# Problemas de Sistemas Operativos

**Facultad de Informática, UCM**

**Módulos 3.1 y 3.2: Gestión de Procesos y Planificación**

## Problemas Básicos

**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 globalVar;

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

  globalVar=10;
  printf("I am the original process. My PID is %d\n", getpid());
  fflush(NULL);

  pid = fork();
  if (pid ==-1) {
    perror("Can't fork()\n");
    exit(-1);
  }
  if (pid == 0 ) {
    // Child process
    globalVar = globalVar + 5;
    localVar = localVar + 5;
  }
  else {
    // Parent process
    wait(NULL);
    globalVar = globalVar + 10;
    localVar = localVar + 10;
  }
  printf("I am the process with PID %d. Mi parent is %d Global: %d Local %d\n", getpid(),
       getppid(),globalVar, localVar );
}
```

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?

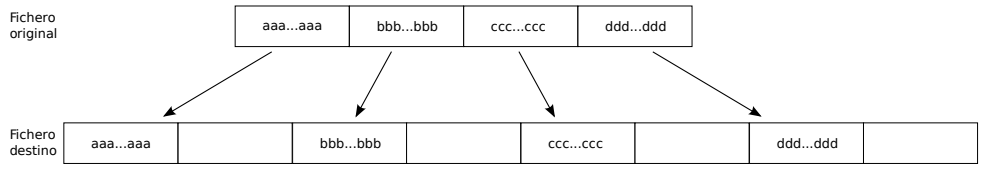
**3.-**Considere el siguiente código:

```
int a = 3;
void main() {
    int b=2;
    int p;
    for (i=0;i<4;i++) {
         p=fork();
         if (p==0) {
             b++;
             execlp("command":...);
             a++;
         }
         else {
              wait();
              a++;
              b--;
         }
     }
    printf("a=%d,b=%d\n",a,b);
}
```

*a*) ¿Cuántos procesos se crean en total? (sin contar el padre original) ¿Cuántos coexisten en el sistema como máximo?

*b*) ¿Qué imprimirá por pantalla la última sentencia del programa?

**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 programador crea una primera versión de la aplicación que realiza la copia intercalada de un fichero que consta de 4 bloques usando 4 procesos. El código del programa es el siguiente:

```
1  #define BLOCK 4096
2  char buf[BLOCK] = "xxxxxxx...xxxxx";

4  void copy_block(int fdo, int fdd) {
5      read(fdo,buf,BLOCK);
6      write(fdd,buf,BLOCK);
7  }

9  void main() {
10 pid_t pid;
11 int fdo,fdd,i;
12 fdo = open("Origin",O_RDONLY);
13 fdd = open("Destination",O_RDWR|O_CREAT|
       O_TRUNC,0666);
14 for (i=0; i < 4; i++) {
15    lseek(fdo,i*BLOCK, SEEK_SET);
16    lseek(fdd,2*i*BLOCK,SEEK_SET);
17    pid = fork();
18    if (pid==0){
19        copy_block(fdo,fdd);
20        exit(0);
21     }
22  }
23  while (wait(NULL)!=-1) { };
24  read(fdd,buf,BLOCK);
25  lseek(fdd,0,SEEK_SET);
26  read(fdd, buf,BLOCK);
27 }
```

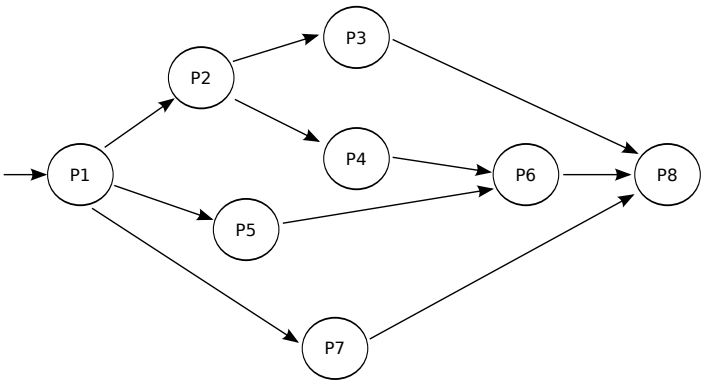
Responde a las siguientes preguntas, suponiendo que **la prioridad es para el proceso padre**, y después para cada uno de los hijos en el orden de creación.

*a*) Indica el contenido del array `buf` inmediatamente después de la ejecución de las líneas 24 y 26. Justifica tu respuesta.

*b*) ¿Realiza el código la copia intercalada correctamente? En el caso de que no sea así, escribe una versión correcta de la copia intercalada **paralela** que no haga suposiciones artificiales acerca de la planificación.

*c*) Sea el sistema de ficheros de tipo Linux (nodos-i), con 3 punteros directos y un indirecto simple, tamaño de bloque de 4KiB (4096 bytes) y 4bytes por puntero. Dibuja un posible estado final de toda la información del sistema de ficheros relativa al fichero “Destination”.

**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 asociados a cada proceso (por ejemplo, "p1" para el proceso `P1`, etc.) y se respete la precedencia mostrada en la figura (por ejemplo, `P6` no puede comenzar hasta que no hayan acabado `P4` y `P5`).




**6.-**Considere el siguiente código:

```
int fd = -1;
char buf1[4]="aaaa";
char buf2[4]="bbbb";

int main() {
   pthread_t tid1, tid2;
   pthread_create(&tid1,NULL,thread1,NULL);
   pthread_create(&tid2,NULL,thread2,NULL);
   pthread_join(tid1,NULL);
   pthread_join(tid2,NULL);
   close(fd);
}

void* thread1(void* arg) {
   fd=open("test",O_RDWR|O_CREAT|O_TRUNC,0666);
   write(fd,buf1,4);
}
void* thread2(void* arg) {
   while (fd==-1) {};
   write(fd,buf2,4);
}
```

Indique si cada una de las siguientes afirmaciones es cierta o falsa justificando la respuesta:

*a*) El hilo 2 (función `thread2()`) nunca saldrá del bucle while inicial.

*b*) La escritura del hilo 2 (función `thread2()`) producirá un error por no haber abierto antes el fichero.

*c*) El contenido final del fichero *test* será o bien, "aaaa" o bien "bbbb".; ninguna otra alternativa esposible.

*d*) La llamada a close() del programa principal no debería devolver '-1' (esto es, no debería producir ningún error).

**7.-**En un sistema monoprocesador los siguientes procesos llegan a procesarse en los instantes indicados. La tabla mostrada a continuación indica el instante de llegada de cada proceso así como su perfil de ejecución. El desglose de los tiempos, se refiere al tiempo requerido para ráfagas de uso de CPU y de E/S alternativamente.

| **Proceso** | **Tiempo de llegada** | **CPU** | **E/S** | **CPU** | **E/S** |
|---|---|---|---|---|---|
| P1 | 0 | 1 | 5 | 1 | |
| P2 | 1 | 3 | 1 | 1 | 1 |
| P3 | 0 | 5 | 4 | 1 | |
| P4 | 3 | 3 | 2 | 1 | 1 |

Para los distintos algoritmos de planificación enumerados a continuación indique los tiempos de ejecución (o de retorno) y de espera para cada uno de ellos, los tiempos de ejecución y de espera promedios, así como la productividad (*throughput*) y el porcentaje de uso de CPU del sistema. Suponga que en el instante t=0, P1 se encola antes que P3 en la cola del planificador.

*a*) FCFS
*b*) SJF
*c*) RR con quanto=3
*d*) RR con quanto=2

**8.-**Cinco trabajos por lotes (*batch*), de A a E, llegan casi al mismo tiempo a un centro de cálculo dotado de un único procesador. La estimación de sus respectivos tiempos de CPU 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*) Un planificador ideal completamente justo (RR q$\approx$ 0) que asegura una distribución equitativa del tiempo de CPU con independencia del intervalo de planificación considerado.
*b*) Por prioridad estricta
*c*) Por orden de llegada (FCFS)
*d*) Por menor tiempo de CPU (SJF)

**9.-**Considere un sistema monoprocesador con una política de planificación de procesos de 3 niveles (MLF) 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)

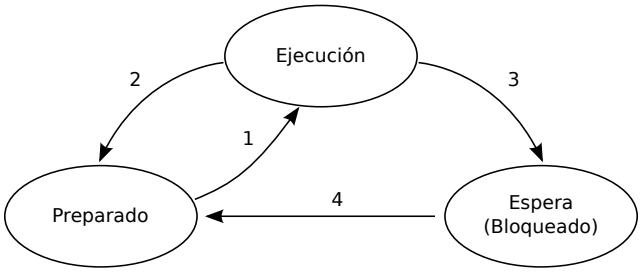
Las colas de los otros dos niveles están vacías. Considere que cuando un proceso consume su cuanto, éste se mueve al nivel de prioridad inmediatamente inferior. Por el contrario, cuando no consume su cuanto (p.ej. se bloquea por E/S antes), el planificador incrementa su prioridad. Usando un diagrama de tiempos muestre:

1. qué proceso está ejecutándose y
2. 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.

# Problemas Adicionales

**10.-**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?

- $2 \rightarrow 1$ (por el hecho de que ocurra una transición tipo 2, hay una transición 1)
- $3 \rightarrow 2$
- $4 \rightarrow 1$

*d*) ¿Bajo qué circunstancias, si las hay, las transiciones 1, 2, 3 y 4 NO producirían ninguna otra transición inmediata?

**11.-**Considere un sistema monoprocesador 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 $T_i = 2i \cdot q$ , 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 `T1` 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.

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

**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 para entornos monoprocesador 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: Se

| | | | |
|---|---|---|---|
| `Proceso A:` | 160 ms CPU, | 50 ms E/S de disco, | 50 ms CPU |
| `Proceso B:` | 20 ms CPU, | 50 ms E/S de disco, | 60 ms CPU |

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.