SISTEMES OPERATIUS

ENGINYERIA TÈCNICA INFORMÀTICA DE SISTEMES

ENGINYERIA TÈCNICA INFORMÀTICA DE GESTIÓ

ESCOLA TÈCNICA SUPERIOR D'ENGINYERIA

Enunciat de la Pràctica

Curs 2006-2007

Nucli d'un Sistema Operatiu

M. Àngels Moncusí

Albert Solé

Carlos Carrillo

Índex

1. Introducció

Estructura d'un Sistema Operatiu

Donada la complexitat d'un sistema operatiu, resulta convenient i desitjable, separar d'una manera clara i precisa la seva implementació en capes ben estructurades. De manera que cada capa implementi un subconjunt de funcions d'acord amb la seves característiques temporals i el seu nivell d'abstracció. Un nivell superior es recolza en el seu nivell immediatament inferior per tal d'implementar noves funcions i amagar els detalls d'implementació de les del nivell inferior. De la mateixa manera cada nivell ofereix els seus serveis al seu nivell superior. Això permetrà que les decisions o canvis que es realitzin en un nivell no afectin als altres nivells.

Per tal de permetre aquesta estructura en capes d'un sistema operatiu, cada capa ha de tenir definida la seva interfície amb les altres capes.

En el cas concret de la pràctica, únicament hi haurà dos nivells, un pel sistema operatiu i un pels processos d'usuari. El nivell inferior o nucli treballarà directament amb el hardware de la màquina i oferirà al nivell superior una sèrie de funcions. El segon nivell usarà les operacions que proporciona el nivell nucli i oferirà als processos d'usuari una sèrie de crides al sistema. D'aquesta manera, quedarà completament separat l'espai de sistema de l'espai d'usuari.

Entre els nivells es creen interfícies d'accés. Aquestes interfícies permeten no només estructurar la implementació del Sistema Operatiu, sinó que prohibeixen la compartició de dades i funcions entre els nivells. Un nivell només pot fer crides a les rutines del nivell immediatament inferior que estan establertes en la interfície del nivell inferior. Això implica que una crida al sistema no podrà invocar a una altra crida al sistema del mateix nivell. Entre el nivell d'usuari i el nivell de sistema s'implementarà un mecanisme que impedirà que els processos d'usuari puguin accedir a variables i estructures que pertanyen intrínsecament al nivell de sistema. Aquests mecanismes, com es veurà més endavant, seran les crides a funcions mitjançant **traps**.

Nucli d'un sistema operatiu

El nucli d'un sistema operatiu és la capa més interna. El nucli treballa directament amb el hardware de la màquina i proporciona totes les funcions de control de processos, sincronització, tractament de la memòria i tractament dels dispositius d'entrada i sortida a baix nivell.

Per tant, ha d'oferir una interfície ben definida per a la utilització fàcil i eficient dels recursos hardware de la màquina. Així doncs, el nucli proporcionarà una sèrie de funcions que podran ser usades pel nivell immediatament superior, sense que aquest nivell s'hagi de preocupar de com estan implementades aquestes funcions.

L'objectiu de la pràctica és realitzar una part del nucli d'un sistema operatiu, en concret la part de gestió de processos, una part de sincronisme, i l'accés a dos dispositius físics. No se implementarà la part de gestió de memòria ni el sistema de fitxers.

2. Processos

El procés serà l'estructura lògica que usarà el nucli per tal de controlar l'execució de les aplicacions. Aquests processos s'executaran de forma concurrent, per tant, competiran per tal d'aconseguir els diferents recursos del sistema, ja sigui la CPU, la memòria, el teclat o qualsevol altre recurs.

El nucli identifica un procés pel seu PCB (*Proces Control Block*), que és una estructura de dades interna al nucli que conté tota la informació necessària per l'execució correcta del procés. Donat que

no s'implementarà la part de gestió de memòria, no es podrà usar memòria dinàmica. Per tant el nucli tindrà un número finit i limitat de PCBs. Això implica que el número de processos que poden estar en un moment determinat actius en el sistema està limitat, i és igual, al nombre de PCBs existents.

La informació que el nucli necessita conèixer d'un procés pot ser la següent:

- PID (identificador del procés) únic durant tota la vida del sistema.
- Estat del procés
- Zona de memòria (codi, dades, pila)
- Context del procés
- Recursos assignats al procés
- Informació útil pel planificador
- Informació estadística

Per tal de permetre l'execució concurrent dels diversos processos, el nucli ha de mantenir diverses estructures on s'associaran els PCBs dels processos depenent de l'estat en que estiguin. Aquestes estructures internes del nucli són:

- Cua de PCBs Disponibles: cua on estaran els PCBs que no estan actius (no els té assignat cap procés).
- Cua de RUN: cua on estarà encuat el procés que té la CPU assignada en aquell moment. En un sistema monoprocessador, sempre hi haurà, com a màxim, un procés en aquesta cua.
- Cua de RDY: cua on estaran els processos que estan esperant a que se'ls assigni el recurs CPU, això voldrà dir que els processos que estan en aquesta cua estan preparats per a ser executats.
- Cues de BLK: processos que estan esperant un esdeveniment (event) determinat. Com que els processos bloquejats poden estar esperant diferents serveis, en el nucli hi pot haver diverses cues d'aquest tipus.

Quan un procés es crea, se li assigna un PCB de la cua de PCBs disponibles i se li dóna un context (espai de codi, espai de pila, espai de dades, ...). A continuació s'encua a la cua de RDY esperant que li arribi el torn d'utilitzar el recurs CPU. Durant l'execució d'un procés, per motius de protecció, sols podrà accedir a les posicions de memòria que se li assignen. Per tant, aquest no podrà accedir a cap estructura interna del nucli. Si necessités consultar-ne alguna ho haurà de fer a través de les crides al sistema que el nucli li proporciona.

Les crides al sistema podran provocar canvis en l'estat propi del procés i en l'estat d'altres processos.

Un procés pot finalitzar per que es destrueix, o el destrueixen. Això provocarà l'eliminació del seu entorn, que implica l'alliberació del seu PCB, de la seva zona de pila i de tots els recursos que tingui assignats.

Donat que en el nucli hi haurà diversos processos que volen executar-se, s'ha de decidir en quin ordre s'executaran els processos. La rutina que ho decideix és el planificador de curt termini. El planificador de curt termini que volem implementar és el **Round Robin sense prioritats i quantum variable.**

Hi haurà un procés creat al inici de la pràctica. Aquest procés l'anomenarem procés **NUL** amb quantum 0 que sols ha d'aconseguir la CPU quan no hi hagi cap més procés a la cua de RDY. De la mateixa manera el procés nul serà desbancat de la CPU quan entri un altre procés a la cua de RDY. El procés nul no farà cap càlcul, però s'ha de mostrar per pantalla que s'està executant.

En el nostre cas, un procés deixarà la CPU i per tant provocar un canvi de context, pels següents motius:

- en sol·licitar un servei que no pot ser atès de manera immediata (passa a una cua de BLK),
- en adormir-se,
- en finalitzar el procés.
- en finalitzar el quantum del procés

Les interrupcions de rellotge permeten disposar d'una base de temps. La rutina de tractament d'aquesta interrupció s'anomenarà *multiplexar*. La seva funció principal serà tractar tots els fenòmens temporals que s'utilitzaran, com ara despertar un procés quan hagi transcorregut el temps que havia sol·licitat d'adormir-se.

Estructures de dades pel control de processos

Per tal de poder implementar el nucli, necessitarem una sèrie d'estructures de dades.

