Operating Systems
Final exam 27th May 2014

Bachelor's Degree in Computer Science and Engineering

Exercise 1 (20 points). QUIZ

Answer the Quiz questions writing the letters of each correct answer. Each wrong answer deducts 3 points from the exercise grade. Non answered questions don't deduct any points (they are ignored by the reviewer).

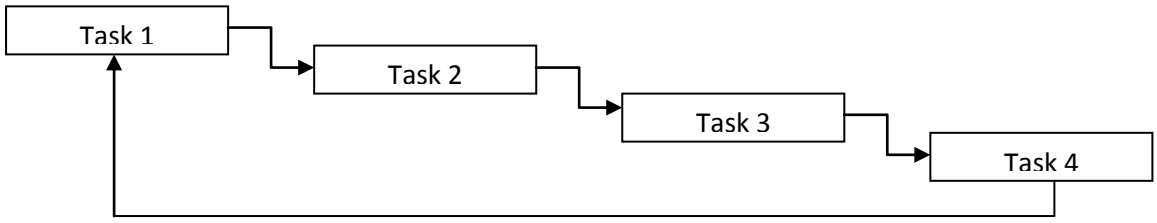
SOLUTION:

Quiz type (A or B)	1	2	3	4	5	6	7	8	9	10	11	12	13
A	A	B	A	B	C	D	D	D	C	B	B	D	A

Quiz type (A or B)	1	2	3	4	5	6	7	8	9	10	11	12	13
B	C	D	C	A	D	B	B	C	B	C	C	D	A

Exercise 2 (35 points for continuous evaluation, 20 points without continuous evaluation)

It is necessary to implement a program with n concurrent tasks. Each tasks runs a given code and all tasks are connected in a pipeline fashion. The following figure shows an example for n=4 tasks.

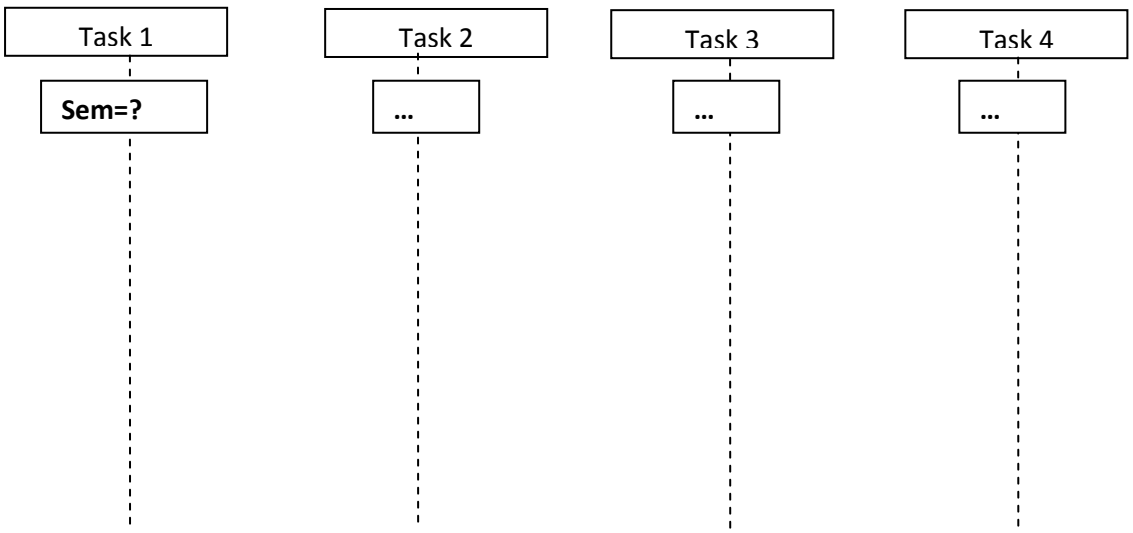


The tasks are executed in order: initially Task 1 is executed while the rest of the tasks are suspended. When a given $task_i$ completes, it notifies the following one ($task_{i+1}$) that it can start the execution. After that, $task_i$ exists and $task_{i+1}$ starts the execution. The results of each task are private. That is, it is not shared between tasks.

