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. Terms shell, system calls, time sharing and software interrupt or trap are essential when it comes to an operating system like UNIX. Write a brief description of every term and indicate what they are useful for.

(**1.2 points** = 0,3 + 0,3 + 0,3 + 0,3)

1	**a)** Shell (define and explain its utility)
	b) System call (define and explain its utility)
	c) Time sharing (define and explain its utility)
	d) Software interrupt or trap (define and explain its utility)

2. Given the following C program named Test.c, that has generated the executable file named "Test":

```
 1 /*** Test.c ***/
 2 #include "all required: stdio.h, stdlib.h, unistd.h"
 3
 4 int main(int argc, char *argv[]) {
 5   pid_ t val;
 6
 7   if (argc==1) {
 8     if (execl("/bin/ls", "ls","-la",NULL)<0) {
 9       printf ("Message 1\n",1);  exit(1);
10     }
11   } else if (argc==2) {
12     val= fork();
13     if (execl("/bin/cat", "cat",argv[1],NULL)<0) {
14       printf("Message 2\n");  exit(2);
15     }
16   }
17   while (wait(NULL)!=-1) printf ("waiting \n");
18   printf("Message 3\n");
19   exit(0);
20 }
```

Explain how many processes are created, the relationship between them and the information displayed on standard output by its execution with the following commands:

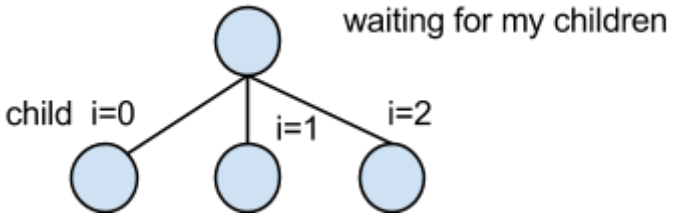
(1.5 points = 0,75 + 0,75**)**

2	**a)** `$ ./Test`
	b) `$ ./Test Test.c`

3. Complete the following code in order to create the process scheme shown on the picture, in such a way that:

(a) All children have to stay zombies for about 10 seconds, they have to end calling to exit(i), where "i" is the variable that controls the children creation loop.

(b) The parent must wait for all his children and then it has to show on the terminal the value passed by every child through exit().



(**1.4 points**)

```
3 /*** Example.c ***/
  #include "all required.h"
  #define N 3
  int main(int argc, char *argv[]) {
    int i, status;
    pid_t val;

    printf("Parent \n");
    for(i=0; i<N; i++) {









    }









    printf("END \n");
    exit(0);
  }
```

4. A short term scheduler on a timesharing system has two queues: F-Queue that is FCFS and R-Queue that is Round Robin with q = 10ut. New processes and those coming from I/O always go to R-Queue. R-Queue has more priority than F-Queue, a process is demoted to F-Queue after consuming a CPU quantum. Inter-queue scheduling is preemptive priorities. All I/O operations are performed on a single device with FCFS queue. In this system the following set of processes is requested to run:

Process	Arrival time	CPU and I/O bursts
A	0	30 CPU + 10 I/O + 30 CPU + 10 I/O + 30 CPU
B	2	10 CPU + 30 I/O + 10 CPU + 30 I/O + 10 CPU
C	4	20 CPU + 20 I/O + 20 CPU + 20 I/O +20 CPU

a) Obtain the execution timeline, filling the table below. Notice that time intervals are 10 ut .

(**2.3 points** = 1.3 + 0.6 + 0.4)

T	R-Queue	F-Queue	CPU	I/O Queue	I/O	Event
0						`Arrival A(0),B(2),C(4)`
10						
20						
30						
40						
50						
60						
70						
80						
90						
100						
110						
120						
130						
140						
150						
160						
170						
180						
190						
200						

4 **b)**	Obtain the waiting times and the turnaround times for every process.

