Operating Systems
Final exam 26th June 2014

Bachelor's Degree in Computer Science and Engineering

Rules:

- Read the exam carefully before starting
- Mobile phones, electronic devices, books and notes are not allowed.
- Mobile phones have to be completely disconnected
- Write the exam using a pen (not a pencil)
- **Exam duration is 150 minutes**
- **Use a different page for each exercise. If you don't answer one question, use a white page with the exercise number and your name.**

Exercise 1 (25 points). QUIZ

Answer the Quiz questions writing the letters of each correct answer. Three wrong answers deduct a correct answer. Non answered questions don't deduct any points (they are ignored by the reviewer).

NAME: **GROUP:**

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

SOLUTION

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
B	C	C	A	A	A	C	A	C	C	B	D	A	C	C	A	C	B	A	D

Exercise 2 (25 points).

In a given real-time system the processes are executed according their priority. This system executes the processes depicted in the following table. For each process, it is shown the arrival time, execution time and priority.

PROCESS	ARRIVAL TIME	EXECUTION TIME	PRIORITY
P1	0	0,5	2
P2	0,2	0,3	3
P3	0,3	0,4	1
P4	0,5	1	3
P5	0,7	0,6	2

NOTE1: When the priorities are the same, the FCFS algorithm is used.

NOTE2: The smaller numerical value, the bigger priority.

A. Consider two different scenarios: (A) preemptive and (B) non-preemptive scheduling. For each scenario obtain the execution timeline of all the processes and the waiting time and turnaround time of each process.

Example: Pa is executed at 0,7seg during 0.2 seconds, and Pb is executed at 0,9seg during 0.6 seconds.

0 0,1 0,2 0,3 0,4 0,5 0,6 0,7 0,8 0,9 1,0 1,1 1,2 1,3 1,4 1,5 1,6 1,7 1,8 --- 2,8

Pa

Pb

(A) pre-emptive scheduling

0 0,1 0,2 0,3 0,4 0,5 0,6 0,7 0,8 0,9 1,0 1,1 1,2 1,3 1,4 1,5 1,6 1,7 1,8 --- 2,8

P1

P2

P3

P4

P5

PROCESS	TURNAROUND TIME	WAITING-TIME
P1		
P2		
P3		
P4		
P5		

(B) non-preemptive scheduling

0 0,1 0,2 0,3 0,4 0,5 0,6 0,7 0,8 0,9 1,0 1,1 1,2 1,3 1,4 1,5 1,6 1,7 1,8 --- 2,8

P1

P2

P3

P4

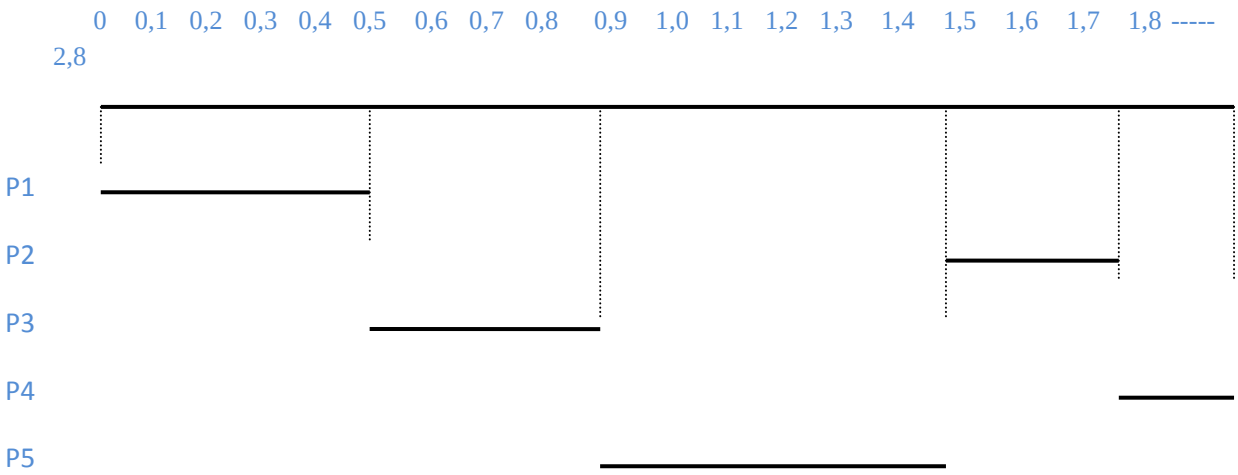
P5

PROCESS	TURNAROUND TIME	WAITING-TIME
P1		
P2		
P3		
P4		
P5		

B. Briefly describe the problem that could appear when a fixed-priority scheduling algorithm is employed. Describe a solution that can be used to solve this problem.

