Sistemas Operativos UCM 2012/2013

Módulo 2.1 Procesos

1. Dado el siguiente programa ejecutado bajo UNIX:

```
void main(int argc, char *argv[]) {
      int i;

      for (i=1; i<=argc; i++)
            fork();
      …
}
```

   a. Dibuje el esquema jerárquico de procesos que se genera para *argc = 3*
   b. ¿Cuántos procesos se crean para *argc* = n?

2. Considere el siguiente código:

```
int varGlobal;

void main() {
   int varLocal=3;
   pid_t pid;

   varGlobal=10;
   printf("Soy el proceso original. Mi PID es %d\n", getpid());
   fflush(NULL);

   pid = fork();
   if (pid ==-1) {
          perror("Error en fork()\n");
          exit(-1);
   }
   if (pid == 0 ) {
          // CODIGO DEL PROCESO HIJO
          varGlobal = varGlobal + 5;
          varLocal = varLocal + 5;
   }
   else {
          // CODIGO DEL PADRE: pid contiene el pid del hijo
          wait(NULL);
          varGlobal = varGlobal + 10;
          varLocal = varLocal + 10;
   }
   printf("Soy el proceso con PID %d. Mi padre es %d Global: %d Local %d\n", getpid(),
getppid(),varGlobal, varLocal );
}
```

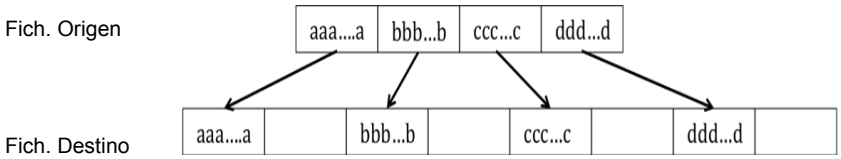
Asumiendo que el proceso original tiene PID 100 y es hijo del proceso *init (*PID=1), indica qué se mostrará por pantalla al ejecutar el código. ¿Es posible que los valores finales de las variables varíen de una ejecución a otra en función del orden de planificación? ¿Puede cambiar el orden en el que se muestran los diferentes mensajes por pantalla?

3. Considere el siguiente código:

```
int a = 3;
void main() {
    int b=2;
    for (i=0;i<4;i++) {
          p=fork();
          if (p==0) {
              b++;
              execlp("comando":...);
              a++;
          }
          else {
               wait();
               a++;
               b--;
          }
     }
    imprime(a,b);
}
```

a. ¿Cuántos procesos se crean en total? (sin contar el padre original) ¿Cuántos coexisten en el sistema como máximo?
b. ¿Qué se imprimirá al mostrar los valores de *a* y *b?*

4. Un programador poco avezado pretende hacer una aplicación que realice *copias intercaladas* de ficheros en paralelo. El concepto de copia intercalada se ilustra en la figura: se copia el primer bloque íntegro, y se deja un bloque vacío. Se copia el segundo bloque del fichero origen al tercer bloque del destino, y así sucesivamente.



El código de su programa para la *copia intercalada* de un fichero de 4 bloques mediante 4 procesos, es el siguiente:

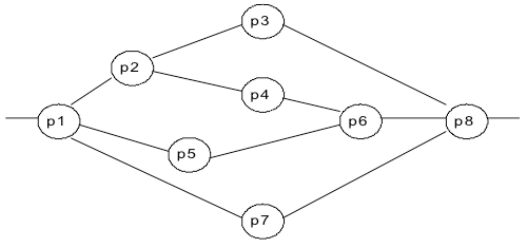
```
1: #define BLOCK 4096
2: char buf[BLOCK] = "xxxxxxx…xxxxx";
3: void main() {
4: pid_t pid;
5: int fdo,fdd;
6: fdo = open(«Origen»,O_RDONLY);
7: fdd = open(«Destino»,O_RDWR|O_CREAT|O_TRUNC,0666);
8: for (int i=0; i < 4; i++) {
9:    lseek(fdo,i*BLOCK, SEEK_SET);
10:   lseek(fdd,2*i*BLOCK,SEEK_SET);
11:   pid = fork();
12:   if (pid==0){
13:        copia_bloque(fdo,fdd);
14:        exit(0);
15:    }
16: }
17: while (wait(NULL)!=-1) { };
18: read(fdd,buf,BLOCK);
19: lseek(fdd,0,SEEK_SET);
20: read(fdd, buf,BLOCK);
}
```

```
void copia_bloque(int fdo, int fdd)
{

    read(fdo,buf,BLOCK);
    write(fdd,buf,BLOCK);

}
```

Responde a las siguientes preguntas, suponiendo que la **prioridad es para el proceso padre**, y después para cada uno de los hijos en el orden de creación.

