Examen Parcial de Sistemas Operativos - Marzo de 2014

Grado en Ingeniería Informática

NOTAS:

* La fecha de publicación de las notas, así como de revisión se notificarán por Aula Global.
* Para la realización del presente examen se dispondrá de 1:50 horas.
* No se pueden utilizar libros ni apuntes
* Será necesario presentar el DNI o carnet universitario para realizar la entrega del examen

Ejercicio 1 (3 puntos). Autotest.

Responda a las preguntas del autotest en los cuadros adjuntos indicando la letra de la respuesta válida. Recuerde que por cada 3 fallos se quita un punto. No contestadas no penalizan.

NOMBRE: GRUPO:

Solución Test 1:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
C	C	D	A	A	B	D	D	B	B	C	A	A	C	A

Solución Test 2:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
C	D	C	B	B	D	C	A	A	B	D	D	B	D	B

Examen Parcial de Sistemas Operativos - Marzo de 2014

Grado en Ingeniería Informática

Ejercicio 2 (3 puntos). Un ingeniero informático debe decidir entre usar un algoritmo de planificación FCFS (First Come First Serve) o Round Robin en un nuevo sistema operativo. Para ello, el ingeniero se plantea el siguiente escenario:

- Un proceso A llega en el instante de tiempo t=0 y su duración (tiempo de servicio) es de 4 unidades de tiempo.
- Un proceso B llega en el instante de tiempo t=2 y tiene una duración de 4 unidades de tiempo.
- Un proceso C llega 2 unidades de tiempo después que B y una duración igual a 2 unidades de tiempo.
- Un proceso D llega 2 unidades de tiempo después de C y tiene una duración igual a 5 unidades de tiempo.
- El proceso B invoca una llamada al sistema 1 unidad de tiempo después de iniciar su ejecución, que tarda en ser completada 2 unidades de tiempo.

Se pide:

a) Complete el siguiente diagrama indicando la planificación de los procesos si se emplea FCFS.

	t=0	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8	t=9	t=10	t=11	t=12	t=13	t=14
A															
B															
C															
D															

b) Complete el siguiente diagrama de temporización indicando los procesos que ejecutan en cada unidad de tiempo si se emplea Round Robin con rodaja de tiempo q=2.

	t=0	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8	t=9	t=10	t=11	t=12	t=13	t=14
A															
B															
C															
D															

c) Complete las siguientes tablas para la planificación FCFS y Round Robin:

FCFS	Llegada	Servicio	T. Inicio	T. Fin	T. Retorno	T. Espera	T. Retorno Normalizado
A							
B							
C							
D							

Round Robin	Llegada	Servicio	T. Inicio	T. Fin	T. Retorno	T. Espera	T. Retorno Normalizado
A							
B							
C							
D							

d) Si se asume que todos los procesos en el sistema son procesos batch y el ingeniero pretendiera que todos estos trabajos se inicien cuanto antes, ¿qué algoritmo de planificación seleccionaría de entre estas dos alternativas?

Solución:

a) FCFS:

A															
B															
C															
D															
	t=0	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8	t=9	t=10	t=11	t=12	t=13	t=14

b) Round Robin (q=2):

A															
B															
C															
D															
	t=0	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8	t=9	t=10	t=11	t=12	t=13	t=14

c)

FCFS	Llegada	Servicio	T. Inicio	T. Fin	T. Retorno	T. Espera	T. Retorno Normalizado
A	0	4	0	4	4-0=4	4-4=0	4/4=1
B	2	4	4	15	15-2=13	13-4=9	13/4=3,25
C	4	2	5	7	7-4=3	3-2=1	3/2=1,5
D	6	5	7	12	12-6=6	6-5=1	6/5=1,2

