Entrada / Sortida

1.- En el sistema operativo Onion, las funciones de E/S ***llegir*** y ***escriure*** son asíncronas con la posibilidad de hacerlas síncronas utilizando la llamada al sistema ***esperar***(id_io). En concreto, *esperar(llegir/escriure*(canal,buff,long)) requiere de dos llamadas a sistema para realizar la E/S síncrona.

Qué modificaciones tendríamos que realizar, si hiciera falta, para que las mismas llamadas al sistema *llegir* y *escriure* pudieran ser síncronas o asíncronas?

- Indicad los parámetros de la nueva llamada.
- Describid las estructuras de datos internas que se ven modificadas, respecto al modelo visto en clase.
- Comentad brevemente los procesos y/o rutinas de sistema afectados.

2.- La llamada al sistema ***tancar***(n) de Onion espera a que finalicen todas las operaciones de E/S realizadas sobre el canal n. Queremos modificarla para que el proceso que la llame no tenga que esperar por unos resultados que ya no quiere utilizar. tampoco las debe cancelar.

1) Especificad este nuevo comportamiento, dando las modificaciones de las rutinas y estructuras de datos afectadas.

2) Indicad además cuales de las siguientes estructuras de datos no debe ser liberada por la propia rutina de cerrar:

a) Los iorb's que tengan el id_io igual a alguno de los id_io's asociados al canal n.

b) Las entradas a la cola io_finalizadas que tengan el id_io igual a alguno de los id_io's asociados al canal n.

c) De la tabla de canales, todas las entradas de la cola de id_io correspondientes al canal n.

d) La entrada de la tabla de canales asociada al canal n.

3.- Volem oferir a Onion la possibilitat de comunicar-se dos ordinadors amb aquest sistema operatiu a través de la línia sèrie. Aquest dispositiu línia sèrie rebrà el nom de TERMINAL i volem que es pugui manipular amb les crides habituals d´Entrada/Sortida vistes a classe. Considerarem que només un procés per ordinador pugui estar accedint a aquest dispositiu i només es podrà accedir per un canal.

Responeu a les següents qüestions:

a) Quins camps addicionals es necessiten al DD_TERM i a l´IORB_TERM? Es necessitarà tenir un camp de propietari? Com es controlarà que només s´obri el TERMINAL per un canal? Què es fa amb aquells processos que vulguin obrir el dispositiu i el trobin ocupat?

b) Quants DD_TERM es necessiten?

c) Quants gestors es necessiten? Un per lectura i un altre per escritura? Canvia l´estructura DD_TERM la teva resposta?

d) Dibuixa el diagrama de funcionament. A quina capa situarias el buffer? Es necessitaria algun procés addicional (per omplir el buffer)?

e) Si ara volem que més d´un procés es pugui comunicar i oferim molts dispositius lògics terminal (TERM00-TERM99) anyadint al missatge una capçalera on digui a quin terminal va dirigit, canviaria el disseny de les estructures DD_TERM, IORB_TERM, el número de gestors i el número de DD_TERM necessaris? Com?

4.- Es vol implementar a 'Onion la sincronització mitjançant semàfors. Per a això es defineixen els dispositius SEM00 a SEM99, que es poden manipular amb les crides d'entrada/sortida habituals: *obrir, tancar, llegir, escriure, esperar.* El comportament és el següent:

err = obrir (canal, SEMxx, mod, prot) :Obre un dispositiu de tipus SEM, creant-lo si és necessari, i l'associa al canal corresponent.

err = tancar (canal):Indica que no es seguirà utilitzant SEMxx associat al canal.

id_io = escriure (canal, buff, lon)

Si el buffer conté l'string "**INIT n**", on *n* es un decimal >=0, llavors:

- Si el comptador del semàfor no estava inicialitzat, s'inicialitza al valor *n*.
  La crida *esperar* retornara el valor 0 (acabament correcte).
- Si estava inicialitzat, *esperar* retornarà un codi d'error (-2).

Si el buffer conté l'string "**SIGNAL**", llavors:

- Si no estava inicialitzat, *esperar* retornarà un codi d'error (-1).
- Si estava inicialitzat, es farà l'operació de signal.
  La crida *esperar* retornarà el valor 0 (acabament correcte).

id_io = llegir (canal, buff, lon)

Implementa el **"WAIT"**. El parametre *buff* no té efecte.

