22.623 - Operating Systems Course

PRACTICAL ACTIVITY 2

Presentation

This practice proposes a series of activities with the objective that the student can apply some of the concepts introduced in the last modules of the subject on a Unix system.
The student must perform a series of experiments and answer the questions raised. You will also have to write small programs in C language.

Restrictions on the use of C functions
To do the practice you must use the Unix system calls. Under no circumstances can input/output C functions (getc, scanf, printf, ...) be used, the read and write system calls will be used instead. You can use C functions for string formatting (sprintf, sscanf, ...). You also can't use the sleep function, instead you'll use the pause system call.

Skills

Transversal:
• Skills to adapt to technologies and future environments by upgrading professional skills.

Specific:
• Skills to analyse a problem at the appropriate level of abstraction for each situation and apply the skills and knowledge acquired to address and solve it.
• Skills to design and build computer applications through development, integration and reuse techniques.

Statement

We want to simulate the card game of "seven and a half" between a series of players and a croupier.

To facilitate the development of the practice, we propose a series of incremental steps until the final solution.

We also provide you with the pr2so.zip file that contains the following files:

- The first step of the source code of the player process (player1.c).
- A first version of the croupier process (croupier.c)
- Executable files (player1ok, croupier1ok, ...) that behave as your application should at each step. Use them to test your solution.

To unzip this file, you need to run the command **unzip pr2so.zip**

Step 0: Familiarization with the programs provided

This step aims to familiarize yourself with the programs provided.

- player1.c : Simulates a game of seven and a half by a single player.

  Randomly generates the cards and decides whether to fold. To compile and run it you can use the commands:

  cc -o player1 player1.c

  ./player1

  Notice that, after a few seconds, the last line written by the program is displayed in red, yellow or green.
- croupier.c: expects as a parameter, the number of player processes to be created. The program executes the players sequentially, that is, it creates player (I) only when player (i-1) has finished his game. Once compiled, you can test the program by doing, for example:

  ./croupier 5

  Notice that the lines shown by a player do not mix with those shown by the others and that the croupier writes some lines in blue.

Step 1: Concurrent execution of player processes

Copy croupier.c over croupier1.c and modify croupier1.c so that player i is created even if player (i-1) has not finished executing (therefore, lines printed by different players may be mixed).

Observations:

- The dealer must finish his execution when all the players have finished.
- The behavior of this application should be like that of the executable **croupier1ok** (remember that you have a reference executable for each step of the practice).
- You should not modify the program player1.c
- The number of players is not limited (in this and the following steps).

Step 2: Determining the winner using exit/wait calls

Until now, the croupier does not know how the player processes finished. You need to have that the players inform the croupier of their final score so that the croupier can decide who won (in case of a tie, assume the player with the lower pid wins). To do this communication, the player processes will use the exit code (parameter of the exit system call). Copy croupier1.c and player1.c to croupier2.c and player2.c respectively and make the necessary changes to the new files.

Step 3: Determining the winner using as many files as players

Now you will use files to have the player processes inform the croupier process of what score they finished. When the croupier has collected the information from all the player processes, the croupier will show who won. Copy croupier2.c and player2.c to croupier3.c and player3.c respectively and make the necessary changes to the new files.

Observations:

- To check that your solution is correct, also show who the winner is using the mechanism you implemented in the previous step; the winner must be the same in both cases.
- It is recommended to create the tokens needed to perform the communication in the tmp directory and include some numerical identifier in the name. For example, to create a token name that contains the process pid you can use the following code fragment:

```
char filename[80];
sprintf(filename, "/tmp/%d", getpid());
```

- When finished, the croupier must delete the temporary tokens created **using the unlink system call**. Consult its man page.

Step 4: Determination of the winner using a single file

Implement the functionality of step 3 but using only one file to report the results. Copy croupier2.c and r2.c to croupier4.c and player4.c respectively and make the necessary changes to the new files.

Step 5: Questions

Finally, answer the following questions about the practice:

a) In step 2, you used the process termination code to communicate the result of the game to the croupier. However, this termination code is an integer while the result of the game can have a decimal. How did you solve this problem?

b) In step 3, how does the croupier know that each player has already written his result?

c) In step 4, how do you guarantee that there can be no concurrency problems between the players when writing to the file?

Resources

- Modules 1, 2, 3, 4, 5 and 6 of the subject.
- You have to give a justified response to the next questions based on Silberschatz, A.; Galvin, P.; Gagne, G. (2008). Operating Systems Concepts (8th ed.). John Wiley & Sons
- Document "Introduction to UNIX programming" (available in the classroom) or any other similar manual.
- "UNIX shell" document (available in the classroom) or any another similar manual.
- The "Operating Systems Laboratory" classroom (you can raise your own doubts about the Unix environment, programming, ...).
- Any basic C language manual.

Evaluation criteria

To do the practice you must use Unix system calls. You cannot use C input/output functions (getc, scanf, printf, ...), instead you will use the read and write system calls. You can use C functions for string formatting (sprintf, sscanf, ...).

The correct use of system calls, error control in their use and the correct programming, structure and operation of the application will be assessed.

As a reference, submitting the 5 correct steps will be evaluated with an A, submitting 4 with a B, submitting 3 with a C+, submitting 2 with a C- and submitting less than 2 with a D.

Format and delivery date

Create a zip o tar file with all the source files of the practice.

The file name will have the following format: "LastName1LastName2PRA2.tar". Last names will be written without accents. For example, the student Juan García Pérez will use the following name: "GarciaPerezPRA2.tar"

Delivery date: midnight on December 23, 2024