ÍNDEX

Pràctica Sistemes Operatius 2

1.- Móduls

Fitxer: main.c
Descripció: Mòdul amb el codi de les rutines d' interrupció multiplexar, trap i teclat, a més de totes les rutines internes del nucli i el programa principal.

Fitxer: nucli.c
Descripció: Mòdul amb les implementacións de les rutines (crides a sistema) a nivell nucli.

Fitxer: estruct.h
Descripció: Capçalera amb les estructures principal i constants útils per la configuració del kernel abans de compilar.

Fitxer: salvrest.asm
Descripció: Mòdul en assamblador amb les funcions necessaries del kernel.

Fitxer: crides.asm
Descripció: Mòdul en assamblador amb la traducció de les funcions de nivell nucli i del mòdul nucli.c que volem proporcionar als processos de prova.

Fitxer: procs.c
Descripció: Mòdul amb els processos de prova que fan us de les crides al sistema proporcionades pel mòdul crides.asm.

Per compilar-la s'ha creat un fitxer de procés per lots: compilar.bat amb el següent contingut: "tcc main.c nucli.c procs.c crides.asm ruts_so.lib cues.c salvrest.asm"

L'executable que inicia el sistema s'anomena main.exe.

2.- Descripció

Des d'un principi hem intentat realitzar un nucli amb la màxima facilitat de configuració abans del compilat, per això hem definit diverses constant que podem modificar facilment el comportament del sistema:

```
#define MIDA_PILA 200
#define NPROC 20  // Nombre de processos màxim
#define NSEM 20 // Nombre de semafors que te el sistema
#define BUFFERTECLAT 200 // Nombre de caracters del buffer teclat
#define QUANTUM_CUA1 4
#define QUANTUM_CUA2 8
```

Aquestes constant es troben al arxiu estruct.h. Les prioritats del processos aniran des de 0 fins a 9, sent 0 la prioritat més baixa i 9 la més elevada.

3.- Estructures

En general disposarem de les seguents estructures:

```
Cua de run, ready1 (+ prioritat, quantum 4), ready2 (-prioritat, quantum 8), blocked,
pcbs lliures i adormits.
  struct cua cuaRUN, cuaRDY1, cuaRDY2, cuaBLK, cuaFREE, cuaSLEEP;

Vector de PCBs
  tPCB pcbs[NPROC];

Identificador de proces que s'incrementarà en la creació de nous processos
  char identificadorProc='  A'  ;

Apuntador al proces en execució, ens facilitarà el seu accés en certes ocasions
  tPCB *procRun;

Quantum consumit
  int Quantum=QUANTUM_CUA1;

Indicador de la cua d'on procedeix el procés en execució, útil per planificar
  int cuaProcedent=1;      // (1=cua prioritaria, 2=cua menys prioritaria)

Vector de semàfors
  tSemafor Semafors[NSEM];
  int SemaforsLliures=NSEM;      // Útil per saber rapidament si encara queden

Semafor per accedir a la pantalla (separat del altres per evitar que un usuari accedeixi)

  tSemafor SemaforPantalla;

Buffer del teclat
  char bufferTeclat[BUFFERTECLAT];
```

4.- Implementació, decisions de diseny i altres estructures

Per portar a terme aquesta pràctica hem seguit rigurosament l'ordre d'implementació especificat en l'enunciat, es a dir, hem anat en aquest ordre:

- Disseny d'estructures -

Vam decidir que el PCB del processos tingues aquesta forma:

```
typedef struct TipusPCB
{
        struct item Fil;    // de cues.h
        int IdProces;
        int IdPCB;
        int Estat;
        PTR SS_SP;
        WORD pila[MIDA_PILA];
        int Dormir;         // Temps que ha de dormir quan esta en estat ASLEEP
        int Prioritat;
        int llegir;         // Quantitat de lletres que espera rebre al buffer teclat
        char *buffer;       // Punter al buffer que contindra el que ha demanat per teclat
        int QSencer[2];   // Historic de la ultima i penultima execució:
                                //          true: quantum utilitzat completament / false: contrari
} tPCB;
```

Cal destacar l'historic en forma de array d'enter a on guardarem l'evolució del procés indicant si a les últimes 2 execucions ha sigut consumit tot el quantum. Això ens serà útil per poder implementar l'algorisme de planificació de curt termini, el qual escollirà a quina cua de Ready ha de encuar el procés en base a aquest històric.