You have to complete the following objectives:

a) Design a program that includes this synchronization scheme for n=4 using lightweight processes (threads) and semaphores. In order to do that, complete the following figure

showing the initial value of the semaphore and writing + connecting the synchronization primitives (wait and signal) necessary to achieve the pipeline. Note: vertical axis is the time.

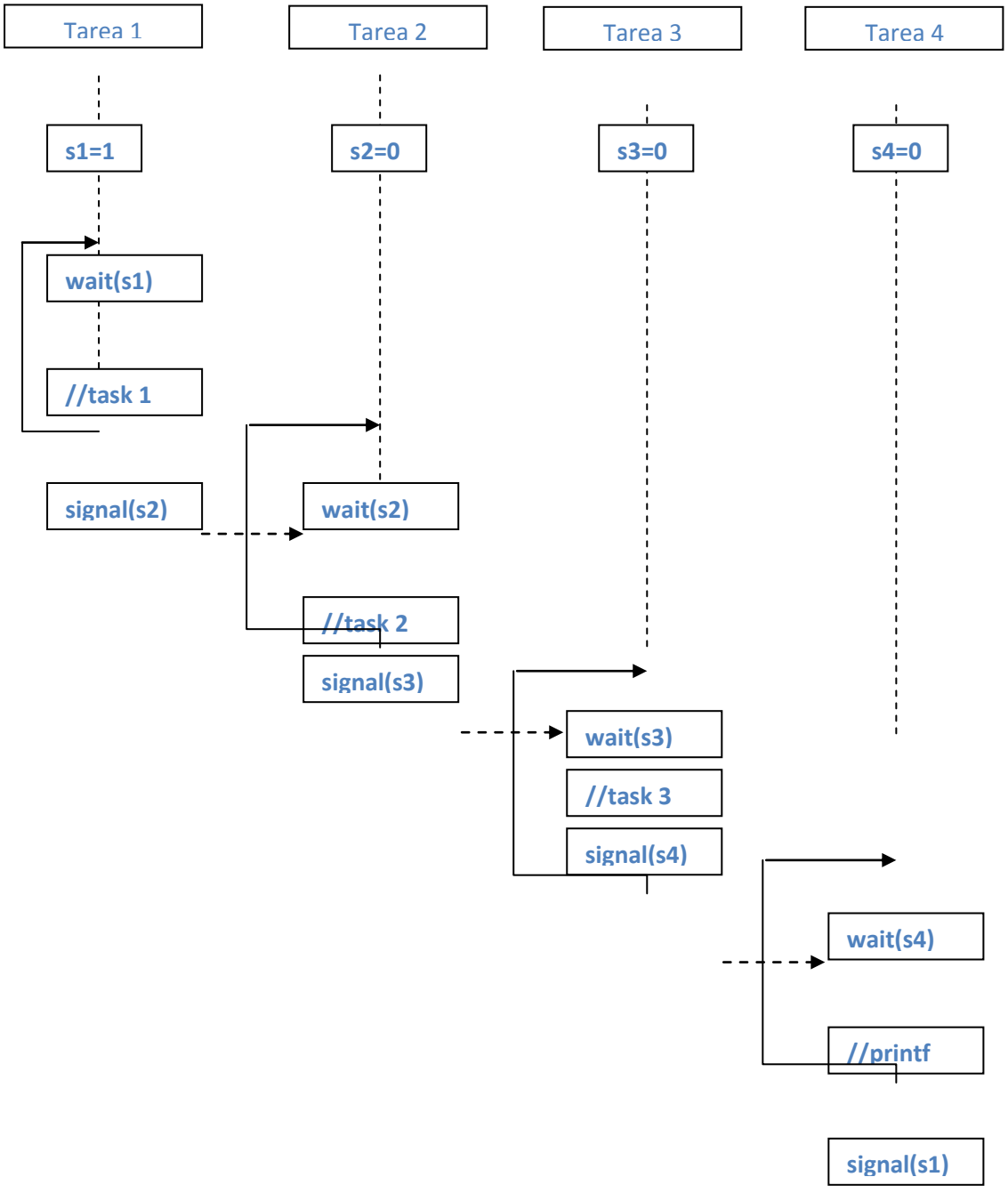


b) White a program in C programming language that implements the previous design for n=4. Provide an explanation of the code lines.