- Si el semàfor està inicialitzat, es farà l'operació de wait.
  La crida *esperar* retornarà el valor 0 (acabament correcte).
- Si no està inicialitzat, *esperar* retornarà un codi d'error (-1).

res = esperar (id_io) :Bloquejarà el procés que la invoqui fins que l'operació indicada al parametre *id_io* hagi finalitzat.

Si l'operació és un *wait*, aquesta no finalitzarà fins que decrementi en una unitat el comptador del semàfor o un signal desperti al procés.

Les estructures de dades relacionades amb aquest nou dispositiu són:

```
struct DD_sem {
  struct dd_comu common;
  int n_opens;
  boolean_t init;
  int counter;
  struct cua blocked;
  struct cua *q_iorb;
  struct cua *q_iofin;}
```

```
struct iorb_sem {
        struct item element;
        char *user_buffer;
         int  *user_bytes;
        struct DD_sem *dd;
        int operacio:
        int id_io;}
```

1- En el cas d'una crida a "escriure", cal llegir el buffer d'usuari per tal d'identificar la comanda ("INIT" o "SIGNAL"). Aquesta informació s'obtindrà:

a) A la rutina "escriure independent de dispositiu".

b) A la rutina "llegir dependent de dispositiu".

c) Al gestor.

2- Quan fem una operació "obrir" sobre un semàfor que ja estava obert,

a) Crearem un "DD_sem" nou.

b) Utilitzarem el mateix "DD_sem" utilitzat pels altres processos que tenen obert aquell semàfor.

c) Només hi haurà un "DD_sem" per a tots els semàfors oferts (00-99) pel sistema , per tant, sempre es reaprofitarà el "DD_sem".

3- Quan s'obre un semàfor, on es detecta que "SEMxx" correspon a un nom de dispositiu de tipus semàfor?

a) A la rutina obrir "independent de dispositiu".

b) A la rutina obrir "dependent de dispositiu".

c) Al gestor.

4- El gestor encarregat dels semàfors el posarieu

a) A nivell de sistema, perquè els semàfors són dispositius lògics i els podem tractar integrament a aquest nivell.

b) A nivell BFS, perquè encara que els SEMxx siguin dispositius lògics,el nivell BFS és el que s'encarrega de la gestió de tots els dispositius.

c) A nivel de nucli, per a que sigui més eficient.

5- A quines estructures de dades accedeix la rutina obrir independent de dispositiu?

a) Taula de canals, Descriptor de dispositiu, TFA.

b) Descriptor de dispositiu, taula de canals, llista de descriptors de semàfors usats.

c) Cua d'operacions I/O pendents, taula de canals, PCB.

6- El codi del gestor

a) Farà sempre crides a BFS per a gestionar els dispositius semàfor.

b) A vegades farà crides a BFS per a gestionar els dispositius semàfor.

c) Mai no farà crides a BFS per a gestionar els dispositius semàfor.

7- Si en fer una lectura d'un semàfor, el contador associat al SEMxx corresponent val zero, el proces s'hauria de bloquejar. Aixo vol dir que:

a) La crida llegir es bloquejarà i no retornarà a usuari fins que arribi el signal corresponent al SEMxx.

b) La crida llegir retornarà immediatament perquè es asincrona i el resultat serà un codi d'error indicant el que ha passat.

c) La crida llegir retornarà immediatament, i el resultat serà un id_io. Encara que la crida retorni, l'operació no haurà finalitzat fins que arribi el signal.

8- Indica si s'executarà la rutina proces() en el següent codi:

```
...
w1=escriure(SEM03, "INIT 0", 6);
r = llegir(SEM03, buffer, &n);
proces();
w2=escriure(SEM03, "SIGNAL", 6);
esperar(w1);
esperar(w2);
esperar(r);
```

a) No s'executarà mai.

b) S'executarà sempre.

c) Depen de si altres processos estan utilitzant el mateix semàfor

9- El codi del gestor té el seguent aspecte:

```
gestor_sem() {
    for(;;) {
        ...
            /* obtenir iorb */
        ...
        switch (determinar_operacio(iorb)) {
            case INIT: gestionar_init(iorb); break;
            case SIGNAL: gestionar_signal(iorb); break;
            case WAIT: gestionar_wait(iorb); break;
            case ERROR: ... }
        ...
```

Implementeu el codi de les rutines
gestionar_init(struct iorb_sem iorb);
gestionar_signal(struct iorb_sem iorb);
gestionar_wait(struct iorb_sem iorb);

5.- Examen Final Q2 02-03. Donat l'enunciat anterior amb la modificació de que un signal sobre un semàfor despertaria al proces més prioritari dels que hi haguessin bloquejats a aquell semàfor. Si sabem que volem implementar el sistema amb un únic gestor per tots els semàfors, es demana:

a) Quants descriptors de dispositiu seran necessaris? Quins camps específics seran necessaris? És necessari algun camp addicional als IORBs? En cas afirmatiu, quina rutina obté aquesta informació i d'on?

b) Escriu el codi del gestor del dispositiu.

c) Al haver-hi un únic gestor, aquest no es podrà bloquejar mai. Necessiteu un procés auxiliar? Per què? En cas afirmatiu, escriviu el codi.

d) Descriu **breument** que modificarieu per convertir la crida de llegir (el wait) en síncron.

