laboratori: 2 hores i 30 minuts

1. Introducció

Objectius

- Comprendre els conceptes bàsics relacionats amb diversos tipus de RAID, en particular les característiques: capacitat d'emmagatzematge, velocitat de lectura, velocitat d'escriptura i tolerància a fallades.
- Aprendre a crear i administrar la majoria de classes de RAID.

Recursos

En aquestes sessions de laboratori s'usa un PC i la màquina virtual NETinVM. Per a executar la màquina virtual es necessita el reproductor VMware, que és gratuït i es pot descarregar del lloc web de l'empresa VMware. Els arxius de la màquina virtual Linux estan disponibles a l'Aula Virtual. Una vegada descomprimit l'arxiu, es pot executar en el reproductor de la màquina virtual (vegeu l'annex II).

2. Treball previ al laboratori

S'han de completar les activitats següents a fi d'aprofitar al màxim aquesta sessió de laboratori. És important dedicar-hi el temps indicat. Tots els membres del grup de treball han de llegir els annexos d'aquest document.

Després de llegir els annexos, s'han de contestar les qüestions següents:

Q1. S'ha d'explicar com és possible assolir més velocitat de lectura amb un RAID 0.

Q2. Per què la velocitat d'escriptura és molt més lenta que la de lectura en un RAID 1?

Q3. Quina és la capacitat d'emmagatzematge en un RAID 10, amb quatre discos actius d'un TiB?

Q4. Quina és la velocitat d'escriptura d'un RAID 5 amb quatre discos de 200 IOPS?

Q5. S'ha de comparar la tolerància a fallades d'un RAID 6 amb un RAID 10, tots dos amb quatre discos.

Q6. Com és possible executar 'exta', directament des de 'base', sense la interfície gràfica? i com és possible obrir una consola en 'exta' des de 'base'?

Q7. Quina és la utilitat de l'ordre *mdadm*?

Q8. Què són els RIOPS i els WIOPS?

Primer punt de control: s'han de lliurar les respostes a les qüestions Q1-Q8 i respondre a les preguntes del professorat sobre aquestes qüestions.

3. Treball al laboratori

En l'annex II hi ha instruccions sobre com s'inicia la màquina NETinVM amb VMware Player. NETinVM és una única imatge de màquina virtual VMware que conté, a punt per a executar-se, una sèrie de màquines virtuals KVM (*kernel virtual machine*) que, quan s'inicien, formen una xarxa completa dins de la màquina virtual VMware. NETinVM inclou les màquines 'exta' i 'base'. Tots els RAID s'han de crear en 'exta'. La màquina 'base' només s'usa per a configurar 'exta'.

Per a començar des d'un estat controlat, s'han de completar els passos següents i exactament en aquest ordre. Per a fer-ho, s'han de completar les seccions descrites en l'annex II amb el mateix nom.

1. **Descàrrega i inici de NETinVM.** No cal descarregar la màquina en un PC de laboratori on la màquina ja estiga descarregada. Però sí per a preparar la pràctica a casa.
2. **Eliminació dels discos en 'exta'**.
3. **Configuració i inici d''exta'**.
4. **Preparació dels discos en 'exta'**.

A fi de confirmar que els discos estan connectats, s'ha d'executar l'ordre descrita en la secció "**Confirmar que els discos estan connectats**" de l'annex II.

Creació d'un RAID 0 i mesurament del rendiment

A continuació s'explica la creació del primer RAID, per la qual cosa és important que no sols s'executen les ordres, sinó que s'entenga què s'està fent. D'aquesta manera, la resta de la sessió serà senzilla, ja que amb CRTL+R es podran recuperar les ordres i es podran editar correctament. A més a més, la tecla de la fletxa cap amunt en el teclat es pot usar per a recuperar ordres anteriors. Inicialment, les ordres es poden copiar d'aquest document i enganxar a la consola 'exta' amb un clic dret del ratolí a la consola i després cal triar *Enganxa*.

Exemple

Es proposa crear un RAID 0 amb dos discos: *sda* i *sdb*, amb el nom `elmeuraid0`. No s'usen discos de recanvi en calent (*hot-spare*) perquè aquesta mena de RAID no implementa redundància. La creació del RAID es pot fer amb l'ordre següent (no s'hi ha d'incloure el símbol "#" de l'intèrpret d'ordres):

```
# mdadm --create /dev/md/elmeuraid0 --level=0 --raid-devices=2 /dev/sd[a-b]
```

Una manera de saber quants RAID hi ha en el sistema és executar l'ordre següent:

```
# cat /proc/mdstat
```

Es poden obtenir detalls del RAID usant `mdadm --detail`, amb l'especificació del nom del RAID. L'ordre torna informació sobre RAID com ara capacitat d'emmagatzematge, estat dels discos, etc.

```
# mdadm --detail /dev/md/elmeuraid0
```

En aquest cas, cal destacar que la capacitat és l'esperada amb un RAID 0 de dos discos de 300 MiB. L'espai final disponible al RAID és el de la suma dels dos discos.

Per a mesurar la velocitat de lectura s'ha d'executar l'ordre *fio* que es mostra a continuació. Atenció, el nom del RAID s'ha de canviar segons siga necessari. En aquest primer cas de RAID 0 (`elmeuraid0`), el nom correcte ja està escrit, de manera que només s'ha de copiar i enganxar:

```
# fio --filename="/dev/md/elmeuraid0" --direct=1 --norandommap --rw=randread --ioengine=libaio --bs=512B --iodepth=16 --runtime=60 --name=totaliops200
```

S'iniciarà el mesurament de la velocitat de lectura i el percentatge de progrés es mostrarà a la consola. Es necessita una mica de temps fins que arribe al 100%. La informació final tindrà un aspecte similar a la què es mostra en la figura 1 (es recomana fer la terminal prou gran a fi d'evitar confondre's amb els salts de línia).

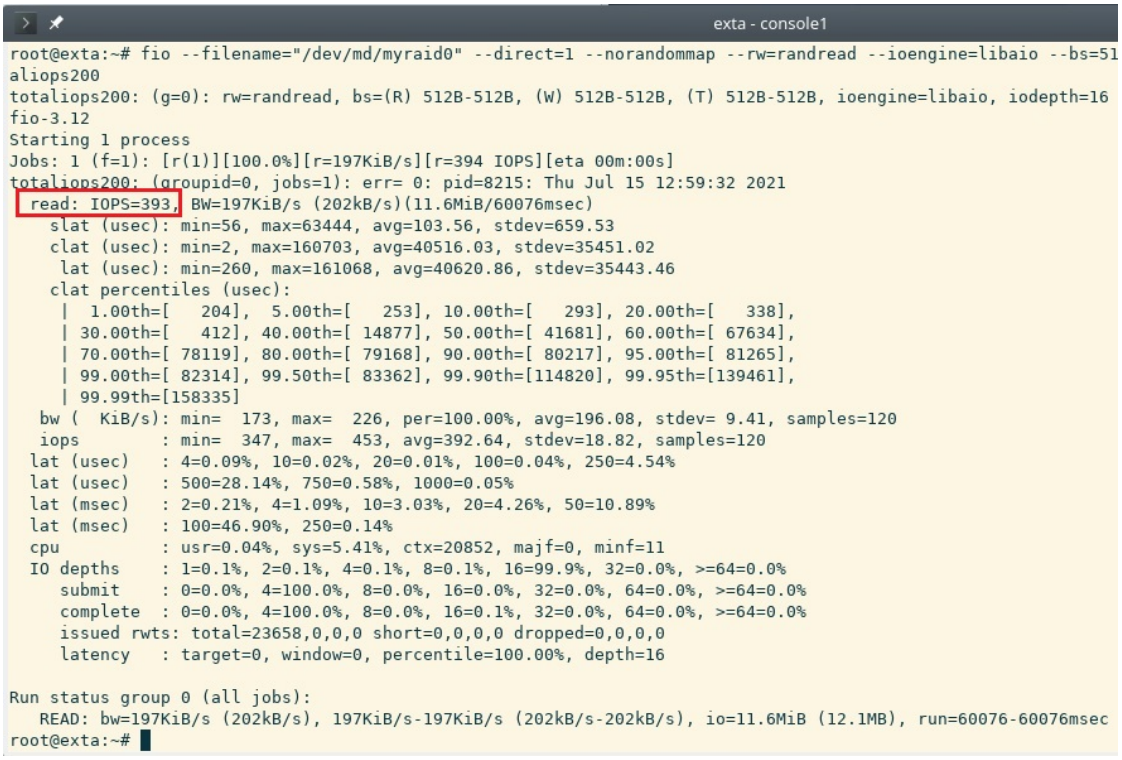


Figura 1. Execució de l'ordre `fio` per a mesurar la velocitat de lectura del RAID.

En l'exemple de la figura es veu que IOPS = 393; però pot ser lleugerament diferent. És interessant comparar aquest resultat amb el resultat esperat. Com que és un RAID 0, la velocitat esperada és la d'un disc multiplicada pel nombre de discos. En aquest cas, com que la velocitat de lectura d'un disc és de 200 IOPS i hi ha dos discos al RAID, la velocitat esperada és de 400 IOPS. El resultat experimental és de 393 IOPS, xifra molt pròxima, i l'error relatiu és (400-393)/400*100=1,75%. Si el resultat obtingut és força diferent del valor esperat, com ara un error relatiu més gran del 20%, podria ser una bona idea executar novament l'ordre *fio*, revisar si tots els paràmetres són correctes, etc. També és bona idea tornar a calcular el valor esperat per tal de poder aclarir la font de l'error.

De la mateixa manera, per a mesurar la velocitat d'escriptura, l'ordre que s'executa és:

```
# fio --filename="/dev/md/elmeuraid0" --direct=1 --norandommap --rw=randwrite --ioengine=libaio --bs=512B --iodepth=16 --runtime=60 --name=totaliops200
```

El procés de mesurament començarà de nou i tardarà uns quants segons a arribar al 100%. El resultat final serà una cosa semblant al que es mostra en la figura 2:

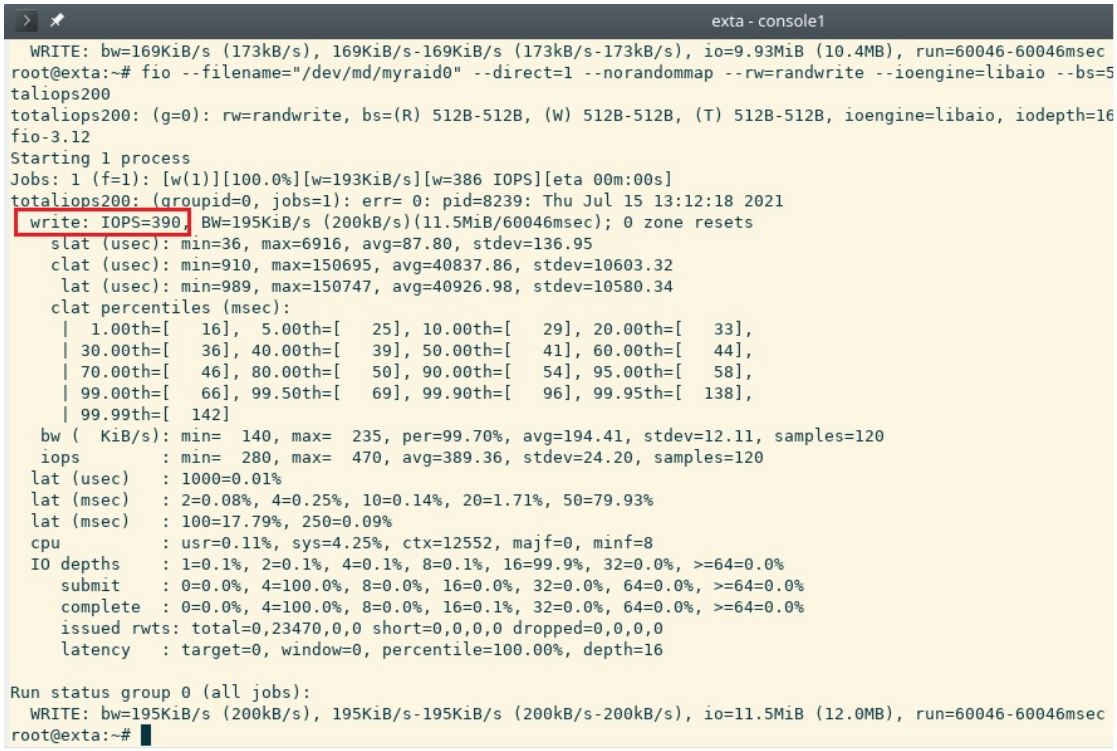


Figura 2. Execució de l'ordre `fio` per a mesurar la velocitat d'escriptura del RAID.