Pels processos

```
typedef struct TipusPCB
{struct item Fil;
 uint IdProces;
 uint Estat;
 long SS_SP;
 uint Prioritat;
 ...
}
```

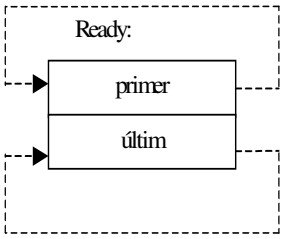
Per a les cues (Veure: Annex 2)

```
typedef struct item
{   struct item *seguent;
    struct item *anterior;
    int ordre;
}
```

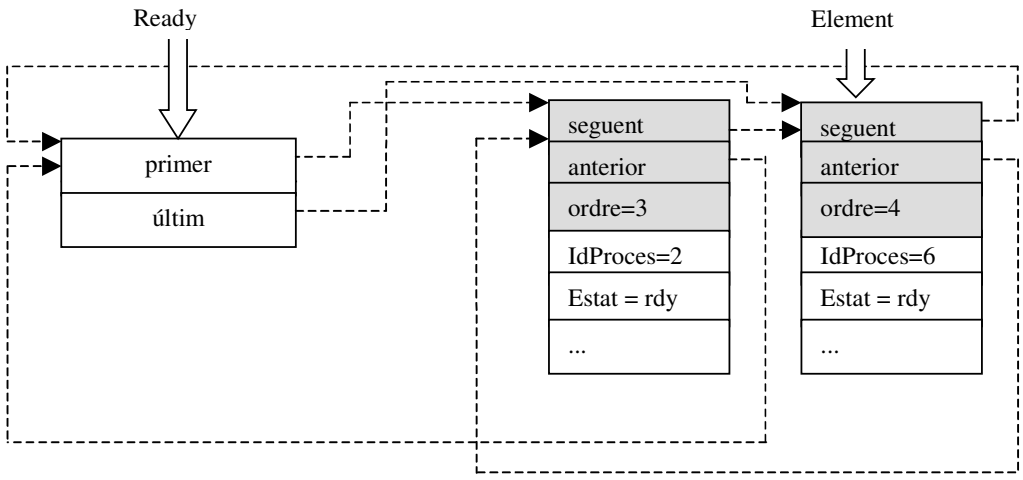
```
typedef struct cua
{   struct item *primer;
    struct item *ultim;
}
```

A continuació veurem un exemple de com s'usen les cues per encuar/inserir PCBs, encara que es poden usar per encuar qualsevol estructura de dades on el primer camp és un ítem.

InicialitzarCua (ready)



Inserir (ready, element, 4)



Observació:

Per a un PCB d'un procés que està a la cua de RDY, els camps *ordre* i *prioritat* tenen el mateix valor. És a dir:

```
pPCB->Fil.ordre == pPCB->Prioritat
```

Crides al sistema per a la manipulació de processos

Les crides al sistema manipulen les estructures de dades internes del nucli, per tal de realitzar l'acció o accions necessàries que s'indiquen. Al formar part de la interfície del programador, la sintaxi i significat de les crides no es pot modificar. És aquesta restricció la que dóna independència de la implementació que s'usi.

```
int CrearProces (void (*codi)(), uint Quantum)
```

Crea un nou procés i el posa a la cua de RDY. Els paràmetres indiquen l'adreça de la primera instrucció a executar del procés i el quantum del mateix.

Retorna:

L'identificador del procés (IdProces) que s'ha creat si tot ha funcionat de forma correcta.

-1 si no s'ha pogut crear per falta de PCBs lliures.

-5 si s'ha produït qualsevol altre error.

```
int DestruirProces(uint IdProces)
```

Destrueix un procés, eliminant el seu context i alliberant tots els recursos que té assignats.

Si el procés està bloquejat el treu de la cua corresponent realitzant les accions compensatòries que es necessitin. Si el procés no existeix retorna error.

Retorna:

0 si tot ha funcionat de forma correcta.

-1 si el procés no existeix.

-2 si s'intenta destruir el procés nul.

```
uint QuiSoc(void)
```

Retorna l'identificador del procés (IdProces) que el crida.

```
void DormirProces (long NTics)
```

Atura el procés que l'executa durant el número de interrupcions de rellotge (NTics) que sol·licita. S'ha de tenir en compte que l'execució d'aquesta rutina comportarà un canvi de context immediat.

Rutines de suport internes al nucli

Les rutines mínimes necessàries pel tractament de processos són:

```
void multiplexar(void)
```

Rutina que tracta la interrupció de rellotge. HA de fer el tractament del quantum i ha de decidir quan s'ha de despertar a un procés aturat mitjançant la crida al *DormirProces*.

```
struct TipusPCB * scheduler(void)
```

Rutina que escull el procés de les cues de *ready* que passarà a executar-se.

```
void dispatcher(struct TipusPCB *proces)
```

Rutina que posa un procés en estat de *run*.

3. Entrada i sortida

En el nostre nucli es vol usar dos dispositius d'entrada i sortida. Aquests dos dispositius seran compartibles. Un és el teclat i l'altre la pantalla. Tots els processos, en el moment de la seva creació tindran assignats i oberts aquests dos dispositius compartibles com a entrada i sortida estàndard.

Per el dispositiu teclat es demana que s'implementi "*buffering*", es a dir, quan l'usuari premi tecles i no hi hagi cap procés llegint de teclat, aquestes es guarden en un buffer del sistema operatiu assignat al dispositiu. El proper procés que faci una lectura de teclat se li passaran les tecles emmagatzemades en aquest buffer, sense necessitat de quedar-se bloquejat en el cas que hi hagi totes les tecles que demana.

Crides al sistema per la entrada i sortida

```
int LlegirTeclat(char *string, uint longitud)
```

- Rutina que llegeix del teclat el nombre de tecles especificats per *longitud*, guardant els caràcters llegits on apunta l'adreça del vector *string*.

  S'estableix que el caràcter de control "carriage return" també finalitzarà la lectura de caràcters encara que no s'hagi llegit el nombre de caràcters sol·licitats per *longitud*. En qualsevol cas, la crida retorna el nombre de caràcters llegits i emmagatzemats en el vector *string*.

Retorna:

El nombre de caràcters llegits realment si tot ha funcionat de forma correcta.
-1 si s'ha produït algun error.

```
int EscriurePantalla(uint x, uint y, char *string, uint longitud, uint atribut)
```

- Rutina que escriu per pantalla des de la posició x,y la infornació que es troba a partir de l'adreça *string* el nombre de caràcters especificats per *longitud* amb els atributs especificats a *atribut*.

Retorna:

0 si tot ha funcionat de forma correcta.
-1 si s'ha produït algun error.

4. Semàfors

Utilitzarem semàfors n-àris per tal de realitzar el sincronisme entre diferents processos.

Estructures de dades pel control del sincronisme

Per tal de poder implementar les rutines de sincronisme, necessitarem disposar de la estructura TipusSemafor definida de la següent manera:

```
typedef struct TipusSemafor
{   struct cua blk;
    uint IdSemafor;
    uint valor;
}
```

Aquesta estructura disposa d'una cua FIFO, ja que els semàfors han de garantir la condició d'espera limitada tal com es va veure a l'assignatura d'ISO.

Al igual que els PCBs, en la inicialització del nucli, s'haurà de reservar memòria per tal de contenir el màxim nombre de semàfors que es vulguin utilitzar.

