Sistemas Operativos Ingeniería Informática. Segundo Curso.

SOLUCIÓN Examen Final. Mayo de 2009.

No olvide poner en la hoja su nombre, apellidos, asignatura, titulación y fecha.

Además, indique claramente en cada hoja una de las dos opciones siguientes:

PARCIAL: si ha aprobado el parcial

FINAL: si hace todo el examen porque no aprobó el parcial o quiere mejorar nota.

Duración prevista. FINAL: 3 horas. PARCIAL: 1,45 horas.

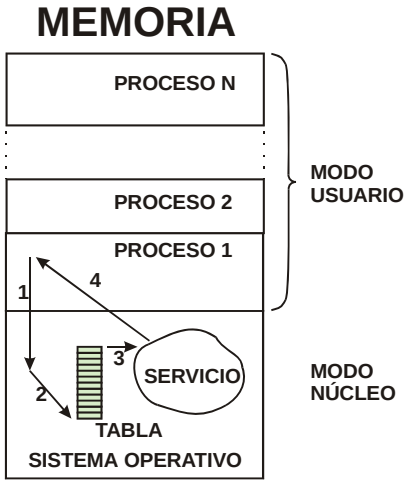
TEORÍA (3 puntos)

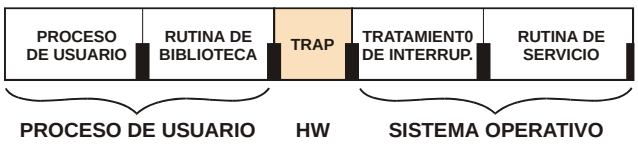
Responda a las preguntas cortas siguientes. Extensión máxima de cada pregunta: media página. Alumnos con el parcial aprobado, realizar solo las preguntas 3 y 4.

Pregunta 1. ¿Cómo se solicita una llamada al sistema operativo? Dibuje un esquema con los pasos.

Solución:

Se solicita mediante un mecanismo de interrupciones. Cuando es un proceso en ejecución el que la solicita, éste utiliza una instrucción TRAP que genera una interrupción, se consulta la RTI y se buscan en la memoria los argumentos de las llamadas.





Cuando se programa en un lenguaje de alto nivel, la solicitud de servicios al sistema operativo se hace mediante una llamada a una función determinada, que se encarga de generar la llamada al sistema y el trap correspondiente.

Pregunta 2 ¿Cuál de las siguientes políticas de planificación es más adecuada para un sistema de tiempo compartido? Primero el trabajo más corto. *Round-Robin*. Prioridades. FIFO.

Solución:

Un sistema de **tiempo compartido** permite ejecutar varios procesos de distintos usuarios en el computador de forma concurrente, con lo que los usuarios tienen la sensación de que tienen todo el computador para cada uno de ellos. Para poder implementar estos sistemas de forma eficiente es imprescindible tener un sistema multiproceso y un planificador que permita cambiar de un proceso a otro según los criterios de la política exigida. Ahora bien, los criterios de la política de planificación influyen mucho sobre el comportamiento del computador de cada a los usuarios, por lo que es importante decidir qué política se debe usar en cada caso.

- La política del **Primero Trabajo Más Corto** consiste en seleccionar para ejecución el proceso listo con tiempo de ejecución más corto, por lo que exige conocer a priori el tiempo de ejecución de los procesos. Este algoritmo no plantea expulsión: el proceso sigue ejecutándose mientras lo desee. Además tiene puede tener problemas de inanición de procesos. No es adecuado para tiempo compartido.
- Un algoritmo de planificación **FIFO** ejecuta procesos según una política "primero en entrar primero en salir": un proceso pasa al final de la cola cuando hace una llamada al sistema que lo bloquea y si esto no ocurre el proceso se ejecutará de forma indefinida hasta que acabe. Este algoritmo es inadecuado para tiempo compartido porque no hay ninguna seguridad en cuanto al tiempo de respuesta a los usuarios, que puede ser muy lento.
- La política de **Round Robin** realiza un reparto equitativo del procesador dejando que los procesos ejecuten durante las mismas unidades de tiempo (rodaja). Con este mecanismo se encarga de repartir el tiempo de UCP entre los distintos procesos, asignando de forma rotatoria intervalos de tiempo de la UCP (slot) a cada uno de ellos. Esto algoritmo es especialmente adecuado para los sistemas de tiempo compartido por que se basa en el concepto de rodaja de tiempo (slot) y reparte su atención entre todos los procesos, lo que al final viene a significar entre todos los usuarios. Los procesos están organizados en forma de cola circular y cuando han consumido su rodaja de tiempo son expulsados y pasan a ocupar el último lugar en la cola, con lo que se ejecutan otro proceso. Con este mecanismo, los usuarios tienen la sensación de avance global y continuado, lo que no está garantizado con los otros algoritmos descritos.
- Con la política de **Prioridades** se selecciona para ejecutar el proceso en estado listo con mayor prioridad. Se suele asociar a mecanismos de expulsión para que un proceso abandone el procesador cuando pasa a listo un proceso de mayor prioridad. Con esta política es necesario utilizar otros algoritmos para decidir que proceso de cada cola de prioridad se elige: Round Robin en el sistema interactivo y FIFO en el sistema batch. Además tiene puede tener problemas de inanición de procesos. No es adecuado para tiempo compartido si no se asocian políticas de rodaja de tiempo a las colas de prioridad y no se evitan los posibles problemas de inanición.