Bachelor's Degree in Computer Science and Engineering

SOLUTION:

a) Four semaphores are needed to fulfill the exercise objective. The following figure shows a solution to this problem:



b)

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

#define         CONT    4

/* VARIABLES GLOBALES */
sem_t sem1;
sem_t sem2;
sem_t sem3;
sem_t sem4;

tarea1(){
        int i=0;
        while(i<CONT){
                sem_wait(&sem1);
                printf("Proceso 1...\n");
                sem_post(&sem2);
                i++;
        }
        pthread_exit(0);
}

tarea2(){
        int i=0;
        while(i<CONT){
                sem_wait(&sem2);
                printf("Proceso 2...\n");
                sem_post(&sem3);
                i++;
        }
        pthread_exit(0);
}
tarea3(){
        int i=0;
        while(i<CONT){
                sem_wait(&sem3);
                printf("Proceso 3...\n");
```

```
                sem_post(&sem4);
                i++;
        }
        pthread_exit(0);
}
tarea4(){
        int i=0;
        while(i<CONT){
                sem_wait(&sem4);
                printf("Proceso 4...\n");
                sem_post(&sem1);
                i++;
        }
        pthread_exit(0);
}
int main(){
        pthread_t th1, th2, th3, th4;

        sem_init(&sem1, 0, 1);
        sem_init(&sem2, 0, 0);
        sem_init(&sem3, 0, 0);
        sem_init(&sem4, 0, 0);

        pthread_create(&th1,NULL, (void*)tarea1, NULL);
        pthread_create(&th2,NULL, (void*)tarea2, NULL);
        pthread_create(&th3,NULL, (void*)tarea3, NULL);
        pthread_create(&th4,NULL, (void*)tarea4, NULL);

        pthread_join(th1, NULL);
        pthread_join(th2, NULL);
        pthread_join(th3, NULL);
        pthread_join(th4, NULL);

        sem_destroy(&sem1);
        sem_destroy(&sem2);
        sem_destroy(&sem3);
        sem_destroy(&sem4);

