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 9 questions, everyone has its score specified.**

1. A computer manages its 1MB of logical memory space by paging with 512-byte pages. The memory map of a given process reserves space for three contiguous regions: one for 1100 instructions of 4 byte each instruction (all contiguous), another for 130 global variables of 8 byte each variable (also contiguous) and a third region for the stack. The system dynamically allocates the minimum number of pages for the stack

(1,2 points = 0,3 + 0,3 + 0,3 + 0,3)

1

a) How many bits are reserved for the page id in the logical address ?

b) How many pages are needed to store the instructions of this process? And how many for its global variables? ?

c) By the time the stack has grown up to 1,800 bytes, how many entries in the process page table will be valid?

d) Compute the internal fragmentation in bytes at the time instant considered on item **c)**

2. Let consider a timesharing system with a short-term scheduler with a single ready queue. At time t = 0 processes A, B and C arrive in this order to the ready queue. All three processes are CPU bounded with the same execution profile: :

1000 CPU	10 E/S	1000 CPU	10 E/S	10 CPU

Obtain with explanation the turnaround time and the waiting time for every process (A, B and C) on the following cases:

(1,2 points = 0,5 + 0,5 + 0,2)

2 **a)** FCFS scheduling

Turnaround time			Waiting time		
A=	B=	C=	A=	B=	C=

b) RR with q = 10 ut

Turnaround time			Waiting time		
A=	B=	C=	A=	B=	C=

c) Taking into consideration the cost for context switching, explain what algorithm FCFS or RR gives better performance in this situation.

3. Given the following code that has generated the executable file named "Example1":

```
1  /*** Example1.c ***/
2  #include "all_required_headers.h"
3  #define N 2
4
5  main() {
6    int i=0, j=0;
7    pid_t pid;
8    while (i<N) {
9      pid = fork();
10     if (pid ==0) {
11       for (j=0; j<i ; j++) {
12          fork();
13          printf("Child i=%d, j=%d \n", i, j);
14          exit(0);
15       }
16     } else {
17       printf("Parent i=%d, j=%d \n", i, j);
18       while (wait(NULL) != -1);
19     }
20     i++;
21   }
22   exit(0);
23 }
```

Answer with enough explanation the following items:

(1,2 points = 0,6 + 0,6)

3	**a)** The number of processes that are generated and draw their parentship graph.
	b) Write the messages that are printed on the screen when executing "Example1".

4. The following program `threads.c` includes function `rever()` that reverses the string passed as argument:

```
 1 #include "all_required_headers.h"
 2 #define ARG_MAX 20
 3 int done=0;
 4
 5 void *rever( void *ptr ){
 6  int i;
 7  char temp;
 8  char *pfirst= (char*) ptr;
 9  char *plast= pfirst+strlen(pfirst)-1;
10  while (plast > pfirst){
11    temp= *pfirst;
12    *pfirst= *plast;  *plast= temp;
13    pfirst++; plast--;
14  }
15  done++;
16 }
17 int main(int argc, char* argv[]){
18  int i;
19  pthread_t thr[ARG_MAX];
20  pthread_attr_t atrib;
21  pthread_attr_init(&atrib);
22
23  for(i=1;i<argc;i++)
24    pthread_create(&thr[i],&atrib,
                rever,(void *)argv[i]);
25
26  for(i=argc-1;i>0;i--){
27    pthread_join(thr[i],NULL);
28    printf("%s ",argv[i]);
29  }
30 }
```

Answer the following items considering that the program is compiled and executed as follows: :

```
gcc threads.c -o threads -lpthread
./threads once upon a time
```

(1,0 points = 0,25 + 0,25 + 0,25 + 0,25)

4

a) What is the maximum number of threads that execute concurrently?

b) What is shown in the standard output after the program execution?

c) How could affect the output generated by the program if removing line 27?

d) If in the original program line 27 is removed and the following line is inserted into line 25:

```
while (done < argc-1);
```

Would it work properly, resulting in the same output than the original program?