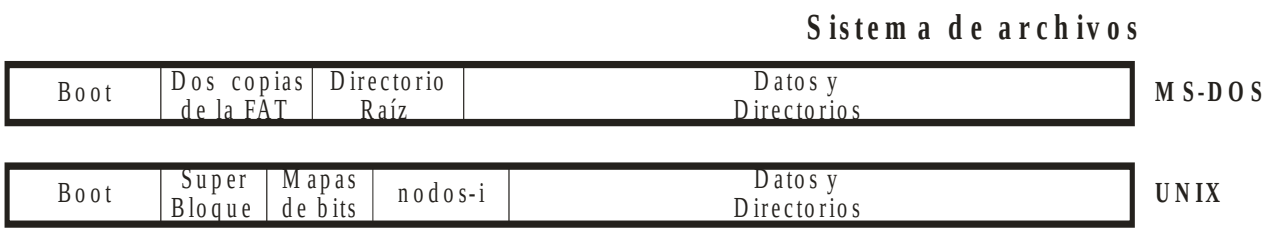
Pregunta 3. Un semáforo binario es un semáforo cuyo valor solo puede ser 0 o –1. Indique cómo puede implementarse un semáforo general utilizando semáforos binarios.

Solución:

```
pthread_mutex_t mutex;  /* mutex de acceso al contador compartido */
pthread_cond_t  lleno;  /* controla el llenado contador */
sem_t  sem;
pthread_mutex_init(&mutex, NULL);
pthread_cond_init(&lleno, NULL);

Init (sem_t  * sem ) {
    pthread_mutex_lock(&mutex);  /* acceso al recurso */
    sem = 50;  // Cargar valor inicial semáforo; puede ser otro valor
    pthread_mutex_unlock(&mutex);
}

wait (sem_t  * sem) {
    pthread_mutex_lock(&mutex);  /* acceso al recurso */
    while (*sem < 0)
           pthread_cond_wait(&lleno, &mutex);
    *sem = *sem -1;  // se puede pasar. Ocupar 1 del semáforo;
    pthread_mutex_unlock(&mutex);
}

Signal  (sem_t  * sem) {
    pthread_mutex_lock(&mutex);  /* acceso al recurso */
    *sem = *sem + 1;  // Liberar 1 del semáforo;
    pthread_cond_signal(&lleno); // indicar libertad
    pthread_mutex_unlock(&mutex);
}
```

Pregunta 4. Dibuje un esquema de un sistema de ficheros UNIX y un FAT32. ¿Qué problemas tiene el FAT si el disco y los archivos son muy grandes?

Solución:



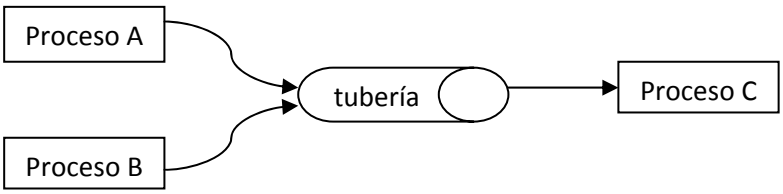
El esquema de FAT con punteros de 32 bits, solo puede direccionar 2^^32, es decir un máximo de 4 GBytes por archivo. Si se usan archivos muy grandes, el tiempo de acceso a los bloques de los archivos es muy alto, porque hay que recorrer muchos enlaces de la FAT.

