RULES:

- The final grades and the review dates will be anounced in Aula Global. The exam duration is **two hours and a half.** Books and notes are not allowed. A valid ID document will be necessary to submmit the exam.

NAME: **GROUP:**

Exercise 1 (20 points) . QUIZ (20 minutes)

Answer the Quiz questions writing the letters of each correct answer. Three wrong answers deducts one correct answer. Non-answered questions don't reduce the grade (are ignored).

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
D	B	C	B	D	B	B	A	C	B	D	B	A	A	D

Exercise 2 (20 points).
Given a hard disk with a Unix-like filesystem using an i-node structure with a data block size of 1024Bytes. The filesystems uses 10 direct pointers to data blocks, 1 single indirect pointer, 1 double indirect pointer and 1 triple indirect pointer. The size of the addresses is 4 bytes.

The filesystem contains the information shown next:

I-node table:

I-node number	2	3	4	5	6
Type	Directory	Directory	File	Symbolic link	
Link counter	3	2	2	1	
Data block address	11	12	13	14	
…….					

Data block:

Block number	11	12	13	14	15
Content	. 2	. 3	File data	/tests/data	
	.. 2	.. 2			
	tests 3	data 4			
		report 4			
		chart 5			

Answer the following questions:

a) Obtain the maximum file size taking into account only i-node configuration.

b) Obtain how many disk blocks are used by a 10 Mbyte file (taking into account both the data blocks and the addresses).

c) Show what are the results in the i-node and data block configurations after the following operations:

1. rm /tests/data
2. rm /tests/report
3. rm /tests/chart
4. mkdir /tests/Dir1

Fill the following table:

SOLUTION:

1) I-node table after rm /tests/data:

I-node number	2	3	4	5	6
Type	Directory	Directory	File	Enlace Simbolico	
Link counter	3	2	**1**	1	
Data block address	11	12	13	14	

Data blocks after rm /tests/data:

Block number	11	12	13	14	15
Content	. 2 .. 2 tests 3	. 3 .. 2 report 4 ------ chart 5	File data	/tests/data	

2) I-node table after rm /tests/report:

I-node number	2	3	4	5	6
Type	Directory	Directory	----	Link	
Link counter	3	2		1	
Data block address	11	12		14	
…….					

Data blocks after rm /tests/report:

Block number	11	12	13	14	15
Content	. 2 .. 2 tests 3	. 3 .. 2 ----- Chart 5	----	/tests/data	

3) I-node table after rm /tests/chart:

I-node number	2	3	4	5	6
Type	Directory	Directory		…..	
Link counter	3	2			
Data block address	11	12			
…….					

Data blocks after rm /tests/charts:

Block number	11	12	13	14	15
Content	. 2 .. 2 tests 3	. 3 .. 2		…..	

4) **I-node table after mkdir /tests/Dir1:**

I-node number	2	3	4	5	6
Type	Directory	Directory	**Directory**		
Link counter	3	2	**2**		
Data block address	11	12	**13**		
…….					

Data blocks after mkdir /tests/Dir1:

Block number	11	12	**13**	14	15
Content	. 2 .. 2 tests 3	. 3 .. 2 **Dir1 4**	**. 4** **.. 3**		

a) File maximum size: #direct pointers*block size +

first level indirect pointers *BlockSize/AddressSize*BlockSize +

second level indirect pointers *(BlockSize/AddressSize)^2*BlockSize +

third level indirect pointers *(BlockSize/AddressSize)^3*BlockSize=

10*1024+1*1024/4*1024+1*(1024/4)^2*1024*(1024/4)^3*1024 = **16 GBytes**

b) For a 10 MB file it is needed:

#BLocks= FileSize/BlockSize = 10*2^{20}/2^{10} = 10*1024 = **10240 blocks**

10 First 10 blocks are addresses by direct pointers. We will need to use the indirect pointers to address the remaining 10230 blocks.

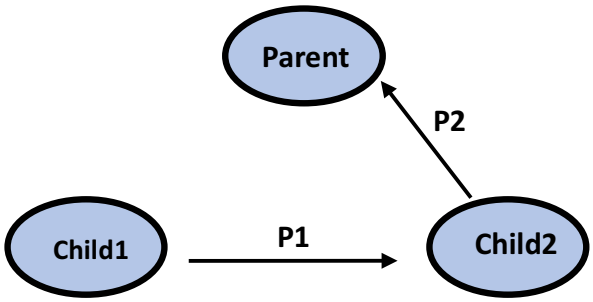
The indirect pointer points to a block with 1024/4 addresses, i.e., data blocks. So a second-level indirect pointer is also required to point to the 9974 remaining blocks. We

will need an extra block (1 more) to store these pointers. This block contains pointers that point to blocks that have pointers (256 per block). Given that we need 9974 blocks of data, we will need 9974/256=38.9 blocks, that are 39 effective blocks to keep all the related pointers.

In overall, we need 10240 data blocks + 1 block of firts-level indirect pointer+ 1 block of second-level indirect pointer + 39 blocks with pointers (that are pointed by the second-level indirect pointer). This adds up **10281 block plus** file i-node.

Exercise 3 (30 puntos).

Given the following concurrent program specification:



- There are three processes: Parent and 2 children called Child1 and Child2
- There are two pipes:
  - P1 that links Child1 with Child2
  - P2 that links Child2 with Parent
- Child 1 executes the Linux system command (command prompt) that lists in the standard output the files and directories existing in the current directory. The command output is sent to Child2 by means of pipe P1.
- Child 2 executes the Linux system command (command prompt) that displays in the standard output the number of characters received from the standard input. That is, given an input string, the command returns the number of characters on it. This command has to be employed using P1's output as standard input and P2's input as standard output.
- Parent process reads and displays in screen the output of P2. This output will be the number of characters of the list that contains the existing files and directories in the current directory.

