LABORATORI DE SISTEMES OPERATIUS

ETAPA 3: - Final (5 setmanes)

Com a punt de partida d'aquesta etapa es farà servir el codi base proporcionat a la web de l'assignatura (http://studies.ac.upc.es/FIB/SO/Welcome.html). Aquest codi cubreix la segona etapa de la realització d'ONION, amb les particularitats que es detallen a continuació:

* La planificació utilitza apropiació immediata
* Els processos tenen 2 contextos: un d'usuari i un de kernel
* Els processos poden canviar de context a nivell de kernel
* Es fa una gestió de PCBs lliures i de piles lliures
* Hi ha una taula amb les crides a sistema
* Les inicialitzacions que es fan al boot.c utilitzen rutines que estan implementades al nivell adequat
* Es dóna implementada la crida al sistema `retardar` descrita a l'apartat A d'aquest enunciat per tal que tingueu un exemple de com funciona el nou canvi de context a nivell de kernel

Tenint en compte la quantitat de modificacions que s'han realitzat, i el canvi conceptual d'algunes d'elles (per exemple, el canvi de context a nivell de kernel), es recomana que estudieu el codi abans d'anar al laboratori.

En aquesta darrera etapa afegirem la resta de crides al sistema. El mecanisme per cridar a serveis del sistema serà igualment mitjançant un *trap* dins de les rutines de la llibreria `onionlib.lib`. A la rutina `trap()` s'avaluarà quin servei es demana i executarà la rutina corresponent de nivell sistema.

> **Recordeu:** La implementació en assemblador de les crides al sistema ha d'estar al fitxer `onionlib.asm`, les quals implementen el mecanisme de *trap* passant el corresponent valor al registre `AX`. Heu d'implementar els serveis de nivell sistema, BFS i nucli, tot i que les úniques rutines que executen codi útil són les de nivell nucli. A totes les rutines de nivell nucli heu de comprovar que les especificacions s'ajusten a la documentació d'ONION.

A) Gestió de temps

En el codi base es dóna implementada la crida al sistema per permetre a un procés bloquejar-se durant un temps determinat. La descripció d'aquesta crida al sistema és la següent:

`SINTAXI:`
```
int retardar(n_tics)
int n_tics;
```

`DESCRIPCIÓ:` Aquesta rutina bloqueja el procés que la invoca durant el nombre de tics indicat per la variable `n_tics`. Només es permet que hi hagi un procés bloquejat per aquest motiu. En cas de més d'una sol.licitud es retornarà un error.

`RETORN:` El valor de retorn 0 indica una execució correcte. Si hi ha més d'una sol.licitud de retard el valor retornat serà -1

B) Sincronització a nivell de sistema

Cal implementar crides de sincronització entre processos. Serveixen per a que diferents fragments de codi de sistema es puguin executar en exclusió mútua.

Necessitarem 2 nivells de semàfors. En el primer, s'hauran d'implementar les crides que utilitzarà el kernel quan vulgui aconseguir exclusió mútua. La descripció d'aquestes crides és la següent:

`SINTAXI:`
```
int kinit_sem(n_sem, valor_inicial)
int n_sem, valor_inicial;
```

`DESCRIPCIÓ:` Aquesta rutina inicialitza el valor del semàfor `n_sem` a `valor_inicial`. Al mateix temps inicialitza la cua d'aquest semàfor i les dades necessàries per a la seva correcta utilització.

`RETORN:` El valor de retorn 0 indica una execució correcte. Si `n_sem` no és un identificador vàlid per a un semàfor o ja es troba inicialitzat el valor retornat serà -1.

`SINTAXI:`
```
int kdestruir_sem(n_sem)
int n_sem;
```

`DESCRIPCIÓ:` Aquesta rutina allibera el semàfor `n_sem`.

`RETORN:` El valor de retorn 0 indica una execució correcte. Si `n_sem` no és un identificador vàlid per a un semàfor o no esta inicialitzat el valor retornat serà -1.

`SINTAXI:`
```
int kwait(n_sem)
int n_sem;
```

`DESCRIPCIÓ:` Si el valor del semàfor `n_sem` és més petit o igual que zero, aquesta crida bloqueja en aquest semàfor el procés que la ha invocat. En cas que `n_sem` sigui més gran que zero, aquesta crida decrementa el valor del semàfor.