        exit(0);
}
```

Exercise 3 (45 points for continuous evaluation, 20 points without continuous evaluation)

A given filesystem has the following specifications:

- Block size: 1024 bytes.
- Block address size: 2 bytes.
- Number of sectors per block: 2
- Sector read time: 1 ms.
- i-node size: 1 block.
- i-node fields:
    - i-node identifier (ID)
    - Metadata (file attributes, user and group identifiers, etc.)
    - Type of elements: directory (dir) , file (fil) or link (lnk) .
    - Link counter (CE)
    - 1 direct block (PD) ,
    - 1 single indirect block (PIS)
    - 1 double indirect block (PID)

The following figure shows the configuration of the file system. A blank entry means that the corresponding value is empty (void/nill type).

Block0	Block1	Block2	Block3	Block4	Block5	Block6	Block7	Block8
Superblock Root i-node: 0	ID: 0 Metadata Type: dir CE: 3 PD: 51 PIS: PID:	ID: 1 Metadata Type: fil CE: 1 PD: 100 PIS: PID:	ID: 2 Metadata Type: fil CE: 1 PD: 103 PIS: 52 PID:	ID: 3 Metadata Type: dir CE: 4 PD: 53 PIS: PID:	ID: 4 Metadata Type: dir CE: 2 PD: 54 PIS: PID:	ID: 5 Metadata Type: lnk CE: 1 PD: 55 PIS: PID:	ID: 6 Metadata Type: dir CE: 2 PD: 56 PIS: PID:	ID: 7 Metadata Type: fil CE: 1 PD: 120 PIS: 121 PID: 57

Operating Systems
Final exam 27th May 2014

Bachelor's Degree in Computer Science and Engineering

Block51	Block52	Block53	Block54	Block55	Block56	Block57	Block58	Block59
. 0	104	. 3	. 4	/Murcia	. 6	130	131	
.. 0	105	.. 0	.. 3		.. 3	58	132	
Madrid 1	106	North 4	City 5				133	
Lugo 2		South 6						
Murcia 3		Central 7						

Complete the following objectives:

1. Draw the file/directory tree structure. What problem can happen when we traverse the complete filesystem searching for a file?
2. What is the maximum file size? Justify your answer.
3. Describe how the following operation is performed and what changes are necessary to apply to the file system. Note: rm command deletes a file.
   rm /Murcia/North/City
4. Obtain the access time necessary to read the **first** byte of /Madrid file. Justify your answer.
5. Obtain the access time necessary to read the **last** byte of /Murcia/Central. Justify your answer.

SOLUCIÓN:

1. File/directory tree structure:

/Madrid	(file)
/Lugo	(file)
/Murcia/Norte/Ciudad	(Link)
/Murcia/Sur	(Empty directory)
/Murcia/Centro	(File)

The problem is a circular link of: /Murcia/Norte/Ciudad that points to /Murcia. In case of search, it generates a loop that can produce a deadlock.

2.

A direct block points to a single 1 KB block.

Given that the block size if 1KB and the address size is 2B, each single indirect block points to block that contains 1KB/2B=512 direct block pointers. In this way, we can address 512 KB of data.

A double indirect pointer points to a block that contains 512 single indirect pointers, each one of them points to 512 KB of dada. In this way we can address: 512*512KB of data=256MB

The maximum file size will be: 1 KB + 512 KB + 256 MB.

Un puntero indirecto doble direcciona un Block que contiene 1KB/2B=512 punteros indirectos simples, cada uno de los cuales direcciona 512 KB. Pudiendo direccionar: 512 * 512KB = 256 MB

3.

The sequence of accessed blocks is:

B0 -> B1 -> B51 ->B4 -> B53 -> B5 -> B54 -> B6

B0 contains the FS information. It is accessed to identify and access to the root i-node (0 in block 1). Then, we access to its content (B51). Using its information, we access to i-node 3 (B4) and its content (B53). Then, we access to i-node 4 (B5) and its content (B52). Using this information, we access to i-node 5 (B6) and we notice that the link counter is 1 thus we can delete this link. Block i-node 5 and B55 are freed. The result is:

Block 0	Block 1	Block 2	Block 3	Block 4	Block 5	Block 6	Block 7	Block 8
SuperBlock i-nodo raíz: 0	ID: 0 Metadata Type: dir CE: 3 PD: 51	ID: 1 Metadata Type: fil CE: 1 PD: 100	ID: 2 Metadata Type: fil CE: 1 PD: 103	ID: 3 Metadata Type: dir CE: 4 PD: 53	ID: 4 Metadata Type: dir CE: 2 PD: 54	ID: 5 ~~Metadata~~ ~~Type: lnk~~ ~~CE: 1~~ ~~PD: 55~~	ID: 6 Metadata Type: dir CE: 2 PD: 56	ID: 7 Metadata Type: fil CE: 1 PD: 120

	PIS: PID:	PIS: PID:	PIS: 52 PID:	PIS: PID:	PIS: PID:	~~PIS:~~ ~~PID:~~	PIS: PID:	PIS: 121 PID: 57

Block 51	Block 52	Block 53	Block 54	Block 55	Block 56	Block 57	Block 58	Block 59
. 0	104	. 3	. 4	~~/Murcia~~	. 6	130	131	
.. 0	105	.. 0	.. 3		.. 3	58	132	
Madrid 1	106	North 4	~~City~~ ~~5~~				133	
Lugo 2		South 6						
Murcia 3		Central 7						

4.

We need to access the following sequence of blocks:

B0 -> B1 -> B51 -> B2 -> B100

We need to access to 5 blocks (5 associated sectors). The access time will be 10 ms.

We need to access the following sequence of blocks:

B0 -> B1 -> B51 -> B4 -> B53 -> B8 -> B57 -> B58 -> B133

Note that for accessing the last file byte we need to Access to a double indirect block, so when we access to i-node 7 (B8) we access to the last entry of B57 and then the last entry of B58. Its last entry points to the last file block (B133).

We need to access to 9 blocks (18 sectors) in 18 ms.

Exercise 4 (20 points) . Only for students without continuous evaluation.

It is necessary to implement an interface that improves the file system access time. This interface is based on POSIX standard calls and includes the following user interface:

- `int replace(char *file_in, char *file_out, char character_old, char character_new)`
  - It receives: (1) the input file with absolute path, (2) the output file with absolute path, (3) the old character that is necessary to replace, and (4) the new character that replaces the old one.
  - This function opens the input file and replaces (in case of existing) all the occurrences of the old character with the new one. The result is written in the output file.
  - The function results the number of characters that were replaced and -1 value in case of any error.

You need to complete the following objectives:

a) Write and explain the replace function pseudocode.
b) Implement the replace function in C programming language. It has to fulfil all the previous requirements.
c) Write a main function in C programming language. This function has to receive as input arguments: file_in, file_out, character_old and character_new. In case of not receiving these four input values it displays an error message and exists. Otherwise, it calls the replace function and prints the result.

SOLUTION:

B)