En aquest cas, la velocitat d'escriptura és de 390 IOPS, també molt pròxima a la velocitat d'escriptura esperada de 400 IOPS.

Per a eliminar el RAID, com que encara no s'ha creat un sistema d'arxius i no s'ha muntat en el sistema, n'hi ha prou amb aturar el RAID, indicant el nom del RAID, amb l'ordre següent:

```
# mdadm --stop /dev/md/elmeuraid0
```

És bona idea eliminar qualsevol informació de configuració del RAID i, així, preparar els discos per a usar-los en RAID posteriors. Això es pot fer amb aquesta ordre:

```
# mdadm --misc --zero-superblock /dev/sd[a-b]
```

El RAID `elmeuraid0` ja no hauria d'estar present. Això es pot verificar amb aquesta ordre:

```
# cat /proc/mdstat
```

El sistema ara està a punt per a crear un RAID nou.

3.1. Es proposa crear un RAID 1, anomenat *elmeuraid1*, amb dos discos actius. En aquest cas, com que aquest tipus de RAID admet tolerància a fallades, és possible afegir-hi discos de recanvi en calent, de manera que s'hi ha d'agregar un disc addicional d'aquest tipus. La secció *create* de l'ordre *mdadm* en l'annex III mostra l'ús de l'opció `--create` per a crear RAID amb discos de recanvi. L'ordre que apareix en aquesta secció s'ha de personalitzar de la forma adequada. En aquest cas, es proposa el nom *elmeuraid1*. L'opció `--level` ha de ser 1 perquè és un RAID 1. La quantitat de discos actius és 2, de manera que l'opció `--raid-devices` és 2. El disc de recanvi s'especifica amb `--spare-devices=1`. Els discos emprats són `/dev/sd[a-c]`. El sistema tarda uns quants segons a crear-se. És possible observar l'estat del sistema, fins i tot el percentatge de creació, amb `cat/proc/mdstat`.

S'ha de mesurar la mida de l'emmagatzematge disponible al RAID, juntament amb la velocitat de lectura i escriptura. ¿La mida observada coincideix amb les prediccions explicades a l'annex I?

Es pot simular una fallada d'un disc, per exemple de `sda`, i s'ha d'observar què succeeix en el sistema. L'ordre per a marcar el disc `sda` com a defectuós és:

```
# mdadm /dev/md/elmeuraid1 --fail /dev/sda
```

Tan prompte com el disc `sda` es marca com a defectuós, es pot veure com es reconstrueix el sistema amb l'ordre `cat /proc/mdstat` o també amb `mdadm --detail /dev/md/elmeuraid1`.

Una vegada acabada aquesta part, el RAID s'ha de parar i s'ha d'eliminar-hi qualsevol informació de configuració amb les ordres ja introduïdes en l'exemple del RAID 0.

3.2. El RAID 10, tal com es pot llegir a l'annex I, combina els avantatges del RAID 0 i del RAID 1. Es proposa crear un RAID 10 amb el nom *elmeuraid10*, inicialment amb quatre discos en RAID i dos discos de recanvi en calent. El temps de creació és més llarg que en el cas anterior perquè ara la redundància es fa amb més discos. És una bona idea observar el procés de construcció amb l'ordre `cat /proc/mdstat` fins que es complete.

De la mateixa manera, com s'ha fet abans, s'ha de mesurar la mida de l'emmagatzematge disponible al RAID, juntament amb la velocitat de lectura i escriptura. ¿La mida i velocitat observades coincideixen amb les prediccions explicades a l'annex I?

Novament, el RAID ha de parar-se i s'ha d'eliminar-hi qualsevol informació de configuració.

Segon punt de control: cal mostrar al professor o professora totes les ordres, els resultats de capacitat d'emmagatzematge i la velocitat mesurada, tant per al RAID *elmeuraid1* com per al RAID *elmeuraid10*. També cal provocar una fallada en un dels discos dels RAID i veure com es recupera el sistema.

Una vegada que s'ha creat qualsevol tipus de RAID, s'hi pot crear un sistema d'arxius i afegir-lo a un sistema d'arxius existent. Atès que tots aquests conceptes s'aprendran més endavant en l'assignatura Sistemes Operatius, en aquesta sessió de laboratori no s'inclouen exercicis amb sistemes de fitxers.

Es pot fer créixer un RAID agregant discos al sistema. S'ha de destacar que fer créixer un RAID consisteix a agregar discos actius al RAID, no agregar-hi discos de recanvi, encara que també és possible afegir-hi un disc, que per defecte s'agregarà al RAID com a disc de recanvi, per a després incorporar-lo activament al sistema. Per exemple, si *elmeuraid5* fora d'un RAID 5, amb tres discos actius i un disc de recanvi seria possible augmentar la quantitat de discos en el RAID usant el disc de recanvi. Per a fer-ho s'ha d'executar l'ordre següent:

```
# mdadm --grow --raid-devices=4 /dev/md/elmeuraid5
```

No obstant això, s'hi pot afegir directament un disc actiu. Per a fer-ho cal usar l'opció `--grow`, amb la indicació del nou disc, en aquest cas `sde`, i de la nova configuració RAID amb cinc discos amb l'opció `--raid-devices`:

```
# mdadm --grow /dev/md/elmeuraid5 --add /dev/sde --raid-devices=5
```

Per a afegir-hi un altre disc de recanvi, en aquest cas `sdf`, es fa amb l'opció `--add-spare`. Això fa que passem al mode d'administració, que no té una opció com a tal per a ser seleccionada. L'ordre seria la següent:

```
# mdadm /dev/md/elmeuraid5 --add-spare /dev/sdf
```

3.3. S'ha de crear un RAID 5, *elmeuraid5*, amb quatre discos actius i un disc de recanvi. Cal mesurar la capacitat d'emmagatzematge que té, la velocitat de lectura i d'escriptura.

A continuació, el RAID s'ha de fer créixer convertint el disc de recanvi en un disc actiu, de manera que el RAID tindria 5 discos actius i cap disc de recanvi. Una vegada més, s'ha de mesurar la capacitat d'emmagatzematge, la velocitat de lectura i escriptura, sempre després de finalitzar el procés de creixement. Cal tenir en compte que aquest procés de creixement pot necessitar un cert temps i es pot observar amb l'ordre `cat/proc/mdstat`.

De nou, el RAID ha de parar-se i s'hi ha d'eliminar qualsevol informació de configuració.

3.4. S'ha de crear un RAID 6, *elmeuraid6*, amb quatre discos actius, ja que no es necessita cap disc de recanvi. La creació pot tardar uns quants minuts. Quan acabe, cal mesurar la capacitat d'emmagatzematge que té, la velocitat de lectura i d'escriptura. Després el RAID s'ha de fer créixer afegint-hi directament un disc actiu. De nou, una vegada finalitzat el procés de creixement, s'ha de mesurar la capacitat que té, la velocitat de lectura i d'escriptura.

Tercer punt de control: s'han de mostrar al professorat totes les ordres emprades, la mida de la capacitat d'emmagatzematge i les velocitats mesurades per als RAID *elmeuraid5* i *elmeuraid6*, abans i després de fer-los créixer. ¿Els resultats coincideixen amb els valors esperats? Quines són les diferències?

Annex I. Revisió de RAID

El terme RAID va nàixer en un article publicat l'any 1988 (Patterson, Garth, & Randy, 1988). Actualment, el terme RAID és un acrònim de matriu redundant de discos independents (***r**edundant **a**rray of **i**ndependent **d**iscs*). Un RAID és una manera d'organitzar la informació en diversos discos durs amb la intenció de tenir protecció contra fallades de disc o un accés més ràpid a la informació.

Un RAID és un disc dur lògic per al sistema operatiu. Les dades s'escriuen en fragments en diversos discos simultàniament. Hi ha diverses configuracions RAID que ofereixen més tolerància a fallades i nivells més alts de rendiment que un sol disc dur o que un grup de discos durs independents.

Originalment, es van definir més nivells de RAID dels que es veuran en aquesta sessió de laboratori. Algunes configuracions teòriques de RAID no s'implementen en l'actualitat per raó de rendiment i cost. Altres configuracions reals són barreges de diversos tipus de RAID. Les configuracions RAID més populars es descriuen a continuació.

RAID 0

Aquest nivell de RAID s'anomena segmentació del disc en fragments (*disc striping*). Consisteix a dividir la informació dels arxius originals en trossos que s'escriuen o es llegeixen en paral·lel en els discos. Com més discos hi haja, més fragmentada està la informació que s'ha d'escriure o llegir dels N discos. Aquesta classe de RAID ofereix la màxima velocitat: amb N discos amb velocitat X hi ha una velocitat de lectura i escriptura de $N \cdot X$. El punt feble d'aquesta configuració és que no té tolerància a fallades.

La capacitat d'emmagatzematge es fa servir completament. Amb N discos de mida S, la capacitat és $N \cdot S$.

El nombre mínim de discos per a un RAID 0 és de dos.

RAID 1

Aquest cas és el contrari del RAID 0. En lloc de segmentar el disc, es fan còpies espill del disc original (*mirroring*), de manera que hi ha tolerància a fallades. Un RAID 1 amb N discos de velocitat X té una velocitat de lectura $N \cdot X$ ja que és possible fer peticions de lectures de diferents fragments als diversos discos en paral·lel. No obstant això, no és possible accelerar el procés d'escriptura perquè s'ha d'escriure el mateix fragment en els N discos al mateix temps, de manera que la velocitat d'escriptura és X. De manera similar, l'espai d'emmagatzematge s'usa en les còpies de seguretat i, per tant, tenir més discos no es tradueix en més espai útil. Amb N discos de mida S hi ha un espai de S, igual que amb un sol disc.

El nombre mínim de discos per a construir un RAID 1 és de dos. Amb dos discos, el RAID pot tolerar la fallada d'un. Amb més discos s'aconsegueix més redundància i, així, es tolera la fallada de més discos.

RAID 10

També conegut com RAID 1+0. Consisteix a combinar les dues estratègies: segmentació i còpies espill. Això fa possible l'augment de la velocitat i, simultàniament, proporciona tolerància a fallades mitjançant redundància. Un RAID 10 amb N discos de velocitat X té una velocitat de lectura $N \cdot X$, ja que és possible llegir la informació per fragments de tots els discos al mateix temps, fins i tot d'aquells que són còpies espill. Per contra, amb escriptures

no és possible aprofitar l'accés als N discos al mateix temps perquè la meitat dels discos s'usen per a copiar l'altra meitat. Per tant, la velocitat d'escriptura és $N \cdot X/2$.

Per tant, l'espai d'emmagatzematge visible és el de la meitat dels discos. Amb N discos de mida S hi ha un espai $N \cdot S/2$.

El nombre mínim de discos per a construir un RAID 10 és quatre.

RAID 5

Aquest RAID ofereix segmentació de dades amb paritat distribuïda. La informació de paritat es distribueix de manera entrellaçada entre els N discos del RAID, sense discos espill, de manera que la informació es pot reconstruir en temps real si falla algun disc del RAID. Amb N discos amb velocitat X, hi ha una velocitat de lectura $N \cdot X$. No obstant això, per a fer una escriptura cal llegir les dades, llegir la paritat, després escriure les dades i escriure la paritat, de manera que la velocitat d'escriptura es penalitza en un factor quatre. Per tant, seria $N \cdot X/4$.

L'espai d'emmagatzematge també es redueix a causa de la informació redundant. Amb N discos de capacitat S, l'espai d'emmagatzematge disponible és $(N-1) \cdot S$.

El nombre mínim de discos per a construir un RAID 5 és tres. En aquest tipus de RAID es tolera la fallada d'un disc.

RAID 6

Aquest nivell de RAID és similar al RAID 5, però ofereix un sistema de doble paritat, de manera que hi ha més redundància i tolerància a fallades. Amb N discos amb velocitat X, la velocitat de lectura és $N \cdot X$. No obstant això, és diferent per a l'escriptura. Per a fer una escriptura, cal llegir les dades, llegir els dos fragments de paritat, després escriure les dades i els dos fragments de paritat, de manera que la velocitat d'escriptura es penalitza en un factor sis i seria $N \cdot X/6$.

