

2. Dado el programa *experiment.c* cuyo archivo ejecutable es “*experiment*”.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/wait.h>
5
6  int main(int argc, char *argv[]) {
7      pid_t  pid;
8      int    i, N=4;
9
10     for (i=0; i<N; i++)
11     { printf("[%d] Iteration %d\n", getpid(),i);
12       if (i%2 == 0)
13       {   pid = fork();
14           if (pid == 0) {
15               printf("[%d] Even iteration (%d)\n", getpid(),i);
16               break; }
17           else if (pid == -1) {
18               printf("Error creating the process\n");
19               exit(-1); }
20       }else
21       if (execlp("wc","wc","-l","experiment.c",NULL)<0)
22           printf("Error running wc\n");
23     }
24     while (wait(NULL)!=-1);
25     printf("[%d] Finished.\n", getpid());
26     exit(0);
27 }
```

Nota: *a % b* devuelve el resto de la división entera *a / b*.

(1.4 puntos=0,7+0,7)

a) Suponga que las llamadas al sistema no producen error e indique de forma justificada el número de procesos creados y el parentesco existe que entre ellos al ejecutar *experiment*

b) Suponga que el PID del proceso *experiment* es 100, y que el sistema operativo asigna los PIDs 101, 102, ... a sus posibles procesos hijos. Escriba una de las posibles secuencias que se mostraría en pantalla al ejecutar el programa.

3. Dado el programa *pro.c* cuyo archivo ejecutable es “*proc*”.

```

1 #include <stdio.h>
2 #include <sys/types.h>
3 #define N 3
4 int X=1; //global variable
5
6 int main(int argc,char *argv[]) {
7     pid_t pid;
8     int i,status;
9
10    for (i=0; i<N; i++) {
11        pid = fork();
12        if (pid==0) {
13            X=10+X*i;
14            sleep(10);
15            exit(X);
16        }
17    }
18    X=100+i;
19    while (wait(&status)>0)
20        printf("Process status: %d.\n", status/256);
21
22    printf("X value is: %d \n",X);
23    return(0);
24 }
```

(1.2 puntos=0,6+0,4+0.2)

a) Indique de forma justificada los procesos que se crean al ejecutar *proc*, el parentesco existente entre ellos y los mensajes que se imprimen en el terminal.

b) Dadas las instrucciones de las líneas 13, 14, 15 y 19, marque el estado(s) en que podría estar el proceso(s) tras ejecutar dicha instrucción y antes de ejecutar la siguiente instrucción

Línea	Suspendido	Preparado	En ejecución	Terminado	Zombie	Huerfano
13						
14						
15						
19						

c) Indique de forma justificada si puede haber o no condición de Carrera en el acceso a la variable global X

4. En un sistema de tiempo compartido su planificador a corto plazo consta de dos colas, una gestionada con SRTF (ColaS), en la que en caso de empate se utiliza el orden de llegada como criterio de desempate, y otra gestionada con Round Robin (ColaR) con un quantum de 2 unidades de tiempo ($q=2$ ut). Los procesos nuevos y los procedentes de E/S siempre van a la ColaR. La ColaR es la más prioritaria y un proceso es degradado a la cola ColaS tras consumir un quantum q en CPU. La gestión intercolas es por prioridades expulsivas. Todas las operaciones de E/S se realizan en un único dispositivo con política de servicio FCFS.

En este sistema se solicita ejecutar el siguiente grupo de trabajos:

Proceso	Instante de llegada	Ráfagas de CPU y E/S
A	0	5 CPU + 2 E/S + 1 CPU
B	1	4 CPU + 4 E/S + 4 CPU
C	2	3 CPU + 1 E/S + 5 CPU

(2.0puntos =1.4+ 0.6)

4 a)	Indique diagrama de uso de CPU, rellenando la tabla adjunta. Los procesos aparecerán en las colas ordenados en base al criterio de planificación, de forma que el proceso de la derecha será el primero. Además en cada instante de tiempo t ponga entre paréntesis (ej. (A)), en la cola de la que provenga, el proceso al que asigne la CPU.					
T	ColaR-->	ColaS-->	CPU	Cola E/S-->	E/S	Evento
0						
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						
19						
20						
21						
22						
23						

4 b)	Indique los tiempos de espera y de retorno de cada proceso		
		T.espera	T.retorno
	A		
	B		
	C		

5. Dado el siguiente programa *endth.c* cuyo ejecutable es *endth*.

(1.1 puntos=0.8 +0.3)

<pre>#include <... all headers...> pthread_t th1, th2; pthread_attr_t atrib; int N=0; void *Func2(void *arg) { int j=2; sleep(5); printf("END THREAD 2 \n"); pthread_exit(&j); }</pre>	<pre>void *Func1(void *arg) { int j=1; pthread_create(&th2, &atrib, Func2, NULL); N=N+j; sleep(20); printf("END THREAD 1 \n"); pthread_exit(&j); }</pre>
<pre>int main (int argc, char *argv[]) { int b; printf("START MAIN \n"); pthread_attr_init(&atrib); pthread_create(&th1, &atrib, Func1, NULL); N= N+10; sleep(10); pthread_join(th2, (void**) &b); printf("END MAIN \n"); exit(0); }</pre>	

5	a) Indique las cadenas que imprime el programa en la Terminal tras su ejecución. Justifique su respuesta
	b) Justifique si hay o no posibilidad de que ocurra una condición de carrera en el acceso a las variables N, j y b del programa.

6. Indique cuáles de las siguientes afirmaciones sobre sección crítica son ciertas y cuáles son falsas. Debe justificar adecuadamente sus respuestas, indicando qué cumplen, qué no cumplen y por qué:

(1.2 puntos=0.4+0.4+0.4)

6	<p>a) Si se utiliza un protocolo basado en semáforos para resolver el problema de la sección crítica, como se muestra a continuación, el valor inicial del semáforo S ha de ser 0</p> <div data-bbox="427 349 1173 477"><pre>Protocolo Entrada → P(S) Sección crítica(); Protocolo de salida → V(S)</pre></div> <p>b) No siempre podemos utilizar la solución hardware basada en las instrucciones de Deshabilitar/Habilitar interrupciones, que plantea el código siguiente:</p> <div data-bbox="422 808 1173 936"><pre>Protocolo Entrada → DI (Deshabilitar) Sección crítica(); Protocolo de salida → EI (Habilitar)</pre></div> <p>c) La solución basada en la instrucción <i>test_and_set</i> que aparece en el cuadro cumple las condiciones de Exclusión mutua, Progreso y Espera limitada, con independencia del planificador que se utilice</p> <div data-bbox="422 1296 1393 1453"><pre>//llave es una variable global e inicializada a 0 Protocolo Entrada → while (test_and_set(&llave)); Sección crítica(); Protocolo de salida → llave = 0;</pre></div>
---	---

7. El siguiente código es una variación del ejercicio de suma de filas visto en la práctica de threads. En concreto, en este programa son los hilos sumadores (función **AddRow**) y no el hilo **main** el que se encargan de sumar a la variable **total_addition** el resultado de la suma de la fila. Para ello, se han realizado las siguientes modificaciones:

- Se ha eliminado de la estructura **row** el elemento **addition**.
- La variable **total_addition** se ha declarado global.
- Se ha declarado también la variable local **local_addition** en **AddRow**.
- Se ha declarado el mutex **m**.

