Operating System

Computadors – Grau en Ciència i Enginyeria de Dades – 2020-2021 Q2

Facultat d'Informàtica de Barcelona

This is the second Lab session focused on OS. In this case, it aims at getting experience with system calls to create and to manage processes.

Files distributed with this Lab session (FilesS8.tar.gz)

Within the collection of files distributed with this session you will find:

```
FilesS8/
    Makefile  -- how to compile the support files
    processes.cpp   -- initial version from which to start the development of Exercise 1
    writer.cpp   -- support program for Exercise 4
```

Process Creation/Termination

In this section we will analyze the difference between sequential and concurrent child process creation.

Exercise 1

Continue the development of the "processes.cpp" program, that accepts a number as its first parameter. You have to modify the file that you will find in the FilesS8 package provided with this Lab session.

The main process will create as many child processes concurrently as indicated by the parameter.

The child processes only have to print a message to the screen (e.g. "Hello World! My PID is %d\n", where the PID can be obtained with "getpid()" system call) and then finish with a specific return code that you decide.

The parent process, on its side, has to execute an infinite loop at the end of the code (e.g. an active waiting loop with "while(1);").

Once the source code is compiled, launch the program in background (using "&") and use the "ps" command to show the processes in the system. Now, check the output and the "Status" value of the "/proc/<PID>/status" file of every process (parent and child processes).

Edit a new "answers.txt" file to write your findings.

If you list (in the full format) the contents of the "/proc/<PID>" folder of the above child processes, once they have finished, but not yet the parent process, you will see some contents that have been disabled and/or flushed.

Issue a "jobs" command and see that there exists a job on the current shell, with the "processes" process. Note that you can terminate jobs by using the command "#> kill %<jobnumber>".

Exercise 2

Write the command line to finish the execution of the parent process ("#>kill -9 <PID>") launched before (in the Exercise 1) and check again the processes with the "ps" tool. Edit the "answers.txt" file to explain why the defunct child processes, that is, the zombie processes have been automatically eliminated if you have particularly only finished the parent process.

Exercise 3

Now copy the "processes.cpp" code and rename it to "processes2.cpp". Change the infinite loop of the parent process to a loop of "waitpid" to release the child processes previously finished. Launch the program and check the output of the "ps" tool. Edit the "answers.txt" file to briefly explain the consequences of this code behavior change.

Exercise 4

Now copy the "processes2.cpp" code and rename it to "processes3.cpp". Change the child process code in such a way it mutates the process (with the "execlp" system call) to execute another program we have developed, called "writer". The "execlp" system call will receive twice the name of the "writer" (as filename, and argv[0]), and also include an array of chars containing the iteration number. This one will be received as argv[1] by writer. Also, end the "execlp" system call parameters with a NULL pointer, indicating to the OS that there are no more parameters for "writer".

"writer" first waits for 5 seconds (using "sleep(5);") and then prints "New Hello World (from pid %d with parameter %d\n", where pid is obtained with the "getpid()" system call and the parameter is received on the command line of the process.

Check that the N "writer" processes are created and completed successfully.

Exercise 5

Now copy the "processes3.cpp" code and rename it to "processes4.cpp". Change the schema of child process creation in such a way that the child processes are sequentially created and executed, rather than concurrently. Take into account that now the experiment will take more time to execute.

Comment, on your "answers.txt" file, what are the minor number of changes you need to do to obtain "processes4.cpp" from "processes3.cpp" (hint: use the "diff" command 😊)

Upload the Deliverable

To save the changes you can use the tar command as follows:

#tar czvf session8.tar.gz answers.txt *.cpp Makefile

Now go to RACO and upload this recently created file to the corresponding session slot.