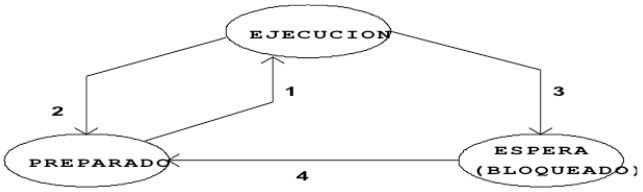
a) Indica el contenido del array **buf** inmediatamente después de la ejecución de las líneas 18 y 20. Justifica tu respuesta.
b) Sea sistema de ficheros de tipo Linux (nodos-i), con 3 punteros directos y un indirecto simple, tamaño de bloque de 4KB (4096 bytes) y 4bytes por puntero. Dibuja un posible estado final de toda la información del sistema de ficheros relativa al fichero "*Destino*". ¿Realiza el código la *copia intercalada* correctamente? Justifica tu respuesta.
c) Escribe una versión correcta de la *copia intercalada* paralela.

5. Use las llamadas *fork()*, *exec()*, *exit()* y *wait()* de UNIX para describir la sincronización de los ocho procesos cuyo grafo general de precedencia es el siguiente. Para ello escriba un función *main()* en la que se vayan creando los 8 procesos mediante llamadas a *fork(),* se ejecuten los binarios asociadoos a cada proceso (por ejemplo, *p1.out* para el proceso *p1*….) y se respete la precedencia mostrada en la figura (por ejemplo, *p6* no puede comenzar hasta que no hayan acabado *p4* y *p5).*



Alternativamente, use señalización para forzar las precedencias anteriores, en una situación donde los ocho procesos coexisten simultáneamente.

6. Podemos describir gran parte de la gestión del procesador en términos de diagramas de transiciones de estado como éste:



a. ¿Qué "evento" causa cada una de las transiciones marcadas?
b. Si consideramos todos los procesos del sistema, podemos observar que una transición de estado por parte de un proceso podría hacer que otro proceso efectuara una transición también. ¿Bajo qué circunstancias podría la transición 3 de un proceso provocar la transición 1 inmediata de otro proceso?
c. ¿Bajo qué circunstancias, si las hay, podrían ocurrir las siguientes transiciones causa-efecto?
   i. 2 -> 1 (por el hecho de que ocurra una transición tipo 2, hay una transición 1)
   ii. 3 -> 2
   iii. 4 -> 1
d. ¿Bajo qué circunstancias, si las hay, las transiciones 1, 2, 3 y 4 NO producirían ninguna otra transición inmediata?

7. Suponga que los siguientes trabajos llegan a procesarse en los instantes indicados. ¿Cuáles son los tiempos de retorno (*turnaround*) y de espera para cada uno de ellos, los tiempos de retorno y de espera promedios, así como la productividad (*throughput*) del sistema, aplicando las diferentes estrategias de planificación listadas? Aplique los algoritmos de planificación sin expropiación y base las decisiones en la información de que se dispone en el momento de tomarlas. El desglose de los tiempos de los trabajos se refiere al tiempo requerido para cadenas de uso de UCP y de E/S alternativamente.

Trabajo	Llegada	Tiempo Total	CPU	E/S	CPU	E/S
1	0.0	8	3	4	1	
2	0.5	4	1	1	1	1
3	1.0	2	0.5	0.5	1	
4	3.0	6	0.5	5	0,5	

a. FCFS: Primero en llegar, primero en pasar
b. SPN: Primero el de menor tiempo de UCP siguiente
c. RR: Prioridad circular con cuanto = 1
d. SPN, pero dejando la CPU inactiva durante la primera unidad de tiempo

8. Cinco trabajos de lotes, de *A* a *E*, llegan a un centro de cálculo casi al mismo tiempo. La estimación de sus respectivos tiempos de ejecución es de 10, 6, 2, 4, y 8 minutos. Sus prioridades, determinadas externamente, son 3, 5, 2, 1 y 4, respectivamente, siendo 5 la prioridad superior. Para cada uno de los siguientes algoritmos de planificación determine el tiempo medio de retorno de los procesos. (Ignorar el tiempo de conmutación de procesos).

a. Por turnos (round-robin) con q ≈ 0, es decir, empleando la política de compartición del procesador y midiendo los tiempos al final de los turnos completos.
b. Por prioridad estricta.
c. Por orden de llegada (FCFS)
d. Por menor tiempo de ejecución (SPN)

9. Considere un sistema con una política de planificación de procesos de 3 niveles con realimentación. Cada nivel usa a su vez una política de planificación circular (*round robin*) cuyos cuantos de tiempo son 2, 4 y 8, respectivamente. Al principio hay 3 procesos en la cola del nivel 1 (máxima prioridad). Los patrones de ejecución de los procesos son los siguientes (y se repiten indefinidamente):

P1 (3-CPU,5-E/S) P2 (8-CPU,5-E/S) P3 (5-CPU,5-E/S)

Las colas de los otros dos niveles están vacías. Cuando acaba una operación de E/S, los procesos entran en la cola de mayor prioridad. Usando un diagrama de tiempos muestre:

a. qué proceso está ejecutándose y
b. qué procesos hay en cada nivel, durante las 30 primeras unidades de tiempo de ejecución.

Calcule además la utilización de CPU y los tiempos de espera de cada proceso.