6.-De cara a optimizar el retraso provocado en la entrega de prácticas, se pretende dotar a cada PC, que trabaja con el sistema operativo ONION, de cuatro impresoras.

Diseñad un mecanismo para que la salida por el dispositivo IMPRESORA: vaya dirigido a la que se encuentre libre, sin que se entere el usuario.

El dispositivo IMPRESORA: debe mantener todas las características modificables por el comando SET.

7.- En un sistema UNIX modificado no existen las llamadas al sistema fork ni exec. En lugar de ellas disponemos de la siguiente llamada:

```
crear_proces (nom, par1, par2, ..., (char*)0);
char * nom;
char * par1, *par2, ...;
```

En este sistema los canales del proceso padre no se heredan.

Por defecto, los canales 0, 1 y 2 del nuevo proceso están asociados a un terminal que viene determinado por el identificador del usuario (uid).

a) Cómo afecta esta modificación a la idea de filtros de UNIX?

b) Comentar esta modificación desde el punto de vista de la independencia de dispositivos.

8.- Queremos añadir a ONION un sistema de cuatro ventanas a la cónsola. Cada ventana representa un dispositivo de tipo CONSOLA: (pantalla y teclado) con sus características.

Suponemos que las ventanas se pueden identificar a nivel sistema por P1:, P2:, P3:, P4:.

Para saber en qué ventana está asociado el teclado en un momento determinado se dispone de las teclas especiales F1, F2, F3, y F4 respectivamente

a) Definir las estructuras de datos necesarias a nivel sistema y a nivel bfs para poder gestionar las ventanas.

b) Decir, si hace falta, los cambios necesarios en los parámetros de las llamadas entre niveles. Detallar los niveles y las llamadas.

c) Hacer un esbozo de la secuencia de llamadas que se generan a todos los niveles de ONION a la hora de hacer un obrir(...) y un escriure(...).

Describir para cada una de ellas la parte relevante del código.

9.- Examen Final Q1 00-01. Volem modificar les crides a sistema d'ONION `llegir` i `escriure` de tal forma que es pugui especificar un *timeout*. La nova sintaxi de les crides serà la següent:

```
llegir(canal, buffer, tamany, timeout)
escriure(canal, buffer, tamany, timeout)
```

El paràmetre timeout especifica un temps en segons. El seu significat és que si l'operació de lectura o d'escriptura no s'ha començat a tractar (és a dir que el gestor no ha agafat l'IORB) abans del nombre de segons especificat per timeout, l'operació no es farà i al IO_FIN es col.locarà l'error E_TIMEOUT. Si el timeout és 0, vol dir que no hi ha temps especificat i que per tant es comportarà com sempre (ignorant el timeout).

Es demana que responeu a les següents qüestions:

a) Quines modificacions s'han de fer als DDs? i als IORBs?

b) Quines parts del sistema cal modificar: la depenent, la independent o les dues? Raoneu la vostra resposta.

c) Si ens és igual que la crida a sistema esperar(id_io) s'esperi tot i que hagi passat el *timeout*, com haurem de modificar els gestors? És a dir, si ha passat el *timeout*, l'operació no es farà, però pot ser que passi un cert temps fins que s'encui el IO_FIN i per tant esperar pot quedar-se bloquejat fins que es detecti que s'ha passat el temps (cosa que pot passat molt després del *timeout*). Escriu el codi que s'ha d'afegir als gestors.

d) Si ara volem que esperar(id_io) no es bloquegi si ja s'ha acabat el *timeout*, com es pot fer? Dona la idea intuitiva en no més de 5 línies.

e) Com es pot calcular el temps per saber si ja ha caducat el *timeout*? N'hi ha prou amb els serveis que ens ofereix el nivell BFS o cal afegir alguna crida nova?