- Implementació CrearProcés -

Agafa 1 PCB lliure si n'hi ha
Inicialitza la seva pila
Compara la prioritat del nou proces amb el que esta en execució
  Si es més prioritari el nou, fem apropiació
  Sino encuem a la primer cua de Ready

Per inicialitzar la pila, vam implementar una funció en ensamblador:

```
_iniPila proc far
   push bp
   mov bp, sp

   mov cx, ss      ; guardem la posició de la pila actual
   mov bx, sp

   mov ax, ss:[bp+6]      ; segment del procés --> AX
   mov dx, ss:[bp+8]      ; desplaçament del procés --> DX

   mov sp, ss:[bp+12]
   mov ss, ss:[bp+10]      ; redireccionem cap a la nostra pila

   pushf              ; fiquem els flags a la pila
   pop bp                        ; ens assegurem que el flag d'  interrupcio ha d'  estar activat
   or bp, 200h
   push bp

   push dx             ; fiquem segment del procés
   push ax             ; fiquem desplaçament del procés

   push 0      ; bp
   push 1      ; cx
   push 2      ; ax
   push 3      ; bx
   push 4      ; si
   push 5      ; di
   push ds
   push es
   push 6      ; dx

   mov dx, ss      ; retornem la posició final de la pila
   mov ax, sp

   mov ss, cx      ; restaurem la pila
   mov sp, bx

   pop bp
   ret
_iniPila endp
```

A l'adreça de memòria especificada per paràmetre, guardem els flags (primer en asegurem que el flag i estigui activat per que pugui rebre interrupcions quan es restauri el procés), després guardem l'adreça d'inici del codi del proces passat per paràmetre també i finalment fem espai per tots els registres que seran recuperat amb al restaurar context, però que no tindran importància perque es tractarà de la primera execució.

- Implementació de l'algorisme de planificació -

Aquest va ser un punt clau de la pràctica, haviem de crear unes funcions en ensamblador per guardar i restaurar el context del procesos, a més d'implementar les funcions en C: multiplexar, scheduler i dispatcher, les qual havien de decidir i portar a terme el canvis de context necessaris segons l'algorisme de planificació plantejat a l'enunciat.

Segons està especificat a la pràctica, hi ha dues cues de Ready amb una política Round Robin amb prioritats estàtiques i apropiacions. A la cua més prioritària (pRDY1) el quantum serà de quatre tics de rellotge, mentre que a la segona cua (pRDY2) serà de vuit tics. Un procés de la primera cua que consumeixi dues vegades consecutives tot el quantum assignat, es degradarà a la segona cua. Mentre que un procés de la segona cua que consumeixi menys del quantum que li toca es promocionarà a la primera cua. Sempre que un procés entri a la cua de Ready des d'un estat diferent de Run anirà a la primera cua.

Per guardar el context d'un procés fem ús de la següent funció:

```
_salvarContext proc far
   push bx
   mov bx, sp
   xchg ax, ss:[bx+2]        ; guardem ax a la pila i agafem adreça de retorn de la RSI
   xchg cx, ss:[bx+4]  ; idem anterior

   push si
   push di
   push ds
   push es
   push dx

   push cx     ; restaurem l'  adreça de retorn
   push ax

   mov ax, DGROUP
   mov ds, ax
   mov es, ax

   mov dx, ss      ; retornem el ponter cap a la nostra pila
   mov ax, sp
   add ax, 4       ; saltem l'  adreça de retorn
   ret
_salvarContext endp
```

Guarda el context a la pila de l' ultim procés que s' estava executant, com que ha sigut interromput per una int, l' adreça a on s' ha quedat l' execució esta a la pila, per tant, aprofitarem aquesta situació. A la pila tindrem l' adreça de retorn a la RSI i la del proces interromput, per tant, l'adreça de retorn a la RSI la ficarem al cap damunt de tot el que guardem perque retorni correctament al fer el ret.

Per recuperar el context:

```
_restaurarContext proc far
   add sp, 4       ; saltem l'  adreça de retorn
   pop ax          ; ens situem en l'  adreça de la pila
   pop ss          ; que se li ha passat per paràmetre
   mov sp, ax

   pop dx          ; restaurem els registres
   pop es
   pop ds
   pop di
   pop si
   pop bx
   pop ax
   pop cx
   pop bp
```

```
    iret
_restaurarContext endp
```

Restaurem context i amb iret saltarem al punt on va ser interromput el procés.