Crides al sistema pel sincronisme de processos

```
int IniSem(uint IdSemafor, uint valor)
```

Inicialitza un semàfor amb l'identificador del semàfor i el valor que se li passa com a paràmetre. Els identificadors del semàfor sol poden ser inicialitzats una única vegada. Es dóna l'identificador per a facilitar la compartició entre processos d'usuari.
Retorna:

0 si tot ha funcionat de forma correcta.
-1 si no s'ha pogut crear per falta de semàfors lliures.
-2 si el semàfor ja estava inicialitzat.

```
int WaitS(uint IdSemafor)
```

Si el comptador del semàfor val 0, el procés es queda bloquejat en una cua associada al semàfor produint un canvi de context immediat, en cas contrari decrementa en 1 el valor del comptador associat al semàfor *IdSemafor* i retorna el control al procés.
Retorna:

0 si tot ha funcionat de forma correcta.
-1 si el semàfor sol·licitat no existeix o no està inicialitzat.

```
int SignalS(uint IdSemafor)
```

Si la cua associada a aquest semàfor està buida, incrementa en 1 el valor del comptador associat al semàfor que se li passa com a paràmetre, altrament desbloqueja al primer procés que hi està encuat.
Retorna:

0 si tot ha funcionat de forma correcta.
-1 si el semàfor sol·licitat no existeix.

```
int ElimSem (uint IdSemafor)
```

Allibera el semàfor l'identificador del qual passem per paràmetre. No es pot eliminar un semàfor que tingui processos bloquejats a la seva cua.
Retorna:

0 si tot ha funcionat de forma correcta.
-1 si el semàfor sol·licitat no existeix .
-2 si el semàfor sol·licitat té processos bloquejats en la cua associada al semàfor.

5. Desenvolupament de la pràctica.

En aquesta pràctica, tal com s'ha dit anteriorment, s'ha d'implementar el nivell més intern d'un sistema operatiu. A més hi haurà el nivell d'usuari, on hi haurà els processos de prova, els quals accediran al sistema mitjançant crides al sistema, implementades a través d'un *trap*. (Tingueu en compte que no es permès invocar a una crida al sistema dins d'una altra crida al sistema). Aquests processos realitzaran un joc de proves adient per tal de comprovar que tot funciona correctament.

Es recomana desenvolupar el nucli en diverses fases, de manera que no es començarà una nova fase si no s'està segur que l'anterior funciona correctament, doncs s'hi fonamentaran. Això permet anar desenvolupant el joc de proves i la documentació a mesura que es va evolucionant en la implementació.

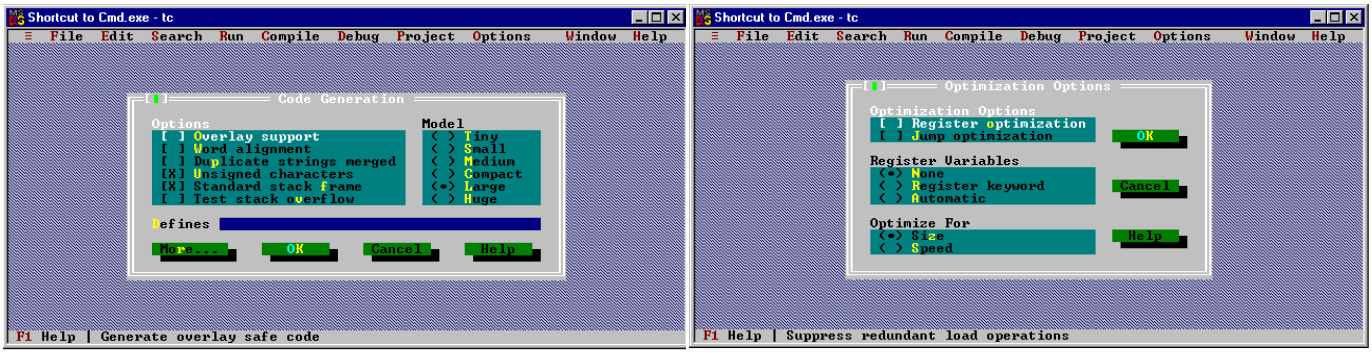
S'ha d'anar en compte de separar clarament el que correspon al sistema, i el que correspon als processos que es realitzen per tal de provar la implementació. Per exemple, un procés d'usuari mai accedirà a variables del nucli.

El desenvolupament de la pràctica s'anirà realitzant en les diferents sessions de laboratori.

Llenguatge d'implementació

El nucli s'ha d'implementar en C usant el model de memòria *large*, i en assamblador quan calgui. No es pot usar codi assamblador entremig del codi de C, ni cap de les funcions de les llibreries de C. El motiu és que les llibreries no estan pensades per a processos concurrents, de manera que depenent on es faci el canvi de context pot ser que la pila quedi descontrolada i no podrem retornar al procés de manera correcta. A més a més hi ha funcions de la llibreria de C que criden al SO host on desenvoluparem la pràctica i aquest SO s'ha desplaçat del sistema pel nostre nucli.

Per tal de compilar la pràctica, es crearà un fitxer *project* de Turbo C on hi posarem les següents opcions de compilació i d'optimització.



Per tal de realitzar la depuració del codi es recomana l'ús del **Turbo Debugger**, que permet seguir la evolució de la pila dels processos d'una manera senzilla i clara.

Laboratori 1: Disseny d'estructures i implementació de CrearProcés

En aquest laboratori s'haurà de:

- Decidir les estructures de dades internes del nucli: Cues, PCBs, Piles.

El més senzill i eficient és tenir definit un vector de PCBs, però no accedir-hi usant l'índex del vector, sinó usant les rutines de cues que us facilitem. La declaració d'aquest vector serveix únicament per a reservar memòria i no tenir que usar mecanismes de memòria dinàmica, i el fet de no usar l'índex del vector per tal d'accedir-hi, permetrà poder modificar la implementació del vector fàcilment.

En tot cas, l'Identificador del procés no ha de tenir res a veure amb la posició que ocupa el PCB del procés dins del vector, ja que les posicions del vector seran reutilitzables, mentre que els Identificadors no s'haurien de reutilitzar durant tota l'execució de la pràctica.

Si el vostre codi ha de escriure alguna cosa per pantalla mitjançant les rutines que s'especifiquen en el mòdul "ec.h" inicialment s'ha de cridar a la rutina BufferPantalla. Altrament no es veuria res.

- Crear els processos a executar.

Donat que no hi ha gestió de memòria implementada, el codi dels processos es carreguen juntament amb el nucli. Hauran d'estar definits en un mòdul apart i no podran usar cap estructura ni variable del nucli, sols tindran accés a les crides al sistema especificades.

En el moment de la creació d'un procés se li ha de assignar l'adreça de memòria on es troba el codi que ha d'executar el procés en qüestió (en C la adreça del codi d'una rutina és equivalent al nom de la rutina) i una zona de memòria que el procés usarà com a pila.

Per tal de simplificar al màxim les estructures del procés, en lloc de tenir una zona de dades globals, es pot usar únicament variables locals al procediment definit, que el propi C ens deixarà en la pila del procés. En el moment d'iniciar el procés s'ha de tenir en compte aquesta zona i deixar el lloc adient a la pila.

En el moment de la creació del procés se li haurà de donar un entorn d'execució i encuar el PCB del procés a la cua de RDY. La creació de l'entorn consistirà en donar una pila i posar-li uns valors inicials (context del procés), per tal que en el moment de treure el procés de *ready* i iniciar-lo, es pugui fer restaurant el context del procés que hagi escollit el planificador. Els valors imprescindibles són: La paraula d'estat (s'ha d'anar en compte en deixar les interrupcions habilitades), els registres CS i IP, els registres SS i SP, el registre DS i l'enllaç dinàmic (BP) si l'useu.

La rutina *CrearProces* que encuarà el nou procés a la cua de *ready*.

Inicialment es pot fer que els processos no acabin, es a dir, executin un bucle sense fi.

Laboratori 2: Implementació de l'algorisme de planificació.

