Fundamentos de los Sistemas Operativos (FSO)

Departamento de Informática de Sistemas y Computadoras (DISCA)
Universitat Politècnica de València

Ejercicios de consolidación

(I)

fSO

Ejercicio 1

Un sistema operativo utiliza un algoritmo de planificación basado en "**clases de prioridad**" con dos niveles, y **la política de planificación para cada nivel es FCFS. La planificación entre los dos niveles es expulsiva basada en prioridades**, siendo la prioridad más alta la clase 1. **Cuando un proceso es interrumpido, pasa al principio de su cola.** Los procesos comparten **un solo disco que se planifica FCFS**. El sistema operativo asigna procesos a una de las dos clases dependiendo de su uso de CPU y consumo de disco. **Si un proceso ha consumido más tiempo de CPU que el disco, se asigna a la clase 2, de lo contrario se asigna a la clase 1.** Inicialmente los procesos están en la clase 1. El sistema determina la clase para cada proceso en cada unidad de tiempo. Tres procesos, A, B, C, llegan simultáneamente al sistema en el orden A, B y C. Los perfiles de ejecución se muestran en la siguiente tabla.

Obtenga la línea de tiempo de procesamiento, el tiempo de respuesta promedio y el tiempo de espera promedio.

Process	Execution profile
A	2 CPU + 1 DISK + 7CPU
B	1CPU + 3 DISK + 1 CPU + 2 DISK + 1 CPU + 2 DISK + 1 CPU
C	2CPU + 1 DISK + 1CPU + 2 DISK + 1 CPU + 2 DISK + 1 CPU

Ejercicio 3

Proceso	Perfil de Ejecución
A	2 CPU + 1 DISCO + 7CPU
B	1CPU + 3 DISCO + 1 CPU + 2 DISCO + 1 CPU + 2 DISCO + 1CPU
C	2CPU + 1 DISCO + 1CPU + 2 DISCO + 1 CPU + 2 DISCO + 1 CPU

t	Preparados		CPU	Cola Disco	Disco	Comentarios
	Clase 1	Clase 2				
0	C B (A)		A			Llegan A, B y C
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						
19						
20						
21						
22						

Ejercicio 3: Solución

Proceso	Perfil de Ejecución
A	2 CPU + 1 DISCO + 7CPU
B	1CPU + 3 DISCO + 1 CPU + 2 DISCO + 1 CPU + 2 DISCO + 1CPU
C	2CPU + 1 DISCO + 1CPU + 2 DISCO + 1 CPU + 2 DISCO + 1 CPU

t	Preparados		CPU	Cola Disco	Disco	
	Clase 1	Clase 2				
0	C B (A)		A			Llegan A, B y C
1	C (B)	A	B			
2	(C)	A	C		B	
3		C (A)	A		B	
4		(C)	C	A	B	
5	(B)		B	C	A	
6		(A)	A	B	C	
7		C (A)	A		B	
8		C (A)	A		B	
9	(B)	C A	B			
10		C (A)	A		B	
11		C (A)	A		B	
12	(B)	C A	B			
13		C (A)	A			Acaba B
14		C (A)	A			
15	(C)		C			Acaba A
16					C	
17					C	
18	(C)		C			
19					C	
20					C	
21	(C)		C			
22						Acaba C

Tiempo de retorno (15,13,22))
Tiempo de espera (4,1,11)

Ejercicio 2

Sea un sistema operativo en el cual el algoritmo de planificación consta de cuatro clases de prioridad numeradas del 0 al 3. El algoritmo de planificación es **Round-Robin para las clases 0, 1 y 2,** y es **FCFS para la clase 3**. La clase más prioritaria es la 0. Los cuantos de tiempo, qi , para las clases 0,1 y 2 vienen dados por la siguiente fórmula: **qi = i + 1**. El algoritmo **de planificación inter-colas es expulsivo**.

Los procesos que entran en el sistema son admitidos inicialmente en la **clase más prioritaria (0)**, existiendo un mecanismo de degradación de la prioridad obedeciendo a la siguiente regla: "un proceso permanece en su clase hasta que ha consumido dos cuantos de tiempo, tras lo cual es **degradado** a la clase de prioridad inmediatamente inferior". Todo proceso que llega a la clase 3 permanece en ella hasta que termina su ejecución.

Calcular el **tiempo de retorno y la clase en que terminan tres procesos, P1, P2, P3,** que llegan en dicho orden en el instante cero.