4 **c)**	Briefly explain if the scheduler gives some kind of preference to CPU bound processes or I/O bound processes

5. Access protocols to the critical section should meet three requirements: mutual exclusion, progress and limited waiting. Indicate for each of the following proposals if mutual exclusion is verified (YES) or not (NO). In case of verifying mutual exclusion then answer if the other two requirements are met or not.

(**1.6 points** = 0,4 + 0,4 + 0,4 + 0,4)

5		Mutual exclusion	Progress	Limited waiting

	`semaphore S=1 // shared variable with FIFO queue`
	`Input protocol -> P(S)`
	`  Critical section();`
	`Output protocol -> V(S)`			

	`int key = 0;  // shared variable`
	`Input protocol -> while (key ==1);`
	`  key = 1;`
	`  Critical section();`
	`Output protocol -> key = 0;`			

	`// Hardware solution`
	`Input protocol -> DI; // Disable interrupts`
	`  Critical section();`
	`Output protocol -> EI;  // Enable interrupts`			

	`int llave=0;  // shared variable`
	`Input protocol -> while (test_and_set(&key));`
	`  Critical section();`
	`Output protocol -> key = 0;`			

6. A swimming pool has set a maximum number of 100 simultaneous swimmers. A swimmer has to perform operation GetHat() in order to be able to do Swim(). A "swimmer" is implemented as a thread that executes function FuncSwim(). There is a thread "collector" that puts hats on a shelf executing function FuncCollect(). Add the required operations on the semaphores declared and initialized in main(), so the following code comply with these requirements:

- Both functions GetHat() and PutHat() are executed in mutual exclusion.
- A “swimmer” thread has to call to GetHat() before calling to Swim().
- The hat shelf size is 100.
- Thread “collector” has to be able to call to PutHat() if there is a gap on the hat shelf (number of hats < 100)
- If the hat shelf if full then “collector” thread has to be suspended.
- The maximum number of “swimmer” threads simultaneously on Swim() is 100.
- Every time a “swimmer” ends Swim() another one will do Swin() if it has already a hat.
- If there are 100 “swimmer”s on Swim() then another one trying to Swim() has to be suspended.

Note. You can use for semaphore operations Disktra notation P() and V() or POSIX sem_wait(), sem_post().

(1.0 point)

6

```
#include <all requred...>
 sem_t swim_swimmers, hats, mutex;    // semaphores declaration

int main(int argc, char *argv[]) {
    pthread_attr_t attr;
    pthread_t swimmer[300], collector;
    int i;
    pthread_attr_init(&attr);
    sem_init(&swim_swimmers,0,0); sem_init(&hats,0,100); sem_init(&mutex,0,1);
    pthread_create(&collector, &attr, FuncCollect, NULL);
    for (i = 0; i < 300; i++) pthread_create(&swimmer[i], &attr, FuncSwim, NULL);
    for (i = 0; i < 300; i++) pthread_join(swimmer[i], NULL);
}
void *FuncSwim(void *arg) {
    // Complete to comply with enunciate requirements


    GetHat();


    Swim();

}
void *FuncCollect(void *arg) {
    // Complete to comply with enunciate requirements
    while(1) {



        PutHat();



    }
}
```

7. Obtain the strings that will be printed on the terminal after running the following program. Explain your answer.

(1.0 point)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

pthread_t  th1, th2;
pthread_attr_t atrib;

void *Func_th1(void *arg) {
  int i,j;
  i= *((int *)arg);
  j=1;
  sleep(10+i);
  pthread_join(th2,NULL);
  printf("th1 is awake\n");
  pthread_exit(&j);
}
```

```
void *Func_th2(void *arg) {
  int i,j;
  i= *((int *)arg);
  j=2;
  sleep(20+i);
  printf("th2 is awake\n");
  pthread_exit(&j);
}
```

```
int main (int argc, char *argv[]) {
  int i;

  pthread_attr_init(&atrib);
  printf("Pthread message: \n");
  i= rand();       // function that provides a random number
  pthread_create(&th1, &atrib, Func_th1,&i);
  pthread_create(&th2, &atrib, Func_th2,&i);

  printf("END \n");
  exit(0);
}
```

7	