- Implementar les rutines Salvar i Restaurar

  Salvar: Aquesta rutina és la primera que es cridarà dins d'una rutina de servei d'interrupció (RSI) escrita en "C", i s'ha d'encarregar de guardar a la pila el valor de tots els registres del processador, per tal de poder recuperar després els valors originals, donat que el codi en "C" de la RSI els pot haver modificat. També inicialitzarà el registre DS per a que apunti al segment de dades correcte.

Restaurar: Aquesta rutina és la complementària de l'anterior, i és l' última que es cridarà dintre de qualsevol RSI escrita en "C". La seva funció és treure de la pila els registres guardats per la rutina Salvar, eliminant les possibles variables locals i retornant de la RSI amb IRET.

- Donar el control als processos.

Programar la rutina d'interrupció del rellotge (*multiplexar*), per tal de portar la base temporal que necessita l'algorisme de planificació que volem implementar. Aquesta rutina, quan ho consideri oportú cridarà a la rutina *scheduler* (planificador) per tal de decidir quin és el següent procés que s'ha d'executar, i quan ho hagi decidit, ho comunicarà al *dispatcher* per a que realitzi el canvi de context. Inicialment, s'ha de comprovar que la rutina *multiplexar* és capaç de salvar i restaurar el context de cada procés de forma correcte. (Això es pot fer si únicament hi ha un procés carregat, i per tant no es realitza cap canvi de context)

Des del procés main s'ha de restaurar el procés que es vol que s'executi en primer lloc.

Com en aquestes fases s'està treballant a molt baix nivell (per exemple modificant valors i índexs a la pila) succeirà freqüentment que la pràctica es descontrolarà, podent deixar penjat el sistema i havent de reengegar la màquina.

✍ *Per estalviar-vos uns quants resets us recomanem que treballeu en **Windows**, obriu una sessió, maximitzeu la pantalla i executeu la pràctica des del **Turbo Debugger**.*

- En el exemple del laboratori 2, hi ha un exemple amb l'ús de la rutina multiplexar que provoca un canvi de context. Heu de tenir en compte que en aquest exemple sols s'usa un única pila per a tots els processos, mentre que en la pràctica cada procés usarà una pila pròpia.

Laboratori 3,4: Implementació del TRAP.

- Introduir la separació entre la capa de sistema i la capa d'usuari.

Una vegada estan implementades les anteriors crides al sistema i s'ha comprovat que funcionen correctament, es pot introduir la separació entre els diferents nivells. Per tal de fer això s'ha d'implementar la funció **trap** (veure apartat 6. Estructuració de la Pràctica). El **trap** estarà associat a una interrupció software, per tant s'executarà amb les interrupcions inhibides. Al ser una RSI haurà de cridar les rutines Salvar i Restaurar que usa el multiplexar.

Si voleu, heu de tenir en compte, que mentre s'està executant la rutina **trap**, poden arribar altres interrupcions (per exemple la del rellotge) que provoquin un canvi de context en el procés de RUN. Aquest fet obliga a tenir varis contexts (un per interrupció que es pugui provocar) guardats en l'estructura del PCB. En el moment de restaurar el context, s'haurà de restaurar en ordre invers a com s'ha salvat.

També podeu implementar el **trap** de manera que no permeteu que us arribin interrupcions mentre esteu realitzant les crides al sistema. Si ho feu així, tan sols es podrà produir un canvi de context quan ho sol·liciti/imposi la crida al sistema que s'executa.

- En el exemple del laboratori 3, hi ha un exemple de l'ús del trap usant els processos tal com els havíem definit en els exemples del laboratori 2, es a dir, usant una única pila per a tots els processos.

- Modificar les rutines de *CrearProces* per a que passi a través del trap.

Abans de continuar amb el següent punt, s'ha de verificar que continuen funcionant correctament. S'ha de tenir en compte, que el compilador de C, depenent de com estigui implementada la rutina, pot decidir emmagatzemar a la pila altres registres a més a més del BP.

A partir d'aquest moment totes les crides al sistema que aneu implementant ja hauran de passar sempre a través del **trap**.

- Implementar la rutina QuiSoc a través del trap.

La rutina *QuiSoc* retornarà el identificador del que l'executa, permetent així la destrucció del propi procés, i possibilitant per tant que els processos puguin finalitzar en lloc de ser bucles infinits.

- Implementar la rutina *DestruirProces* a través del trap.

La crida DestruirProces ha d'eliminar tot el context del procés i forçar l'elecció del nou candidat a executar-se en cas que es vulgui destruir el procés de RUN. Aquesta crida també podrà destruir el processos de la cua de Ready, els processos bloquejats en semàfors, en pantalla i en teclat. No podrà destruir mai al procés Nul.

Si un procés de prova no executa un bucle infinit llavors quan finalitzi el seu codi sempre s'haurà de cridar a la rutina DestruirProces, altrament, el procés (que implementem com una subrutina) retornarà. Al usar una pila pròpia aquest procés no sabria on retornar, dependria de que hagués en aquell moment al capdavall de la pila.

Per tal de comprovar el seu bon funcionament, heu de fer una sèrie de processos que creïn altres processos els quals finalitzin la seva execució cridant a la rutina de *DestruirProces*. S'ha de tenir en compte que aquesta rutina pot destruir el procés que l'executa o pot destruir qualsevol altre procés que es passi com a paràmetre i no sigui el procés nul.

- Implementar les rutines de *DormirProces.*

Heu de tenir present, que el tractament dels processos dormits es farà des de dins de la rutina d'interrupció del rellotge, i per tant haurà de ser el més eficient possible.

Laboratori 5: Entrada i Sortida.

- Implementar les crides al sistema d'Entrada i Sortida.

En aquest punt, s'hauran d'implementar les crides al sistema de llegir i escriure. S'ha de tenir en compte que els processos, ja tindran assignats els dispositius teclat i pantalla com a dispositius estàndards d'entrada i sortida respectivament.

- En el exemple del laboratori 5, hi ha un exemple de l'ús de la rutina *LlegitTeclat* usant els processos tal com els havíem definit en els exemples del laboratori anteriors, es a dir, usant una única pila per a tots els processos.

Laboratori 6: Joc de proves de E/S: Shell.

En el moment de carregar el nostre nucli, es crearan dos processos: Un procés *nul* i un procés *shell*: Aquest procés serà la base de tot el nostre joc de proves. Tots els processos que especifiquem han d'estar en un fitxer ***procs.c***

En tot moment s'ha de veure la informació (identificador procés, estat i quantum consumit) dels processos (actius) que hi ha en les diferents cues.

(Es poden crear unes rutines que ens facin l'actualització en pantalla de l'evolució dels processos)

El procés *nul* executarà un bucle sense fi, que escriurà per pantalla un "molinet" de manera continua.

El procés *shell* executa la crida *Quisoc* escrivint per pantalla el seu codi de retorn. A continuació entrarà en un bucle infinit esperant comandes.

Les comandes mínimes a implementar són:

R y => executar la prova número y.

K => mata tots els processos actius excepte el *nul* i la *shell*.

K i => mata el procés amb identificador i.

Així doncs serà aquest procés inicial (la *shell*) el que permetrà anar executant diferents processos que comprovaran el correcte funcionament de la pràctica.

Laboratori 7: Implementació del sincronisme entre processos i Joc de proves definitiu.

- Implementar les rutines de sincronisme entre processos usant la interfície del **trap**.

Aquí s'haurà de decidir quina estructura de dades per semàfors s'usarà. El més senzill és tenir un vector de semàfors definits, on cada semàfor constarà d'un valor i d'una cua on s'encuaran els PCBs dels processos que es quedin bloquejats quan fan un *Wait*.