L'espai d'emmagatzematge també es redueix encara més a causa de l'esquema de doble paritat. Amb N discos de mida S, l'espai d'emmagatzematge disponible és $(N-2) \cdot S$.

En aquest cas, el nombre mínim de discos per a construir un RAID 6 és quatre. El més interessant d'aquest tipus de RAID és que tolera la fallada de dos discos.

Resum de beneficis

La taula següent resumeix la capacitat dels diversos tipus de RAID. Es pressuposa que els N discos són iguals, amb velocitat X i amb capacitat S. La columna **RIOPS** són les operacions d'entrada i eixida de lectura per segon (*read input/output operations per second*). La columna **WIOPS** són les operacions d'entrada i eixida d'escriptura per segon (*write input/output operations per second*). La columna **Capacitat** mostra l'espai d'emmagatzematge disponible al RAID. La columna **Mínim** indica el nombre mínim de discos necessaris per a crear el RAID i la columna **Fallades** mostra el nombre de discos que poden fallar simultàniament.

RAID	*RIOPS*	*WIOPS*	*Capacitat*	*Mínim*	*Fallades*
0	$N \cdot X$	$N \cdot X$	$N \cdot S$	2	0
1	$N \cdot X$	X	S	2	$(N-1)$
10	$N \cdot X$	$N \cdot X/2$	$N \cdot S/2$	4	entre 1 i $N/2$*
5	$N \cdot X$	$N \cdot X/4$	$(N-1) \cdot S$	3	1
6	$N \cdot X$	$N \cdot X/6$	$(N-2) \cdot S$	4	2

*Depèn de quins discos específics fallen en la configuració del RAID.

Annex II. Preparació de la màquina

1. Descàrrega i inici de NETinVM

Tot el programari s'ha d'instal·lar prèviament al laboratori. No obstant això, per a preparar la sessió és bona idea descarregar la màquina Linux i el seu reproductor.

La màquina NETinVM Linux s'ha de descarregar d'ací:

`http://www.netinvm.org`

El reproductor VMware es pot descarregar d'ací:

`https://www.vmware.com/go/downloadplayer`

Una vegada que VMware està instal·lat i l'arxiu descomprimit, la màquina virtual s'ha d'obrir amb el reproductor VMware amb l'opció del menú: ***file->open a virtual machine***.

L'arxiu que s'obri té l'extensió `vmx`. Per exemple: `NETinVM KVM 2020-07-15.vmx`, però pot ser-ne una versió més nova. A continuació s'ha d'executar la màquina: ***virtual machine->power->play virtual machine***.

La primera vegada que s'executa la màquina virtual, el programa pregunta si la màquina s'ha mogut o si s'ha copiat. La resposta ha de ser que s'ha copiat.

NETinVM és una única imatge de màquina virtual VMware que conté, a punt per a executar-se, una sèrie de nuclis de màquines virtuals KVM (*kernel virtual machine*) que, quan s'inicien, formen una xarxa completa dins de la màquina virtual VMware. Per això el nom NETinVM, un acrònim de *NETwork in virtual machine*. NETinVM s'ha concebut principalment com una eina educativa per a ensenyar i aprendre sobre sistemes operatius, xarxes informàtiques i seguretat de sistemes, però en aquesta sessió s'utilitzarà per a construir diversos tipus de RAID. Després d'executar NETinVM mitjançant el reproductor VMware, apareix la interfície gràfica de la màquina, com es mostra en la figura 3.

La interfície gràfica s'executa en la màquina 'base'. Les altres màquines no tenen interfície gràfica. Cal tenir en compte que el tipus de teclat es pot canviar, si cal, a espanyol simplement fent clic a la bandera dels EUA Que apareix a la cantonada inferior dreta de la pantalla. Llavors hauria d'aparèixer la bandera espanyola i el teclat ja ha canviat de tipus.

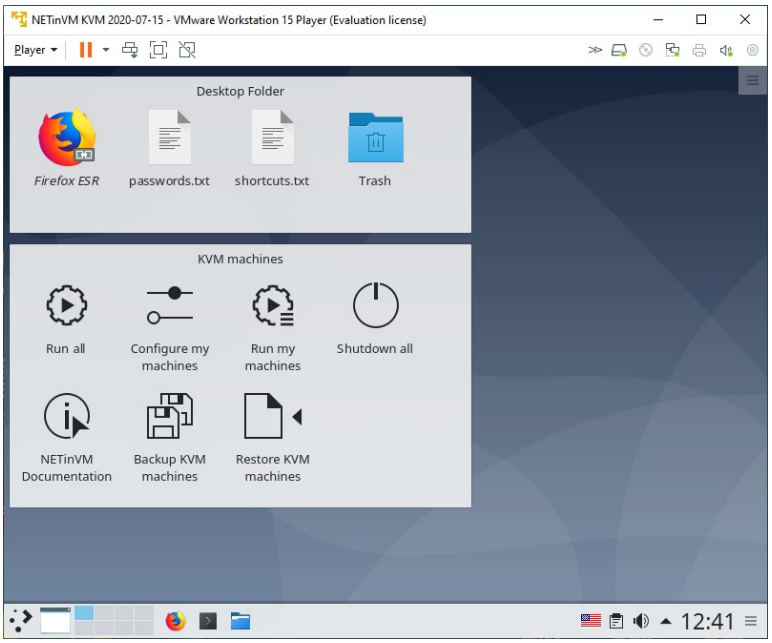


Figura 3. La màquina NETinVM.

Estructura de NETinVM

Aquestes són les màquines incloses en NETinVM:

- `int<X>`: KVM connectades a la xarxa interna. <*X*> pot prendre valors de `a` a `f`, tots dos inclosos.
- `dmz<X>`: KVM connectades a la xarxa perimetral (DMZ). Se suposa que són nodes bastió.
- `ext<X>`: KVM connectades a la xarxa externa (és a dir, Internet).
- `fw`: interconnecta les tres xarxes ('int', 'DMZ' i 'ext'), la qual cosa permet la comunicació i la filtració de paquets.

La carpeta de màquines KVM inclou una interfície gràfica per a diverses accions: configurar, engegar, parar les màquines, etc. El menú està en anglès i inclou (entre parèntesis) el nom de cada opció, tal com apareix en NETinVM. En fer clic sobre els enllaços, fan les accions següents:

- **Executar-les totes** *(run all)*: engega totes les màquines en NETinVM.
- **Configurar les màquines** *(configure my machines)*: llança un editor per a canviar el fitxer d'ordres emprat per a executar les meues màquines.
- **Executar les meues màquines** *(run my machines)*: engega el subconjunt de KVM especificat.
- **Parar-les totes** *(shutdown all)*: para totes les màquines KVM.
- **Documentació NETinVM** *(NETinVM documentation)*: inicia un navegador que mostra una copia local de la documentació.
- **Còpia de seguretat** *(Backup KVM Machines)*: crea una còpia de seguretat de tota la xarxa NETinVM. Totes les màquines s'han de parar abans de fer aquesta còpia.
- **Restaurar les màquines KVM** *(Restore KVM machines)*: elimina l'estat actual de la màquina KVM i la restaura a un estat anterior. El fitxer de còpia de seguretat es pot seleccionar durant el procés. Totes les màquines s'han d'aturar abans de restaurar una còpia de seguretat.

Configuració i inici d''exta'

El primer pas per a configurar la màquina 'exta' és iniciar l'entorn NETinVM mitjançant el programa de virtualització VMware.

A continuació, a fi de començar amb la màquina en un estat controlat, aquesta ha de restaurar-se des d'un punt de recuperació. Per a fer aquesta restauració, totes les màquines KVM han d'estar parades. Totes les màquines es poden parar (excepte 'base', és clar) prement el botó *Parar-les totes* a la carpeta *Màquines KVM*.

Una vegada a l'escriptori de la màquina 'base', s'ha d'usar un navegador web per a accedir a la pàgina de l'assignatura a l'Aula Virtual. Per exemple, es pot usar Firefox, la icona del qual és a la cantonada superior esquerra de l'escriptori. Una vegada que s'ha accedit a la pàgina, s'ha de descarregar l'arxiu de còpia de les màquines KVM, `kvm_machines_backup_EC.tgz`, i es desa en la carpeta de descàrregues de la màquina 'base'.

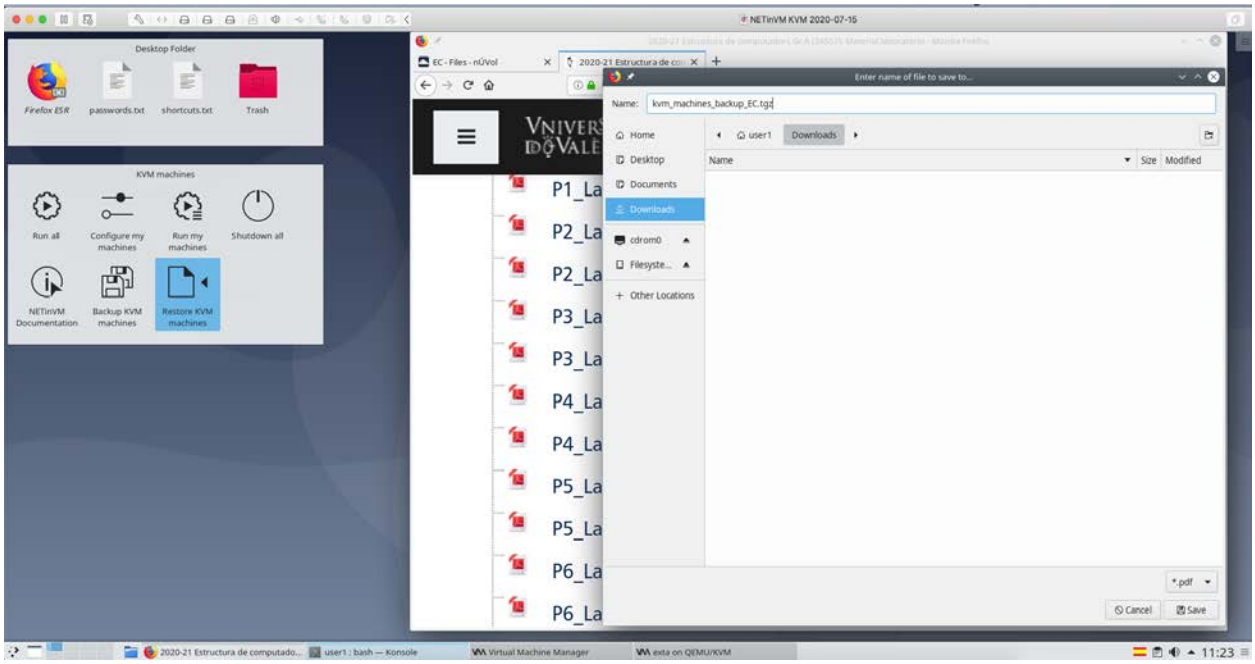


Figura 4. Descàrrega de l'arxiu kvm_machines_backup_EC.tgz *de l'Aula Virtual.*

Una vegada l'arxiu s'ha descarregat, les màquines KVM s'han de restaurar usant l'opció *Restaurar les màquines KVM* disponible al menú *Màquines KVM* a l'escriptori. Quan es completa aquesta acció, s'obri una finestra perquè s'indique l'arxiu de restauració que es vol usar. En aquest cas, l'arxiu és `kvm_machines_backup_EC.tgz` que s'ha descarregat a la carpeta de descàrregues.