10.- Examen Final Q2 99-00. Volem afegir el dispositiu de *pipe* de UNIX. Per això, a més de les crides de fitxers tradicionals d´Onion (llegir, escriure, esperar, cancel·lar i tancar) hem d´afegir la de creació d´una *pipe* (`pipe`). com volem mantenir la semàntica de UNIX, no hi ha un nombre predefinit de pipes (és a dir, podem tenir tantes com vulguem).

```
int pipe(fdr, fdw)
  fdr i fdw són els canals de lectura i escriptura
```

Nota: Les pipes guarden la informació a memòria i no a disc.

a) Quins nivells (sistema, BFS, nucli) hauríem de modificar per oferir aquestes pipes. en cas de que calgui modificar la capa de sistema, hauríem de modificar la part independent del dispositiu, la depenent o les dues?

b) Quants DDs necessitarem? I aquests DD es crearan de forma estàtica (com els MBXs) o de forma dinàmica (com els fitxers)?

c) Quines de les següents opcions són possibles?

Tenir un sol gestor per a totes les pipes.

Tenir N gestors (on N és independent del nombre de pipes).

Tenir un gestor per pipe.

d) Trieu l´opció que us sembli millor i doneu el codi del gestor/gestors (en pseudocodi).

Només calen les idees més significatives.

e) Si volguéssim que les pipes fossin permanents (guarden el buffer en un fitxer normal), quines modificacions hauríem de fer sobre BFS? I sobre nucli?

f) Seria possible afegir una característica a les pipes per indicar que el buffer es guarda a memòria (tradicionals) o que es guarda a disc (com els mailboxes permanents)? A on s´hauria de guardar aquesta característica? Modifica això el nombre de gestors necessaris? I les crides a BFS?

11.- Volem oferir als usuaris d'Onion un servei de temporització per als seus processos. El servei s'oferirà a través d'un dispoitiu anomenat TIMER, que es podrà manipular amb les crides d'entrada/sortida habituals: obrir, tancar, llegir i escriure. El comportament d'aquestes crides es descriu a continuació:

err = obrir (canal,"TIMER", mod, prot): Obre el dispositiu de tipus TIMER, i l'associa al canal corresponent.
err = tancar (canal): Indica que no es seguirà utilitzant el dispositiu TIMER associat al canal.
id_io = escriure (canal, buff, lon): El buffer conté un enter amb el nombre de tics que ens volem retardar.
id_io = llegir (canal, buff, lon): No té cap efecte sobre el dispositiu TIMER. L'operació sempre és correcta.
res = esperar (id_io): Bloquejarà el procés que la invoqui fins que l'operació indicada al parametre *id_io* hagi finalitzat.

Diversos processos (o els mateixos processos en crides successives) han de poder utilitzar el dispositiu TIMER simultàniament. El temps d'espera ha de ser l'indicat a la crida al sistema 'escriure', independentment del nombre de processos que estiguin utilitzant el dispositiu TIMER en aquell moment. Es a dir, si dos processos criden a 'escriure' al mateix temps i demanen retards de 5 i 12 tics respectivament, el primer s'haurà de despertar al cap de 5 tics, i el segon 7 tics més tard. Es poden programar diversos retards a través del mateix canal sense cap problema: cada cop que es cridi a 'escriure' es programarà un nou retard sense invalidar els anteriors.

Les rutines de nivell bfs disponibles són les següents (considereu-les implementades):
tics = retardar_bfs(int ntics): bloqueja el procés que fa la crida durant 'ntics'. Retorna 0 si tot va bé o el nombre de tics que faltaven per a acabar el retard, en cas que la crida hagi estar cancel.lada. Només hi pot haver UN (1) procés bloquejat per retard. Si n'hi ha més d'un, el sistema fa un reset.
tics = cancelar_retard_bfs(): desperta el procés que estava bloquejat per retard, encara que no hagi finalitzat el temps de retard programat. Retorna el nombre de tics que li quedaven per esperar al procés (o 0 si no hi havia cap proés bloquejat).
tics = temps_sistema_bfs(): retorna el nombre de tics que han passat des del boot del sistema.

Contesteu BREUMENT les següents preguntes
a) Quants descriptors de dispositiu es necessiten per a gestionar el TIMER: un descriptor global per a tot el sistema? un descriptor nou cada cop que s'obri el dispositiu? o un descriptor per a cada procés que estigui utilitzant el TIMER? Raoneu la resposta.
b) Indiqueu quins seran els camps no comuns per al descriptor de dispositiu.
c) Indiqueu quins seran els camps de l'estructura IORB.
d) Quants gestors calen per a gestionar el dispositiu TIMER? Fa falta algun procés auxiliar?

e) Dibuixeu un diagrama del sistema de gestió del dispositiu, indicant les estructures de dades que cal utilitzar i els processos implicats. Indiqueu de forma clara quines estructures són pròpies d'aquest dispositiu específic.

f) Escriviu el pseudocodi del gestor que implementa la funcionalitat de 'escriure'

g) Escriviu el pseudocodi dels processos auxiliars que puguin caldre per a gestionar el dispositiu TIMER (si es que n'hi ha algun).