Cal fer un joc de proves específic, doncs el seu correcte funcionament és vital per a les següents fases. Els diferents processos podran 'compartir' un semàfor, ja que el seu identificador és assignat pel programador, el qual, a la vegada, haurà de ser molt curós i assegurar-se que els processos que usen un semàfor són els que l'han d'usar.

Realitzar el Joc de proves definitiu:

- Programar una sèrie de processos per tal de provar cada una de les crides al sistema, tenint en compte que s'ha de poder comprovar el seu funcionament per pantalla.

S'ha de comprovar el funcionament correcte de totes les crides al sistema, tant en els casos correctes com en els casos erronis.

No us oblideu de comprovar:

- ✓ el funcionament correcte de l'algoritme de planificació implementat.
- ✓ la creació de més processos dels que té previst el sistema.
- ✓ que el tractament dels processos encuats en els dispositius i en els semàfors, siguin realment FIFO

El joc de proves que es demana és el següent:

Processos de prova:

1. Comprovació del Round Robin.

Executar el mateix procés 6 vegades amb un comportament orientat a CPU i igual quantum. Heu de comprovar que els processos es vagin executant sempre amb el mateix ordre i durant el mateix temps.

2. Comprovació del quantum i la destrucció de processos.

Crear un procés que crearà tres processos. El primer amb un quantum de 4, el segon amb un quantum de 8 i el tercer amb un quantum de 16. A continuació escriurà un missatge de finalització i s'autodestruirà. Després de la crida *DestruirProces*, es comprovarà el codi de retorn i entrarà en un bucle infinit escrivint lletres per pantalla (Aquestes lletres no han d'apareixer per pantalla donat que el procés s'ha autodestruit.

Tots processos de quantum 4 i quantum 8 escriuran lletres per pantalla en fileres consecutives. El proces de quantum 16 ha d'escriure vuit caràcters i ha de destruir els altres dos processos.

3. Comprovació de *DormirProces*

Crear 3 processos idèntics que escriguin lletres per pantalla i, a més es vagin dormint en aquest ordre: el primer procés es dormirà 15 segons, el següent es dormirà 20 segons i l'últim es dormirà 10 segons. Abans de dormir-se ho ha de notificar per pantalla i tant aviat com es desperti també. Els tres processos han de fer les accions pertinents 5 vegades.

4. Comprovació de la reutilització dels PCB's

Engegar tots els processos creats anteriorment al mateix temps, es a dir, crearem més processos que el nombre de pcb's definits.

5. Comprovació del comportament del teclat

Crear tres processos que en un bucle vagin llegint strings per teclat i els escriguin a la pantalla, juntament amb el nombre de caràcters sol·licitats i els realment llegits.

6. Comprovació del Buffering de teclat i la inserció de la tecla <CR>.

Crear tres processos que en un bucle llegeixin per teclat 1,10 i 100 caràcters respectivament. Cada vegada que aconsegueixin llegir caràcters s'adormiran durant cinc segons i tornaran a demanar caràcters. La idea és interaccionar amb ells, inserint <CR> al mig del buffer del sistema i arribar a desbordar-lo (que no hi càpiguen més tecles).

7. Comprovació del comportament dels semàfors

Crear cinc processos idèntics que escriguin diferents lletres per pantalla de manera sincronitzada usant un únic semàfor, en cada moment, sols un d'ells pot escriure a la pantalla.

8. Comprovació del comportament dels semàfors n-aris

Crear cinc processos idèntics que escriguin diferents lletres per pantalla de manera sincronitzada usant un únic semàfor inicialitzat amb el valor de 2. S'ha de poder veure que hi ha un màxim de dos processos simultanis escrivint per pantalla.

9. Implementeu el problema del filòsofs (implementar 6 filòsofs) que pensen i mengen usant semàfors. Implementeu-lo de manera que estigui lliure de deadlocks (estancaments).

10. Altres comprovacions que creieu convenients per tal de fer un joc de proves més complert.

Documentació.

La documentació de la pràctica es pot anar realitzant a mesura que es vagi avançant en el desenvolupament, sobretot a partir de la realització de la 3ª fase, de manera que al final sols quedi ajuntar-ho tot en un únic document.

- La documentació final a lliurar ha d'estar composada per:
  - ✓ Nom de tots els membres del grup que han treballat en la pràctica
  - ✓ Índex
  - ✓ Decisions de disseny amb la seva justificació
  - ✓ Millores i parts opcionals realitzades
  - ✓ Estructures de dades i dimensionat justificat
  - ✓ Pseudocodi de les principals rutines implementades
  - ✓ Explicació del joc de proves realitzat
  - ✓ Tot el codi, dividit en mòduls segons la seva funcionalitat i amb els comentaris adients (constants, estructures, rutines internes del nucli, crides al sistema, processos de prova ...)
  - ✓ Índex de les principals funcions
- Lliurament pel moodle tenint estructurada la pràctica amb els següents noms de fitxers:
  - ✓ Tots els codis fonts documentats: nucli.c, trap.c, procs.c,
  - ✓ *Project* de compilació o fitxer .bat per tal de permetre la compilació de la pràctica
  - ✓ Fitxers executables sense virus

6. Estructuració de la pràctica

Per tal de delimitar els diferents nivells del sistema operatiu es muntaran una sèrie de mòduls:

- Mòdul en C on hi haurà totes les definicions de les estructures de dades globals que s'usin a la pràctica i la funció main. (**nucli**.c)

- Mòdul en C on hi haurà el codi de les rutines d'interrupció multiplexar, trap i teclat, totes les rutines internes al nucli, així com el programa principal. (**rsi**.c)

- Mòdul en C on tindrem la implementació de les rutines del nivell nucli. Aquestes rutines seran les mateixes que es vol tenir com a crides al sistema afegint les lletres **_K** (de *kernel*) a continuació del nom. A aquestes rutines sols tindrà accés el nivell superior al nucli. Per exemple la crida al sistema de CrearProces en aquest nivell s'anomenarà CrearProces_K. En aquest mòdul es podran usar les rutines que ens proporciona el nivell nucli. (**crides.c**)

- Mòdul en assamblador on hi haurà la traducció de les funcions de nivell nucli i del mòdul anterior a les crides al sistema que volem proporcionar als processos de prova. Es crida a la funció **trap** mitjançant la instrucció en assamblador del i8086 **`int #`.(syscall.asm)**

- Mòdul en assamblador on hi haurà la resta de funcions amb assamblador. Salvar, restaurar,...(**salvrest.asm**)

- Mòdul en C on hi haurà els processos de prova i que usarà les crides al sistema que proporciona el mòdul en assamblador anterior. Aquest mòdul sols inclourà un fitxer amb els prototipus de les crides al sistema que s'han implementat. (**procs.c**)

A continuació es mostrarà un exemple on es pot veure l'ús dels diferents mòduls.