`RETORN:` El valor de retorn 0 indica una execució correcte. Si `n_sem` no és un identificador vàlid per a un semàfor o bé el semàfor no està inicialitzat el valor retornat serà -1.

`SINTAXI:` `int ksignal(n_sem)`
`int n_sem;`

`DESCRIPCIÓ:` Si no hi ha cap procés bloquejat en el semàfor `n_sem` aleshores aquesta crida incrementa el valor del semàfor `n_sem`. En el cas que hi hagi un procés bloquejat sobre `n_sem`, aquesta crida desbloqueja el primer procés.

`RETORN:` El valor de retorn 0 indica una execució correcte. Si `n_sem` no és un identificador vàlid per a un semàfor o bé el semàfor no està inicialitzat el valor retornat serà -1.

En el segon nivell de semàfors, cal implementar les rutines per a la gestió de semàfors com si fossin crides al sistema, és a dir, usant el mecanisme de *trap* i implementant les rutines al nivell sistema, per tal que puguin ser cridades pels usuaris. Fixeu-vos que aquestes rutines hauran d'acabar cridant a les rutines de gestió de semàfors que utilitza el kernel. La descripció d'aquestes crides és la següent:

`SINTAXI:` `int init_sem(n_sem, valor_inicial)`
`int n_sem, valor_inicial;`

`DESCRIPCIÓ:` Aquesta rutina inicialitza el valor del semàfor `n_sem` a `valor_inicial`. Al mateix temps inicialitza la cua d'aquest semàfor i les dades necessàries per a la seva correcta utilització.

`RETORN:` El valor de retorn 0 indica una execució correcte. Si `n_sem` no és un identificador vàlid per a un semàfor o ja es troba inicialitzat el valor retornat serà -1.

`SINTAXI:` `int wait(n_sem)`
`int n_sem;`

`DESCRIPCIÓ:` Si el valor del semàfor `n_sem` és més petit o igual que zero, aquesta crida bloqueja en aquest semàfor el procés que la ha invocat. En cas que `n_sem` sigui més gran que zero, aquesta crida decrementa el valor del semàfor.

`RETORN:` El valor de retorn 0 indica una execució correcte. Si `n_sem` no és un identificador vàlid per a un semàfor o bé el semàfor no està inicialitzat el valor retornat serà -1.

`SINTAXI:` `int signal(n_sem)`
`int n_sem;`

`DESCRIPCIÓ:` Si no hi ha cap procés bloquejat en el semàfor `n_sem` aleshores aquesta crida incrementa el valor del semàfor `n_sem`. En el cas que hi hagi un procés bloquejat sobre `n_sem`, aquesta crida desbloqueja el primer procés.

`RETORN:` El valor de retorn 0 indica una execució correcte. Si `n_sem` no és un identificador vàlid per a un semàfor o bé el semàfor no està inicialitzat el valor retornat serà -1.

Decidiu el nombre de semàfors que voleu implementar (entre 20 i 30), reserveu una part de manera que només els pugui utilitzar el sistema (si una crida de l'usuari intenta utilitzar-los haureu de retornar un error), i definiu-vos les estructures de dades que creieu oportunes. Feu les proves necessàries per garantir l'ús correcte de les crides de sincronització, tant per implementar una exclusió mútua, com per sincronització entre processos.

C) Gestió de processos

Afegirem a la crida de creació d'un procés, les crides al sistema per destruir un procés, per consultar l'identificador de procés (*pid*), per consultar les característiques d'un procés, i per modificar les característiques d'un procés.

Concretament, les crides que heu de implementar són les següents:

`SINTAXI:`
```
int destruir_pcb(id_proc)
int id_proc;
```

`DESCRIPCIÓ:` Aquesta rutina destrueix el procés identificat per `id_proc`, alliberant les estructures de dades corresponents. No es pot destruir el procés nul (el qual té com a identificador de procés el 0).

`RETORN:` El valor de retorn 0 indica una execució correcte. Altrament el valor retornat serà -1.

`SINTAXI:`
```
int quisoc()
```

`DESCRIPCIÓ:` Aquesta rutina retorna l'identificador del procés que ha invocat la crida.

`RETORN:` Identificador del procés.