En cuanto al disco, ocurre igual, la FAT solo podría tener como mucho 4 Gbloques. Si el disco es grande, la FAT ocuparía muchísimo y la copia no cabría en memoria, por lo que al acceder a archivos, habría mucha entrada/salida a disco debida al recorrido de la FAT.

PROBLEMAS 5 y 6. (Alumnos con parcial aprobado → sólo los problemas 7 y 8)
(Final: 3.5 puntos)

Problema 5. Escribir una función en C sobre UNIX que cree tres procesos comunicados mediante una tubería, de manera que dos de ellos tengan la salida estándar asociada a la tubería y el otro la entrada estándar.

Argumentos: nombres de programa que deberán ejecutar los tres procesos hijos.



Solución:

```
#include <stdio.h>

int main(void)
{
      int tuberia[2];
      int pid1, pid2;

      /* el proceso padre, que crea el pipe, será el proceso p1 */

      if (pipe(tuberia) < 0) {
            perror("No se puede crear la tubería");
            eixt(0);
      }
     /* se crea el proceso p2 */
     switch ((pid1=fork()) {
           case -1:
                 perror("Error al crear el proceso");
                 /* se cierra el pipe */
                 close(tuberia[0]);
                 close(tuberia[1]);
                 exit(0);
           case 0: /* proceso hijo, proceso P2 */
                 /* cierra el descriptor de lectura del pipe */
                 close(tubería[0]);

                 /* en esta sección de código el proceso P2 */
                 /* escribiría en la tubería */
                 /* utilizando el descriptor tubería[1] */

                 break;
           default:
                 /* el proceso padre crea ahora el proceso P3 */
                 switch((pid2 = fork()) {
                       case -1:
                             perror("Error al crear el proceso ")
                             close(tuberia[1]);
                             close(tuberia[1]);

                             /* se mata al proceso anterior */
                             kill(pid1, SIGKILL);
                             exit(0);
                       case 0:
                             /* el proceso hijo, el proceso P3, *
                             /* lee de la tubería */
                             /* cierra el descriptor de escritura *
                             close(tuberia[1]));

                             /* en esta sección de código el pro-
                             ceso P3 */
                             /* lee de la tubería */
                             /* utilizando el descriptor tube-
                             ría[0] */
                             break;

                       default:
                             /* el proceso padre, proceso P1, */
                             /* escribe en la tubería */
                             / * cierra el descriptor de lectura *
                             close(tuberia[0]);

                             /* en esta sección de código el pro-
                             ceso */
                             /* P3 lee de la tubería */
                             /* utilizando el descriptor tube-
                             ría[0] */
                 }
           }
}
```

Problema 6. Se tienen los siguientes trabajos a ejecutar:

Proceso	Unidades de tiempo	Prioridad
1	8	2
2	5	4
3	2	2
4	7	3

Los trabajos llegan al mismo tiempo en el orden 1, 2, 3 y 4 y la prioridad más alta es la de valor 1, se pide:

a) Escribir un diagrama que ilustre la ejecución de estos trabajos usando:

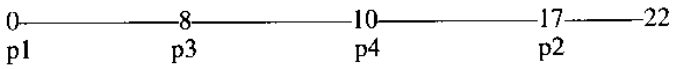
1. Planificación de prioridades no expulsiva
2. Planificación cíclica con una rodaja de tiempo de 2
3. FIFO

b) Indicar cuál es el algoritmo de planificación con menor tiempo medio de espera.

Solución:

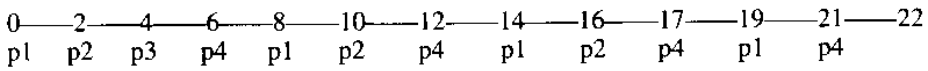
a) Diagramas de GANTT.

1. Planificación con prioridades no expulsiva.



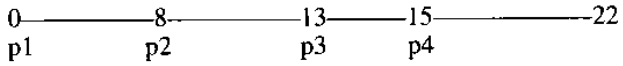
Tiempo medio espera = (0+8+10+17)/4 = 8 u.t.
Tiempo medio respuesta = (8+10+17+22)/4 = 14,25 u.t.