5. It is wanted to manage the entrance to an amusement park limited to 5,000 people. To do this a process implements the park entering and leaving tasks by two threads that execute functions "func_enter" and "func_leave" respectively. These two functions have to monitor that no more entries will happen when the park capacity is reached and no outputs are accounted when there is nobody inside the park. In addition we want to know at every moment the actual number of visitor inside the park, that will be stored on variable “visitors” that the threads will modify properly.

To solve the synchronization problems that arise three semaphores are used:

- mutex : controls critical section access
- vacants : indicates the number of available entrances
- people : indicates the number of visitors inside the park

a) Complete the code for the main program initializing the defined semaphores.

b) Complete the code for functions "func_enter" and "func_leave", performing the required operations on the appropriate semaphores.

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

5

a)

```
#include <semaphore.h>
#define N 5000
int visitors;
sem_t mutex, vacants, people;
pthread_t enter, leave;
void main () {
  // Semaphore initialization




  //
  pthread_create(&enter, NULL , func_enter, NULL);
  pthread_create(&leave, NULL , func_leave, NULL);
  ...
}
```

b)

```
void *func_enter(void *p) {
  while (1) {



    visitors = visitors + 1;



  }
}
```

```
void *fun_leave(void *p) {
  while (1) {



    visitors = visitors - 1;



  }
}
```

6. Given a system with **virtual memory** by **demand paging** and **GLOBAL LRU replacement** algorithm, pages are 4KB, main memory is 40 KB (10 frames) and the maximum size of a process is 16KB (4 pages). The LRU algorithm is implemented using counters, annotating the time instant when the page is referenced. The allocation of frames in main memory is in ascending order. At t = 10, there are 3 processes with the following content on their page tables:

Page table for process A			
Page	**Frame**	**Counter**	**Val. bit**
0	2	4	v
1	5	1	v
2			i
3	1	5	v

Page table for process B			
Page	**Frame**	**Counter**	**Val. bit**
0	0	2	v
1	3	3	v
2			i
3			i

Page table for process C			
Page.	**Frame**	**Counter**	**Val. bit**
0	4	10	v
1	8	7	v
2	9	8	v
3			i

Consider the state of the system at t = 10 and answer the following questions:

a) Indicate the process and page number occupying every main memory frame. Fill in the table with the following format: A0 corresponds to page 0 from process A .

b) From t = 10, the following logical addresses are referenced (**addresses in hexadecimal**): (A, 2F2C), (C, 3001), (A, 11C1), (B, 3152). Fill the content of main memory and page tables at the end of the reference sequence.

(1,2 points = 0,6 + 0,6)

6

(TABLE item a))	
Frame	
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

(TABLE item b))	
Frame	
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

PAGE TABLES FOR PROCESSESS A, B y C (item b))