`SINTAXI:`
```
int info_proc(id_proc, p_info)
int id_proc;
struct info_proc *p_info;
```

`DESCRIPCIÓ:` Aquesta rutina omple el *buffer* apuntat per `p_info` amb la informació del procés `id_proc`. L'estructura `info_proc` és de la forma:

```
struct info_proc {
```

```
        int prio;
        int quantum;
        long t_cpu;
    }
```

`RETORN:` El valor de retorn 0 indica una execució correcte. Altrament el valor retornat serà -1.

`SINTAXI:`
```
int modif_proc(id_proc, p_info)
int id_proc;
struct info_proc *p_info;
```

`DESCRIPCIÓ:` Aquesta rutina modifica els paràmetres prioritat i quantum del procés `id_proc`, segons el contingut de l'estructura `p_info`. No es podran modificar els paràmetres del procés nul.

`RETORN:` El valor de retorn 0 indica una execució correcte. Altrament el valor retornat serà un -1.

D) Gestió del teclat

La següent crida al sistema que heu d'implementar serveix per llegir un caràcter del teclat..

La descripció de la rutina és la següent:

`SINTAXI:`
```
char llegir_teclat()
```

`DESCRIPCIÓ:` Aquesta rutina permet a un procés llegir un caracter del teclat. El sistema bloqueja el procés fins que es premi una tecla.

`RETORN:` Retorna el caracter llegit del teclat.

Heu d'implementar la rutina de la llibreria, al igual que les rutines de nivell sistema, BFS i nucli.

Pas 1

La rutina de nivell nucli `llegir_teclat_nuc` ha de bloquejar al flux a nivell de kernel, forçant un canvi de contexte. Podeu assumir que aquesta rutina no serà cridada per més d'un procés alhora. La recollida de la tecla es farà després de tornar del bloqueig.

Per altre banda, heu d'implementar la rutina de servei a la interrupció del teclat `rsi_teclat()`. Aquesta rutina usarà la rutina `llegir_teclat_hw()` de la llibreria `solib.lib` per llegir un caracter del registre de dades del controlador de teclat. És important tenir en compte que sempre que hi ha una interrupció de teclat s'ha de llegir el registre de dades del teclat per tal de generar la seqüència d'*strobe*.

La rutina d'atenció a la interrupció de teclat ha de mirar si hi ha algun flux bloquejat al teclat: en aquest cas li retornarà el caracter i el desbloquejarà; altrament, es cridarà a la rutina `beep()`com a senyal que el caracter s'ha perdut.

En iniciar el sistema, cal connectar una rutina de servei a la interrupció de teclat (recordeu que és la interrupció `0x09`) usant la rutina `vector_int()`. Feu una primera prova en què un procés d'usuari demani un caracter al teclat i l'escrigui a la seva part de la pantalla, fent servir les dues crides a sistema. Comproveu que en bloquejar-se el procés en teclat es posa en marxa el procés `nul`.

Pas 2

Un cop hagueu comprovat que la crida al sistema per llegir un caracter del teclat funciona correctament, cal implementar la optimització de ***buffering***. Aquesta consisteix en mantenir un *buffer* circular de tecles pitjades que no han estat llegides. Quan un procés sol.liciti llegir una tecla agafarà la tecla del *buffer.* Si no hi ha tecles al *buffer* aleshores el procés s'ha de bloquejar fins que hi hagi alguna. La rutina de servei a la interrupció de teclat afegirà les tecles al *buffer.*

Pas 3

S'ha de afegir un gestor de teclat a la capa de sistema. Aquest gestor agafarà la feina d'una cua de peticions (IORB) i cridarà a `llegir_teclat_bfs`. S'ha de modificar `llegir_teclat_sis` perque encui una petició a aquest gestor i esperi a que finalitzi la seva petició per retornar el resultat a l'usuari (funcionament sincron).

La sincronització entre el gestor i els procesos es farà mitjancant les primitives de semafors del kernel. La estructura de les peticions serà la següent:

```
struct iorb_teclat {
        struct item it;
        int id_io;
        char c;
};
```

Cada cop que s'hagi de fer una operació E/S s'haurà de buscar un semàfor dels reservats per el sistema que estigui lliure per utilitzar-lo en la sincronització entre el gestor i el procés d'usuari. Al finalitzar la operació el semàfor s'ha de destruir.