UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Departamento de Informática de Sistemas y Computadores (DISCA)

fSO

FINAL exam

Januray 23th, 2020

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

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.**
- **Remember that you have to appropriately explain your answers to get the full corresponding score**

1. Considering the execution without errors of the following code, answer the following questions: :

(1,2 points = 0,6 + 0,4 + 0,2)

```
#include <all_needed.h>
int main() {
    int i;
    pid_t pid;

    for (i=0; i<2; i++) {
        pid = fork();
        if (pid != 0) {
            sleep(10 - 4 * i));
            break;
        }
    }
    if (i == 2) sleep(2);
    while (wait(NULL) =! -1);
    exit(0);
}
```

1

a) ¿How many processes are created along its execution? Draw the relationship diagram between them.

b) ¿What processes stay on zombie state?¿for how long?

c) Considering a multiprogrammed system and that the execution time of the instructions is negligible compared to sleep times, how long does the program take to run?

2. Be a timeshare system with a short-term scheduler with two ready queues: Queue0 managed by Round Robin with quantum q = 1ut and Queue1 managed by FCFS. The policy between queues is preemptive priorities, having Queue0 the highest priority. New processes and those that come from I/O access go to Queue0, and demote to Queue1 when they consume a CPU quantum while not having ended the actual CPU burst. If several events occur at the same time, they are ordered as: new, coming from I/O and quantum end. There is a single I/O device managed by FCFS. Three processes, A, B and C, arrive to this system whose arrival times and standing alone execution profiles are the following: :

Process	Arrival time	Standing alone profile
A	0	2 CPU + 3 I/O + 4 CPU
B	1	1 CPU + 1 I/O + 2 CPU
C	2	4 CPU + 2 I/O + 1 CPU

(1,6 points = 1,2 + 0,4)

a) Fill the following execution time diagram:

T	Queue1	Queue0	CPU	I/O queue	I/O	Event
0						
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						

b) Indicate turnaround time and waiting time for every process.

	Process A	Process B	Process C
Turnaround time			
Waiting time			

3. In the next piece of code lines 7, 13 and 25 are completed each one with one instruction.

(1,2 points = 0,4 + 0,4 + 0,4)

```
1  #include <all_needed.h>
2
3  int V = 0;
4  pthread_t thr1, thr2;
5
6  void *thread1( void *ptr ){
7     /**complete**/
8     V = 1;
9     printf("T1 V=%d\n",V);
10 }
11
12 void *thread2( void *ptr ){
13    /**complete**/
14    V = V + 10;
15    printf("T2 V=%d\n",V);
16 }
17
18 int main(int argc, char* argv[]){
19    pthread_attr_t atrib;
20    pthread_attr_init(&atrib);
21
22    pthread_create(&thr1,&atrib, thread1, NULL);
23    pthread_create(&thr2,&atrib, thread2, NULL);
24
25    /**complete**/
26    V = V + 100;
27    printf("Main V=%d\n",V);
28 }
...
```

Several possibilities are proposed below to complete these lines. Indicate for every possibility the messages that will be displayed on the screen after the code execution explaining your answer .

a) line 7 - sleep(1);
line 13 - sleep(3);
line 25 - sleep(2);

b) line 7 - sleep(5);
line 13 - sleep(2);
line 25 - pthread_exit(0);

c) line 7 - sleep(5);
line 13 - pthread_join(thr1,NULL);
line 25 - pthread_join(thr2,NULL);

4. Given the following code that uses POSIX threads and semaphores:

```
#include <semaphore.h>

sem_t sem_A, sem_B;

void *compute(void *param) {

   ...
}
```

```
int main(void) {

   phtread_t th[10];
   phtread_attr_t attr;
   int n;
   sem_t sem_C;

   sem_init(&sem_A, 0, 1);
   sem_init(&sem_B, 0, 0);
   sem_init(&sem_C, 0, 4);

   pthread_attr_init(&attr);
   for (n=0; n<10; n++) {
      pthread_create(&th[n], &attr, compute, NULL);
   }

   ...
}
```

The programmer wishes to use the declared semaphores to solve different issues that will arise when the threads that are created are executed concurrently.

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

4	**a)** Indicate three situations or three purposes for which semaphores are useful in concurrent programming.
	b) Considering both the area in which semaphores *semA*, *semB* and *semC* have been declared, as well as their initialization in the proposed code, and taking into account that semaphores will be used by threads *th[n]* inside **compute()** function, indicate which semaphores *semA*, *semB* or *semC* would be adequate for each of the situations described on the previous section a).