Complete the following tasks:

1. Write the Linux command names that are used by Child1 and Child2.

2. Implement the problem in C-language code using the standard system calls (*fork(), pipe(), execlp()*, etc.). **Justify how the code works by means of explaining comments in the code.**

3. Show the changes that have to be done in the code to make the parent process show the error messages of the command executed by Child2. Note: it is only necessary to consider Child2 (the error messages of Child1 are not considered and discarded).

SOLUTION

1.- Child 1 command "ls", child 2 command: "wc –c"

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

int main(int argc, char **argv){
        int id1,id2;
        int p1[2],p2[2];
        char cnt[100];
        int status;
        int val;
        pipe(p1);
        pipe(p2);


        id1=fork();
        if(id1==0) // Child 1
        {
         close(1);
         dup(p1[1]);
         close (p1[0]);
         close (p1[1]);
         close (p2[0]);
         close (p2[1]);
         execlp("ls","ls",(void*)NULL);

         exit(1);
        }

        id2=fork();
        if(id2==0) // Child 2
        {
         close(0);
         dup(p1[0]);
         close (p1[0]);
         close (p1[1]);
         close(1);
         dup(p2[1]);
         // Extra code for the last section
         close(2);
         dup(p2[1]);
         // ********************
         close (p2[0]);
         close (p2[1]);
         execlp("wc", "wc","-c",NULL );
         exit(1);
        }
        close (p1[0]);
        close (p1[1]);
        close (p2[1]);
        read(p2[0],cnt,100);
        printf("\n \n Result:: %s \n",cnt);

        waitpid();
        waitpid();
        return(1);
}
```

2 y 3) code below.

Exercise 4 (30 points).

Write a program that executes in parallel three threads called A, B and C.

- A thread has three code blocks (a1, a2 y a3).
- B thread has four code blocks (b1, b2, b3 y b4)
- C thread has three code blocks (c1, c2 y c3).

a) Code a function called Write() that displays on the screen a test (received as input parameter) and performs after that a random delay between 3 seconds. Note: rand() function generates a random number with a value between 0 and MAX_INT.
b) Write a main program that creates the threads A, B and C. Each thread will use the Write() function to show the code block that is being executed. This operation is performed **without** any concurrency control.
c) Modify the previous program using as many semaphores as necessary to enforce the following execution order:

- c1 will be executed after a1 completion.
- a2 will be executed after b1 completion
- c2 will be executed after b2 completion
- b3 will be executed after a2 completion
- a3 will be executed after c2 completion
- b4 has to be the last data block executed.

Note: show clearly the system calls used to manage the semaphores.

SOLUCION

a)

```
#define STDOUT 1
void mywrite (char *dato){
int num;

for(num= 0; num< 5; num++){
      write(STDOUT, dato, strlen(dato));
      sleep (rand()%3);
      }
}
```

b)

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

#define STDOUT 1

void mywrite (char *dato){
int num;
```

```
for(num= 0; num< 5; num++){
      write(STDOUT, dato, strlen(dato));
      sleep (rand()%3);
      }
}

Void *A (void *p){
      mywrite (" a1");
      mywrite (" a2");
      mywrite (" a3");
      pthread_exit (NULL);
}

Void *B (void *p){
      mywrite (" b1");
      mywrite (" b2");
      mywrite (" b3");
      mywrite (" b4");
      pthread_exit (NULL);
}


Void* C (void *p){
      mywrite (" c1");
      mywrite (" c2");
      mywrite (" c3");
      pthread_exit (NULL);
}


Int main (int argc, char *argv[]){
      pthread_t th1, th2, th3;

      pthread_create (&th1, NULL, A, NULL);
      pthread_create (&th2, NULL, B, NULL);
      pthread_create (&th3, NULL, C, NULL);

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

      Return 0;

}
```

c)

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

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

Sem_t s1, s2, s3, s4, s5, s6;

#define STDOUT 1

void mywrite (char *dato){
int num;

for(num= 0; num< 5; num++){
      write(STDOUT, dato, strlen(dato));
      sleep (rand()%3);
      }
}

Void *A (void *p){
      mywrite (" a1");
      sem_post (&s1);
      sem_wait (&s2);
      mywrite (" a2");
      sem_post (&s4);
      sem_wait (&s5);
      mywrite (" a3");
      sem_post (&s6);
      pthread_exit (NULL);
}

Void *B (void *p){
      mywrite (" b1");
      sem_post (&s2);
      mywrite (" b2");
      sem_post (&s3);
      sem_wait (&s4);
      mywrite (" b3");
      sem_wait (&s6);
      sem_wait (&s6);
      mywrite (" b4");
      pthread_exit (NULL);
}


Void* C (void *p){
      Sem_wait (&s1);
      mywrite (" c1");
      sem_wait (&s3);
      mywrite (" c2");
      sem_post (&s5);
      mywrite (" c3");
      sem_post (&s6);
      pthread_exit (NULL);
}
```

```
Int main (int argc, char *argv[]){
      Pthread_t th1, th2, th3;

      Sem_init (&s1, 0, 0);
      Sem_init (&s2, 0, 0);
      Sem_init (&s3, 0, 0);
      Sem_init (&s4, 0, 0);
      Sem_init (&s5, 0, 0);
      Sem_init (&s6, 0, 0);

      Pthread_create (&th1, NULL, A, NULL);
      Pthread_create (&th2, NULL, B, NULL);
      Pthread_create (&th3, NULL, C, NULL);

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

      Sem_destroy (&s1);
      Sem_destroy (&s2);
      Sem_destroy (&s3);
      Sem_destroy (&s4);
      Sem_destroy (&s5);
      Sem_destroy (&s6);

      Return 0;

}
```