OpeUniversidad Carlos III de Madrid
Computer Science and Engineering Department

Operating Systems Course

RULES:

- The final grades and the review dates will be announced 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 submit the exam.

NAME: **GROUP:**

Exercise 1 (20 points).

Always

It uses different types of pointer inside the iNode (Direct pointer, single indirect pointer ,double indirect pointer, tripple indirect pointer)

Every Time.

Provides hardware support to the OS memory management

Comparing by hardware units every address generated in user mode with base and limit registers.

Paging

All of the above.

lock(m) tries to block the mutex. If it is already locked , lock(m) unlocks it

All of the above.

A and B are correct

To improve performance by sending some processes to swap area

Register

Executes an especific service.

Is changed only at the startup of the system

switches the execution of the processes so fast that the user believes to be interacting with them as if it was in parallel.

Exercise 2 (20 points).

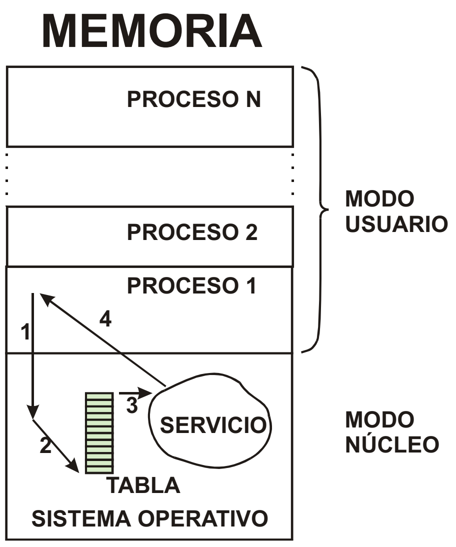
Answer the following questions. Justify each answer.

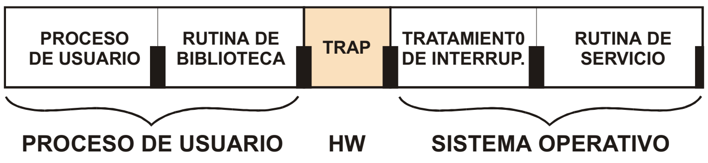
a) Why it is not possible to perform a system call using a function similar to CALL?

SOLUTION

You cannot use the operating system with a CALL function because you have to go to

supervisor mode.The proper way to do this is through a system call, which is responsible for causing a TRAP having first prepared the necessary parameters. In figure making a system call is as follows:





b) Consider the following program called prog1.c

```
// prog1.c

#include <stdio.h>
#include <unistd.h>
Int main()
{
printf("one\n");
write(STDOUT_FILENO, "two\n", 4);
return 0;
}
```

This program produces the following output when it is executed in command prompt:

```
$ ./prog1
one
two
$
```

However, when the program's output is send to the cat command by means of a pipe, the output is the following one:

```
$ ./prog1 | cat
two
one
```

Explain this behaviour.

SOLUTION

printf () uses an input buffer / stdout, while write () is a system call that immediately invokes the kernel to generate output data.

The stdio buffer is emptied when there is a line break "\ n" but only when the buffer is connected to the standard output (screen or terminal), if stdout is connected through a pipe as in the second run emptying buffer delayed until completion of the program.

c) An user executes in a bash terminal the command **cat < main.c | grep main**
In the following table show each one of the following events:
1. System calls related with the processes: fork, exec, exit, kill, wait/pid
2. System calls related to the file descriptors: open, close, pipe, dup, dup2

The different rows can be used to represent the event synchronization. For instance, if an event n happens after event m, then n has to be placed in a row after event m.

Table for Exercise 2.c

The input is shown in **bold** font and the output is shown in *italic* font.	Shell process	Child1 process	Child2 process
$ **cat < main.c | grep main** *int main() {* $			

The input is shown in **bold** font and the output is shown in *italic* font.	Shell process	Child1 process	Child2 process
$ **cat < main.c | grep main** *int main() {* $	pipe([RD, WR])		
	fork()		
	close(WR)	close(RD)	
	fork()	dup2(WR,1)	
	close(RD)	close(WR)	dup2(RD, 0)
		open("main.c") = FD	close(RD)
		dup2(FD, 0)	exec("grep")
		close(FD)	exit()
		exec("cat")	

		exit()	
	waitpid(-1)		
	waitpid(-1)		

Dup2 cannot be executed in the shell process that affect descriptors stdin / stdout files.
Note: Close calls are only displayed in response to complete the scheme but could be accepted as valid response without them.