10. Considere un sistema con una planificación *MLF* (multinivel con realimentación) donde el número de niveles es $n = 10$, y el intervalo de planificación para el nivel i es $T_i = 2^i \cdot q$, siendo q es el valor del intervalo básico de planificación o *quanto*. En nuestro sistema sólo hay tres procesos, se encuentran inicialmente en la cola de máxima prioridad T_1 y sus tiempos respectivos hasta la siguiente petición de E/S son 3, 8, y 5 *quantos*. Cuando un proceso alcanza su petición de E/S permanece suspendido (sin competir por la CPU) durante 5 unidades de tiempo, tras las cuales vuelve a entrar en la cola de máxima prioridad. Los tiempos de CPU requeridos hasta la petición de E/S siguiente son de nuevo 3, 8, y 5. Usando un diagrama de tiempos, muestre

a. Qué proceso estará ejecutándose y qué procesos estarán en qué cola durante cada una de las primeras 30 unidades de tiempo de ejecución.
b. Cuáles son los valores de las siguientes medidas de rendimiento: productividad, tiempos retorno (*turnaround*) individual y promedio, y tiempos de espera individual y promedio.

11. Repita el ejercicio anterior pero suponiendo que $Ti = 2 \cdot q$ (es decir, constante), para todos los niveles de prioridad.

12. En un sistema UNIX se quiere programar un proceso que lea de un archivo de usuarios /etc/users para ver si un usuario está dado de alta. Dicho archivo es un array de entradas struc user:

```
struct user {
      char nombre[32];
      char descripción[254];
}
```

El proceso debe leer sin verse afectado por las señales SIG_INT y SIG_QUIT, pero debe poner un temporizador, de 20 segundos, de manera que si el proceso de comprobación dura más tiempo, debe imprimir un mensaje de error y terminar.

Se pide programar dicho proceso en C, resaltando claramente las llamadas al sistema. Opcionalmente se puede hacer en pseudo-código (manteniendo al menos la nomenclatura y los parámetros para las llamadas al sistema).

13. Se tiene un proceso productor y otro consumidor. El proceso consumidor debe crear al proceso productor y asegurarse de que está vivo. El proceso consumidor llama a una función (leer_entero), que devuelve un valor entero, generado por el productor, y lo guarda en un archivo (datos). Si el valor leído es 0, es señal de que el productor ha muerto por algún motivo, por lo que deberá crear otro proceso productor. El proceso consumidor debe leer un valor cada 10 segundos, cuando haya leído 100 valores, deberá matar al productor y acabar la ejecución.

Suponiendo que tanto el proceso productor como la función leer_entero están ya escritos, se pide programar el proceso consumidor en C, resaltando claramente las llamadas al sistema. Opcionalmente se puede hacer en pseudo-código (manteniendo al menos la nomenclatura y los parámetros para las llamadas al sistema).

14. Sea un sistema operativo que sigue el modelo cliente-servidor en el que existe un proceso servidor SF (Sistema de Ficheros) y un proceso CD (Controlador de Disco). Las prioridades de los procesos SF y CD son mayores que la de los procesos de usuario. En un determinado instante la cola de PREPARADOS de este sistema contiene dos procesos de usuario A y B, en dicho orden. Las características de ejecución de A y B son las siguientes:

a. Proceso A: 160 ms CPU; 50 ms E/S de disco; 50 ms CPU
b. Proceso B: 20 ms CPU; 50 ms E/S de disco; 60 ms CPU

Se pide construir un diagrama de tiempos donde se muestre, a partir del instante en que aparecen los procesos A y B en el sistema, los estados de estos dos procesos (EJECUCIÓN; PREPARADO; BLOQUEADO). Las operaciones de E/S de disco conllevan una petición por parte del usuario al SF, que a su vez realizará una petición al proceso CD. Cuando los datos solicitados al CD estén listos, éste avisará al proceso invocante (SF) que a su vez notificará al proceso de usuario. Se supone que el tiempo de ejecución de los procesos CD y SF es despreciable. El *quanto* de planificación aplicado a los procesos de usuario es de 100 ms.

15. Determinar el rendimiento (%uso de CPU y %sobrecarga del S.O.) obtenido al planificar dos procesos multi-hilo en un sistema donde los hilos se gestionan en espacio de usuario. El algoritmo de planificación del SO es RR con q=100, y el cambio de contexto ente procesos tarda 20. El planificador de la biblioteca de hilos reparte el *q* del proceso entre los hilos aplicando RR con *q*=10 y sin coste de cambio de contexto entre sus hilos. Los dos procesos tienen el siguiente comportamiento:

Proceso A	Proceso B
Hilo 1 (30-CPU; 110-E/S; 40-CPU)	Hilo 1 (20-CPU; 50-E/S; 60-CPU)
Hilo 2 (50-CPU)	Hilo 2 (40-CPU; 110-E/S; 20-CPU)
Hilo 3 (30-CPU)	
Hilo 4 (20-CPU; 60-E/S; 40-CPU)	

¿Sería diferente la planificación final si los hilos fuesen gestionados por el kernel?