SISTEMAS OPERATIVOS - CUESTIONES
3 de septiembre de 2020

Nombre _____ DNI _____

Apellidos _____ Grupo _____

Cuestión 1. (2 puntos)

Un usuario desea dar formato a una partición de disco de 8 GiB de capacidad, barajando para ello diversas opciones. Contesta razonadamente a las siguientes preguntas:

a) (0,5 puntos) Si el usuario plantea utilizar un sistema de ficheros FAT16 (es decir, utilizando 16 bits para direccionar un bloque de datos) y un tamaño de bloque de 1 KiB, ¿qué cantidad de espacio en disco quedará inutilizable?

b) (0,5 puntos) En el anterior sistema de ficheros, y despreciando el tamaño que ocupa la tabla FAT y el reservado para el directorio raíz, ¿cuál es el tamaño de bloque mínimo que le permitiría utilizar de forma completa la partición?

c) (0,5 puntos) Fijando como tamaño de bloque el calculado en el apartado b), ¿qué porcentaje de disco se desperdiciará suponiendo que todos los ficheros a almacenar tienen un tamaño de 131 KiB?

d) (0,5 puntos) Considerando un sistema de ficheros basado en i-nodos con bloques de 1 KiB, donde cada i-nodo consta de dos índices directos, dos indirectos simples y dos indirectos dobles, y para referenciar un bloque se utilizan punteros de 128 bits, ¿qué cantidad de datos de un fichero que ocupa 131 KiB puede ser irrecuperable en el caso de que un bloque de la partición resulte ilegible? Considera y analiza todos los casos posibles.

Cuestión 2. (2 puntos)

Considera el siguiente programa:

```c
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

#define N 4

char fname[] = "argXX";

int write_args(int i)
{
    FILE* f;

    snprintf(fname, sizeof(fname),
        "arg%02d", i);
    f = fopen(fname, "w");
    fprintf(f,
        "my pid is %lld and my arg is: %d\n",
       (long long int) getpid(), i + 1000);
    fclose(f);
}
```

```c
int main(int argc, char *argv[])
{
    int i,pid,n;

    for (i = 0; i < N; i++) {
        pid = fork();
        if (pid < 0) {
               perror("fork");
               exit(EXIT_FAILURE);
        } else if (pid == 0) {
               write_args(i);
               execlp("cat", "cat",
                      fname, NULL);
               exit(EXIT_FAILURE);
        }
    }

    while (wait(NULL) > 0);

    exit(EXIT_SUCCESS);
}
```

Sabiendo que el proceso se ejecuta con el pid 100, y a los nuevos procesos creados se les asignan pids consecutivos a partir de éste (101, 102, …), se pide:

a) (0.5 puntos) Explica razonadamente cuántos procesos serán creados y cuantos ejecutarán potencialmente de forma concurrente.

b) (0.5 puntos) Justifica si puede determinarse o no el orden en que se ejecutarán estos procesos, indicando en tal caso dicho orden.

c) (0.5 puntos) Explica la salida que obtendremos al ejecutar el programa en un terminal.

d) (0.5 puntos) ¿Cuantos ficheros se habrán creado? Indica cómo se puede modificar el programa para que esos ficheros se borren una vez que los procesos hijo hayan completado su ejecución.

Cuestión 3. (2 puntos)

En general, los algoritmos de planificación se configuran con distintos parámetros. Por ejemplo, el algoritmo RR necesita de un parámetro para definir el quanto o timeslice. Las colas multinivel con realimentación se definen con parámetros para establecer el número de colas, el algoritmo de planificación en cada una de ellas, el criterio a utilizar para mover procesos entre las colas, etc.