- Mòdul en C on hi haurà el programa principal i les definicions de les variables del nucli. (nucli.c)

```
#include "estruc.h"

void main(void)
{  inicialitzacions();
   CanviarVectInt (0x8,Multiplexar); ;
   CanviarVectInt (0x40,trap);
   Crear_proces_nul ();
   Crear_proces_init ();  /* al nivell de l'usuari: serà el pare de la resta */
   Dispatcher();
   Restaurar(proces_run->ss_sp);
}
```

- Mòdul en C on hi haurà totes les rutines que proporciona el nivell nucli: (crides.c)

```
int CrearProces_K(context c, void (*codi)(), uint quantum)
{
/* codi necessari per tal de crear un procés. Context és una estructura de dades
creada per tal de facilitar l'accés als paràmetres de les rutines:
    struct context
    {  int es,ds,ax,bx,cx,dx,di,si,bp; /*valors dels registres del processador*/
       long @_retorn_del_trap_al_codi_asm;
       int flag;
       long @_retorn_crida_sistema_al_proces_usuari;
    } */
}
....
```

- Mòdul en C on hi haurà tot el codi de les interrupcions: multiplexar, trap i teclat (rsi.c)

```
void trap(void)
{   /* prun apunta al procés que esta en RUN */
    prun->ss_sp=salvar(); /* ss_sp apunta al top de la pila del procés de RUN */
    switch (rutina_retorna_ax())
/*rutina és un tipus enumerat, el valor del qual s'ha de passar a través de la
pila del procés (associada a AX), on hi ha quina crida al sistema que s'ha
realitzat */
    {  case crear: CrearProces_K(); break;
       ...
      default: break;
    }
    restaurar(prun->ss_sp) ;
}
```

Procureu mantenir un grau d'independència alt entre el mòdul del nucli i aquest

Recordeu que les crides al sistema són una interfície que us ve definida i que no la podeu canviar (ni el nom, ni els paràmetres, ni el que retornen).

- Mòdul en assamblador on s'oferirà al nivell dels processos de prova les crides al sistema: (syscall.asm)

```
SYSCALLS SEGMENT BYTE PUBLIC 'CODE'

Assume cs:SYSCALLS

Public _CrearProces

_CrearProces proc far
  mov ax, CodiCrearProces   ; indicarà quina crida al sistema es vol realitzar
  int 40h
  ret
_CrearProces endp
.....
SYSCALLS ENDS

END
```

- Mòdul en C on tindrem els processos d'usuari per provar les crides al sistema (procs.c)

```
#include "syscalls.h"
#define NORMAL 0x7

void Init(void)     /* Proces de sistema que executarà el nostre joc de proves*/
{
    CrearProces(procesA,3);
    ...
    DestruirProces(Quisoc()); /* si no esta implementada es pot fer for(;;); */
}

void Nul(void) /* Proces de sistema que s'executarà quan no hi hagi ningú més*/
{   int i;
    char roda[4]={|,/,-,\};
    for (;;)
    { for(i=0;i<4;i++) { EscriureCar(quisoc(),0,roda[i],NORMAL); retard(10); } }
}

void procesA(void)
{   int identificador;  /* variables locals */
    int i=0;
    identificador=CrearProces(procesB,3);
    for (;;)
    {   EscriureCar(quisoc(),i,"A",NORMAL); retard(10);
        i++;
        if  (i=80) i=0;
    }
}
```

✍ *Recordeu que, en aquest mòdul, cap procés ha d'usar **variables** que no siguin del seu 'espai d'adreces' ni cridar a altre **codi** que no siguin les crides al sistema !*

7. Lliurament de la pràctica

Data límit de lliurament convocatòria extraordinària: 15 de gener del 2007

Data límit de lliurament primera convocatòria: 21 de maig del 2007

Data límit de lliurament segona convocatòria: 11 de juny del 2007

El lliurament de les pràctiques en primera convocatòria està subjecte al lliuraments de quatre parts de manera escalonada: Multiplexar (M), Trap (T), Entrada/Sortida(E) i Sincronisme(S). (Mireu la plana del moodle de SOP per tal de veure les condicions i els terminis). La setmana del 21 de maig es faran les entrevistes de la pràctica en hores concertades pel moodle.

El lliurament en segona convocatòria i en convocatòria extraordinària obliga, també, a lliurar totes les parts de la pràctica. Una vegada lliurada tota la documentació es fixarà la data de l'entrevista amb tots els membres del grup.

Les pràctiques es podran passar a recollir des de l'octubre al desembre del 2007.

8. ANNEX 1: Rutines de suport.

Podreu usar les següents rutines

typedef unsigned int **WORD**; /* Natural de 16 bits */

typedef unsigned char **BYTE**; /* Natural de 8 bits */

typedef void * **PTR**; /* Punter genèric (far) */

PTR **Ptr** (WORD Segment, WORD Desplacament); /* Converteix el segment i desplaçament en un punter */

WORD **Segment** (PT0R Adreca); /* Retorna el segment del punter donat */

WORD **Desplacament** (PTR Adreca); /* Retorna el desplaçament del punter donat */

WORD **Word** (BYTE msb, BYTE lsb); /*Retorna un WORD amb part baixa i alta els donats*/

BYTE **MSB** (WORD Valor); /* Retorna el byte alt del word donat */

BYTE **LSB** (WORD Valor); /* Retorna el byte baix del word donat */

WORD **Inhibir** (void) /* Inhibeix les interrupcions (posa IF=0), retornant el valor de la paraula d'estat (flags) abans d'inhibir. Per llegir els flags es útil la instrucció PUSHF. */

void **Desinhibir** (WORD Flags); /* Carrega el valor passat com a paràmetre com a nou valor de la paraula d'estat (flags). Per escriure els flags podem s'usarà la instrucció POPF. */

void **EOI** (void); /* Envia al controlador d'interrupcions (i8259) una comanda de final d'interrupció no específica (EOI). */

void (***CanviarVectInt** (BYTE NumInt, void (*NovaRSI)()))(); /* Assigna una nova rutina de servei d'interrupció a una de les posicions del vector d'interrupcions (V.I.), retornant l'adreça de l'anterior rutina de servei, per tal de restaurar-la més endavant. Quan s'estigui accedint al Vector d'Interrupcions les interrupcions han d'estar inhibides, deixant-les després en el mateix estat (inhibides o no) que quan es va cridar aquesta rutina. */

PTR **BufferPantalla** (void); /* Retorna l'adreça on està mapejada la pantalla. Emmagatzema internament aquesta informació per ser usada per les rutines posteriors. En cas que el mode de pantalla no sigui reconegut retorna un valor que indica que s'ha produït un error. */

void **EscriureCar** (BYTE fila, BYTE columna, char caracter, BYTE atribut); /* Escriu a pantalla el caràcter que se li passa, amb l'atribut indicat, a les coordenades especificades. La informació d'on es troba mapejada la pantalla l'obté de la variable que ha inicialitzat la rutina BufferPantalla(). */

void **EscriureString** (BYTE fila, BYTE columna, char * string, BYTE atribut); /* Escriu a pantalla la cadena de caràcters que se li passa, a partir de la posició donada, i amb el mateix atribut per tots els caràcters. La seva implementació utilitza instruccions de tractament de strings de l'i8086 (LODSB i STOSW). La informació d'on es troba mapejada la pantalla l'obté de la variable que ha inicialitzat la rutina BufferPantalla(). */

void **EsborrarPantalla** (BYTE atribut); /* Omple tota la pantalla de caràcters blancs (espai, codi ASCII 32) amb l'atribut que se li indica. També fa servir instruccions de tractament de strings (concretament, REP STOSW). La informació d'on es troba mapejada la pantalla l'obté de la variable que ha inicialitzat la rutina BufferPantalla().*/

