SURNAME		NAME		Group
ID		Signature		

- **Keep the exam sheets stapled.**
- **Write your answer inside the reserved space.**
- **Use clear and understandable writing. Answer briefly and precisely.**
- **The exam has 7 questions, everyone has its score specified.**

---

1. In each section, briefly define the first term that is cited and relate to it the sentence in C language or the shell command given below .

(1,5 points = 0,5 + 0,5+0,5)

1	**a)** System call (define) printf (" This message is printed on the standard output \n"); // in a program (relate)
	b) User and kernel execution modes (define) b= a / 0; // division by zero in a user program (relate)
	c) Timesharing (define) $ ls -la | wc -l (relate)

2. Given the program ***experiment.c*** whose executable file is ***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             }
21         } else 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*** returns the remainder of the integer division ***a / b***.

(1,4 points = 0,7 + 0,7)

	a) Supposing that the system calls used do not produce any error, justify the number of processes created and the relationship between them when executing *experiment*
	b) Suppose that the PID of process *experiment* process is 100, and that the operating system assigns PIDs 101, 102, ... to its possible child processes. Write one of the possible sequences that would be shown on the screen when running the program .

3. Given the ***pro.c*** program whose executable file is ***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 points = 0,6 + 0,4 + 0.2)

3 **a)** Explain the processes that are created when executing proc, the relationship between them and the message they print in the terminal.

b) Given the instructions in lines 13, 14, 15 and 19, mark the state(s) in which the process(es) could be after executing the line sentence and before executing the following line sentence.

Line	Suspended	Ready	Execution	Ended	Zombie	Orphan
13						
14						
15						
19						

c) Explain if there may or may not be a *race condition* in the access to global variable 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

(2.0 points = 1.4 + 0.6)

4 a) **I**ndique diagrama de uso de CPU, rellenando la tabla adjunta. Represente los procesos en las colas por orden de llegada a dicha cola, siendo el de la derecha el primero de la cola (al que se asignaría CPU en un algoritmo FCFS). 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 points=0.8 +0.3**)

```
#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);
}
```

```
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);
}
```

```
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);
}
```

5 **a)** Indique las cadenas que imprime el programa en la Terminal tras su ejecución. Justifique su respuesta

b) Indique si hay o no posibilidad de que ocurra una condición de carrera para cada una de las variables declaradas en dicho programa. Justifique su respuesta para las variables N, j y b.

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

(**1.2 points=0.4+0.4+0.4**)

6

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

```
Protocolo Entrada   → P(S)
                      Sección crítica();
Protocolo de salida → V(S)
```

b) No siempre podemos utilizar la solución hardware basada en las instrucciones de Deshabilitar/Habilitar interrupciones, que plantea el código siguiente:

```
Protocolo Entrada   → DI (Deshabilitar)
                      Sección crítica();
Protocolo de salida → EI (Habilitar)
```

c) La solución basada en la instrucción *test_and_set* que aparece en el cuadro cumple las condiciones de Exclusión mutua, Progreso y Espera limitada, con independencia del planificador que se utilice

```
//llave es una variable global e inicializada a 0
Protocolo Entrada   → while (test_and_set(&llave));
                      Sección crítica();
Protocolo de salida → llave = 0;
```

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 points= 0.4+0.6+0.6)

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

	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.
	c) Sin tener en cuenta los cambios realizados en el point 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.