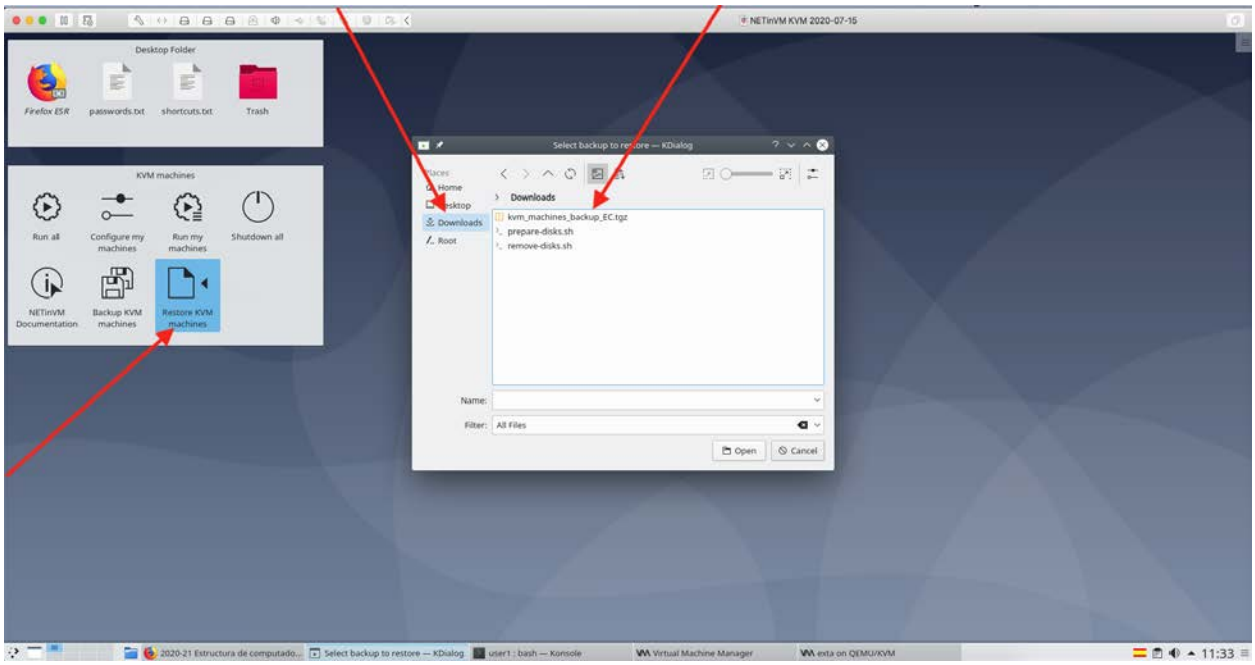


Figura 5. Menú per a restaurar les màquines KVM amb l'arxiu kvm_machines_backup_EC.tg.

Com a informació útil, cal dir que és possible executar una màquina directament, sense la interfície gràfica. Per a fer-ho, simplement s'ha d'executar des d'un intèrpret d'ordres en la màquina 'base' (sent 'user1'):

```
$ netinvm_run exta
```

Una vegada que la màquina s'ha iniciat, apareix una consola. Es pot iniciar la sessió com a administrador (*root*). La contrasenya és la mateixa i s'emmagatzema en l'arxiu `passwords.txt` a la carpeta de l'escriptori.

Si accidentalment es tanca la finestra de la consola, es pot tornar a obrir-ne una des de 'base' sent 'user1' amb l'ordre següent:

```
$ netinvm_console exta
```

Preparació dels discos en 'exta'

S'ha preparat un *script* que crea vuit discos de 300 MiB (`sda`, `sdb`, ... `sdh`), amb velocitat limitada (200 IOPS), i els connecta a 'exta'. Una vegada que s'ha iniciat NETinVM, totes les màquines s'han de parar prement el botó *Parar-les totes* a la carpeta de l'escriptori de les màquines KVM. Una vegada fet això, amb l'script `prepare-disks.sh`, com a 'user1' en 'base', es poden afegir els discos addicionals a qualsevol màquina KVM. Per a fer-ho, primer s'ha de descarregar l'script `prepare-disks.sh` de l'Aula Virtual. Després s'ha de copiar la màquina 'base'. Es pot copiar i enganxar de l'administrador d'arxius de l'explorador (Windows) a l'administrador de fitxers *Dolphin* de la màquina 'base'en NETinVM.

Una vegada l'arxiu s'ha copiat, aquest ha de tenir els permisos d'execució adequats. Per a fer-ho, des de l'intèrpret d'ordres en 'base', en el mateix directori de treball, s'ha d'executar l'ordre:

```
$ chmod u+x prepare-disks.sh
```

Açò afegeix a l'usuari (expressat amb `u`) el permís d'execució (expressat amb `x`).

Per a executar l'script i afegir els discos a 'exta', s'ha d'executar en 'base':

```
$ ./prepare-disks.sh exta
```

Eliminació dels discos d''exta'

De manera similar, es poden eliminar tots els discos agregats a 'exta' amb l'script `remove-disks.sh`. De primer s'ha de descarregar l'script `remove-disks.sh` de l'Aula Virtual. Després s'ha de copiar la màquina 'base'. Es pot copiar i enganxar de l'administrador d'arxius de l'explorador (Windows) a l'administrador de fitxers *Dolphin* de la màquina 'base' en NETinVM.

Una vegada l'arxiu s'ha copiat, aquest ha de tenir els permisos d'execució adequats. Per a fer-ho, des de l'intèrpret d'ordres en 'base', en el mateix directori de treball, s'ha d'executar l'ordre següent:

```
$ chmod u+x ‘remove-disks.sh
```

Açò afegeix a l'usuari (expressat amb `u`) el permís d'execució (expressat amb `x`).

Per a executar l'script i afegir els discos a 'exta', s'ha d'executar en 'base':

```
$ ./remove-disks.sh exta
```

Confirmar que els discos estan connectats

És possible verificar en 'exta' que els discos estan connectats i llestos simplement executant l'ordre:

```
# lsblk
```

Això mostra una vista en mode arbre dels volums d'emmagatzematge. La figura 4 mostra una imatge de NetinVM amb les dues consoles obertes: l'una en 'base' i l'altra en 'exta', després d'executar l'ordre `lsblk` en 'exta'.

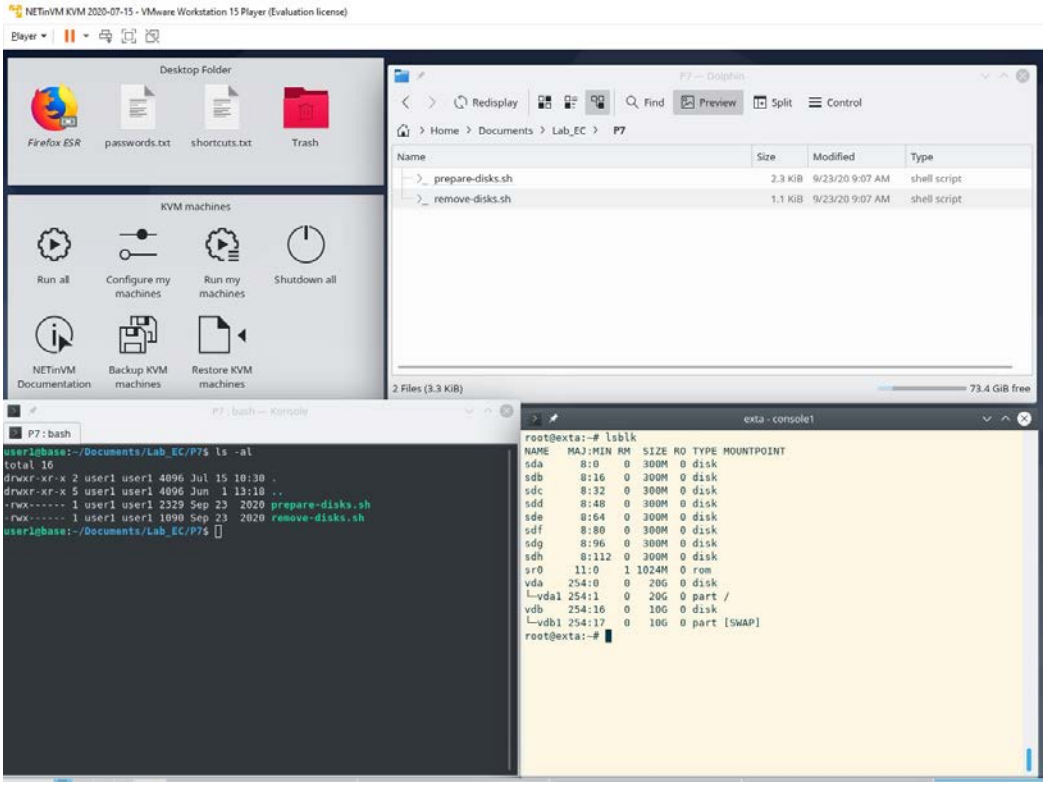


Figura 6. NET VM amb consoles obertes en 'base' i 'exta'.

Annex III. L'ordre `mdadm`

`mdadm` té diversos modes de funcionament. En general, l'ordre `mdadm` té el format següent:

```
# mdadm [mode] <raiddevice> [options] <component discs>
```

Per a executar `mdadm`, es necessiten privilegis d'administrador, de manera que es pressuposa que `mdadm` s'executa com a administrador (*root*).

Els modes es poden especificar amb opcions. De vegades, quan s'estableix una opció determinada, es pressuposa un determinat mode sense necessitat d'especificar-ho explícitament. Si es necessita un mode determinat, es poden usar les opcions següents:

`--create` o de manera equivalent **`-C`**

Crea un nou RAID, amb el nom especificat, i el nivell s'indica després de l'opció `--level` i els dispositius especificats amb l'opció `--raid-devices`. Per exemple, per a crear un RAID 5, anomenat `elmeuraid5`, amb tres discos actius que formen el RAID: `sda`, `sdb` i `sdc`, deixant `sdd` com a disc de recanvi en calent:

```
# mdadm --create /dev/md/elmeuraid5 --level=5 --raid-devices=3 --spare-devices=1 /dev/sd[a-d]
```

`--assemble` o de manera equivalent **`--A`**

Inicia un RAID prèviament definit. És possible que el RAID s'haja definit abans en la línia d'ordres o que s'haja especificat en `/etc/mdadm.conf`. Això generalment es fa en els scripts d'inici després d'iniciar el sistema. Per exemple, l'ordre següent engega tots els RAID definits en `mdadm.conf`:

```
# mdadm --assemble --scan
```

`--follow` o **`--monitor`** o de manera equivalent **`--F`**

Vigila un o més RAID i actua en cas de qualsevol canvi d'estat, i pot configurar `mdadm` com un dimoni per a enviar alertes o executar ordres quan falle un disc. Per defecte, els missatges d'alerta s'envien a l'usuari administrador del sistema. L'ordre següent mostreja successos cada 60 segons en el RAID `elmeuraid5` i envia un correu electrònic a l'administrador del sistema:

```
# mdadm --monitor --mail=sysadmin --delay=60 /dev/md/elmeuraid5
```

`--build` o de manera equivalent **`--B`**

Construeix un RAID sense *superblocs*. Per a aquests tipus de RAID, `mdadm` no pot diferenciar entre la compilació inicial i l'acoblament posterior. A més a més, no es pot comprovar que els discos siguen adequats. Aquest mode no s'ha d'usar llevat que s'entenga completament el funcionament de l'ordre.

`--grow` o de manera equivalent **`--g`**

Es fa servir per a fer créixer, reduir o remodelar un RAID d'alguna manera. Un exemple seria, donat l'exemple de RAID nivell 5, `elmeuraid5`, amb tres discos actius i un disc de recanvi, augmentar el nombre de discos en el RAID usant el disc de recanvi. Per a fer-ho, simplement s'ha de fer:

```
# mdadm --grow --raid-devices=4 /dev/md/elmeuraid5
```

Per a afegir-hi directament un disc actiu, per exemple `sde`, com el cinquè disc del RAID:

```
# mdadm --grow /dev/md/elmeuraid5 --add /dev/sde --raid-
devices=5
```

Per a afegir-hi un altre disc de recanvi en calent, en aquest cas `sdf`, es pot fer simplement amb l'opció `--add-spare`. Això fa que s'entre en el mode d'administració, que no té una opció com a tal per a ser seleccionat. L'ordre seria la següent:

```
# mdadm /dev/md/elmeuraid5 --add-spare /dev/sdf
```

`--misc` o de manera equivalent **`--M`**

Aquest mode inclou totes les opcions que no hi ha en els altres modes. Per exemple, si s'inclou un disc com a primer paràmetre, o si la primera opció és `--add`, `--re-add`, `--add-spare`, `--fail`, `--remove` o `--replace`, llavors se suposa que està en mode de gestió. Qualsevol altra opció farà que s'assumisca el mode `misc`. Exemples útils dins d'aquest mode serien els següents per a veure l'estat del RAID o d'algun component:

```
# mdadm --misc --detail /dev/md/elmeuraid5
# mdadm --examine /dev/sdd
```

Una opció interessant és marcar un disc com a defectuós. Això vol dir que si el RAID implementa redundància, el sistema es reconfigura correctament. Cal assenyalar que en el cas d'un RAID amb discos grans, aquest temps pot ser molt llarg. En el cas de la sessió, amb discos de 300 MiB, aquests temps no superen el minut. En l'exemple següent, el disc `sda` es marca com a defectuós:

```
# mdadm /dev/md/elmeuraid5 --fail /dev/sda
```

Bibliografia

Patterson, D. A., Garth, G. i Randy, K. H. (1988). "A case for redundant arrays of inexpensive discs". *SIGMOD International Conference on Management of Data* (p. 109-116). Chicago, IL, EUA: ACM.