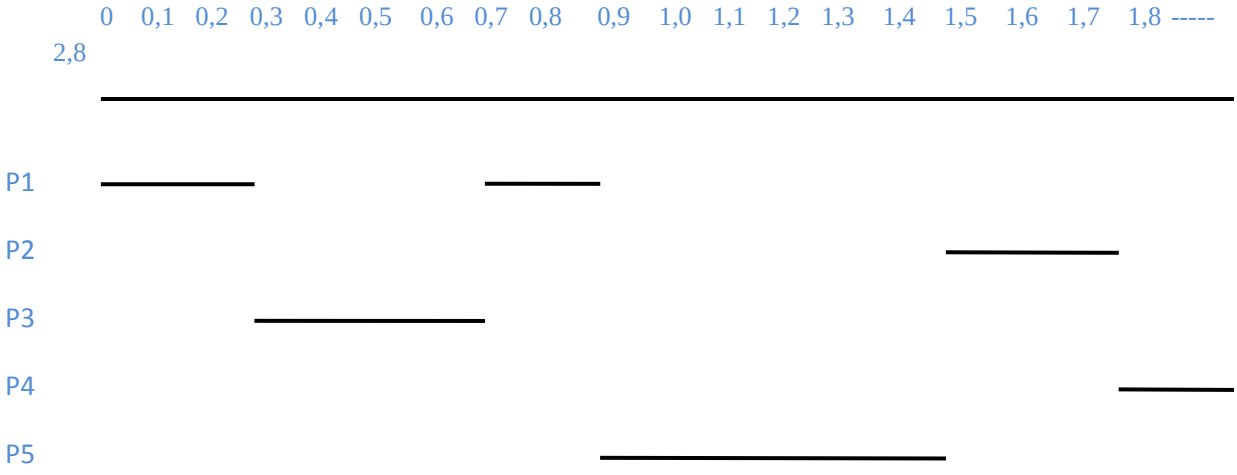
SOLUCION

a) Non-preemtive.



PROCESS	TURNAROUND TIME	WAIT TIME
P1	0,5	0
P2	1,6	1,3
P3	0,6	0,2
P4	2,3	1,3
P5	0,8	0,2

b) Preemptive



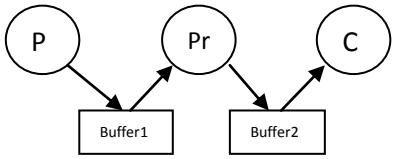
PROCESS	TURNAROUND TIME	WAIT TIME
P1	0,9	0,4
P2	1,6	1,3
P3	0,4	0
P4	2,3	1,3
P5	0,8	0,2

c) The problem arises when a large number of high-priority processes are executed, leaving the low-priority processes in the wait state. The solution is the aging technique that allows to increase the priority of the waiting process.

Exercise 3 (25 points).

Design a thread-based pipeline consisting of three phases: producer, processor and consumer. This design uses two buffers to exchange data.

- You need to design and implement the thread synchronization by means of **condition variables**. You need to modify the provided code adding: (1) the variable declaration and (2) the instructions and function calls necessary to achieve the required objective.
- Show the code section in which a deadlock can be produced. Justify how the proposed design avoids the deadlock.




```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#define MAX_BUFFER 250
int buffer1[MAX_BUFFER];
int buffer2[MAX_BUFFER];

void *producer(void *arg) {
        int * pdatos = (int*) arg;  // Pointer to the parameter
       int data_to_produce= *pdatos; // Number of data to produce
        int pos = 0;

        for (int i = 0; i < data_to_produce; i++) {
                …
                buffer1[pos] = i;
                …
        }
}

void *processor(void *arg) {
        int pos1=0, pos2=0, received_data, processed_data;
        int data_to_process = *(int *)arg;

        for (int i = 0; i < data_to_process; i++) {
                …
                processed_data = processing_function(received_data);
                …
        }
}

void *consumer(void *arg) {
        int data_to_consume = *(int *)arg;
        int pos = 0;

        for (int i = 0; i < data_to_consume; i++) {
                …
                printf("< Consuming: %d\n",buffer2[pos]);
                …
        }
}

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

        pthread_t th1, th2, th3;      int num_items = 1000;

        pthread_create(&th1, NULL, producer, (void *) &num_items);
        pthread_create(&th2, NULL, processor, (void *) &num_items);
        pthread_create(&th3, NULL, consumer, (void *) &num_items);

        pthread_join(th1, NULL); pthread_join(th2, NULL);
        pthread_join(th3, NULL);
```

```
        printf("Main thread exiting\n");
        return 0; }
```