Exercise 3 (30 points).

A new Company offers support to investors by means of an automatic learning algorithm based on a decision tree. These tree is used by all the clients and it is constantly updated with new information.

This decision tree is implemented with two routines that are not thread-safe. That is, they are not designed to be used in a concurrent fashion. The names of the routines are eval_investment (to evaluate a given investment) y update_tree (to provide new information to the decission tree).

```
assessment_t* eval_investment(investment_t* input);

void update_tree(features_list* new_features);
```

It is necessary to provide concurrent access to the eval_investment routine and guarantee that when the decision tree is being modified (update_tree) by one thread, no other thread can execute neither eval_investment nor update_tree.

Complete the following tasks:

a) Discuss the related problems of the following solution to the previous requirements.

```
assessment_t* eval_investment_safe( investment_t* input ) {
      pthread_mutex_lock(&m);
      assessment_t* result = eval_investment(input);
      pthread_mutex_unlock(&m);
      return result;
}

void update_tree_safe( features_list* values ) {
      pthread_mutex_lock(&m);
      update_tree(values);
      pthread_mutex_unlock(&m);
}
```

b) Implement an alternative solution for multiple readers/writers using condition variables. The solution has to fulfil the previous requirements.
c) Modify the previous solution (b) to introduce priorities in the writers. That is, if a thread is executing update_tree_safe, then the new incoming threads wait in order to perform the update operation.

NOTES:

- Include the declaration of the shared variables, condition variables and mutexes used in the solution.
- It is not necessary to write the main() function. It is only necessary to write the eval_investment_safe() and update_tree_safe() functions.

SOLUTION

a) The proposed solution avoids race conditions on access to the tree but has a problem in that only one thread can access the critical section, even when we only read. This involves a significant performance penalty.

b) A possible solution could be next. It is important to use pthread_cond_broadcast to allow all waiting threads eligible for critical section, especially if you are reading threads:

```
pthread_mutex_t m;
pthread_cond_t turn;
int clients = 0;
int updating = 0;

assessment_t* eval_investment_safe(investment_t* input) {

assessment_t* result;

pthread_mutex_lock(&m);
while(updating) {
      pthread_cond_wait(&turn, &m);
clients++;
pthread_mutex_unlock(&m);

result = eval_investment(input);

pthread_mutex_lock(&m);
clients--;
pthread_cond_broadcast(&turn);
pthread_mutex_unlock(&m);

return result;
}

void update_tree_safe(features_list* new_values) {
pthread_mutex_lock(&m);
while (clients || updating)
      pthread_cond_wait(&turn, &m);
updating++;
pthread_mutex_unlock(&m);

update_tree(features_list);

pthread_mutex_lock(&m);
updating--;
pthread_cond_broadcast(&turn);
pthread_mutex_unlock(&m);
}
```

C) In this case, another possible solution is to take a simple count of how many writers are active at all times and check whether readers is nonzero or not:

```
pthread_mutex_t m;
pthread_cond_t turn;
int clients = 0;
int updating = 0;
int writers = 0;

assessment_t* eval_investment_safe(investment_t* input) {

assessment_t* result;

pthread_mutex_lock(&m);
if (writers)
      pthread_cond_wait(&turn, &m);
while(updating) {
      pthread_cond_wait(&turn, &m);
clients++;
pthread_mutex_unlock(&m);

result = eval_investment(input);

pthread_mutex_lock(&m);
clients--;
pthread_cond_broadcast(&turn);
pthread_mutex_unlock(&m);

return result;
}

void update_tree_safe(features_list* new_values) {
pthread_mutex_lock(&m);
writers++
while (clients || updating)
      pthread_cond_wait(&turn, &m);
updating++;
pthread_mutex_unlock(&m);

update_tree(features_list);

pthread_mutex_lock(&m);
updating--;
writers++;
pthread_cond_broadcast(&turn);
pthread_mutex_unlock(&m);
}
```

Exercise 4 (20 puntos).
Given a standard Linux (ext3) filesystem where the inodes have 10 direct pointers, one single indirect pointer, one double indirect pointer and one triple indirect pointer.