Vadala, D. (2002). *mdadm: A New Tool for Linux Software RAID Management*. Obtingut d'ací: <http://www.linuxdevcenter.com/pub/a/linux/2002/12/05/RAID.html>.

Vadala, D. (2009). *Managing RAID on Linux. Fast, Scalable, Reliable Data Storage.* O'Reilly Media.

Lab 8

Laboratori 8
Perifèrics
Targeta gràfica

(Actualitzat el dia 16/09/2021)

Temps de treball a casa (previst): 1 hora i 30 minuts
Temps de laboratori: 2 hores i 30 minuts

1. Introducció

En aquesta pràctica s'estudia el funcionament de la targeta gràfica de l'ordinador. S'hi mostra la forma en la qual es poden obtenir els paràmetres de configuració de la targeta i com s'efectua l'accés a la memòria de vídeo per a representar imatges i gràfics amb diferents colors. Implementem diversos programes en llenguatge C per a treballar amb la targeta gràfica a través del dispositiu de memòria intermèdia d'imatges (*framebuffer*), disponible en el sistema operatiu Linux.

Objectius

L'estudiant, en acabar aquesta pràctica, ha de ser capaç de:

- Conèixer l'estructura de la memòria de vídeo i la manera d'accedir-hi.
- Usar les funcions de llibreria que permeten accedir al dispositiu de memòria intermèdia d'imatges de Linux per a usar la targeta gràfica a baix nivell.
- Consultar els paràmetres de configuració de la targeta gràfica.
- Dissenyar i implementar programes en C que accedeixen a la targeta gràfica a baix nivell en Linux.

Recursos

En aquesta pràctica s'usa la mateixa màquina virtual de les dues sessions anteriors. Per a executar la màquina virtual cal tenir instal·lat el programa VMware Player, que és gratuït i es pot descarregar de la pàgina web de la companyia VMware. L'entorn NETinVM té preinstal·lat tot el programari necessari per a compilar i executar els programes que cal fer durant la pràctica. L'entorn NETinVM es pot descarregar de l'enllaç següent: http://www.netinvm.org.

2. Treball previ al laboratori

En aquest treball previ, tots els membres de l'equip han de llegir els annexos I, II i III d'aquest document. L'annex I revisa conceptes sobre la targeta VGA i sobre el dispositiu de memòria intermèdia de Linux. L'annex II descriu les funcions que s'usen en l'accés a aquest dispositiu de

Linux. En aquest annex es mostren dos programes exemple. Tots dos implementen l'accés a baix nivell per a comprovar la configuració de la targeta i per a escriure sobre la memòria de vídeo. L'annex III mostra com es configura la màquina 'exta' per a compilar i executar els fitxers de codi en llenguatge C. També s'han de revisar els conceptes sobre la targeta gràfica de l'ordinador descrits en el tema 3: perifèrics. Una vegada efectuada la lectura dels annexos i els apunts de classe, cal respondre a les qüestions següents:

Q1. S'ha de descriure quin tipus de dispositiu és la memòria intermèdia d'imatges de Linux i com es presenta a les aplicacions que volen fer-la servir.

Q2. S'ha d'explicar quina utilitat té la funció *mmap*.

Q3. Per a què s'usa la funció *ioctl*?

Q4. S'han de llegir els programes exemple de l'annex II i descriure què fan.

Q5. S'ha d'explicar el càlcul que s'ha fet en el programa exemple per a saber la mida de la memòria de vídeo:

- `mida_pantalla = vinfo.xres * vinfo.yres * vinfo.bits_per_pixel / 8;`

Q6. S'ha d'explicar el càlcul efectuat per a obtenir la posició dins de la memòria de vídeo que correspon a un cert píxel les coordenades en pantalla del qual són `c_x` i `c_y`:

- `ubica = c_x * (vinfo.bits_per_pixel/8) + c_y * finfo.line_length;`

Q7. Quins píxels ocupen posicions consecutives en la memòria de vídeo, els que són en la mateixa fila o en la mateixa columna?

Q8. S'ha d'explicar per què l'assignació de colors a un píxel s'efectua amb les assignacions següents:

- `*(fbp + ubica + vinfo.blue.offset/8) = blau;`
- `*(fbp + ubica + vinfo.green.offset/8) = verd;`
- `*(fbp + ubica + vinfo.red.offset/8) = roig;`

Primer punt de control: a l'inici de la sessió s'han de lliurar les respostes a les preguntes Q1-Q8 i s'han de poder explicar al professor o professora.

3. Treball al laboratori

En aquesta sessió es veu de forma pràctica la programació i gestió a baix nivell de la targeta gràfica a fi d'entendre les possibilitats i limitacions que té. Per a fer-ho, es proposa dissenyar diversos programes que inclouen l'ús d'aquest perifèric.

Per a comprovar el funcionament dels programes proposats en aquesta sessió s'usa l'entorn NETinVM. Els programes s'executen en la màquina KVM anomenada 'exta' aprofitant la seua consola gràfica. En l'annex III es mostra de forma detallada com es configura i s'engega la màquina 'exta' per a poder executar els programes.

3.1. En primer lloc, es proposa executar l'exemple 1 de l'annex II en la màquina 'exta'. Per a executar el programa cal fer les accions següents:

- El codi del programa es copia en una finestra de l'editor de text KWrite en la màquina 'base'.
- El programa es desa a la carpeta `/home/user1/shared` de la màquina 'base', amb l'extensió `.c`.

- A continuació s'inicia una sessió en la màquina 'exta' amb l'usuari administrador o 'root' que obri el terminal gràfic.
- En la màquina 'exta' es copia el programa de la carpeta `shared` al directori arrel:

```
cp shared/exemple1.c ./
```

- El programa es compila en la màquina 'exta' usant el compilador `gcc` amb l'ordre:

```
gcc exemple1.c -o exemple1
```

- Finalment, el programa s'executa en 'exta' amb l'ordre següent:

```
./exemple1
```

Una vegada executat el programa, s'han de contestar de forma raonada les preguntes següents:

3.1.1. Quanta memòria de vídeo té la màquina 'exta'?

3.1.2. Quina és la resolució en files i columnes?

3.1.3. Quants *bytes* s'usen per a donar la informació del color d'un píxel?

3.1.4. Quin significat tenen els bits de desplaçament per a cada component de color RGB?

3.2. L'exemple 2 s'executa a la màquina 'exta'. El procediment per a executar-lo és similar al mostrat en l'exemple 1. Una vegada executat el programa, s'ha de resoldre de forma raonada l'exercici següent:

3.2.1. Quines són les coordenades X i Y que s'han d'usar per a situar el quadrat que parpelleja en les següents posicions de la pantalla?: cantonada superior esquerra, cantonada superior dreta i cantonada inferior esquerra.

3.3. S'ha d'implementar un programa en llenguatge C que moga un quadrat de 20x20 píxels de color roig per la pantalla. Inicialment, el quadrat ha de situar-se al centre de la pantalla i s'ha de mostrar durant cinc mil·lisegons. Una vegada passat aquest temps, s'ha d'apagar sobreescrivint el mateix quadrat, però usant el color negre. A continuació s'ha de modificar la posició del quadrat incrementant en un píxel la seua posició tant en l'eix X com en l'eix Y. D'aquesta manera, fa l'efecte que el quadrat es desplaça per la pantalla.

Quan algun dels píxels del quadrat arribe a situar-se en la vora de la pantalla, s'ha de produir l'efecte rebot. És a dir, si s'ha arribat al final de la pantalla en l'eix X, les pròximes posicions del quadrat en aquest eix es calculen restant un píxel al valor actual. De forma similar, quan s'arribe al final de la pantalla en l'eix Y es comença a restar un píxel a la posició del quadrat en l'eix Y. Per contra, quan el quadrat arribe a l'inici de la pantalla, ja siga en l'eix X o en l'eix Y, es torna a incrementar la posició en aquests eixos en el càlcul de les futures posicions.

El programa ha d'executar-se de manera indefinida mitjançant un bucle infinit que contínuament estiga repetint les mateixes accions. Per a parar el programa es pot usar la combinació de tecles `Ctrl-C`, que en finalitza l'execució. S'ha d'executar el programa i contestar de forma raonada la pregunta següent:

3.3.1. Quins límits cal comprovar per a determinar que el quadrat està situat sobre una vora de la pantalla, ja siga per a arribar al límit màxim o mínim en l'eix X o en l'eix Y?

Segon punt de control: s'ha de mostrar al professorat de pràctiques el funcionament dels programes i la resposta a les qüestions, i també respondre a preguntes sobre el disseny i el funcionament.

3.4. Modifica la funció *acoloreix* del programa de l'exemple 2 perquè puga pintar rectangles. Indica com a paràmetres dues mides de costat diferents, l'una per a l'eix X i l'altra per a l'eix Y. Amb aquesta nova funció implementa un programa que mostre els colors de tots els tons de blau, verd i roig que es poden fer de forma independent en cada component de color. Així, les primeres 256 línies horitzontals han de tenir un degradat dels 256 tons de color roig que es poden usar i els components verd i blau s'han de mantenir a zero. Les 256 línies següents han de mostrar els tons de degradats del component verd i mantenir els components roig i blau a zero. Finalment, l'última franja de 256 línies ha de mostrar el degradat dels tons en blau i mantenir els components roig i verd a zero. S'ha d'executar el programa i contestar les preguntes següents:

3.4.1. Quants píxels han de tenir els costats del rectangle en l'eix X i en l'eix Y emprat per a pintar una fila horitzontal de píxels?

3.4.2. Cal modificar la posició de començament del rectangle en l'eix X quan es passa d'una fila horitzontal a la següent? I en l'eix Y?

3.5. En el cas que hi haja temps en la sessió, es pot proposar algun programa addicional, com ara pintar banderes del món o fer alguna animació senzilla.

Tercer punt de control: s'ha de mostrar al professorat de pràctiques el funcionament dels programes i la resposta a les qüestions, i també respondre a preguntes sobre el disseny i el funcionament.

Abans de finalitzar la sessió s'han d'entregar per escrit les respostes a totes les preguntes proposades en el guió.

Annex I. Accés a la memòria de vídeo en Linux

Les principals característiques de la VGA en mode gràfic són la resolució de 640x480 píxels i l'extensió del color, que usa una paleta de 2^{18} = 262.144 colors. D'aquests colors, pot mostrar-ne 16 de manera simultània, amb la resolució esmentada, o 256 colors de forma simultània si la resolució es redueix a 320x200 píxels. La VGA va equipada amb una memòria de vídeo de 256 kibibytes, com a mínim, i un mebibyte com a màxim.

En el mode VGA amb 256 colors per píxel, el color d'un píxel s'assigna mitjançant un *byte*. La memòria RAM de vídeo està organitzada amb una simple disposició lineal, en la qual un *byte* correspon a un píxel. El valor de l'octet especifica el color del píxel. Aquest mode requereix 320 *bytes* (`0x140` en hexadecimal) per línia, que corresponen a 320 píxels. La memòria gràfica consta de 64 kibibytes (`10000h` *bytes*), però només 64.000 *bytes* (64 Kbytes) s'usen realment. Els altres 1.536 *bytes* resten lliures. El píxel (0,0) és a la cantonada superior esquerra. La coordenada `c_y` indica el número de fila i la coordenada `c_x` el número de columna. Tenint en compte això, l'adreça de començament del píxel amb coordenades `c_x` i `c_y`, quan cada píxel ocupa un *byte* en la memòria, seria:

```
adreça (c_x, c_y) = inici + (0x140*c_y)+ c_x //en hexadecimal
adreça(c_x, c_y) = inici + (320*c_y) + c_x //en decimal
```

L'adreça `inici` seria la posició de començament de la memòria de vídeo al mapa de posicions de l'ordinador. Aquesta posició en l'estàndard VGA estava definida com l'adreça `0xA000`.