La rutina multiplexar haurà de fer en general:

Salvem context del procés a Run
Si la cua de dormits no esta buida -> actualitzarla
Si el quantum es 0 cridar al scheduler i el proces que ens retorni el pasem al dispatcher
pq el fiqui en execució
Sino decrementem quantum i restaurem procés que estava a Run

La rutina scheduler consisteix en:

Treu de Run el procés actual i segons el seu historial d'execució encua a la cua de
Ready 1 o 2
Si la cua de Ready 1 no esta buida retorna l'element més prioritari d'ella
Sino retorna l'element més prioritari de la cua de Ready 2

La rutina dispatcher realitza la següent feina:

Prepara per introduir a Run el procés que li han passat
Restaura el seu context

- Implementació del TRAP -

El trap farà que la capa nucli estigui separada de la capa d'usuari de forma que quan un procés faci una crida a sistema, realment es produirà una interrupció que tractarà el trap cridant a la funció adient. Hem implementat el trap de forma que no permetem que ens arribin interrupcions, es a dir, les crides a sistema es realitzen de forma atòmica. Al iniciar el trap guardarem el context del procés en execució i al final el recuperarem.

-Implementació de les crides a sistema: QuiSoc, DestruirProcés i DormirProcés -

QuiSoc simplement retorna l'identificador de procés:

```
int QuiSoc_K(struct enviroment context)
{
 int id;
 id=procRun->IdProces;
 context.ax=id;
 return (id);
}
```

Com a paràmetres rebrem de forma automàtica l'estructura enviroment:

```
extern struct enviroment
{
  int dx,es,ds,di,si,bx,ax,cx,bp;
  long retTrap;                 // Adreça de retorn del trap
  int flag;
  long retCridaSys;             // Adreça de retorn de la crida de sistema
};
```

Com que el trap fa un salvarContext i crida la funció QuiSoc_K() la pila contindra la informació del struct enviroment en el mateix ordre. Per tant, sense passar directament res per paràmetre estem tenint accès al que hi ha a la pila, podent així modificar el context del procés en execució i retornant-li valors mitjançant el registre AX.

La rutina DestruirProces buscarà per tot el sistema (per totes les cues) el procés indicat i el treura alliberant els recurs ocupats. Tot i això vam decidir que no sigui possible eliminar el proces null ja que sempre hi ha d'haver com a mínim 1 procés al sistema perque tot funcioni correctament.

La rutina DormirProces simplement treu de Run el proces i el fica a la cua SLEEP, la qual serà actualitzada (actualitzarDormits()) cada tic de rellotge des de multiplexar.

actualitzarDormits() decrementarà el temps que porten dormits el processos i si hi ha algun que hagui de despertar el ficarà a la cua de Ready 1, a no ser que sigui més prioritari que el que esta en execució, en aquest cas farà apropiació.

- Implementació del sincronisme entre processos -

Vam decidir fer un vector de semàfors, els qual podrien ser inicialitzats i eliminats. L'estructura d'un semàfor és la següent:

```
typedef struct TipusSemafor
{
  struct cua cuaBLK;
  struct cua *pBLK;
  int id;
  int valor;
} tSemafor;
```

Tindrem una cua a on encuar els processos que haguin fet un WaitS i haguin de quedar bloquejats, un identificador de semafor i un valor que serà inicialitzat i modificat per SignalS i WaitS.

Per inicialitzar un semàfor necesitarem assignar-li un identificador i un valor inicial. Es farà amb la funció IniSem, aquesta seguirà el següents pasos:

```
  Comprova que hi haguin semàfors lliures (farem ús d'una variable global que ens
                                          indicarà quants semàfors ens queden)
  Comprova que l'identificador especificat no estigui repetit
  Inicialitza el semafor
```

Per eliminar un semàfor s'utilitzar ElimSem al qual només li hem d'indicar l'identificador del semàfor que volem esborrar. En cas de que no existeixi o tingui processos esperant a la seva cua de bloquejats es retornarà error.

La funció WaitS buscarà el semàfor indicat i si el seu valor es igual a 0, encuarà el procés a la cua de bloquejats del semafor i posarà a Run un altre, en cas contrari només decrementarà el valor.

La funció SignalS buscarà el semàfor indicat i incrementarà el seu valor en 1 unitat, seguidament comprova la cua de bloquejats i si no esta buida desencuarà un procés i el ficarà a la cua de Ready 1, a no ser que sigui més prioritari que el que estigui a Run, llavors efectuarà l'apropiació.

NOTA: A totes les crides de sistema s'utilitza el mateix sistema descrit a la rutina QuiSoc per retornar paràmetres.

