Internet Programming

Assignment 1

1. Micro-shells

Mysh1

This shell accepts only one command without parameters except if the command is “cd” which accepts a path as a parameter. If parameters are indicated after a valid command, only the command will be executed omitting the parameters.

The program uses the following functions:

ask_command: It prompts a “$ “ and waits for the user to input a command.

count_spaces: Used to separate the input of the user into different parameters. (Needed in case the user inputs “cd”)

make_param_vector: Used to make a vector with the parameters entered by the user.

The command is executed with *execvp* omitting the parameters inserted by the user.

The function *chdir* is use to change the directory of the program.

Mysh2

This shell accepts a command and its parameters. The functions used are the same as in mysh1. The only difference is that the parameters entered by the user are given to the function *execvp.*

Mysh3

This shell accepts piped commands.

The program used the same functions of *mysh2* plus the following:

execute_program: The logic of executing a command is in this function. It creates a child and executes the command and its parameters with *execvp*.

pipe_commands: Create n-1 pipes (n: number of commands). It overwrites the standard input and output of the process executing the commands so they are piped.

Mysh4

This shells accepts piped commands. The program is the same as *mysh3* since it could already pipe more than two commands.

2. Synchronization

Process synchronization

syn_process_1:

In order to synchronize both messages, one semaphore is declared and initialized to 1. In every loop we do the operation down on the semaphore so only one can display the message. After displaying the message we do up to the semaphore so we can print again Helloworld or Bonjour monde.

syn_process_2:

In this case “ab” needs to be printed before “cd”. To do so, two semaphores are declared (let’s call them semA and semB). SemA will control the “ab” printing and semB the “cd” part. We initialize semA to 1 and semB to 0. When going to display “ab” we do semA down and after displaying, semB up. When displaying “cd” we do semB down and after, semA up. This way we ensure that “ab” is always going to be displayed before “cd”.

Thread synchronization

syn_thread_1:

The logic is the same as in *syn_process_1* but in this case we use a lock. We lock the mutex before displaying “Helloworld” and unlock it after. We do the same with “Bonjour monde”. This way we ensure only one message is displayed at a time.

syn_thread_2:

The logic is the same as in *syn_process_2* but in this case we use locks with predicates. We have two predicates that are triggered after “ab” is printed so “cd” can print afterwards, and another predicated triggered after “cd” so “ab” can be printed again.

Java threads

Syn1:

In order to make synchronization, both threads share the same instance of Display. The class Display has the display method declared as synchronized so both thread cannot access it at the same time. This makes both threads to be synchronized.