Ing. Tec. Informática de Gestión  SS.OO. Examen de junio de 2006

## Ejercicio 1. (2,5 puntos).

a) (2 puntos) Codificar un programa en lenguaje C que genere dos procesos. El primer proceso lee números enteros del teclado y los envía a un pipe para que los lea el otro proceso. El segundo proceso recibirá los números del pipe sumándolos. Cuando el primer proceso lea el número 0 de teclado enviará una señal SIGALRM al proceso hijo para mostrar el resultado de la suma y terminar.

b) (0,5 puntos) ¿Podría el primer proceso enviar la señal SIGKILL para indicar al segundo proceso que debe mostrar el resultado de la suma y terminar? Razona la respuesta.

## Ejercicio 2 (2,5 puntos).

a) (1,25 puntos) Considérese la siguiente cadena de referencias a página:

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

Rellenar una tabla como la siguiente, indicando cuántos fallos de página ocurrirían con los siguientes algoritmos de reemplazo LRU, Óptimo y FIFO, suponiendo que el sistema tiene 1 marco de memoria , suponiendo que tiene 2, ... , suponiendo que tiene 7 marcos:

| Número de Marcos | LRU | FIFO | OPTIMO |
|---|---|---|---|
| 1 | | | |
| 2 | | | |
| 3 | | | |
| 4 | | | |
| 5 | | | |
| 6 | | | |
| 7 | | | |

(Recordar que todos los marcos están inicialmente vacíos por lo que siempre existe una fallo inicial de página en cada marco).

b) (1,25 puntos) Considérese un sistema de paginación por demanda donde el grado de multiprogramación está fijado a 4. El sistema ha sido recientemente analizado para determinar el porcentaje de utilización de CPU y la paginación a disco. Suponiendo los siguientes resultados decir que está pasando en cada uno de los casos: ¿Es útil aumentar el grado de multiprogramación? ¿Está ayudando la paginación? ¿Cómo mejorar el rendimiento?

a. Utilización de CPU 17 % y utilización de disco de 97 %
b. Utilización de CPU 87 % y utilización de disco de 3 %
c. Utilización de CPU 13 % y utilización de disco de 3 %

## Ejercicio 3 ( puntos).

a) (1 punto). Supóngase el siguiente código:

```
int fd1;
int fd2;
int pid;
fd1 = open ("/usr/libc.c", O_RDONLY);
pid =  fork ();
switch (pid){
   case -1:
      perror(" error durante la creación de proceso");
      exit(-1);
      break;
   case 0:
      lseek(fd1,805,SEEK_SET); /*SEEK_SET indica desplaz. desde el comienzo*/
      break;
   default:
      fd2 = fd1;
      lseek (fd2,999,SEEK_SET);
      break;
}
```

Indicar a que posición queda apuntando cada uno de los descriptores tanto del padre como del hijo. Razonar si es posible conocer donde quedan apuntando los descriptores fd1 y fd2 al finalizar la ejecución del programa y si esta ejecución produce un resultado determinista.

b) (1,5 puntos) Dado un sistema de archivos tipo UNIX donde el tamaño del bloque lógico es de 1 KB y el tamaño de las direcciones a bloques de disco es de 32 bits. ¿Cuál será el número de accesos a disco necesario para abrir un archivo que se encuentra en el directorio de trabajo y acceder a los bytes 67.382.000 y al 67.392.250 de forma consecutiva.
Se debe suponer que: durante estas operaciones ningún otro proceso acceder al sistema de archivos ; que actualmente ningún proceso tiene abierto este archivo; y que el tamaño de la cache del sistema de archivos es de 10 bloques y se utiliza un algoritmo de reemplazo LRU.

## Ejercicio 4. (2,5 puntos).

Un famoso firma autógrafos en una tienda. El famoso solo puede firmar un autógrafo a la vez. La sala donde se firma tiene un aforo limitado de 20 plazas. El famoso dice que solo saldrá a firmar autógrafos si en la sala hay más de 5 personas. Si no hay al menos 5 personas en la sala, dormirá hasta que las haya (en el momento que haya 4 personas o menos se pondrá a dormir ). Las personas que quieran firmar y no puedan entrar a la sala por rebasar el aforo permitido se irán sin poder recibir el autógrafo, las que reciban el autógrafo abandonarán la sala.

El famoso representa a un proceso ligero de un tipo que permanece siempre en el sistema y ejecuta la función famoso. Las personas representan a procesos ligeros que ejecutan la función fan.

```
void famoso()
{
   while(1)
   {
         //Código del proceso famoso
   }
}
```

```
void fan()
{
   // Código del fan
}
```

Dadas las siguientes definiciones compartidas por todos los procesos:

```
#define AFORO_MAX 20


int ocupacion=0;     // Almacena la ocupación de la sala
pthread_mutex_t m;      //Mutex para región crítica

pthread_cond_t famoso_durmiendo;
pthread_cont_t autografo;

void Firmar();    //Función a la que debe llamar el famoso
```

famoso_durmiendo es una variable condicional para que el famoso espere dormido hasta que entren 5 personas a la sala
autografo es una variable condicional para que las personas esperen hasta haber recibido su autógrafo.
Codificar las funciones famoso y fan utilizando el mutex y las variables condicionales dadas.