12.- Examen Final Q2 98-99. Volem utilitzar un PC amb Onion per a controlar un robot en un entorn industrial. El control es farà a través del dispositiu ROBOT implementat amb el mecanisme d'entrada/sortida vist a classe. El robot pot enviar informació a l'ordinador i rebre'n comandes. La funcionalitat de control i supervisió a nivell de BFS ja està implementada i consta de les rutines següents:

- *int llegir_robot_bfs(struct robot_status *dades)*: retorna tota la informació sobre l'estat del robot en el moment actual (posició, velocitat, estat, etc.) i la deixa dins l'estructura robot_status apuntada pel paràmetre dades. Aquesta rutina s'ha de poder executar en qualsevol moment (encara que ja hi hagi una altra comanda que estigui executant-se).
- *int comanda_robot_bfs(struct robot_command *dades)*: envia una comanda al robot. Aquesta rutina no retorna fins que la comanda no ha estat realitzada. No es pot executar una altra rutina comanda_robot_bfs fins que l'anterior no hagi finalitzat.
- *int aturar_robot_bfs()*: atura immediatament l'activitat del robot. Es pot enviar en qualsevol moment, encara que hi hagi una comanda en curs.

El treball des del nivell d'usuari es farà a través de l'interfície d'entrada/sortida asíncrona d'Onion. El nom del dispositiu serà ROBOT i les funcions seran les següents:

- *obrir(...)*: permetrà obrir el dispositiu i començar a treballar amb el robot. El procés que obri el dispositiu per a escriptura tindrà capacitat de control sobre el robot, mentre que els processos que obrin per a lectura només en podran consultar l'estat. Només un procés pot tenir obert simultàniament el dispositiu ROBOT en mode escriptura (als altres se'ls retornarà un error); aquest procés, però, tindrà permís per a obrir el ROBOT en mode escriptura a través de tants canals com li calgui.
- *llegir(...)*: s'utilitzarà per a obtenir informació de l'estat del robot. Es retornarà al buffer de l'usuari les dades retornades per llegir_robot_bfs (si el buffer és prou gran; si és més petit, es retornarà un error). Cal que qualsevol procés pugui fer una lectura i obtenir aquesta informació en qualsevol moment.
- *escriure(...)*: s'utilitzarà per a enviar comandes al robot. El contingut del buffer serà una estructura del tipus robot_command. Si el contingut del buffer és vàlid, s'enviarà al robot amb comanda_robot_bfs.
- *tancar(...)*: tancarà el canal especificat.
- *posicionar(...)*: no té cap efecte sobre aquest dispositiu.

Amb aquesta informació, es demana que contesteu BREUMENT i CLARAMENT les següents preguntes:

a) Quants descriptors de dispositiu utilitzaríeu? Un únic descriptor per al ROBOT, un descriptor per procés, un descriptor per a lectures i un altre per a escriptures, o un descriptor nou cada cop que s'obri el dispositiu?

b) Quins camps no comuns tindria el descriptor de dispositiu?

c) Quants gestors calen per a gestionar el dispositiu? Cal algun procés auxiliar? Justifiqueu la resposta.

d) Dibuixeu un diagrama del sistema d'entrada/sortida per a aquest dispositiu. Indiqueu clarament quins processos i quines estructures de dades (cues, descriptors, etc.) es veuen afectats

quan hi ha una lectura o una escriptura en aquest dispositiu.

e) Indiqueu el pseudocodi de la rutina dependent obrir_robot().

f) Indiqueu el pseudocodi del gestor/gestors/processos_auxiliars que calguin per a gestionar el dispositiu.

g) Suposeu que es vol implementar una nova funcionalitat: si el tamany de les dades de la crida escriure es 0 s'utilitza la comanda aturar_robot_bfs per a fer una parada d'emergència, encara que hi hagi una comanda en curs. Canvia això la resposta a la pregunta c)? Raoneu-lo.