SOLUTION

A)

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

#define MAX_BUFFER 250

int funcion_procesamiento(int);

int buffer1[MAX_BUFFER]; // El primer buffer se usa para pasar datos del productor
al procesador
int buffer2[MAX_BUFFER]; // El segundo buffer se usa para pasar los datos del
procesador al consumidor


int cont1 = 0, cont2 = 0;  // Variables compartidas. Contadores de items
actualmente en cada buffer.

pthread_cond_t no_lleno1;
pthread_cond_t no_vacio1;
pthread_mutex_t mutex1;

pthread_cond_t no_lleno2;
pthread_cond_t no_vacio2;
pthread_mutex_t mutex2;

void *productor(void *arg) {
       int datos_a_producir = *(int *) arg;
       int pos = 0;

       for (int i = 0; i < datos_a_producir; i++) {

              pthread_mutex_lock(&mutex1);

              while (cont1 == MAX_BUFFER)
                     pthread_cond_wait(&no_lleno1, &mutex1);

              buffer1[pos] = i;
              printf("> Produciendo: %d\n",i); fflush(stdout);

              pos = (pos + 1) % MAX_BUFFER;
              cont1++;

              pthread_cond_signal(&no_vacio1);
              pthread_mutex_unlock(&mutex1);
       }

       printf("Productor: No más items.Saliendo...\n");
}

void *procesador(void *arg) {
       int dato_recibido, dato_procesado, pos1=0, pos2=0;
       int datos_a_procesar = *(int *)arg;

       for (int i = 0; i < datos_a_procesar; i++) {
```

```
		pthread_mutex_lock(&mutex1);

		while (cont1 == 0)
			pthread_cond_wait(&no_vacio1, &mutex1);
		dato_recibido = buffer1[pos1];

		pos1 = (pos1 + 1) % MAX_BUFFER;
		cont1--;

		pthread_cond_signal(&no_lleno1);
		pthread_mutex_unlock(&mutex1);

		dato_procesado = funcion_procesamiento(dato_recibido);


		pthread_mutex_lock(&mutex2);

		while (cont2 == MAX_BUFFER)
			pthread_cond_wait(&no_lleno2, &mutex2);
		buffer2[pos2] = dato_procesado;
		pos2 = (pos2 + 1) % MAX_BUFFER;
		cont2++;

		pthread_cond_signal(&no_vacio2);
		pthread_mutex_unlock(&mutex2);
	}

	printf("Procesador: No más items.Saliendo...\n");
}

void *consumidor(void *arg) {

	int datos_a_consumir = *(int *)arg;
	int pos = 0;

	for (int i = 0; i < datos_a_consumir; i++) {
		pthread_mutex_lock(&mutex2);

		while (cont2 == 0)
			pthread_cond_wait(&no_vacio2, &mutex2);

		printf("< Consuming: %d\n",buffer2[pos]);	fflush(stdout);

		pos = (pos + 1) % MAX_BUFFER;
		cont2--;

		pthread_cond_signal(&no_lleno2);
		pthread_mutex_unlock(&mutex2);

	}
	printf("Consumidor: No más items.Saliendo...\n");
}

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

	pthread_t th1, th2, th3;

	if (argc != 2) {
		printf("Introduzca solo el numero de items a generar\n");
		exit(-1);
	}

	int num_items = atoi(argv[1]);

	pthread_create(&th1, NULL, productor, (void *) &num_items);
```

```
pthread_create(&th2, NULL, procesador, (void *) &num_items);
pthread_create(&th3, NULL, consumidor, (void *) &num_items);

pthread_join(th1, NULL);
pthread_join(th2, NULL);
pthread_join(th3, NULL);