NOTA: No hay que inicializar los mutex ni las variables condicionales, se suponen ya inicializadas.

## Ejercicio 1. Solución

### a)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <wait.h>

int total=0;

void ImprimirResultado()
{
    printf("El total de los números es %d\n",total);
    exit(0);
}

int main()
{
    int fd[2];  //Descriptores para el pipe
    int pid;
    int num;
    int estado;
    struct sigaction a;

    pipe(fd);

    pid=fork();

    if(pid!=0)  // Proceso padre
    {
          scanf("%d",&num);
          while(num!=0)
          {
                write(fd[1],&num,sizeof(num));
                scanf("%d",&num);
          }
         kill(pid,SIGALRM);
         wait(&estado);
    }
    else
    {
          a.sa_handler=ImprimirResultado;
          a.sa_flags=0;
          sigaction(SIGALRM,&a,NULL);
          while(1)
          {
             read(fd[0],&num,sizeof(num));
             total+=num;
          }
    }

    return 0;
}
```

### b)

Si el primer proceso envía SIGKILL al segundo proceso, el segundo proceso terminaría inmediatamente sin dar tiempo a que se muestre el resultado de la suma por pantalla. Además, por restricciones del sistema de señales, el segundo proceso no se puede armar para recibir la señal SIGKILL y por tanto tampoco podría cambiar el comportamiento por defecto. Por tanto, la respuesta es NO PODRÍA.

## Ejercicio 2. Solución:

1. Solución en la siguiente tabla:

| Numero de Marcos | LRU | FIFO | OPTIMO |
|---|---|---|---|
| 1 | 20 | 20 | 20 |
| 2 | 18 | 18 | 15 |
| 3 | 15 | 16 | 11 |
| 4 | 10 | 14 | 8 |
| 5 | 8 | 10 | 7 |
| 6 | 7 | 10 | 7 |
| 7 | 7 | 7 | 7 |

2.

a. Está ocurriendo el fenómeno de hiperpaginación.
b. El % de utilización de CPU es lo suficientemente alto como para dejar las cosas actuar sola e incrementar el grado de multiprogramación.
c. Incrementar el grado de multiprogramación.

## Ejercicio 3. Solución:

1. Cuando un proceso crea un proceso hijo utilizando la llamada al sistema fork(), el nuevo proceso hijo mantiene abiertos todos los archivos abiertos en el proceso padre. Además, ambos procesos comparten el puntero de la posición, que se almacenan en UNIX en la tabla intermedia situada entre las tablas de descriptores abiertos y la tabla de nodos-i. De acuerdo a esto el descriptor fd1 (y a su vez fd2 pues es el mismo) estarán en la posición 805 ó 999 dependiendo de si el padre ó el hijo ejecutaron antes si código (esta ejecución es indeterminista).

2. En primer lugar hay que indicar que cada bloque de índice permite guardar 1KB /32 bits = 256 bloques. Los bytes a leer se encuentra en los bloques 65803 y 65813 (67.382. 000/1024 = 65802,734 y 67.392.240/1024 = 65812,744). Para direccional estos bloques necesitaremos los punteros de triple indirección del nodo-i (con los de doble indirección se llega al bloque 10+256+256*256 = 65802).
   Los accesos requeridos son los siguientes:

   - Un acceso para acceder al bloque del directorio de trabajo donde encontrar el número-i de nuestro archivo (suponemos que la entrada al archivo s encuentra en el primer bloque del directorio).
   - Un acceso para acceder al nodo-i del archivo.
   - Tres accesos para acceder a los tres niveles de indirección
   - Un acceso para acceder al bloque donde se encuentre el primer byte.
   - Un acceso para acceder al bloque donde se encuentra el segundo byte (los bloques de triple indirección de este byte son los mismos que los del primer byte y por tanto se encuentra en la cache).

Ejercicio 4. Solución.

```
void famoso()
{
  while(1)
  {
     pthread_mutex_lock(&m);
     while(ocupacion < 5)
       pthread_cond_wait(&famoso_durmiendo,&m);

     Firmar();

     ocupacion--;
     pthread_cond_signal(&autografo);
     pthread_mutex_unlock(&m);
  }
}


void fan()
{
  pthread_mutex_lock(&m);

  if(ocupacion != AFORO_MAX)
  {
    ocupacion++;
    pthread_cond_signal(&famoso_durmiendo);
    pthread_cond_wait(&autografo,&m);
  }
  pthread_mutex_unlock(&m);
}
```