Fitxer suport.asm

```
public _Ptr
public _Segment
public _Desplacament
public _Word
public _MSB
public _LSB
public _BufferPantalla
public _EscriureCar
public _EscriureString
public _EsborrarPantalla
public _QuinCarAtr
public _Inhibir
public _Desinhibir
public _EOI
public _Salvar
public _Restaurar
public _CanviarVectInt
public _InhibirIRQ
public _DesinhibirIRQ

_DATA  segment word public 'DATA'
       SegmPant      dw    ?
_DATA  ends
_BSS   segment word public 'BSS'
_BSS   ends

DGROUP group  _DATA, _BSS

assume cs:EC_CODI, ds:DGROUP

EC_CODI segment byte public 'CODE'
```

```
_Ptr   PROC   FAR
       PUSH   BP
       MOV  BP,SP
       MOV  AX,[BP+8]
       MOV  DX,[BP+6]
       POP  BP
       RET
_Ptr   ENDP
```

```
_Desplacament PROC FAR
       PUSH   BP
       MOV  BP,SP
       MOV  AX,[BP+6]
       POP  BP
       RET
_Desplacament ENDP
```

```
_MSB   PROC   FAR
       PUSH   BP
       MOV  BP,SP
       MOV  AL,[BP+7]
       POP  BP
       RET
_MSB   ENDP
```

```
_Segment   PROC   FAR
       PUSH   BP
       MOV  BP,SP
       MOV  AX,[BP+8]
       POP  BP
       RET
_Segment   ENDP
```

```
_Word  PROC   FAR
       PUSH   BP
       MOV  BP,SP
       MOV  AL,[BP+8]
       MOV  AH,[BP+6]
       POP  BP
       RET
_Word  ENDP
```

```
_LSB   PROC   FAR
       PUSH   BP
       MOV  BP,SP
       MOV  AL,[BP+6]
       POP  BP
       RET
_LSB   ENDP
```

```
_BufferPantalla PROC FAR
       PUSH   ES
       XOR  AX,AX
       MOV  ES,AX
       MOV  AL,ES:[0449h]
       CMP  AL,7
       JE   Monocrom
       CMP  AL,2
       JE   Color
       CMP  AL,3
       JE   Color
Error: MOV  DX,0
       JMP  Sortida
Color: MOV  DX,0B800h
       JMP  Sortida
Monocrom:  MOV  DX,0B000h
Sortida:   MOV  SegmPant,DX
       XOR  AX,AX
       POP  ES
       RET
_BufferPantalla ENDP
```

```
_EscriureCar  PROC   FAR
       PUSH   BP
       MOV  BP,SP
       PUSH   ES
       PUSH   BX
       PUSH   AX
       MOV  ES,SegmPant
       MOV  AL,[BP+6]
       MOV  BX,80
       MUL  BL
       MOV  BL,[BP+8]
       ADD  BX,AX
       SHL  BX,1
       MOV  AL,[BP+10]
       MOV  AH,[BP+12]
       MOV  ES:[BX],AX
       POP  AX
       POP  BX
       POP  ES
       POP  BP
       RET
_EscriureCar  ENDP
```

```
_EscriureString PROC FAR
       PUSH   BP
       MOV  BP,SP
       PUSH   ES
       PUSH   DS
       PUSH   AX
       PUSH   BX
       PUSH   SI
       PUSH   DI
       MOV  ES,SegmPant
       MOV  AL,[BP+6]
       MOV  BL,80
       MUL  BL
       ADD  AL,[BP+8]
       ADC  AH,0
       SHL  AX,1
       MOV  DI,AX
       LDS  SI,[BP+10]
       MOV  AH,[BP+14]
       CLD
BUCLE: LODSB
       CMP  AL,0
       JE    FiBucle
       STOSW
       JMP  BUCLE
FiBucle:   POP  DI
       POP  SI
       POP  BX
       POP  AX
       POP  DS
       POP  ES
       POP  BP
       RET
_EscriureString ENDP
```

```
_EOI   PROC   FAR
       PUSH   AX
       MOV  AL,20h
       OUT  20h,AL
       POP  AX
       RET
_EOI   ENDP
```

```
_Inhibir      PROC   FAR
       PUSHF
       CLI
       POP  AX
       RET
_Inhibir      ENDP
```

```
_Desinhibir    PROC  FAR
       PUSH   BP
       MOV  BP,SP
       PUSH   AX
       MOV  AX,[BP+6]
       PUSH   AX
       POPF
       POP  AX
       POP  BP
       RET
_Desinhibir    ENDP
```

```
_EsborrarPantalla    PROC    FAR
       PUSH   BP
       MOV  BP,SP
       PUSH   ES
       PUSH   AX
       PUSH   CX
       PUSH   DI
       MOV  ES,SegmPant
       MOV  DI,0
       MOV  AH,[BP+6]
       MOV  AL,' '
       MOV  CX,80*25
       CLD
       REP STOSW
       POP  DI
       POP  CX
       POP  AX
       POP  ES
       POP  BP
       RET
_EsborrarPantalla    ENDP
```

```
_
QuinCarAtr  PROC  FAR
       PUSH   BP
       MOV  BP,SP
       PUSH   ES
       PUSH   BX
       MOV  ES,SegmPant
       MOV  AL,[BP+6]
       MOV  BX,80
       MUL  BL
       MOV  BL,[BP+8]
       ADD  BX,AX
       SHL  BX,1
       MOV  AX,ES:[BX]
       POP  BX
       POP  ES
       POP  BP
       RET
_QuinCarAtr    ENDP
```

```
_CanviarVectInt PROC FAR
       PUSH   BP
       MOV  BP,SP
       PUSH   ES
       PUSH   BX
       XOR  AX,AX
       MOV  ES,AX
       MOV  AL,[BP+6]
       SHL  AX,1
       SHL  AX,1
       MOV  BX,AX
       MOV  AX,[BP+8]
       MOV  DX,[BP+10]
       PUSHF
       CLI
       XCHG   AX,ES:[BX]
       XCHG   DX,ES:[BX+2]
       POPF
       POP  BX
       POP  ES
       POP  BP
       RET_CanviarVectInt ENDP
```

```
_Salvar PROC  FAR
       PUSH   BX
       MOV  BX,SP
       XCHG   CX,SS:[BX+2]
       XCHG   DX,SS:[BX+4]
       PUSH   AX
       PUSH   DI
       PUSH   SI
       PUSH   ES
       PUSH   DS
       MOV  AX,DGROUP
       MOV  DS,AX
       PUSH   DX
       PUSH   CX
       RET
_Salvar ENDP
```

```
_Restaurar   PROC FAR
       ADD  SP,4
       POP  DS
       POP  ES
       POP  SI
       POP  DI
       POP  AX
       POP  BX
       POP  CX
       POP  DX
       MOV  SP,BP
       POP  BP
       IRET
_Restaurar   ENDP
```

```
_InhibirIRQ    PROC  FAR
       PUSH   BP
       MOV  BP,SP
       PUSH   AX
       PUSH   CX
       MOV  CL,[BP+6]
       MOV  AH,1
       SHL  AH,CL
       PUSHF
       CLI
       IN   AL,21h
       OR   AL,AH
       OUT  21h,AL
       POPF
       POP  CX
       POP  AX
       POP  BP
       RET
_InhibirIRQ    ENDP
```

```
_DesinhibirIRQ  PROC  FAR
       PUSH   BP
       MOV  BP,SP
       PUSH   AX
       PUSH   CX
       MOV  CL,[BP+6]
       MOV  AH,1
       SHL  AH,CL
       NOT  AH
       PUSHF
       CLI
       IN   AL,21h
       AND  AL,AH
       OUT  21h,AL
       POPF
       POP  CX
       POP  AX
       POP  BP
       RET
_DesinhibirIRQ  ENDP
```

```
EC_CODI ends
end
```

9. ANNEX 2: Rutines de gestió de cues

Per tal de estalviar-vos feina, us proporcionem el codi d'unes rutines per a realitzar la gestió de cues d'estructures de dades genèriques. El seu ús és, òbviament, opcional. També, si voleu, en podeu desenvolupar de noves si trobeu a faltar alguna funcionalitat. Finalment, coneguda l'estructura de les cues, òbviament podeu recórrer-les mitjançant punters en el vostre propi codi.

struct cua *(**inicialitzar_cua** (struct cua *c))
: Inicialitza una estructura cua, fa que el punter al primer i últim ítem de la cua apuntin a l'estructura cua, o sigui, a cap element.