Proceso	Instante Llegada	Perfil de Ejecución
P1	0	4 CPU
P2	0	8 CPU
P3	0	12 CPU

Ejercicio 2:

Proceso	Instante Llegada	Perfil de Ejecución
P1	0	4 CPU
P2	0	8 CPU
P3	0	12 CPU

t	Preparados				CPU	
	Cola 3 FCFS	Cola 2 RR q=3	Cola 1 RR q=2	Cola 0 RR q=1		
0				P3 P2	P1	Llegan P1, P2 y P3
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						
19						
20						
21						
22						
23						
24						

Algoritmo de planificación inter-colas expulsivo

Tiempo de retorno
(?, ?, ?)
Clase de finalización
de los procesos
(?,?,?)

Ejercicio 2: Solución

Proceso	Instante Llegada	Perfil de Ejecución
P1	0	4 CPU
P2	0	8 CPU
P3	0	12 CPU

Algoritmo de planificación inter-colas expulsivo

t	Preparados				CPU	
	Cola 3 FCFS	Cola 2 RR q=3	Cola 1 RR q=2	Cola 0 RR q=1		
0				P3 P2	P1	Llegan P1, P2 y P3
1				P1 P3	P2	
2				P2 P1	P3	
3				P3 P2	P1	
4			P1	P3	P2	
5			P2 P1		P3	
6			P3 P2		P1	
7			P3 P2		P1	
8			P3		P2	Acaba P1 desde clase 1
9			P3		P2	
10			P2		P3	
11			P2		P3	
12			P3		P2	
13			P3		P2	
14		P2			P3	
15		P2			P3	
16		P3			P2	
17		P3			P2	
18					P3	Acaba P2 desde clase 2
19					P3	
20					P3	
21					P3	
22					P3	
23					P3	
24						Acaba P3 desde clase 2

Tiempo de retorno (8, 18, 24)
Clase de finalización de los procesos (1,2,2)

- Escribe la salida por consola del siguiente programa tras su ejecución. Explica la respuesta.
- Nota:
  - delay (n) realiza un retraso (suspension del hilo) de n milisegundos;
  - rand() devuelve un número aleatorio en el rango 0.0 .. 1.0

```
void * f1(void * arg) {
     printf("f1\n");
     delay(4000*rand()%1000);
     printf("Thread 1 text\n");
     return 0;
}
```

```
void * f2(void * arg) {
     printf("f2\n");
     delay(4000*rand()%1000);
     printf("Thread 2 text\n");
     return 0;
}
```

```
int main (void) {
  pthread_t th1,th2, th3, th4;
  pthread_attr_t attrib;

  pthread_attr_init( &attrib );

  printf("Start Process ...\n");
  pthread_create( &th1, &attrib, f1, NULL);
  pthread_create( &th2, &attrib, f2, NULL);
  pthread_create( &th3, &attrib, f1, NULL);
  pthread_create( &th4, &attrib, f2, NULL);
  //delay(4000*rand()%1000);
  exit(0);
}
```

- Considere dos archivos ejecutables F1 y F2, obtenidos mediante la compilación de los siguientes programas:
  - **Nota:**
    - La orden shell "`date +%S`" imprime por pantalla el campo de los segundos de la hora actual. Por ejemplo, si la hora actual es “20:30:12”, imprimiría "12".

/** F1.c **/	/** F2.c **/

#include <…>
 	
int pid;
 	
main(){
  pid=fork();
  if (pid==0)
  {sleep(3);}
  execl("./F2","F2",NULL);
}
	#include <…>
	
	int pid;
	
	main(){
	  pid=fork();
	  if (pid==0)
	  { sleep(1);}
	  execl("/bin/date","date","+%S",NULL);
	}

Suponiendo que ambos programas ejecutables se encuentran en el directorio actual y se ejecutan sin errores, responda a las siguientes cuestiones:

/** F1.c **/	/** F2.c **/

#include <…>
 	
int pid;
 	
main(){
  pid=fork();
  if (pid==0)
  {sleep(3);}
  execl("./F2","F2",NULL);
}
	#include <…>
	
	int pid;
	
	main(){
	  pid=fork();
	  if (pid==0)
	  { sleep(1);}
	  execl("/bin/date","date","+%S",NULL);
	}

a) Cuántos procesos se crean cuando se ejecuta la orden ". / F2" y cuál es su parentesco.

b) Qué salida por pantalla se obtiene cuando se ejecuta la orden "./F2" en el momento 09:10:25.