```
1.  int sustituir(char *fichero_in, char *fichero_out, char car_viejo, char car_nuevo) {
2.    FILE *file_in, file_out;
3.    file_in = fopen(fichero_in, "r");
4.    file_out = fopen(fichero_out, "w");
5.    if(file_in == NULL || file_out == NULL) {
6.      printf("Error opening files\n");
7.      return -1;
8.    }
9.    int total = 0;
10.   char c;
11.   while((c = getc(file_in)) != EOF) {
12.     if(c == car_viejo) {
13.       total++;
14.       c = car_nuevo;
15.     }
16.   putc(c, file_out);
17.   }
18.   fclose(file_in);
19.   fclose(file_out);
20.   return total;
21. }
```

C)

```
1.  int main(int argc, char *argv[]) {
2.    if(argc != 5) {
3.      printf("Numero de argumentos incorrecto\n");
4.      return -1;
5.    }
6.    int total = sustituir(argv[1], argv[2], argv[3][0], argv[4][0]);
7.    printf("Numero de sustituciones: %d\n", total);
8.    return 0;
9.  }
```

Exercise 5 (20 points). Only for students without continuous evaluation.

It is necessary to implement a program that plays the Spoof game. The code consists of a main function and a global variable called overall_coins. The program structure is the following one:

- Main function creates 3 threads. Each one receives an identification (ID).
- Each thread generates its assigned coins (local_coin) as random number with a value between 0 and 5.
- Each thread generates a prediction about the total number of coins (pred_total_coin). This value is obtained adding to its assigned number of coins a random value between 0 and 10.
- Each thread adds its assigned coins (local_coin) to the (overall_coins) variable. This variable contains the actual number of overall coins of all the threads.
- When **all** the threads have completed the previous step, each thread compares its prediction (pred_total_coin) with the actual value (overall_coins). If they are the same, it displays its ID and the following text "I win".

You have to complete the following objectives:

a) Write in C programming language a code with a main function that creates 3 threads (each one with a given identifier). The main function waits for the thread termination. Implement the threads based on the previous structure. Note: you can use the following rand() function:
   - rand() % X; generates a number between 0 and X - 1

b) Implement the synchronisation mechanism necessary to access the overall_coin variable. Race conditions have to be avoided.

c) Implement the synchronisation mechanisms necessary to prevent any thread to check if it wins **before** the competition of all the remaining threads (and the update of overall_coin variable).

SOLUTION

```
int main(){

        pthread_t th1[3];

        pthread_create (&th1[0], NULL, jugar, 1);

        pthread_create (&th1[1], NULL, jugar, 2);

        pthread_create (&th1[2], NULL, jugar, 3);

        pthread_join (th1[0], NULL);

        pthread_join (th1[1], NULL);
```

```
        pthread_join (th1[2], NULL);
}

int jugar(void* arg){
        int numHilo = (int)arg;
        int misMonedas = rand() % 6;
        int estimo = misMonedas + rand() % 11;
        pthread_mutex_lock(mutex);
        monedas_totales += misMonedas;
        pthread_mutex_unlock (mutex)
        // BARRERA PUEDE HACERSE CON barrier O CON variables condicionales
        pthread_mutex_lock(mutex2);
        n_acabados++;
        while(n_acabados != 3)
                pthread_cond_wait(condVar);
        pthread_cond_bcast(condVar);
        pthread_mutex_unlock (mutex2)

        if(estimo == monedas_totales){
                printf("Yo gano %d\n", numHilo);
        }
}
```