Super VGA

En els nous modes introduïts per la SVGA, la RAM de vídeo s'organitza igualment com un vector lineal de *bytes*, i cada quatre bits representen un píxel (16 colors) o cada *byte* un píxel (256 colors). En el mode `105h` (1024*768 píxels amb 256 colors), la posició del *byte* que representa el color del píxel a la fila `c_y`, columna `c_x`(`c_y`$\in$ [0,767], `c_x`$\in$ [0,1.023]) en la memòria de vídeo seria:

```
adreça(c_x, c_y) = inici + (0x400*c_y) + c_x //en hexadecimal
adreça(c_x, c_y) = inici + (1024*c_y) + c_x //en decimal
```

El significat del paràmetre `inici` és el mateix que en l'exemple anterior. Igual que en els modes de la VGA, la forma d'emmagatzemar els píxels és mitjançant una matriu lineal en la qual es necessiten tants bits per píxel com siguen necessaris per a la profunditat del color seleccionat. Originalment (i estrictament d'acord amb l'estàndard SVGA), només es podrien representar al mateix temps 256 colors diferents; així, a cada píxel s'assigna un *byte*. Amb la introducció de modes amb més profunditat de color, com el d'alt color amb 256 K colors i el mode de color vertader, de 16 milions de colors, es requereixen 24 bits (3 *bytes*), per cada píxel. Així i tot, en aquests casos l'emmagatzemament dels píxels s'efectua de forma lineal en la memòria RAM de vídeo. La memòria estaria organitzada com un vector lineal de píxels en el qual cada *byte* assignat a un píxel representa un nivell de color dins dels components roig, verd i blau. La combinació d'aquests *bytes* produeix el color final del píxel. També es pot assignar un *byte* addicional que indica el nivell de transparència, de manera que el nombre de *bytes* que ocuparia el color de cada píxel seria de quatre.

Dispositiu de memòria intermèdia d'imatges de Linux

La memòria intermèdia d'imatges de Linux està construïda com un dispositiu virtual de sistema operatiu al qual s'accedeix com si fora un arxiu. Aquest dispositiu té el nom de ***framebuffer*** dins

del sistema de fitxers de Linux. Els processos poden accedir a aquest dispositiu per a llegir i escriure damunt seu com si es tractara d'un fitxer comú. En modificar el contingut del *framebuffer,* canvia el contingut de la memòria de vídeo de la targeta gràfica i es produeixen variacions en les imatges representades en el monitor. Usant la memòria intermèdia d'imatges, una aplicació pot mostrar informació en la pantalla sense preocupar-se dels detalls d'accés al maquinari que constitueix la targeta gràfica. D'aquesta manera, la memòria intermèdia d'imatges subministra una capa d'abstracció mitjançant una interfície d'accés ben definida. Aquesta implementació evita als programes haver d'accedir a la targeta de vídeo a través del controlador corresponent.

La memòria intermèdia d'imatges es representa en el sistema Linux mitjançant un dispositiu de tipus fitxer que està localitzat en el directori `/dev` del sistema de fitxers. Des del punt de vista de l'usuari, aquest dispositiu té el mateix aspecte que qualsevol altre dispositiu en `/dev`. És un dispositiu de tipus arxiu anomenat com `/dev/fb0`. Pot haver-hi més d'un *fb* a la vegada, per exemple, si es té més d'una targeta gràfica. Els dispositius corresponents s'anomenen de manera seqüencial (`/dev/fb0`, `/dev/fb1`, etc.) i funcionen independentment.

La memòria intermèdia d'imatges, com que es comporta com un fitxer normal, permet la lectura i l'escriptura del contingut. Per exemple, es pot fer una captura de pantalla amb aquesta ordre de Linux:

- `cp /dev/fb0 elmeuarxiu`

També es podria sobreescriure el contingut i canviar la imatge de la pantalla, per exemple, amb un patró de colors aleatori:

- `cat /dev/urandom > /dev/fb0`

Per a accedir al contingut de la memòria de vídeo i configurar els paràmetres de la targeta usant el dispositiu *fb* se solen usar dues crides al sistema que faciliten aquesta tasca: `mmap` i `ioctl`.

mmap

La memòria que apareix a l'arxiu especial *fb* és la memòria de la targeta gràfica i es pot usar la crida al sistema `mmap()` per a mapar-la en l'espai de memòria assignat a un procés. Aquest tipus de recurs, denominat arxiu mapat en memòria, permet que segments de la memòria virtual d'un procés siguen assignats mitjançant una correlació directa *byte* a *byte* amb alguna part d'un fitxer o un altre recurs similar. D'aquesta manera, la crida del sistema `mmap()`fa correspondre el contingut d'un fitxer, o un altre objecte especificat per un descriptor de fitxer, en l'espai de memòria assignat a un procés. Quan es fa correspondre un arxiu dins de l'espai d'adreces del procés, s'hi pot accedir com si fora una matriu del programa. Aquesta és una de les formes més eficients d'accedir a les dades d'un arxiu i proporciona una interfície d'accés a les dades que es pot usar sense haver de recórrer a l'abstracció de la lectura i escriptura d'arxius.

ioctl

La crida al sistema `ioctl()` manipula els paràmetres del dispositiu subjacent. La funció `ioctl()` és una crida de sistema en Linux que permet a una aplicació controlar un controlador de dispositiu o comunicar-s'hi de forma senzilla. Un dels arguments que s'ha de passar a la funció és un descriptor d'arxiu que representa el dispositiu que es vol controlar. Així, moltes característiques de funcionament dels dispositius controlats a través d'arxius especials (per exemple, els terminals de vídeo) es poden modificar amb peticions `ioctl()`.

Annex II. Programes exemple

Programa exemple1

A continuació es mostra un programa que llegeix la configuració de la targeta de vídeo. La informació s'emmagatzema en dues estructures predefinides, els camps de les quals s'actualitzen amb els valors de configuració de la targeta.

```
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>
#include <linux/fb.h>
#include <sys/mman.h>
#include <sys/ioctl.h>

//Variables per a obtenir la informació de la pantalla

struct fb_var_screeninfo vinfo;
struct fb_fix_screeninfo finfo;

int main()
{
    int fbfd = 0;
    long int mida_pantalla = 0;

    // S'obri el dispositiu
    fbfd = open("/dev/fb0", O_RDWR);

    if (fbfd == (-1)) {
        printf("Error: el dispositiu no es pot obrir.\n");
        return(1);
    }

    // Agafem la informació fixa sobre la pantalla gràfica
    if (ioctl(fbfd, FBIOGET_FSCREENINFO, &finfo)) {
        printf("Error en llegir la informació fixa.\n");
        return(2);
    }

    // Agafem la informació variable sobre la pantalla gràfica
    if (ioctl(fbfd, FBIOGET_VSCREENINFO, &vinfo)) {
        printf("Error en llegir la informació variable.\n");
        return(3);
    }

    //Imprimim la grandària de la memòria de vídeo en bytes
    printf("Grandària de la memòria: %d bytes\n", finfo.smem_len);

    //Resolució en píxels de la pantalla en l'eix X (elements en una fila)
    printf("Xres: %d píxels\n", vinfo.xres);

    //Resolució en píxels de la pantalla en l'eix Y (nombre de files)
    printf("Yres: %d píxels\n", vinfo.yres);

    //Bits utilitzats per a acolorir un píxel, amb els valors RGB més nivell de transparència
    printf("Bits per pixel: %d\n", vinfo.bits_per_pixel);

    //Mida de la memòria en bytes necessària per a desar una línia de píxels
    printf("Longitud de línia: %d bytes\n", finfo.line_length);

    //Desplaçament en bits des de la posició de començament d'un píxel
    //en memòria de vídeo per a emmagatzemar cada component RGB
    printf("Desplaçament per al roig: %d bits\n", vinfo.red.offset);
    printf("Desplaçament per al verd: %d bits\n", vinfo.green.offset);
    printf("Desplaçament per al blau: %d bits\n", vinfo.blue.offset);

    //Grandària en bits per als valors dels components de color RGB
    printf("Grandària per al roig: %d bits\n", vinfo.red.length);
    printf("Grandària per al verd: %d bits\n", vinfo.green.length);
    printf("Grandària per al blau: %d bits\n", vinfo.blue.length);


    // Calculem la memòria de vídeo disponible en bytes
    mida_pantalla = vinfo.xres * vinfo.yres * vinfo.bits_per_pixel / 8;
```

```
    printf("Grandària de la memòria de vídeo: %d bytes\n", mida_pantalla);

    close(fbfd);
    return(4);
}
```

Les funcions `open()`i `ioclt()`emprades en l'exemple anterior tenen la funcionalitat següent:

- `open(const char *path, …)`: la funció `open()` estableix la connexió entre un arxiu i un descriptor d'arxiu. El descriptor d'arxiu l'usen altres funcions d'entrada/eixida per a referir-se a aquest arxiu i poder accedir al contingut. L'argument *`path`* apunta a una ruta que identifica l'arxiu.
- `int ioctl( int fd, …)`: la crida al sistema `ioctl()` permet accedir als paràmetres de configuració de la targeta gràfica. Així, moltes característiques de funcionament de la targeta es poden controlar amb peticions `ioctl()`. L'argument *fd* ha de ser un descriptor que identifica l'arxiu associat a un dispositiu de memòria intermèdia d'imatges.

Els camps que contenen les estructures `fb_var_screeninfo` i `fb_fix_screeninfo` per a manipular la configuració del terminal de vídeo són:

- *fb_var_screeninfo* s'usa per a descriure les característiques del terminal gràfic que es poden modificar. Aquests valors es poden llegir, modificar i escriure sobre la targeta usant les crides a la funció `ioctl()`. Alguns dels camps continguts en aquesta estructura són:

```
struct fb_var_screeninfo {
__u32 xres;               /* nombre de píxels per fila */
__u32 yres;               /* nombre de píxels per columna*/
__u32 bits_per_pixel;     /* bits per píxel */

…
struct fb_bitfield red;   /* estructura per a definir el color roig*/
struct fb_bitfield green; /* estructura per a definir el color verd */
struct fb_bitfield blue;  /* estructura per a definir el color blau */
…
}
```

  En què l'estructura *fb_bitfield* té el significat següent:

```
struct fb_bitfield {
__u32 offset;             /*Començament del camp de bits d'aquest color*/
__u32 length;             /* Llargària de camp de bits d'aquest color*/
__u32 msb_right;          /* Ordenació dels bytes en memòria. Si és
                          diferent de zero, l'ordenació és big-endian. Si
                          no, l'ordenació és little-endian*/
};
```

- *fb_fix_screeninfo* s'usa per a descriure les característiques del terminal gràfic que no es poden modificar. Alguns dels camps continguts en aquesta estructura són:

```
struct fb_fix_screeninfo {
…
__u32 smem_len;           /* Longitud del framebuffer en bytes */
__u32 line_length;        /* Longitud d'una línia en bytes */
…
}
```

En l'enllaç següent hi ha més informació sobre aquestes estructures i els altres camps que contenen: <https://www.kernel.org/doc/Documentation/fb/api.txt>