Round Robin	Llegada	Servicio	T. Inicio	T. Fin	T. Retorno	T. Espera	T. Retorno Normalizado
A	0	4	0	5	5-0=5	5-4=1	5/4=1,25
B	2	4	2	12	12-2=10	10-4=6	10/4=2,5
C	4	2	5	7	7-4=3	3-2=1	3/2=1,5
D	6	5	9	15	15-6=9	9-5=4	9/5=1,8

d) Si se pretende que los trabajos batch inicien su ejecución en el sistema lo antes posible, sin importar cuándo acabarán, la mejor opción sería Round Robin, dado que todos los trabajos entrarían a ejecutar en orden después de (n-1)*q unidades de tiempo, siendo n el número de procesos en la cola. Con FCFS, los procesos deben esperar a la terminación de todos los procesos anteriores a él en la cola, por tanto, deberán esperar la suma de los tiempos de servicio de los trabajos anteriores a él.

Grado en Ingeniería Informática

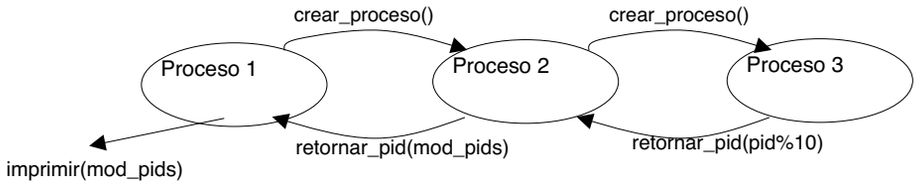
Ejercicio 3 (2 puntos). Se desea crear una jerarquía de 3 procesos en forma de línea, en la que cada proceso excepto el último crea un único proceso hijo, es decir el proceso 1 creará un proceso 2 y éste creará un proceso 3. Cada uno de los procesos hijos (todos menos el proceso 1) realizarán las dos siguientes acciones:

- Imprimir en pantalla un mensaje con su PID.
- Finalizar su ejecución retornando el módulo 10 de su PID (PID%10) al proceso padre.

Cada proceso padre (todos menos el proceso 3) deberá realizar lo siguiente:

- Esperar por la finalización del proceso hijo.
- Acumular en una variable mod_pids el valor que retornó el hijo y el módulo 10 de su PID.
- Finalizar su ejecución retornando mod_pids al proceso padre.

Finalmente, el proceso 1 imprimirá mod_pids cuyo valor será e módulo 10 de la suma de los PIDs de los tres procesos. La interacción entre los procesos de la línea seguirán un esquema similar a:

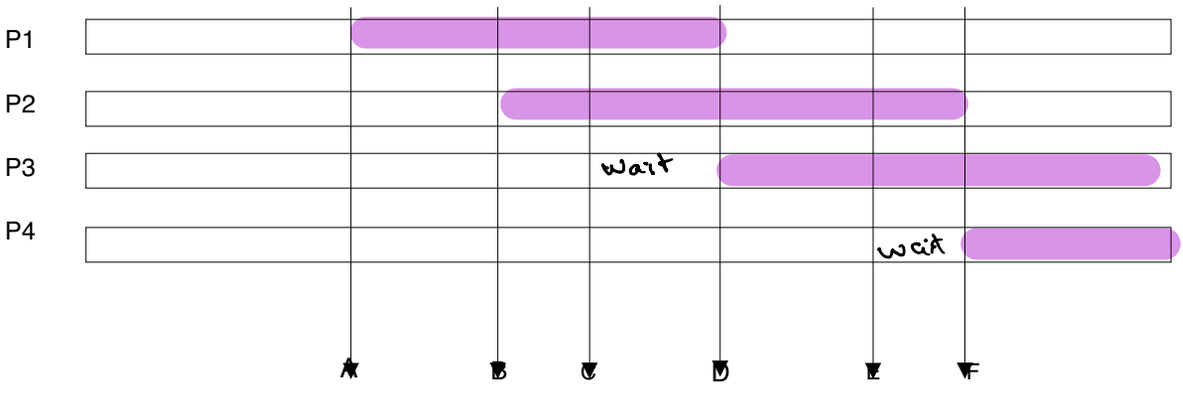


Se pide codificar el programa en C que permita implementar la jerarquía descrita.