13.- Examen Final Q2 00-01. En un sistema basat en ONION es connecta un nou disc, que anomenarem disc secundari (per a diferenciar-lo del disc original d'ONION, que anomenarem disc principal). El sistema escriu seqüencialment un missatge al disc secundari cada cop que finalitza una operació de lectura o escriptura sobre el disc principal.

Aquest missatge consta dels següents camps: el nom del fitxer obert, el tipus d'operació (lectura o escriptura), el valor del punter de lectura/escriptura abans de realitzar la operació, el tamany (nombre de bytes) de la operació, i el pid del procés que ha efectuat la operació. Per tal d'anar escrivint aquesta informació al disc secundari, es disposa d'un gestor a nivell sistema que executa les rutines de nivell BFS i nucli apropiades cada cop que s'efectua una operació sobre el disc principal.

a) Com s'ha d'indicar al gestor del disc secundari que s'ha efectuat una operació sobre el disc principal?

b) En funció del que heu contestat a l'apartat (a), a quin punt (rutina o procés) del subsistema d'entrada/sortida d'ONION hi haurà el codi que indica al gestor del disc secundari que s'ha efectuat una operació sobre el disc principal? Justifiqueu aquesta resposta.

c) Per tal d'indicar al gestor del disc secundari que s'ha efectuat una operació sobre el disc principal, cal conèixer la informació que composa el missatge: nom del fitxer, tipus d'operació, valor del punter, tamany i pid. De quines estructures de dades s'obté cada un d'aquests camps?

d) En quines funcions del subsistema d'entrada/sortida d'ONION s'actualitza la informació de les estructures de dades a les que heu fet referència a l'apartat (c)?

Al marge de com s'actualitza la informació del disc secundari, s'ofereix la possibilitat de que un procés usuari pugui consultar aquesta informació. Per això es defineix el dispositiu DISC2: el qual es pot manipular amb les crides d'entrada/sortida habituals.

Per poder consultar la informació, el dispositiu DISC2 ha d'haver estat obert prèviament. Aquest es pot obrir únicament per lectura. Un cop obert el dispositiu, cada operació de lectura retorna al buffer d'usuari un dels missatges (independentment del tamany especificat a la operació de lectura), començant pel principi i recorrent-los seqüencialment per cada nova operació de lectura. Diferents processos poden tenir obert aquest dispositiu, de forma que cada procés llegeix els missatges independentment dels altres processos. Les altres operacions tenen el comportament habitual (excepte l'escriptura que no es pot fer sobre aquest dispositiu desde un procés usuari).

e) Es pot usar el mateix gestor de disc secundari descrit anteriorment per a executar les lectures, o bé caldrà un o varis nous gestors? Raoneu la vostra resposta.

f) Quants descriptors de dispositiu caldran per a gestionar correctament el funcionament del DISC2? Quins camps específics haurà de tenir cada descriptor de dispositiu? Justifiqueu les vostres decisions.

14.- Examen Final Q2 01-02. Es vol afegir a ONION un nou dispositiu per mesurar la quantia de les plujes en una estació meteorològica. El dispositiu consisteix bàsicament en un pluviómetre que acumula l'aigua de les plujes durant un període de temps programat per l'usuari, anota la quantitat de pluja recollida en un *buffer* a nivell sistema, buida el pluviómetre, i torna a acumular plujes un altre cop.

A nivell lògic, el dispositiu pluviómetre es diu "PLUV:". Per tal de gestionar el dispositiu a nivell de sistema operatiu disposem de les següents operacions:

`void buidar_pluviometre_bfs();` Llença l'aigua recollida en el pluviómetre i es prepara per tornar a recollir plujes. Aquesta operació no es bloqueja mai, és a dir, retorna immediatament.

`int llegir_pluviometre_bfs();` Retorna la quantitat d'aigua recollida en el pluviómetre en el moment de fer la crida. Aquesta operació no es bloqueja mai,és a dir, retorna immediatament.

`void retardar_bfs(int nsecs);` Bloqueja el procés que la invoca durant la quantitat de segons especificada. Aquesta operació, per tant, si que bloqueja.

Les operacions a nivell d'usuari sobre aquest dispositiu són les habituals d'entrada/sortida en ONION, és a dir:

`int obrir(canal,disp,mode)` Indica al sistema que es vol usar el dispositiu i l'associa a un canal. Només un procés pot tenir obert el dispositiu en un moment donat. Si un altre procés demana obrir el dispositiu i aquest ja està ocupat es retorna un error.