Programa exemple 2

A continuació es mostra un programa que accedeix a la memòria de vídeo de la targeta gràfica per a pintar un quadrat de píxels usant el dispositiu de memòria intermèdia d'imatges de Linux:

```
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>
#include <linux/fb.h>
#include <sys/mman.h>
#include <sys/ioctl.h>

// Nombre de píxels que formen el costat del quadrat

#define Bloc 20

//Variables per a obtenir la informació de la pantalla

struct fb_var_screeninfo vinfo;
struct fb_fix_screeninfo finfo;

// Punter al començament de la memòria de vídeo mapada
// dins de l'espai d'adreçament del programa
char *fbp = 0;

// Funció que acoloreix un quadrat de píxels de la pantalla.
// La posició de començament del quadrat està donada per les coordenades c_x i c_y del primer
// píxel, situat a la cantonada superior esquerra del quadrat.
// El color dels píxels del quadrat es dona amb els valors roig, verd i blau.
// El paràmetre bloc és el nombre de píxels que formen el costat del quadrat.
void acoloreix(int c_x, int c_y, char roig, char verd, char blau, char bloc) {

    int i, j;
    long int ubica = 0;
    for ( i = 0; i < bloc; i++ )
            for ( j = 0; j < bloc; j++ ) {

                // La posició de començament d'un píxel en la memòria de vídeo és:
                // c_x * (bytes per píxel) +
                // c_y * (bytes en una línia de píxels)
                // La posició s'incrementa amb i j per a
                // recórrer tots els píxels del quadrat.
                ubica = (c_x + i) * (vinfo.bits_per_pixel/8) +
                          (c_y + j) * finfo.line_length;

                // S'emmagatzemen els tres components de color RGB per a aquest píxel
                // en les posicions de memòria corresponents a cada color
                *(fbp + ubica + vinfo.red.offset/8) = roig;
                *(fbp + ubica + vinfo.green.offset/8) = verd;
                *(fbp + ubica + vinfo.blue.offset/8) = blau;
      }
}

int main()
{
    int fbfd = 0;
    long int mida_pantalla = 0;
    int x, y, k;

    // S'obri el dispositiu framebuffer 0
    fbfd = open("/dev/fb0", O_RDWR);

    if (fbfd == (-1)) {
        printf("Error: no es pot obrir el dispositiu.\n");
        return(1);
    }

    // Agafem la informació fixa sobre la pantalla gràfica
    if (ioctl(fbfd, FBIOGET_FSCREENINFO, &finfo)) {
        printf("Error en llegir la informació fixa.\n");
        return(2);
    }

    // Agafem la informació variable sobre la pantalla gràfica
    if (ioctl(fbfd, FBIOGET_VSCREENINFO, &vinfo)) {
        printf("Error en llegir la informació variable.\n");
        return(3);
```

```
    }

    // Calculem la memòria de vídeo disponible en bytes
    mida_pantalla = vinfo.xres * vinfo.yres * vinfo.bits_per_pixel / 8;

    // Inicialitzem el punter a la memòria de vídeo
    fbp = (char *)mmap(0,mida_pantalla,PROT_READ|PROT_WRITE,MAP_SHARED,fbfd,0);

    if (fbp == (char*)(-1)) {
        printf("Error en situar el punter a la memòria de vídeo.\n");
        return(4);
    }

    // Posició dels píxels de començament del quadrat que conté
    // els píxels per modificar. Se situa el quadrat a la cantonada inferior dreta
    // de la pantalla
    x = vinfo.xres-Bloc;
    y = vinfo.yres-Bloc;

    // Bucles per a canviar el color del quadrat de mida igual a Bloc x Bloc
    // píxels situats a la cantonada inferior dreta de la pantalla.
    // Els píxels parpellegen amb un període de 0,5 segons.
    for ( k = 0; k < 10; k++ )  {
            acoloreix( x, y, 255, 0, 0, Bloc);
            usleep(500000);
            acoloreix( x, y, 0, 0, 0, Bloc);
            usleep(500000);
    }

    munmap(fbp, mida_pantalla);
    close(fbfd);
    return(5);

}
```

Algunes funcions de llibreria emprades en l'exemple anterior són les següents:

- `void * mmap(void *start, size_t length, int prot , int flags, int fd, off_t offset)`: la funció `mmap()` fa correspondre el contingut d'un fitxer especificat pel descriptor de fitxer *fd* a l'espai de memòria assignat a un procés.

  L'adreça `start` és una adreça inicial predefinida per a la realització del mapatge. Si no hi ha un altre mapatge en aquesta adreça, el nucli tria un límit de pàgina pròxim i mapa el dispositiu en aquesta zona; en cas contrari, el nucli tria una nova adreça. Si aquest argument és 0, el nucli pot col·locar l'assignació en qualsevol altre lloc.

  La quantitat de memòria situada es defineix amb el paràmetre `length` i comença amb un desplaçament donat per `offset` des del principi de l'arxiu.

  El punter a la posició de començament de la memòria on s'ha situat el dispositiu és el valor retornat per `mmap()`.

  També es poden especificar arguments per a indicar el grau de protecció de la memòria i si la memòria es comparteix amb altres processos. Per a més informació, es pot consultar l'enllaç següent: <https://linuxhint.com/using_mmap_function_linux/>
- `int munmap(void *start, size_t length)`: la crida al sistema `munmap()` elimina la correspondència en memòria efectuada per la funció anterior.
- `int usleep(useconds_t usec)`: suspèn l'execució del programa durant un període de temps indicat en microsegons.

Annex III. Configuració, engegada i execució dels programes en la consola gràfica de la màquina 'exta'

Configuració i engegada d''exta'

El primer pas per a configurar la màquina 'exta' és engegar l'entorn de NETinVM usant el programari de virtualització VMware.

A continuació, a fi de començar amb la màquina en un estat correcte, cal fer una restauració des d'un punt de restauració. Per a fer aquesta restauració cal assegurar que totes les màquines KVM estan apagades. Es poden apagar totes les màquines (menys 'base', és clar) prement el botó d'apagar-les totes (*Shutdown all*) a la carpeta de les màquines KVM.

Una vegada s'ha accedit a l'escriptori de la màquina 'base', cal usar un navegador web per a accedir a la pàgina de l'assignatura a l'Aula Virtual. Per exemple, es pot usar el Firefox, la icona del qual és a la cantonada superior esquerra de l'escriptori. Una vegada s'accedeix a la pàgina de l'assignatura, cal descarregar el fitxer de punt de restauració de les màquines KVM '`kvm_machines_backup_EC.tgz`', a la carpeta de descàrregues de la màquina 'base'.

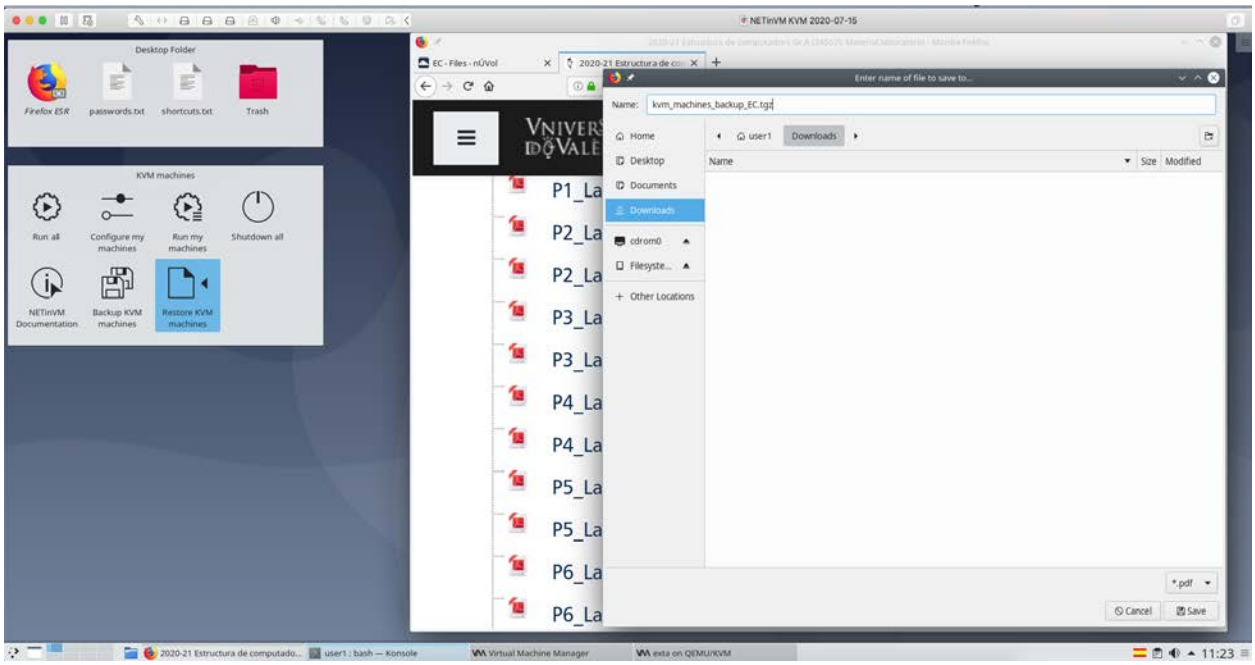


Figura 1. Descàrrega del fitxer kvm_machines_backup_EC.tgz *de l'Aula Virtual.*

Una vegada descarregat el fitxer, cal restaurar les màquines KVM mitjançant l'opció *Restore KVM machines* disponible al menú de les màquines KVM de l'escriptori. Quan fem aquesta acció s'obri una finestra per a seleccionar el fitxer de restauració que es vol usar. En aquest cas, el fitxer és `kvm_machines_backup_EC.tgz` que s'ha descarregat a la carpeta de descàrregues.

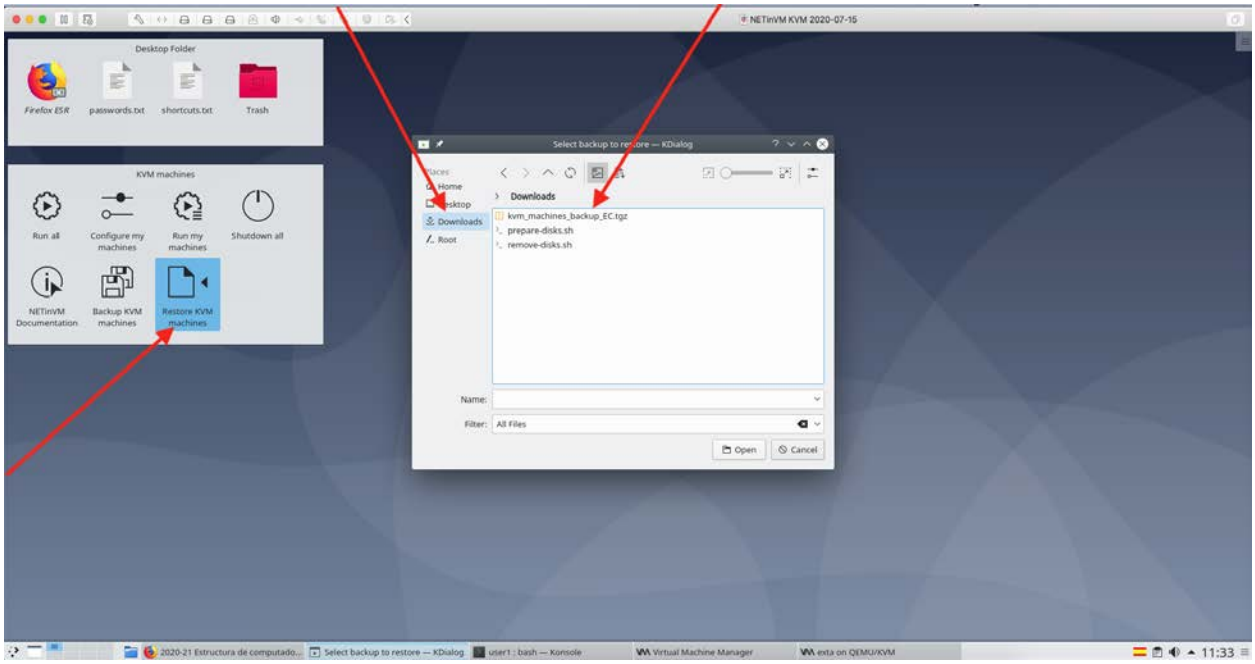


Figura 2. Menú per a restaurar les màquines KVM amb el fitxer kvm_machines_backup_EC.tgz.

Per a engegar la màquina 'exta' dins de l'entorn NETinVM, s'ha d'obrir el gestor de màquines virtuals (opció del menú *Manage virtual machines*) des de la icona d'inici d'aplicacions situat a la cantonada inferior esquerra de l'escriptori, tal com es mostra a la figura 3.