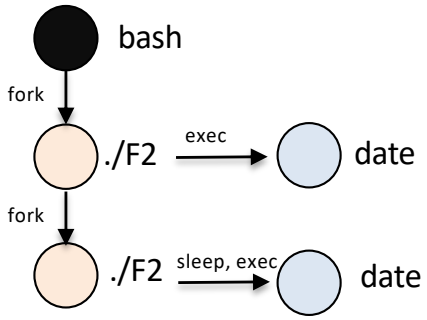
c) Cuántos procesos se crean cuando se ejecuta la orden ". / F1" y cuál es su parentesco

d) Qué salida por pantalla se obtiene cuando se ejecuta la orden "./F1" en el momento 09:10:25.

/** F1.c **/	/** F2.c **/

#include <...>
 	
int pid;
 	
main(){
pid=fork();
if (pid==0)
{sleep(3);}
execl("./F2","F2",NULL);
}
	#include <...>
	
	int pid;
	
	main(){
	pid=fork();
	if (pid==0)
	{ sleep(1);}
	execl("/bin/date","date","+%S",NULL);
	}

a) Cuántos procesos se crean cuando se ejecuta la orden ". / F2" y cuál es su parentesco.

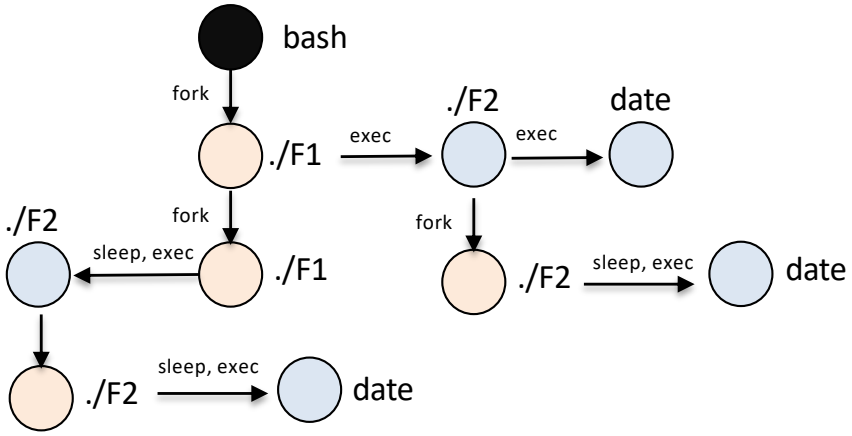


b) Qué salida por pantalla se obtiene cuando se ejecuta la orden "./F2" en el momento 09:10:25.

25

26

c) Cuántos procesos se crean cuando se ejecuta la orden ". / F1" y cuál es su parentesco



d) Qué salida por pantalla se obtiene cuando se ejecuta la orden "./F1" en el momento 09:10:25.

25

26

28

29

- Cuáles de los siguientes elementos pertenecen al Núcleo del Sistema Operativo y cuáles no.

	núcleo	aplicación	dispositivo
“bash”			
sistema de archivos			
controlador de disco			
explorador de archivos			
explorador de internet			
orden “ps”			
gestor de procesos			
Interfaz de llamadas a sistema			
Manejador de interrupción de reloj			
Driver (manejador de dispositivo) del usb			

Ejercicio

- Considere que existe el archivo "hello.txt" en el directorio actual y que contiene el texto "hello \n", de manera que la orden "cat hello.txt" imprime por pantalla una lñinea con la palabra **hello**.

Suponga que compila y ejecuta el siguiente programa (ejecutable a.out) Escriba la salida por pantalla de cada una de las siguientes ejecuciones del programa:

$ ./a.out
$ ./a.out 1
$ ./a.out 2
$ ./a.out 3
$ ./a.out 4
$ ./a.out 5
$ ./a.out 12

```
#include <...all headers...>

int main(int argc, char *argv[]) {
    int X;
    int val = 0;
    int parent_pid = getpid();

    if (argc >1) X = atoi(argv[1]);
    else  X = 0;

    printf("X= %d\n", X);
    if (X >= 3) val = fork();
    if (val == 0) {
        if (X%2 == 1) {
            execl("/bin/cat", "cat", "hello.txt", NULL);
            printf("Err: /bin/cat generates error\n");
        }
        else {
            execl("/cat/bin", "cat", "hello.txt", NULL);
            printf("Err: /cat/bin/cat generates error\n");
        }
    }
    if (getpid() == parent_pid)
        printf("parent\n");
    else
        printf("child\n");
    sleep(2);
    return 0;
}
```

```
$ ./a.out
X= 0
Err: /cat/bin/cat generates error
parent
$ ./a.out 1
X= 1
hello
$ ./a.out 2
X= 2
Err: /cat/bin/cat generates error
parent
$ ./a.out 3
X= 3
parent
hello
$ ./a.out 4
X= 4
parent
Err: /cat/bin/cat generates error
child
$ ./a.out 5
X= 5
parent
hello
$ ./a.out 12
X= 12
parent
Err: /cat/bin/cat generates error
child
```

- Considere 9 hilos de ejecución que ejecutan el siguiente código

```
sem_init(&sem,0,1);

while (1) {
  sem_wait(&sem);
  { Critical Section }
  sem_post(&sem);
}
```

- y un hilo que ejecuta este otro código (total 10 hilos)

```
while (1) {
  sem_post(&sem);
  { Critical Section }
  sem_post(&sem);
}
```

- En un determinado instante ¿cuál es máximo número de hilos que pueden estar ejecutando la sección crítica?
  - 1
  - 2
  - 3
  - Ninguna de las anteriores

- Suponga que queremos sincronizar dos hilos concurrentes P y Q usando semáforos con el siguiente código

- Marque la opción correcta.

  Esta solución asegura:

  – Exclusión mutua
  – Libre de interbloqueo
  – Libre de inanición pero no de interbloqueo
  – Ninguna de las anteriores.

```
P:
while(1)
{
  P(S1);
  P(S2);
  Critical Section
  V(S1);
  V(S2);
}
```

```
Q:
while(1)
{
  P(S1);
  P(S2);
  Critical Section
  V(S1);
  V(S2);
}
```

Considere un bar que dispone de una terraza con una capacidad para 30 personas, una barra en la que hay dos camareros dispensando consumiciones y una puerta que se usa tanto de salida como de entrada por la que pueden pasar como máximo 4 personas simultáneamente. Suponga que cada cliente ejecuta el siguiente algoritmo:

```
Cliente() {
        Entrar()
        PedirConsumicion()
        ConsumirSentado()
        Salir()
}
```

Si en el sistema hay un número arbitrario de clientes, sincronice las actividades de los clientes utilizando semáforos de manera que un cliente no pueda pedir una consumición si no hay sitio libre en la terraza, las consumiciones se soliciten cuando hay un camarero disponible, y no hayan colisiones en la puerta.

Semáforos como contadores de recursos disponibles.

```
Semáforos globales
Sillas = 30; Barra = 2; Puerta = 4;

Cliente() {
        P(Sillas)
                P(Puerta)
                        Entrar()
                V(Puerta)
                P(Barra)
                        PedirConsumicion()
                V(Barra)
                ConsumirSentado()
                P(Puerta)
                        Salir()
                V(Puerta)
        V(Sillas)
}
```

Ejercicio

- Indique todos los posibles valores de las variables X e Y después de la ejecución concurrente de los siguiente hilos.

```
//valores iniciales de las variables
 int x=0, y=0;
 Semaphore: S1=0, S2=1, S3=0;
```

ThA	ThB	ThC

P(S3);
P(S2);
x = x + 1;
y = 2*x + y;
V(S2);
V(S1);
	P(S1);
	P(S2);
	x = 2*x;
	y = x + y;
	V(S2);
		P(S2);
		x = x - 2;
		y = x - y:
		V(S2);
		V(S3);

- Indique todos los posibles valores de las variables X e Y después de la ejecución concurrente de los siguiente hilos.

```
//valores iniciales de las variables
 int x=0, y=0;
 Semaphore: S1=0, S2=1, S3=0;
```

ThA	ThB	ThC

P(S3);
P(S2);
x = x + 1;
y = 2*x + y;
V(S2);
V(S1);
	P(S1);
	P(S2);
	x = 2*x;
	y = x + y;
	V(S2);
		P(S2);
		x = x - 2;
		y = x - y:
		V(S2);
		V(S3);

- ThC: X = -2, Y = -2
- ThA: X = -1, Y = -4
- ThB: X = -2, Y = -6