`int llegir(canal,buf,lon)` Indica al sistema que l'usuari vol iniciar una operació de lectura sobre el dispositiu. Una lectura obté del dispositiu la mesura més antiga que encara no hagi estat llegida. Un cop llegit un valor, aquest és eliminat del *buffer* de sistema. Si no hi ha cap mesura pendent de ser llegida, l'operació es bloqueja fins que la nova mesura estigui llesta.

`int escriure(canal,buf,lon)` Indica al sistema que l'usuari vol iniciar una operació d'escriptura sobre el dispositiu. El contingut del *buffer* indica durant quin interval de temps s'ha d'acumular la pluja en la següent mesura. NOTA: l'operació d'escriure no afecta a l'operació que s'estigui realitzant en aquell moment, sols a la operació que s'iniciarà un cop acabada la lectura en curs. Si es fan dues escriptures abans de l'acabament de l'operació en curs, sols es tindrà en compte el valor de la darrera escriptura.

`int esperar(id_io)` Sincronitza el procés d'usuari amb la finalització de la operació d'entrada/sortida associada a id_io.

`int tancar(canal)` Espera a que acabin les operacions de lectura pendents, i tanca el dispositiu.

Responeu a les seguents preguntes:

a) Quants descriptors de dispositiu seran necessaris? Quins camps específics seran necessaris? Es necessitarà alguna cua auxiliar de IORBs?

b) Caldrà un proces auxiliar que es faci càrrec de gestionar la temporització del dispositiu a nivell de sistema operatiu (recollir pluja durant el temps programat, i acumular les lectures en un *buffer*). Escriviu el codi d'aquest procés.

c) Escriviu el codi del gestor del dispositiu.

15.- Examen Final Q1 02-03. Volem fer servir un ordinador amb Onion per controlar un braç mecànic. El control es farà a través del dispositiu BRAÇ: implementat amb el mecanisme d´entrada/sortida vist a classe. La interfície de comunicació amb el dispositiu està completament implementada a nivell bfs i per executar comandes al braç podeu fer servir:

```
void executar_comanda_bfs(char * buffer, long lon)
```

Executa la comanda indicada en el buffer i no retorna fins que la comanda hagi finalitzat.

Per evitar que un procés s´apropiï indefinidament del braç mecànic, el sistema permetrà que diferents processos puguin estar fent servint el dispositiu simultàniament, però per evitar que les comandes de diferents processos interfereixen entre elles s´agruparan per lots. Un lot es un conjunt de comandes que un procés vol que faci el braç sense interferències de cap altre procés. Una vegada el braç hagi finalitzat un lot de comandes, podrà iniciar un altre del mateix o d´un altre procés. Com els lots poden tenir des d´unes poques a moltes comandes, es va decidir emmagatzemar-les a disc i per això farem servir les crides de fitxers habituals a nivell bfs. Totes les comandes tenen la mateixa longitut (L bytes).

```
int crear_fitxer_bfs()
```

Crea un fitxer temporal obrint-lo per escritura on emmagatzemar les comandes d´un lot i retorna la posició de la Taula de Fitxers Oberts on s´ha ubicat aquest fitxer.

```
void destruir_fitxer_bfs(int id)
```

Tanca i destrueix el fitxer temporal que contenia les comandes i que ocupava la posició pasada com a paràmetre en la Taula de Fitxers Oberts.

```
void escriure_fitxer_bfs(int id, char* buffer, long l, int pos)
void llegir_fitxer_bfs(int id, char* buffer, long l, int pos)
```

Escriu o llegeix a la posició pos del fitxer temporal identificat per la posició a la Taula de Fitxers Oberts un cadena de caracters de longitut l.

A nivell d´usuari disposarem del conjunt de crides a sistema asíncrones vistes a classe tenint funcions especials les crides següents:

```
int llegir(canal, buffer, longitud)
```

No té cap funció associada.

```
int escriure(canal, buffer, longitud)
```

Buffer pot contenir “INICI LOT”, “FI LOT” o una comanda pel braç. La seqüència d´escritura és la següent: com els processos no poden enviar comandes al braç directament sinó que ho han de fer per lots, primer s´ha d´indicar que es comença la descripció d´un lot (INICI LOT), després s´envia una a una les comandes i finalment s´indica que el lot ja està llest per ser executat pel braç (FI LOT) quan estigui disponible. Un cop fet això, pot iniciar-se la descripció d´un altre lot. Qualsevol altre seqüència d´operacions retorna un codi d´error. Aquesta crida no es bloqueja mai.

S´ha pensat que la manera òptima d´implementar això es mitjançant **un únic gestor** i **un spooler**. Es demana:

a) Quants descriptors de dispositiu seran necessaris? Quins camps específics seran necessaris?

b) Escriviu el codi del gestor d’aquest dispositiu.

c) Escriviu el codi del spooler del dispositiu.