2. Planificación con Round Robin (rodaja = 2)



Tiempo medio espera = (13+12+4+15)/4=11 u.t.
Tiempo medio respuesta = (21+17+6+22)/4=16,5 u.t.

3. Planificación FCFS



Tiempo medio espera = (0+8+13+15) / 4 = 9 u.t.
Tiempo medio respuesta = (8+13+15+22) /4 = 14,5 u.t.

b) Se observa que el caso con menor tiempo de espera es el de prioridades no expulsiva. Este tiempo resulta parecido al del FIFO. En realidad el ejemplo es muy sencillo y limitado para sacar ninguna conclusión general válida, puesto que sólo considera cuatro procesos que aparecen todos en el mismo instante.

PROBLEMAS 7 y 8
(Final: 3.5 puntos)
(Parcial: 7 puntos)

Problema 7. Un sistema de ficheros tiene las siguientes características:
1) Tamaño de bloque de 4KBytes y direcciones de bloques de 4 bytes.

2) Al arrancar el sistema las estructuras de i-nodos se traen del disco duro a la memoria en una tabla que sirve también como tabla de ficheros abiertos. Los tipos de ficheros son, D directorio, F archivo y S enlace simbólico.

struct inodo [16];

Inodo: 0 Nombre: "/" Tipo: 'D' Enlaces: 1 Bloque 0: 100 Bloque ind doble: - sem_t lectores sem_t escritores	Inodo: Nombre: Tipo: Enlaces: Bloque 0: Bloque ind doble: - sem_t lectores sem_t escritores	Inodo: Nombre: Tipo: Enlaces: Bloque 0: Bloque ind doble: - sem_t lectores sem_t escritores	Inodo: Nombre: Tipo: Enlaces: Bloque 0: Bloque ind doble: - sem_t lectores sem_t escritores	Inodo: Nombre: Tipo: Enlaces: Bloque 0: Bloque ind doble: - sem_t lectores sem_t escritores

Bloque 100 Bloque 101 Bloque 102 Bloque 103 Bloque 104 Bloque 105 Bloque 106 Bloque 107

. 0 .. 0							

3) En los i-nodos se utiliza solamente un puntero directo y un puntero indirecto doble.

4) Un fichero se puede abrir de sólo escritura (O_WRONLY) sólo por un proceso a la vez y de sólo lectura (O_RDONLY) por un número arbitrario de procesos. Las operaciones de apertura del fichero para escritura tienen prioridad sobre los de lectura.

Responda a las siguientes preguntas:

a) ¿Cuál es el tamaño máximo de un fichero en el sistema de fichero descrito anteriormente?

b) Asumiendo que el sistema de fichero está inicialmente vacío, se pide dibujar la imagen del sistema de ficheros después de la ejecución de la siguiente secuencia de instrucciones:

mkdir ("/a");

mkdir ("/a/b");

creat ("/a/b/c",0666);

symlink ("/a/b/c", "/a/d");

link ("/a/b", "/a/e");

c) ¿Cuántos accesos al disco son necesarios para ejecutar la siguiente secuencia de código:

fd = creat (“/fich”);

fd = write(fd, buf, 8192);

d) Se pide implementar las funciones open (abrir fichero) y close (cerrar fichero) según la descripción del punto 4, y utilizando los semáforos declarados en los i-nodos.

int open(char nombre_fichero, int modo)

//donde modo puede ser “O_WRONLY” o “O_RDONLY”

int close(int fd)

La solución puede utilizar la función int iname (char *nombre_fichero) que recibe en nombre del fichero y devuelve la posición en la tabla del i-nodo asociado al fichero.

Solución:

a) Como no se indica el tamaño del disco duro, el tamaño máximo de un fichero será su tamaño teórico, que vendrá dado por el número de bloques que se pueden reservar en un i-nodo para un fichero:
   - *Puntero directo → 1 bloque de datos*
   - *Puntero indirecto doble → (nº direcciones en un bloque * nº direcciones en un bloque = (4096 / 4) * (4096 / 4)) bloques de datos, donde:*
     - *4096 = 4 Kbytes = tamaño de bloque*
     - *4 = 4 bytes = tamaño de la dirección de un bloque*

   ***TAMAÑO EN BYTES* = 4 KB + (1024 x 1024 x 4 KB) = 4 KB + 4GB**

b)