- Entrada i Sortida -

Per implementar la crida a sistema escriure s'ha hagut de modificar la RSI del teclat per una implementada per nosaltres. En general el que farà serà llegir tecles i en cas de que la cua de bloquejats no estigui buida, els passarà els caracters llegits que haguin demanat o bé fins el primer Enter. Per que això sigui possible disposem d'un buffer de teclat al qual es van guardant les tecles pitjades, a més hem implementat una rutina en ensamblador perque llegeixi les lletres que han sigut pitjades:

```
_llegirTecla PROC FAR
        in      al, 60h         ; AL <- "Scancode" Tecla polsada
    test    al, 10000000b           ; Make/Break?
    jz      NoEsBreak
    mov             al, 1                   ;Ignorem Break, fiquem com a senyal un 1
NoEsBreak:
    mov             ah, 0
```

```
        ret
_llegirTecla ENDP
```

Aquesta rutina només pot ser cridada des de la RSI del teclat, ja que això significa que hi ha una tecla disponible que podem llegir, no es pot cridar des de cap altre banda. Quan ja se li han passat totes les tecles demanades a un proces, el treiem de la cua de bloquejats i el insertem a Ready comprovant sempre si cal fer apropiació.

Un procés que fa la crida LlegirTeclat indicant quants carácter vol, serà encuat a la cua de bloquejats en cas de no disposar ja d'aquest nombre de caracters. Aquest procés sortirà de bloquejats tal i com hem explicat anteriorment, només cal afegir que la crida LlegirTeclat ha de retornar el nombre de caràcters que ha llegit, per poder fer això hem hagut d'implementar una rutina en ensamblador:

```
_canviarAX proc far
   push bp
   mov bp, sp
   push es
   push bx
   push cx

   mov bx,[bp+6]
   mov es,[bp+8]
   mov cx,[bp+10]
   mov word ptr es:[bx+12],cx

   pop cx
   pop bx
   pop es
   pop bp
   ret
_canviarAX endp
```

Quan cridem canviarAX li passarem l'adreça a on es troba la pila del procés i el valor que volem ficar-li al seu AX, que en realitat serà el que retornarà la funció LlegirTeclat.

Per implementar EscriurePantalla hem creat un semàfor perque els diferents procesos pugin accedir de forma concurrent i sincronitzada. Aquest semàfor serà independent als semàfors oferts pel sistema al usuari, d'aquesta forma evitem que pugui ser manipulat. Hem creat SignalSPantalla() i WaitSPantalla() que fan una funció similar a les rutines SignalS() i WaitS(), però només són accesible pel kernel. La rutina EscriurePantalla simplemente fa un Wait al principi, escriu per pantalla lo demanat i realitza un Signal al final.

- Joc de proves i ús del sistema -

Hem realitzat tots els processos demanats a l'enunciat i funcionen correctament. Hem creat un procés null amb prioritat mínima (0) i una shell amb prioritat màxima (9). Des del procés shell es poden executar tots el jocs de proves amb R1, R2...R9, amb la K es poden matar tots el processos i amb Q sortim. A més hem afegit la posibilitat de netejar la pantalla amb la N, llistar el contingut de les cues del sistema amb L (només visualitza les cues que tenen algun procés, les altres les ignora) i amb la A es visualitzarà una petita ajuda a on s'indiquen totes les funcions disponibles.

A l'hora de provar els jocs de prova s'ha de tenir en compte que quan un procés fa una crida a LlegirTeclat normalment queda encuada després del procés Shell (de màxima prioritat) que també esta continuament esperant tecles, per tant la primera pulsació l'agafa la Shell i les següents el procés que ha demanat el teclat.

Per pantalla es visualitzarà continuament certa informació molt útil per veure l'evolució del sistema. En primer lloc trobem "Running X" que indica l'identificador del procés que esta a Run, seguidament trobem "CuaRDY ½" que ens indica d'on hem agafat el procés que esta en execució, si de la cua 1 (+ prioritaria, quantum 4) o de la 2 (- prioritaria, quantum 8), després hi ha "Prioritat 5" que especifica la prioritat del procés en execució. A continuació es troba "Shell(B) – Null(A) /" que són els identificadors de cada procés i un símbol al seu costat que es mostrarà en moviment

quan estiguin a Run. Finalment tenim "Quantum \ 7" que mostrarà l'evolució del quantum, a més el símbol es moura cada cop que la rutina multiplexar sigui executada.