Esto hace que algunos algoritmos de planificación puedan incluir a otros en función del valor de estos parámetros (por ejemplo el FCFS no es más que el RR con un quanto infinito). En este sentido, ¿qué relación (si es que la hay) existe entre los siguientes pares de algoritmos?

a. Prioridad y SJF
b. Prioridad y FCFS
c. RR y SJF

Cuestión 4. (2 puntos)

Un sistema utiliza gestión de memoria virtual con paginación bajo demanda con páginas de 4KiB, direcciones de memoria de 32 bits y memoria principal de 4GiB. Se quiere ejecutar en dicho sistema el siguiente programa:

```
#include <stdio.h>
/* resto de includes */
…
#define BLOQUE (4096)

int retorno=0;

int main(int argc, char *argv[])
{
    int fdin, fdout;
    void *src, *dst;
    size_t tam;

    // Punto A
    fdin = open(argv[1],O_RDONLY);
    fdout=open(argv[2],O_RDWR|O_CREAT|O_TRUNC,0666);
    ftruncate(fdout, BLOQUE);

   // Punto B
    src = mmap(0,BLOQUE,PROT_READ,
               MAP_SHARED,fdin, 0);
    dst = mmap(0,BLOQUE,PROT_WRITE,
               MAP_SHARED,fdout, 0);

   // Punto C
    memcpy(dst, src, BLOQUE);
    munmap(src, BLOQUE);
    munmap(dst, BLOQUE);

    // Punto D
    exit(retorno);
}
```

Considerando que:
- Las llamadas al sistema y el resto de funciones de librería que se incluyen en el programa nunca fallan.
- Hay suficientes marcos de página libres en el sistema.
- Todas las funciones externas se incluyen en una única biblioteca. Dicha biblioteca se enlazó de forma estática al crear el ejecutable.
- El texto (código) del programa tras la compilación ocupa 2,5KiB.
- El contenido inicial de la pila al lanzar a ejecución el programa ocupa 64 bytes.
- Los enteros y el tipo size_t ocupan 4 bytes.

a) Identifique qué regiones lógicas (y su tamaño en páginas) constituyen la imagen de memoria del proceso al lanzar a ejecución dicho programa en los puntos marcados en el código como A, B, C y D

b) ¿Cuántos fallos de página se producirán al pasar del punto A al punto B?. Y del punto B al C?

c) Suponiendo que el fichero que se abre en modo lectura ocupa 2 KiB. ¿Qué funcionalidad realiza el programa descrito?

Cuestión 5 (2 puntos) Considere este fragmento de código (a completar) que describe el comportamiento de dos hilos productores y otro consumidor:

```
#define SIZEBUF 10
int rd_idx=0;
int wr_idx=0;
int buffer[SIZEBUF];
/* Semaphore definitions */
```

```
void producer1(){
 int i=0;

 for(i=0;i<20;i++){
  item=generate_random_num();



  /* Insertion */
  buffer[wr_idx]=item;
  wr_idx=(wr_idx+1)%SIZEBUF;




 }
}
```

```
void producer2(){
 int i=0;

 for(i=0;i<40;i++){
  item=generate_random_num();



  /* Insertion */
  buffer[wr_idx]=item;
  wr_idx=(wr_idx+1)%SIZEBUF;




 }
}
```

```
void consumer(){
 int i=0;

 for(i=0;i<60;i++){



  /* Removal */
  item=buffer[rd_idx];
  rd_idx=(rd_idx+1)%SIZEBUF;




  do_something_with(item);
 }
}
```

Como puede observarse uno de los dos hilos productores (*producer1*) producirá 20 números aleatorios y el otro (*producer2*) 40; el consumidor consumirá 60 elementos del buffer circular compartido.

Añada las sentencias `wait()` y `signal()` necesarias en el código anterior usando los huecos para imponer mediante semáforos las siguientes restricciones de sincronización.

- Los productores insertarán números aleatorios en el buffer de forma alterna. En particular el productor 2 insertará dos números aleatorios seguidos en el buffer, por cada uno del productor 1.
- Se han de satisfacer también las restricciones del problema del productor-consumidor. Es decir, un productor no puede insertar elementos en el buffer cuando está lleno, y un consumidor no puede eliminar elementos del buffer cuando éste esté vacío.

Nótese que sólo pueden añadirse sentencias `wait()` y `signal()`, y sentencias condicionales, además de la definición de los semáforos utilizados, indicando claramente para cada uno cuál es su propósito así como su valor inicial.