Page table for process A			
Page	**Frame**	**Count**	**Val. bit**

Page table for process A			
Page	**Frame**	**Count**	**Val. bit**

Page table for process A			
Page	**Frame**	**Count**	**Val. bit**

7. Assume the following C code, that creates 3 processes that communicate using 3 pipes, executes without errors. Answer to the following sections with enough explanation: :

```
   /*** pipes.c   ***/
 1 #include "all_required_headers.h"
 2 #define nproc 3
 3 #define nt 2
 4
 5 int main(int argc,char* argv[]) {
 6   int pipes[nproc][2];
 7   int i, j, pid, i_1;
 8
 9   for (i=0; i<nproc; i++)
10 pipe(pipes[i]);
11   for (i=0; i<nproc; i++) {
12     pid = fork();
13     if (pid == 0) {
14      dup2(pipes[i][1],STDOUT_FILENO);
15      i_1 = i_previous(i, nproc);
16      dup2(pipes[i_1][0],STDIN_FILENO);
17      for (j=0; j<nproc; j++) {
18        close(pipes[j][0]);
19        close(pipes[j][1]);
20       }
21      comp_and_com(i, nt);
22      exit(0);
23     }
24   }
25   for (i=0; i<nproc; i++) {
26     close(pipes[i][0]);
27     close(pipes[i][1]);
28    }
29   while (wait(NULL) != -1);
30   exit(0);
   }
```

```
31 int comp_and_com(id, nturns) {
32   int v, n;
33   n = 0;
34   while (n < nturns) {
35     if (id == 0 && n == 0) {
36       v = 0;
37       n++;
38       write(STDOUT_FILENO,  &v, sizeof(v));
39     } else {
40       read(STDIN_FILENO,  &v, sizeof(v));
41       v = v + 1;
42       n++;
43       write(STDOUT_FILENO, &v, sizeof(v));
44     }
45   }
46   if (id == 0) {
47     read(STDIN_FILENO,  &v, sizeof(v));
48     fprintf(stderr,"V FINAL VALUE: %d\n",
49 v);
50   }
51   return 0;
52 }
53
54 int i_previous(int i, int np) {
55   if (i > 0) return (i-1)
56   else if (i == 0) return (np-1);
57 }
```

(1,0 points = 0,6 + 0,4)

7 **a)** Draw the communication diagram that is generated between the processes created with `fork()`

b) Explain what will be printed by statement `fprintf` on line 48?

8. Given the following content listing of a directory on a POSIX system :

```
drwxr-xr-x   2 pepe     student         4096 ene  8     2013  .
drwxr-xr-x  11 pepe     student         4096 ene 10     14:39 ..
-rwsr-xr-x   1 pepe     student      1139706 ene  9     2013  copy
-rwxr-sr--   1 pepe     student      1139706 ene  9     2013  append
-rw-------   1 pepe     student       634310 ene  9     2013  f1
-rw----rw-   1 pepe     student       104157 ene  9     2013  f2
-rw-rw----   1 pepe     student       634310 ene  9     2013  f3
```

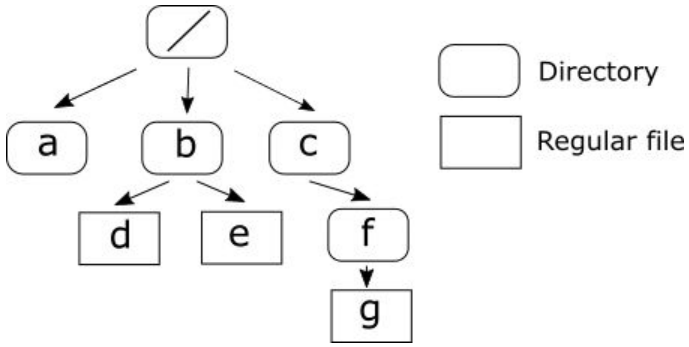
Where `append` and `copy` programs require the names of two files as arguments. The program `append` appends at the end of the file passed as second argument the content first argument file, while `copy` just replaces the content of the file passed as the second argument by the content of the first. Fill in the table and indicate what are the permissions required for the command to succeed and, in case of error, which is the missing permission.

(1,0 points = 0,25 + 0,25 + 0,25 + 0,25)

8

User	Group	Command	Does it work?
ana	**teachers**	**./append f1 f2**	
Explain			
ana	**teachers**	**./copy f2 f4**	
Explain			
pau	**student**	**./append f2 f3**	
Explain			
pepe	**student**	**./copy f3 f1**	
Explain			

9. A Minix file system contains the following file tree:



(1,2 points = 0,5 + 0,7)

9

a) Indicate the content of every directory in this file system, assuming that the i-nodes occupied by every element are : a(4), b(5), c(8), d(12), e(22), f(10) and g(16)

/		a		b		c		f	

b) Consider zones of 1 KByte, 32-bit zone pointers, and i-nodes with 7 direct pointers, 1 single indirect pointer and 1 double indirect pointer, obtain the number of zones occupied by the *d* file if its size is 2 MBytes.