printf("Fin del thread principal\n");
return 0;}
```

Apartado b)

The code section is in the processor function. It arises when we try to lock the second mutex without freeing the first one. Then, if the consumer doesn't free mutex the complete pipeline will be blocked. This solution avoids this problem blocking mutex2 only if mutex1 is freed.

Exercise 4 (25 points).

A 50 MB hard disk with an UNIX-based filesystem has the following configuration parameters:

- Block size: 512 bytes.
- Block address size: 2 bytes.
- Number of i-nodes : 100
- Boot block size: 4 blocks.
- Superblock size: 8 Kbytes.
- The free blocks are managed using a bit map.
- I-node fields:
    - File attributed:
        - Owner and group ids.
        - Read permissions for owner, group and everyone else.
        - Write permissions for owner, group and everyone else.
        - Execute permissions for owner, group and everyone else.
    - Link counter (1 byte).
    - 5 direct block.
    - 2 single indirect block.
    - 2 double indirect block.

The filesystem support both hard and soft links.

Answer the following questions:

a) What is the maximum file size that can be used in this filesystem? Justify your answer.
b) What is the maximum number of hard links (not counting the initial name) that a file can have?
c) What is the maximum number of soft links that a file can have?
d) The filesystem has the following structure:
   /x/y/z
   … and the following commands are executed:

ln /x/y/z /x/y/t
ln -s /x/y/t /x/y/r
rm /x/y/t

show the changes in the filesystem for each one of these commands.

e) After executing all these commands, show the number of disk accesses related to the execution of "cat /x/y/r". Describe the result of this execution. Assume that only the i-node "/" is loaded in the disk.

SOLUTION:

REFERENCES		# BLOCKS
5 Direct pointers		5
2 single indirect	:	2*512/2 = 512
2 double indirect:		2*(512/2)*(512/2)= 131072

The number of blocks is:

5 + 512 + 131072 = 131589 block

The maximum size is::

131589 *512 B = 67373568 B (64 MB approximately)

Give that the HD is 50MB, its size is smaller than the theoretical maximum size. For this reason, the maximum size has to be obtained by the following:

Maximum file size = Overall Number of blocks – Number of blocks used by the rest of the filesystem structures

The filesystem structures are:

Boot	SuperBlock	Bitmaps	i-nodes	Data and directories

Boot size = 2 blocks

SuperBlock size = 4 blocks

Bitmap size = 1 bit/block * 131589 blocks / 512 * 8 size of a block in bits= 32.16 => approximately 33 blocks * 2 bitmaps => 66 blocks

i-node size = 100 blocks

Maximum file size = 50 MB/512 – (2 + 4 + 66 + 100) = 102400 – 172 = 102228 blocks => 49 MB approximately

- The number of hard links depends on the size of the link counter. The counter size is 1 byte so it can store up to 255 hard links. Given that the initial file name is also counted, the overall number of hard links will be 254.
- The number of softlinks depends on the number of files than can be created. Given that there are 100 –nodes and we need at least two of them for the root directory and the file itself, then the maximum number of softlinks will be 98.

- El número de enlaces simbólicos solo depende del número de ficheros que se puedan crear. Dado que el disco solo dispone de 100 inodos, el número máximo de enlaces simbólicos sería de 100, pero como mínimo en el disco existen dos inodos ocupados (el del directorio raíz y el del fichero "original"). Por lo tanto el número máximo de enlaces simbólicos a un fichero puede ser 98.

-

ln /x/y/z /x/y/t

A new entry (with name t) in y directory is created. Its related i-node is the same as file z.

The link counter of z is incremented by 1.

ln -s /x/y/t /x/y/r

A new entry (with name r) of t is created. It has a new i-node with a link-counter value of 1. This i-node points to a new data block that contains /x/y/y

rm /x/y/t

The data block related to t is freed and the link counter value is checked. Given that it has the 1 value, the complete i-node related to t is also freed. The entry of t in the /x/y directory is also removed.

- First, we access to the "/" directory.
  - Then we access to the data block of "/", and we obtain the "x" i-node -> **1 access**
  - We access to "x" i-node, and then to its related data block. We obtain the i-node of y directory. -> **2 accesses**
  - We access to "y" directory, and the to its related data block. We obtain the i-node of r file. **2 acceses**
  - We access to the "r" i-node and to its related data block. -> **2 acceses**
  - We complete the following path "/x/y/t", including the i-node and data blocks of /, x and y. Then, we find that t is missing (it was removed) so an error is produced.