int **buida**(struct cua *c)
: si la cua està buida retorna cert

struct item *(**encuar** (struct cua c, struct item elem))
: posa al final de la cua l'ítem especificat

struct item * (**inserir** (struct cua *c, struct item * elem, int ordre)
: insereix un ítem en una cua ordenada segons el valor del camp *ordre*

struct item *(**primer** (struct cua *c))
: retorna un punter al primer ítem de la cua i el desencua. Si no n'hi ha cap retorna NIL.

struct item *(**extirpar** (struct cua *c, struct item *elem))
: extreu de la cua l'ítem desitjat. Si no el troba retorna NIL

struct item *(**extreure** (struct cua *c, int ordre))
: extreu el primer ítem de la cua amb el camp *ordre* desitjat. Si no n'hi ha cap retorna NIL

Fitxer cues.h

```
/* E.T.S.E / Departament d'Enginyeria Informàtica                    ETIS/ETIG */
/* Sistemes Operatius   CUES:                                 defines & externs */

#ifndef CUES-H
#define CUES-H
#define NIL 0
#ifdef MODUL_CUES

struct item
{     struct item *seguent;
      struct item *anterior;
      int ordre;
};

struct cua
{     struct item *primer;
      struct item *ultim;
      int ordre;                                  /* no usat, sols per coherencia */
};

#else
      extern struct cua *(inicialitzar_cua(struct cua *c));
      extern int buida(struct cua *c);
      extern struct item *(encuar (struct cua *c, struct item *i));
      extern struct item *(inserir (struct cua *c, struct item *i, int ordre));
      extern struct item *(primer (struct cua *c));
      extern struct item *(extirpar (struct cua *c, struct item *e));
      extern struct item *(extreure (struct cua *c, int ordre));
#endif
#endif
```

Fitxer cues.c

```
/* E.T.S.E / Departament d'Enginyeria Informàtica                    ETIS/ETIG */
/* Sistemes Operatius                                                    CUES */

#define MODUL_CUES
#include "cues.h"

/* Funcions per a inicialitzar i mantenir cues doblement encadenades          */
/* Els elements han de ser estructures de manera que els primers camps siguin:*/
/*    struct item *seguent, *anterior;                                        */
/*    int ordre;  criteri d'ordenació (numero d'ordre)                        */
/* NOTA: NO es creen les instanciacions a memòria de les estructures          */

/*--------------------------------------------------------------------------- */
/* Inicialitza una estructura cua, fa que el punter al primer i últim         */
/* ítem de la cua apuntin a l'estructura cua, o sigui, a cap element.         */

struct cua *(inicialitzar_cua (struct cua * c))
{     if (c!=NIL) {     c->ultim = c->primer = (struct item *) c; }
      return(c);
}

/*--------------------------------------------------------------------------- */
/* Funció que pregunta si la cua està buida, i si ho està retorna cert.       */

int buida(struct cua * c)
{     return((c->primer == NIL) || (c->primer == (struct item *) c)); }

/*--------------------------------------------------------------------------- */
/* Procediment que insereix en una cua ordenada un un ítem en un ordre donat  */

struct item * (inserir (struct cua * c, struct item * elem, int ordre))
{     struct item * ant, * post;

      if (c==NIL) return (NIL);                              /* cua no creada */
      ant= c->ultim;

      if (ant==NIL || elem==NIL) return(NIL);       /* cua no inic. o no elem */
      elem->ordre= ordre;

      if (ant == (struct item *) c)       /* cua buida : serà primer i darrer */
      {     elem->seguent= elem->anterior= (struct item *) c;
            c->primer= c->ultim= elem;
            return(elem);
      }
      /* recorrem la cua des de darrera, doncs encuem ! */
      post= (struct item *) c;
      while ( (ant != (struct item *) c) /* !principi de cua i prioritat menor */
            &&    (ant->ordre) < ordre)                 /* ordre(0) < ordre(n) */
      {     post= ant;                  /* ordre = ? => a la cua: RoundRobin */
            ant= ant->anterior;
      }
      /* ja sabem on li toca: fer-ho efectiu */
      ant->seguent= elem;                                       /* pel davant */
      elem->anterior= ant;
      elem->seguent= post;                                   /* i pel darrera */
      post->anterior= elem;
      return(elem);
}
```

```
/*-------------------------------------------------------------------------- */
/* Procediment que fica al final de la cua l'ítem especificat.               */

struct item * (encuar (struct cua * c, struct item * elem))
{     struct item * aux;

      if (c==NIL) return (NIL);                              /* cua no creada */
      aux= c->ultim;
      if (aux==NIL || elem==NIL) return(NIL);       /* cua no inic. o no elem */
      elem->seguent= aux->seguent;                      /* al final de la cua */
      elem->anterior= aux;                           /* recorda darrera qui vas */
      (aux->seguent)->anterior= elem;                   /* == c->ultim= elem; */
      aux->seguent= elem;                              /* el de davant el veu */
      return(elem);
}

/*-------------------------------------------------------------------------- */
/* Funció que extreu de la cua l'ítem desitjat. Si no el troba o la cua      */
/* no està inicialitzada retorna NIL.                                        */

struct item *(extirpar (struct cua *c, struct item *elem))
{     struct item * post, * ant;

      if (c==NIL)       return (NIL);                        /* cua no creada */
      post= c->primer;
      if (post==NIL)    return (NIL);                 /* cua no inicialitzada */

      ant= (struct item *) c;       /* localitzem la seva posició dins la cua */
      while ((post!=(struct item *) c) && (post!=elem))/* !fi de cua i !trobat*/
      {     ant= post;
            post= post->seguent;
      }
      if (post == (struct item *) c) return(NIL);               /* no trobat */
      ant->seguent= post->seguent;                               /* treure'l */
      (post->seguent)->anterior= ant;
      post->seguent= NIL;                             /* ara ja no es a la cua */
      post->anterior= NIL;
      return (post);
}
/*-------------------------------------------------------------------------- */
/* Funció que extreu de la cua ordenada un ítem amb el número d'ordre indicat */
/* Si no el troba o la cua no està inicialitzada retorna NIL.                */

struct item *(extreure (struct cua *c, int ordre))
{     struct item * post, * ant;

      if (c==NIL) return (NIL);                              /* cua no creada */
      post= c->primer;
      if (post==NIL)    return (NIL);                 /* cua no inicialitzada */

      ant= (struct item *) c;       /* localitzem la seva posició dins la cua */
      while ((post!=(struct item *) c)
            && (post->ordre > ordre))                   /* !fi de cua i !trobat */
      {     ant= post;
            post= post->seguent;
      }
      if ((post == (struct item *) c) || (post->ordre != ordre))
            return(NIL);          /* no trobat ningú amb aquest número d'ordre */
      ant->seguent= post->seguent;                                 /* treure'l */
      (post->seguent)->anterior= ant;
      post->seguent= NIL;                             /* ara ja no es a la cua */
      post->anterior= NIL;
      return (post);
}
```

```
/*-------------------------------------------------------------------------- */
/* Funció que retorna un punter al primer ítem de la cua i el desencua.      */
/* Si la cua no està inicialitzada o està buida retorna NIL.                 */

struct item *(primer (struct cua * c))
{     struct item * aux;

      if (c==NIL)       return (NIL);                        /* cua no creada */
      aux= c->primer;
      if (aux==NIL || aux==(struct item *)c) return(NIL);/*cua no inic o no elem*/
      c->primer= aux->seguent;
      (aux->seguent)->anterior= (struct item *) c;
      aux->seguent= NIL;                              /* ara ja no es a la cua */
      aux->anterior= NIL;
      return(aux);
}
```

10. Crides al Sistema: Índex de Referències