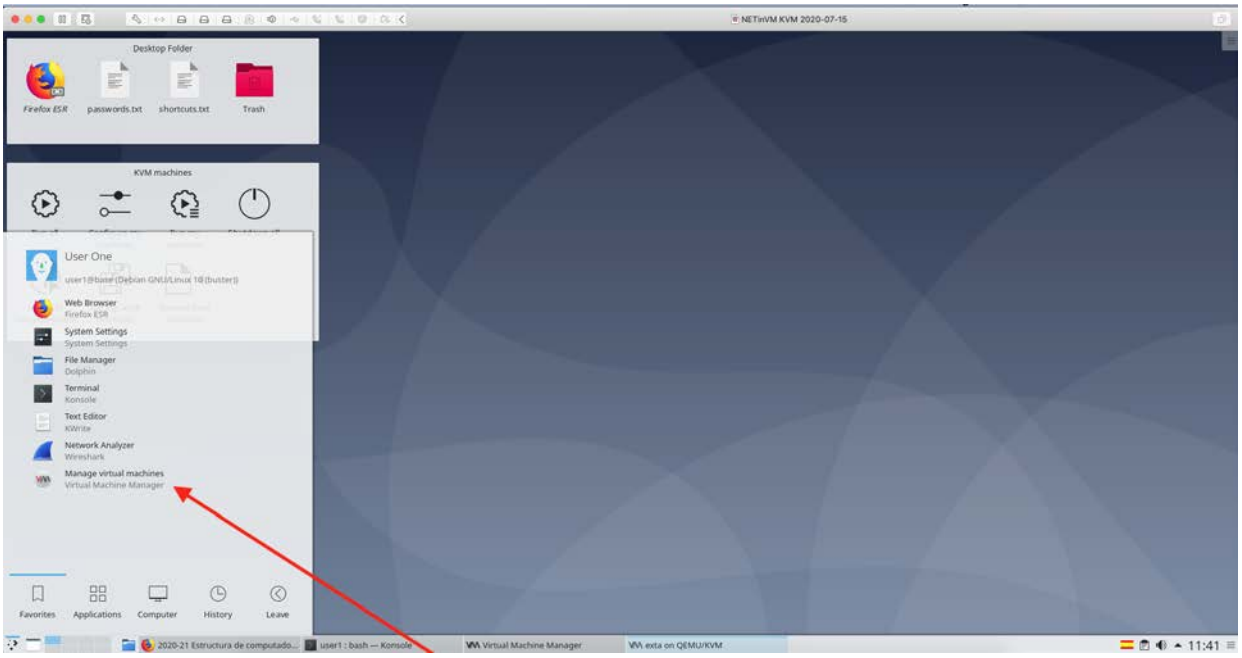


Figura 3. Entorn gràfic de NETinVM amb la situació de l'administrador de màquines virtuals.

Un vegada inicialitzat el gestor de màquina virtuals, s'ha de seleccionar la màquina 'exta' (vegeu la figura 4) i s'ha d'obrir el menú d'aquesta màquina amb l'opció *open*.

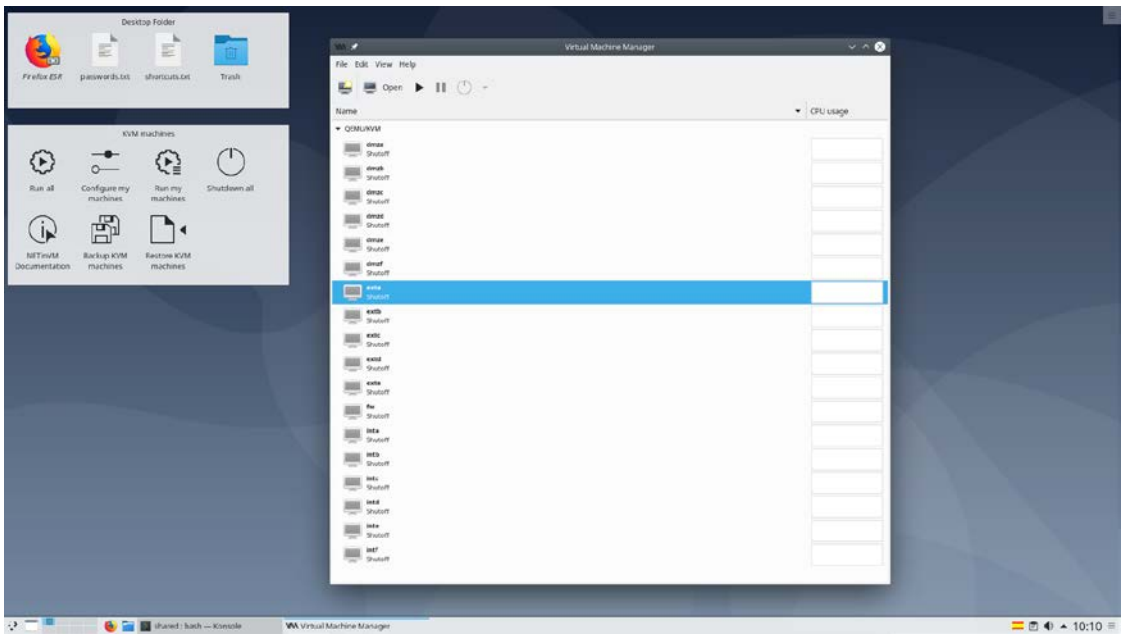


Figura 4. Gestor de màquines virtuals KVM.

Per a configurar de forma adequada la targeta de vídeo de la màquina 'exta', se selecciona la informació del maquinari (icona amb una vinyeta quadrada i la lletra *i* enmig) i s'accedeix a un menú que selecciona el tipus de targeta de vídeo. El tipus de targeta que s'ha de seleccionar és VGA i els canvis s'apliquen amb el botó de menú *Apply*, tal com es mostra a la figura 5.

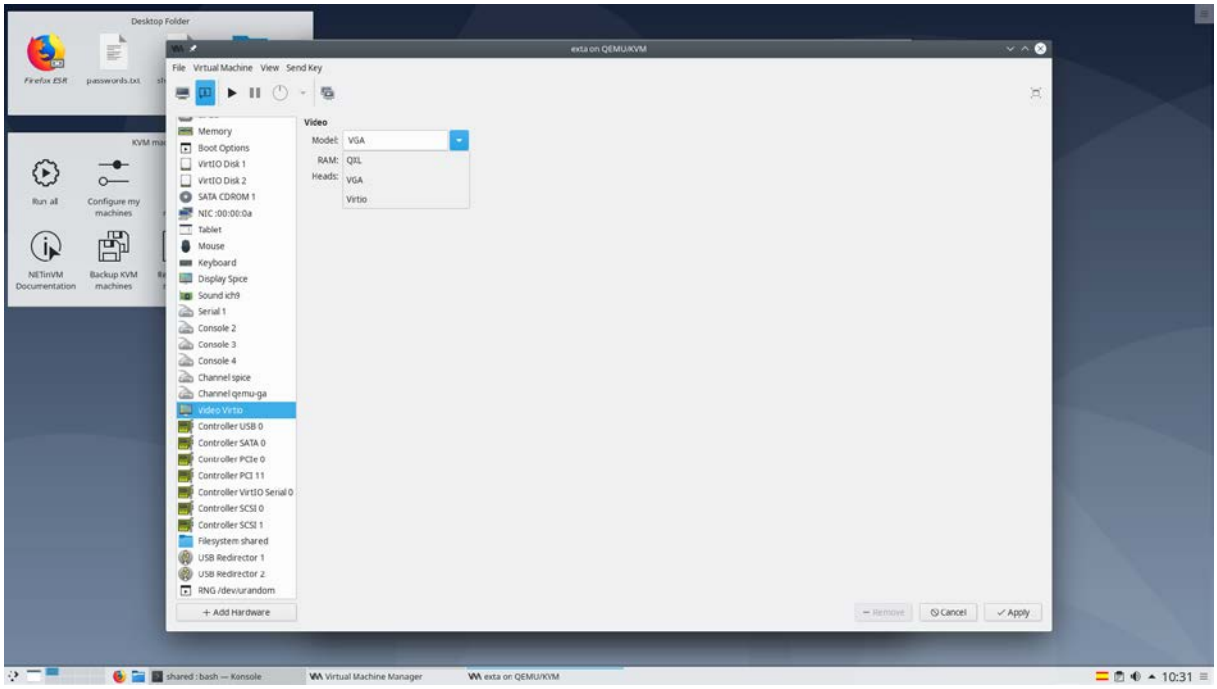


Figura 5. Configuració de la màquina virtual amb targeta VGA.

A continuació, engeguem la màquina prement la icona *Power on* i, uns pocs instants més tard, apareix a la consola gràfica de la màquina l'opció d'introduir l'usuari d'accés, tal com es mostra a la figura 6.

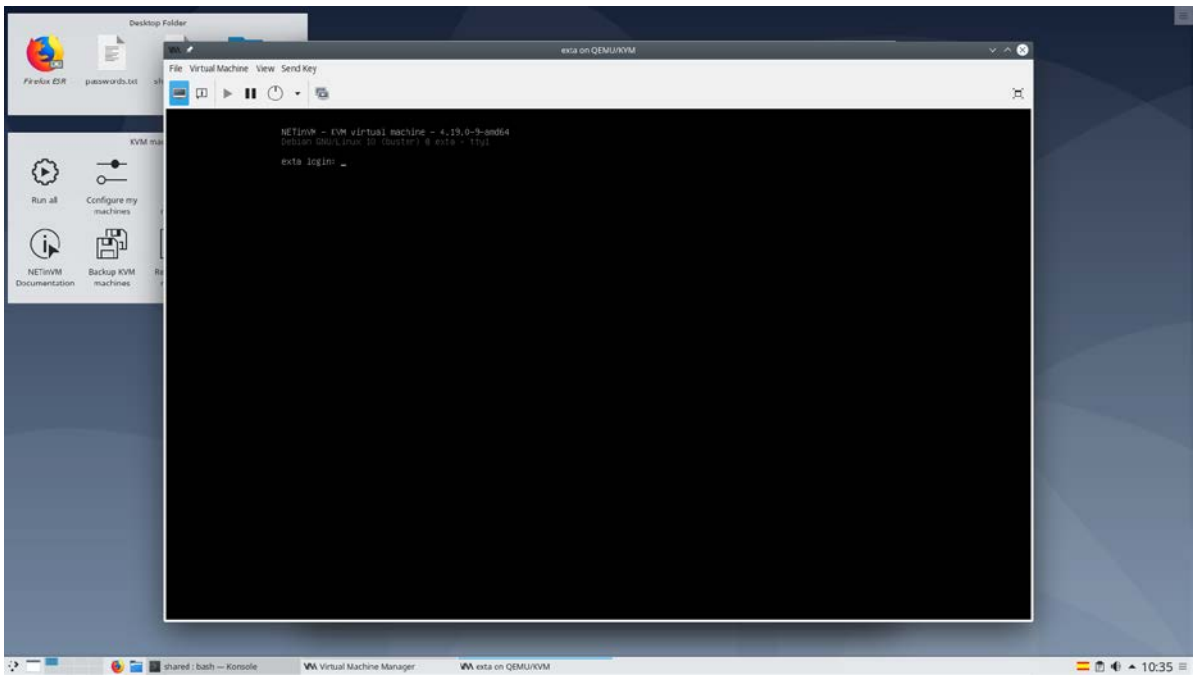


Figura 6. Consola per a l'accés a la màquina.

S'ha d'accedir a la màquina usant l'usuari administrador. La contrasenya s'emmagatzema en el fitxer `passwords.txt`, situat a la carpeta *Escriptori*.

Execució dels programes en 'exta'

La màquina 'exta' no disposa d'entorn gràfic i només s'hi pot accedir a través de la consola. Per a poder editar els programes d'una manera senzilla, es pot utilitzar l'editor de fitxers KWrite dins de l'entorn NETinVM en la màquina 'base'. Aquest editor es pot engegar des de la icona d'engegada d'aplicacions de la màquina 'base', situada a la cantonada inferior esquerra de l'escriptori. Una vegada editats els programes en KWrite, cal desar-los a la carpeta `/home/user1/shared`. Aquesta carpeta és compartida per totes les màquines de l'entorn NETinVM i es pot accedir a aquests programes dins de la mateixa carpeta en la màquina 'exta'.

Per a executar els programes en 'exta', primer cal copiar-los de la carpeta compartida a una altra carpeta de l'usuari administrador. Aquesta acció s'ha de fer perquè, per qüestions de seguretat, no es poden executar fitxers en `shared`. És per aquest motiu que s'executen tres ordres, sempre en la màquina 'exta'. Per exemple, per a copiar el fitxer `exemple1.c` en el directori arrel de l'usuari administrador es pot usar l'ordre següent:

```
1.      cp shared/exemple1.c  ./
```

El segon pas és compilar el programa amb el compilador gcc:

```
2.      gcc exemple1.c -o exemple1
```

El tercer i últim pas és executar el programa a partir del fitxer executable:

```
3.      ./exemple1
```

Si cal modificar el programa, els canvis es poden fer en l'editor KWrite de la màquina 'base' i després el fitxer s'ha de desar. A continuació s'han de fer de nou els tres passos explicats (còpia, compilació i execució) a la màquina 'exta'.