16 .- Examen Final Q1 03-04. El rellotge de la Puerta del Sol es composa d'una part mecànica i d'una altra d'informàtica. La mecànica s'encarrega de moure les agulles del rellotge quan la part informàtica genera els impulsos corresponents. És a dir, **les agulles no es mouen si la part informàtica no ho decideix així (la part mecànica no actua sobre les agulles independentment, només respon a estímuls generats per la part informàtica)**. La interacció entre la part mecànica i la informàtica es fa mitjançant un dispositiu anomenat 'SOL'. A través d'aquest dispositiu el sistema informàtic pot fer moure les agulles del rellotge i també pot consultar l'hora que està marcant el rellotge, és a dir, la posició de les seves agulles.

Es vol integrar a Onion el dispositiu de control del rellotge de la Puerta del Sol (SOL). Es pretén que el dispositiu funcioni de manera que enviï una senyal a la mecànica del rellotge a intèrvals d'un minut per a que les agulles del rellotge es moguin. Per altra banda, el dispositiu també permetrà a les aplicacions que funcionen sobre onion consultar l'hora actual que està mostrant el rellotge. També és necessari que es permeti efectuar una operació d'avançament o endarreriment de l'hora d'acord amb els canvis horaris que s'apliquen dos cops l'any. El rellotge marca hores i minuts, però no segons.

Per tal de gestionar el dispositiu mecànic, disposem de les següents operacions ja implementades:

void retardarbfs(int nsecs) Bloqueja el procés que la invoca durant la quantitat de segons indicada. És una operació bloquejant.

void incrementarminutsbfs(int nmin) Envia la senyal adequada a la mecànica del rellotge per tal que s'avanci l'hora en tants minuts com indica nmin. Aquesta operació és no bloquejant. (Nota: El valor de nmin sempre ha de ser superior a 0 i menor o igual que 661.

timet llegirhorabfs() Consulta a la mecànica del rellotge l'hora actual que marquen les agulles del rellotge. **Aquesta operació és bloquejant**, només retorna un valor en l'instant en què acaba l'execució de l'operació incrementarminutbfs (és a dir, quan es mouen les agulles del rellotge) i només retorna una vegada per cadascun dels moviments d'agulles que efectua la mecànica del rellotge.

Les operacions a nivell d'usuari sobre aquest dispositiu són les habituals d'entrada/sortida en Onion:

obrir(): Serveix per associar el dispositiu a un canal.

tancar(): Desassocia el canal del dispositiu.

esperar(idio): S'espera fins que l'operació d'entrada/sortida amb identificador idio finalitzi

llegir(canal, buf, lon): Retorna l'hora que marquen les agulles del rellotge en el moment del proper pas de minut. **Cada cop que les agulles del rellotge efectuen un moviment, es resolen i donen per finalitzades totes les operacions de lectura pendents**. El resultat de l'operació es deixa en el buffer.

escriure(canal, buf, lon): Aplica el canvi horari al rellotge. Alternativament ha de realitzar canvis d'avançament i endarreriment d'una hora en el rellotge. Els paràmetres s'ignoren. Aquesta operació s'ha de poder aplicar assíncronament en qualsevol moment del temps. **El canvi horari es farà efectiu en les agulles del rellotge en el precís moment en què es produeixi el proper canvi de minut**. (Nota: El primer cop que s'invoca avança una hora el rellotge, el segon cop l'endarrereix, el tercer la torna a avançar, i així successivament).

El sistema informàtic no ha de mantenir cap control de l'hora que marca el rellotge en cada moment, i sempre que se li demana l'hora (operació llegir) la consulta a la mecànica del rellotge (llegirhorabfs).

Com a simplificació a l'enunciat, es pot assumir que mai s'iniciaràn operacions de lectura ni d'escriptura en el mateix instant en què s'efectua qualsevol moviment de les agulles del rellotge. Tampoc s'iniciarà mai una operació d'escriptura si una altra escriptura està en curs.

a) Dibuixeu l'esquema de funcionament (3 nivells d'Onion i nivell d'usuari) per a aquest dispositiu utilitzant el nombre de gestors i/o processos auxiliars que us siguin necessaris.

b) Escriviu el pseudo-codi de tots els gestors i/o processos auxiliars que utilitzeu

c) Quants descriptors de dispositiu fareu servir i quins camps (no genèrics) seran necessaris en cadascun d'ells?