5. Taking into account the process inheritance mechanism in Unix and the POSIX calls, answer to the following sections: **(1,2 points = 0,6 + 0,6)**

5

a) Assuming that there are no errors in system calls, complete the following C program with the necessary instructions and system calls (one underlined line number) so that the next command line be executed:

$ cat < f1 2> ferr | grep ".c" > f2

```
1  #include <unistd.h>
2. #include <fcntl.h>
3. #define readfile O_RDONLY
4. #define newfile (O_RDWR | O_CREAT | O_TRUNC)
5. #define mode644 (S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)
6. int main() {
7.    int pipeA[2];
8.    int fd1,fd2;
9.
10.   if (fork()) {
11.       fd1 = open("ferr", newfile, mode644);
12.       fd2 = open("f1",readfile);
13.
14.
15.
16.       close(pipeA[0]); close(pipeA[1]); close(fd1); close(fd2);
17.       execlp("cat", "cat", NULL);
18.    } else {
19.       fd1 = open("f2",newfile, mode644);
20.
21.
22.       close(pipeA[0]); close(pipeA[1]);close(fd1);
23.       execlp("grep", "grep", ".c", NULL);
24.       }
25.    }
26.    return 0;
27. }
```

b) Fill in the file descriptor tables of the parent process after executing line 15 and the child process after executing line 21. Tables content has to be consistent with section a) requirements and implementation.

Parent file descriptor table on line **15**	
0	
1	
2	
3	
4	
5	
6	
7	

Child file descriptor table on line **21**	
0	
1	
2	
3	
4	
5	
6	
7	

6. Assume a 1-GByte partition formatted with a Minix file system with the following features:

- 32-byte i-nodes with 7 direct pointer to zones, 1 indirect and 1 double indirect
- 16-bit (2 bytes) pointers to zone
- 16-byte (14 for name and 2 for i-node) directory entries
- **1 Block = 1 Zone = 2 KBytes**

(1,2 points = 0,8 + 0,4)

6

a) Obtain the sizes of each header element considering a maximum number of 32768 (32K) i-nodes.

Boot block	Superblock	i-nodes map	Zone map	i-nodes	Data area

b) The file system considered contains the root directory with 30 empty directories and 30 files. Half of the files (15) are regular files and the other half are symbolic links to those regular files. Explain the number of occupied i-nodes.

Explain the size in bytes of the root directory.

7. A system with demand paging with two level paging has a maximum process size of 4GBytes, a page size of 4KBytes and a total of 4096 entries at the first paging level. The following table shows information to instant t = 25 for process P and process S. Process S is an operating system process that has been allocated in memory into frames with the highest addresses available on this system.

Process	Frame	Page	Last access time	Valid bit)
P	0x4A000	0xC71FF	5	1
P	0x4A001	0xC7200	10	1
P	-	0xA70C0	-	0
P	0x4A003	0xA73DC	15	1
S	0xFFFFE	0xB7001	20	1
S	0xFFFFF	0xB7002	25	1

Relying on the information provided answer the following sections:

(1,1 points = 0,5 + 0,2 + 0,4)

7

a) Logical address format and physical address format **indicating name and number of bits for every field**:

Logical address

Physical address

b) Obtain the maximum number of 2nd level page descriptors that process P can use.

c) Obtain the corresponding Physical Addresses, as well as if there is a page fault, when accessing to the Logical Addresses indicated below:

Process → Logical address	***Physical address (or Page Fault)***
P→ 0xA70C0102	
P→ 0xA73DC102	
S→ 0xB7000102	
S→ 0x B7001003	

8. Consider the system described in question 7 and the initial assignment of frames detailed in the table for the instant t = 25. Assume that demand paging is used with **LRU replacement policy, managed by counters**, with **LOCAL scope**, and that the system assigns 4 frames to process P and 2 to process S.

(1,5 points = 0,2 + 1,0 + 0,3)

8

a) From t = 26 processes P and S emit the following logical addresses sequence (in hexadecimal):
P:A70C0102, P:A74D0F02, P:A73DC102, P:C7200C10, P:C7200C11, P:A70C0102, S:B7003A00, S:B7001000

Obtain the reference string corresponding to the previous sequence:

b) Fill in the following table with the evolution of Main Memory content for the reference string obtained on the previous section. In each box, write the corresponding page at the top and the counter value at the bottom. You can fill in only the boxes where there is a change. .

Page→	Initial allocation							
Frame	t= 25	t=	t=	t=	t=	t=	t=	t=
4A000								
4A001								
4A002								
4A003								
FFFFE								
FFFFF								
Mark page faults and replacements:								

TOTAL NUMBER OF PAGE FAULTS =

c) Explain if the LRU replacement policy has achieved the optimal frame management, that is, if the number of page faults has been the same as the optimal replacement algorithm would have obtained.