PRACTICAL STUFF

Oscar Vargas Pabon

My codes where uploaded to

Scripting

Buffer_stuff

"Explain the difference between the output observed on the terminal and that contained in the target piped file Execute the following 'C' program, first interactively, then by redirecting output to a file at the UNIX shell level with a ">". Explain what has happened with the addition of the fflush system call."

The fflush forces the buffer to get empy before the write line. This ensures the order in between the printf and the write. (this file is in scripting/p1.cpp)

Weird program

"Making the minor changes to program 3 above needed to get the code below, execute the following 'C' program several times interactively. Explain how and why the order of the output from this program is different from that of the after program."

Adding the 'wait' before the parent thread executes forces its child to end before it's parent prints "parent %d" this ensures the order is preserved over all executions. Removing/commenting the 'wait' can make the parent to print before its child.

crypting

"Create encrypt_it and decrypt_it programs using C, C++ and bash scripting language. Both receive two arguments: the rotation index and the phrase to be encrypted or decrypted."

Zú iolxgju Ikygx, latioutg! -> Tú cifrado Cesar, funciona!

- Have you reached parallel code? Or parallel behavior?

No

- Does the shell-script code perform parallelism or pipelining?

No

- But if script-based parallel programming is so easy, why bother with anything else?

Because one may require more complex parallel behaviour.

POSIX

Adding array

- Child processes run in parallel mode? Or concurrently? Why?

The child can run paralely because we are creating a new process.

- Is the running time equal for sum and adder programs? Which is faster? Why?

No, sum is faster. This is due to the overhead of the adder program.

- Do parent and child processes use shared memory? Is it important?

They do not share memory, unless it's explicitly stated (like the pipes)

Matrix sum-difference

This program is in posix/matrix. Notes: the constant "const bool use_pthread=0;" was used to change between using threads or not to the comparisongs.

- Which implementation is faster? why?

I made the comparison with n=1000

With threads

```
real    0m0.117s
user    0m0.121s
sys     0m0.010s
```

Withouth threads

```
real    0m0.137s
user    0m0.130s
sys     0m0.000s
```

- How do threads work? Always threads implementations are faster?

Threads allow concurrent behaviour, however it doesn't create a new process. Not always are thread implementations faster as creating/deleting them also have an overhead.

- Threads and processes share the same region of memory? What memory segments share threads?

Threads don't generate new processes, in a way the same process is preserved. Threads work with the same memory as other threads, however they have different information on their execution.

- Can any programming challenge be implemented using threads?

Well… there is always the trivial

"

```
        Int main(){
                pthread_t pd; pthread_create(&pd,NULL,real_main(),NULL);
                pthread_join(pd,NULL);
                return 0;
        }
```

"

I think I didn't understood the question.