Inodo: 0 Nombre: “/” Tipo: ‘D’ Enlaces: 1 Bloque 0: 100 Bloque ind doble: -	Inodo: 1 Nombre: a Tipo: ‘D’ Enlaces: *5* Bloque 0: 101 Bloque ind doble: -	Inodo: 2 Nombre: b Tipo: ‘D’ Enlaces: *4* Bloque 0: 102 Bloque ind doble: -	Inodo: 3 Nombre: c Tipo: “F” Enlaces: 1 Bloque 0: - Bloque ind doble: -	Inodo: 4 Nombre: d Tipo: “S” Enlaces: 1 Bloque 0: 103 Bloque ind doble: -

Bloque 100	Bloque 101	Bloque 102	Bloque 103	Bloque 104	Bloque 105	Bloque 106	Bloque 107
. 0 .. 0 a 1	. 1 .. 0 b 2 d 4 e 2	. 2 .. 1 c 3	/a/b/c				

c) Dado que la tabla de nodos-i se encuentra en memoria, se necesita acceso sólo a los bloques de datos y de punteros indirectos:

- *creat: 1 acceso:*
  - *Al bloque de datos del directorio raíz.*
- *write: 4 accesos:*
  - *Al bloque de datos apuntado por el puntero directo del i-nodo*
  - *Al primer bloque de punteros indirectos dobles*
  - *Al primer bloque de punteros indirectos simples*
  - *Al bloque de datos apuntado a través del puntero indirecto doble*

TOTAL ACCESOS = 5 accesos

d)

```
//controla acceso atómico a la gestión de apertura como escritura
sem_mutexEscritores = 1;

//controla acceso atómico a la gestión de apertura como lectura
sem_mutexLectores = 1;

//controla que no se pueda abrir el fichero como escritura, si ya fue abierto como lectura. Yque no pueda ser abierto nuevamente como escritura.
sem_mutex = 1;

//controla que no se pueda abrir el fichero como lectura, si ya fue abierto como escritura
sem_lectores = 1;

num_lectores = 0; num_escritores = 0;
```

```
int open(char *name, int modo) {
   int fd = iname(name);
   nodoi *n = table_inodo[fd];
   if (n->modo == O_WRONLY) {
       wait(&sem_escritores);
       num_escritores++;
       if (num_escritores == 1){
             wait(&lectores);
       }
       signal(&sem_escritores);
       wait(&sem_mutex);
   else {
       wait(&lectores);
       wait(&sem_lectores);
       num_lectores++;
       if(num_lectores == 1){
             wait(&sem_mutex);
       }
       signal(&sem_lectores);
       signal(&lectores);
   }

   return i;

}
```

```
int close(int fd){
   nodoi *n = table_inodo[fd];
   if (n->modo == O_WRONLY) {
       wait(&sem_escritores);
       num_escritores--;
       if (num_escritores == 0){
             signal(&lectores);
       }
       signal(&sem_escritores);
       signal(&sem_mutex);
   }
   else {
       wait(&sem_lectores);
       num_lectores--;
       if(num_lectores == 0){
             signal(&sem_mutex);
       }
       signal(&sem_lectores);
   }
   return 0;
}
```

Problema 8. Un aparcamiento tiene varias puertas por las que pueden entrar y salir coches. La capacidad máxima es de 250 coches. Para entrar un coche deben existir plazas libres en el garaje (coches < 250). Cuando un coche entra, se decrementa la capacidad y cuando un coche sale se incrementa la capacidad. Si la capacidad llega a cero, los coches no pueden entrar y deben esperar en la barrera.
Se pide implementar usando semáforos las operaciones **entrar** y **salir**, de forma que no haya problemas de concurrencia en el control de capacidad.

Solución:

```
//variables globales

int capacidad=250;
sem_t semaforo;
sem_t semaforoEntrar;

int main()
{
      sem_init(&semaforo,0,1); //garantiza el acceso atómico a
      variable capacidad
      sem_init(&semaforoEntrar,0,250);
      ......
}

void Entrar()
{
      sem_wait(&semaforoEntrar);
      sem_wait(&semaforo);
      capacidad--;
      sem_post(&semaforo);
}

void Salir()
{
      sem_wait(&semaforo);
      capacidad++;
      sem_post(&semaforoEntrar);
      sem_post(&semaforo);
}
```