```

1  #include <...all_headers...>
2  #define DIMROW 1000000
3  #define NUMROWS 20
4  typedef struct row{
5      int vector[DIMROW];
6  } row;
7  row matrix[NUMROWS];
8  long total_addition = 0;
9  pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
11
12 void *AddRow( void *ptr ) {
13     int k;
14     int local_addition;
15     row *fi;
16     fi=(row *) ptr;
17     local_addition=0;
18     for(k=0;k<DIMROW;k++) {
19         total_addition += fi->vector[k];
20     }
21 }
22 int main() {
23     int i,j;
24     pthread_t  threads[NUMROWS];
25     for(i=0;i<NUMROWS;i++) {
26         for(j=0;j<DIMROW;j++) {
27             matrix[i].vector[j]=1;
28         }
29     }
30     pthread_attr_init( &atrib );
31     for(i=0;i<NUMROWS;i++) {
32         pthread_create(&threads[i],NULL,AddRow,(void *)&matrix[i]);
33     }
34     for(i=0;i<NUMROWS;i++) {
35         pthread_join(threads[i],NULL);
36     }
37     printf( "Total addition is: %ld\n",total_addition);
38 }

```

Responda a las siguientes cuestiones.

(1.6 puntos= 0.4+0.6+0.6)

7	a) Identifique, en dicho programa, la(s) línea(s) correspondientes a la sección crítica.
---	--

	<p>b) Utilice el mutex m para proteger la sección crítica e indique de forma justificada qué desventaja puede surgir a nivel de concurrencia con esta solución.</p>
	<p>c) Sin tener en cuenta los cambios realizados en el punto b, escriba el código necesario para modificar el programa de manera que: siga realizando la suma dentro del bucle pero usando la variable local_addition y después del bucle esta variable se sume a total_addition. Si es necesario, en su solución, proteja la sección crítica e indique de forma justificada qué ventaja tiene esta solución respecto a la propuesta en el apartado b.</p>