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

int main(){
	pid_t mi_pid;
	int suma_pids=0;
	int status;

	mi_pid=fork();
	switch(mi_pid){
		case -1: // Error en fork
			printf("Error en el fork. Exit\n");
			exit(-1);
		case 0: // Proceso 2, hijo
			mi_pid=fork();
			switch(mi_pid){
				case -1: // Error en fork
					printf("Error en el fork. Exit\n");
					exit(-1);
				case 0:  // Proceso 3, hijo
					mi_pid=getpid()%10;
					printf("Proceso hijo 3:
						pid=%dpid%10=%d\n",getpid(),mi_pid);
					exit(mi_pid);
				default: // Proceso 2, hijo
					wait(&status);
					suma_pids=(WEXITSTATUS(status)+getpid())%10;
					printf("Proceso hijo 2: Mi_pid=%d, pid
						hijo=%d, suma_pids=%d\n", getpid(),
						WEXITSTATUS(status), suma_pids);
					exit(suma_pids);
			}
		default: // Proceso 1, padre
			wait(&status);
```

```
                suma_pids=(WEXITSTATUS(status)+getpid())%10;
                printf("Proceso padre 1: Mi_pid=%d, pid hijo=%d,
                suma_pids=%d\n", getpid(), WEXITSTATUS(status), suma_pids);
        }
        exit(0);
}
Salida:
<sescolar: guernika>$ ./ejercicio1
Proceso hijo 3: pid=21197pid%10=7
Proceso hijo 2: Mi_pid=21196, pid hijo=7, suma_pids=3
Proceso 1: Mi_pid=21195, pid hijo=3, suma_pids=8

<sescolar: guernika>$ ./ejercicio1
Proceso hijo 3: pid=21200pid%10=0
Proceso hijo 2: Mi_pid=21199, pid hijo=0, suma_pids=9
Proceso 1: Mi_pid=21198, pid hijo=9, suma_pids=7
```

Ejercicio 4 (2 puntos). Dada la siguiente ejecución concurrente de los procesos ligeros P1, P2, P3 y P4. Suponga que el valor inicial del semáforo s es 2.



Considere los siguientes eventos:

- A: P1 realiza wait(s)
- B: P2 realiza wait(s)

- C: P3 realiza wait(s)
- D: P1 realiza signal(s)
- E: P4 realiza wait(s)
- F: P2 realiza signal(s)

Se pide:

a) ¿Cuál es el valor del semáforo s tras los eventos A,B,C,D,E,F?

b) ¿Cuántos procesos pueden ejecutar de manera concurrente sin bloquearse?

c) ¿Qué procesos están ejecutando después de F?

d) ¿Se puede emplear este semáforo para que los procesos P1,P2,P3 y P4 accedan a una sección crítica? Justifique su respuesta.

Solución:

a) Valor del semáforo después de cada llamada wait/signal.

Evento	Sincronización	Valor de s
A	P1→ wait(s)	1
B	P2→ wait(s)	0
C	P3→ wait(s)	-1 (P3 se bloquea)
D	P1→ signal(s)	0 (P3 se desbloquea)
E	P4→ wait(s)	-1 (P4 se bloquea)
F	P2→ signal(s)	0 (P4 se desbloquea)

b) El número de procesos que pueden ejecutar concurrentemente sin bloquearse es igual al valor inicial del semáforo, es decir, 2.

c) Después de F, el valor del semáforo es 0 y se encuentran ejecutando P3 y P4.

d) Cualquier mecanismo de sincronización que resuelva el problema de la sección crítica debe asegurar exclusión mutua, es decir, debe garantizar que sólo existe un proceso ejecutando la SC. Cualquier otro proceso que desee entrar en la SC se bloqueará hasta que el proceso que está dentro de la SC la abandone. Para lograr este comportamiento es necesario inicializar el semáforo con valor igual a 1. Este esquema no se puede utilizar dado que el valor del semáforo es igual a 2, permitiendo que dos procesos puedan ejecutar concurrentemente sin bloquearse.