The disk size is 300MB, and the 96% space is available for data block. The rest of the disk space is used for the resto of the filesystem structures. The block size if 2KB and the block addresses are 32 bits.

Answer the following questions:

a- What is the maximum size (in KByters) that a file can use to store data without using the single double pointer.
b- What is the maximum file size (space used to store data)
c- Given a filesystem with the following information:

i-node table

I-node number	**1**	**2**	**3**	**4**	**5**	6	7
Type	Directory	Directory	File				
Physical link counter	3	2	1				
Data block address	11	12	13				
…….							

Data block

Block number	**11**		**12**		**13**	**14**	**15**	16	17	18
Content	**.**	**1**	**.**	**2**	File data					
	..	**1**	**..**	**1**						
	d	**2**	**f1**	**3**						

Complete the following tables with the results of the execution of the following operations:

ln -s /d /d1 *Symbolic link from directory* /d1 to directory d

cp /d/f1 /f2	File copy from /d/f1 to /f2
mkdir /d/d2	Directory creation in /d/d2

Tables for exercise 4.c

i-node table

I-node number	**1**	**2**	**3**	**4**	**5**	6	7
Type	Directory	Directory	File				
Physical link counter	3	2	1				
Data block address	11	12	13				
…….							

Data block

Block number	**11**		**12**		**13**	**14**	**15**	16	17	18
Content	.	**1**	.	**2**	File data					
	..	**1**	..	**1**						
	d	**2**	**f1**	**3**						

SOLUTION

A)

With 2KBytes a- blocks and addresses of 32 bits = 4Bytes fit 2KB / 4B = 512 addresses per block.

If we cannot use double indirect pointers, thus we can only use 10 direct blocks plus 512 points would give us the targeted block for the simple indirect pointer block so the maximum file size would be 512 + 10 = 522 blocks = 1044Kbytes.

B)

To calculate the maximum size count the number of data blocks with addresses

10 direct pointers point to 10 blocks of the file.

The simple indirect pointer will point to a block of addresses that point to 512 file data blocks. That is 512 = 2^9 blocks

Double indirect pointer point to 512 blocks each of which will address file 512 blocks. That is 512 * 512 = 2^{18} blocks

Triple indirect pointer point to 512 blocks each of which point to 512 blocks each of which will address file 512 blocks. That is 512 * 512 * 512 = 2^{27} blocks

Total 522 blocks + 218 + 227 blocks, which, as each block is 2KB, gives us something more than 2^{28} KBytes

As 300MB (= 300 * 210KB) is less than 228 KB size is not restricted by pointers to blocks but the disk size. Available disk 300 * 0.96 = 288MBytes for data blocks, that is 144M = 147456K blocks.

As the file must be in a directory and this will take at least one block 1 subtract the number of total blocks: 147456K Block-1 = 147455K blocks

To reach that size need:

10 direct pointers

1 single indirect pointer that points to a block of addresses that must be subtracted from the maximum size: 147455K blocks-1 =147454K blocks

To reach the remaining 147454 blocks (blocks -10 - 512 = 146,932 remaining blocks to address) will use the double indirect pointer that points to an address block (subtract 1: 147454 blocks-1 = 147453 maximum size blocks) point to: 146,931 blocks / 512 = 287 address blocks also subtract: 147453 blocks- 287 address blocks = 147166 actual data blocks. Total 147166 * = 2KB per block 294332KB

Nº Inodo	**1**	**2**	**3**	**4**	**5**	**6**
Tipo	Directorio	Directorio	Fichero	Enlace Simbólico	Fichero	Directorio
Contador Enlaces Fis.	3	~~2~~ 3	1	1	1	2
Direcc. Bloque Datos	11	12	13	14	15	16
…….						

Bloques de datos:

Nº Bloque	**11**	**12**	**13**	**14**	**15**	**16**	**.....**
Contenido	**. 1**	**. 2**	Datos del fichero	**/d**	Datos del fichero f2	**. 6**	
	.. 1	**.. 1**				**.. 2**	
	d 2	**f1 3**					
	d1 4	d2 6					
	f2 5						