Universidade de Coimbra
Faculdade de Ciências e Tecnologia
Departamento de Engenharia Electrotécnica e de Computadores

Sistemas Operativos - 2019/2020
Practical Assignment Nº1
Processes, Files, Pipes, FIFOs, I/O Redirection, Unix Sockets, and Signals

1 Objectives

The objective of this practical assignment is to learn to work with files, pipes, I/O Redirection, and Unix Sockets. Furthermore, the clarity and good organisation of the source code is an important objective that should followed in the development of the practical assignment.

2 Material for Preparation and Reference

- [Robbins e Robbins, 2003,
  Chapter 4, "UNIX I/O";
  Chapter 5, "Files and Directories";
  Chapter 6, "UNIX Special Files";
  Chapter 8, "Signals";
  Chapter 18, "Connection-Oriented Communication";
  Chapter 20, "Connectionless Communication and Multicast";
  Appendix B, "Restart Library"].

- [Marshall, 1999,
  "Input and Output (I/O): `stdio.h`";
  "Interprocess Communication (IPC), Pipes";
  "IPC: Interrupts and Signals: `<signal.h>`";
  "IPC: Sockets"].

- [Stevens, 1990,
  Section 2.4 "Signals" (pp. 43-54);
  Section 3.4 "Pipes" (pp. 102-109);
  Section 3.5 "FIFOs" (pp. 110-115);
  Chapter 6 "Berkeley Sockets"].

- [Kernighan e Ritchie, 1988,
  Chapter 7, "Input and Output" (pp. 151-168);
  Chapter 8, "The Unix System Interface" (pp. 169-189);
  Appendix B, "Standard Library" (pp. 241-258)].

- Manual page of the `signal()` routine (`$ man 2 signal`).

- [Araújo, 2014,
  "UNIX Programming: Files and Pipes"
  "UNIX Programming: Signals";].

Students that do not have knowledge of programming in C should overcome this situation. Reading the book [Kernighan e Ritchie, 1988] (ANSI C) is strongly recommended to obtain knowledge about C programming, and in particular about pointers and the main standard C functions (mathematical, string manipulation, and file manipulation).

3 Assigned Work

Problem 1. Processes, pipes, sockets, and redirection. Write a C program that emulates the following shell command line:
"`find . -type f -ls | cut -c 2- | sort -n -k 7 >file.txt ; less <file.txt`"
The program should include the use of the `find`, `cut`, `sort`, and `less` commands, the implementation of the two pipelines "`|`", and the input and output redirections. The "`;`" character separates commands (or pipelines) that are executed in sequence. Thus, in this problem the "`less <file.txt`" command is executed after the "`find . -type f -ls | cut -c 2- | sort -n -k 6 >file.txt`" sequence of pipelines has terminated. You may also need the `wait()` or the `waitpid()` system calls (cf. `man 2 wait`). The processes in a pipeline "`|`" run concurrently.

To implement each of the 2 pipelines "`|`", your program should use two processes (note that one of the processes is involved in more than one pipeline), create an interprocess communication (IPC) mechanism between them, redirecting the *stdout* of the first process to the IPC mechanism, and redirecting the *stdin* of the second process from the IPC mechanism (investigate the `dup()` and `dup2()` functions). Do not forget to close the extra file descriptors, such that each program of each pipeline starts with exactly 3 open file descriptors, as expected. You should also appropriately implement the output redirection of the `sort` command and the input redirection of the `less` command, and run all the commands `find`, `cut`, `sort`, and `less` in combination, as specified in the above command line. The IPC mechanisms used to implement each of the 2 pipelines should be as follows:

- The **pipeline 1** (the pipeline between the "`find`" and "`cut`" commands) should be implemented using a Unix *socket* (i.e. a *socket* of type `AF_UNIX` or `AF_LOCAL`; not of type `AF_INET` nor any other type) as the IPC mechanism. The *socket* (`socket()`) should be used in a connection-oriented setup (e.g. [Stevens, 1990], [Marshall, 1999]), involving the adequate application of the `bind()`, `listen()`, `accept()`, and `connect()` system calls.
- The **pipeline 2** (the pipeline between the "`cut`" and "`sort`" commands) should be implemented using a pipe (`pipe()`) as the IPC mechanism;

Note 1: It is not permitted to employ the `system()` function to implement the program to solve this question. Note 2: Normally, the pipeline "`|`" is implemented by the shell using a *pipe*; However, as specified above, in this question, other IPC mechanism should be used instead of a *pipe* to implement one of the pipelines.

Hint: to organise the solution of the problem the parent creates 3 child processes (to run "`find`", "`cut`", and "`sort`" commands, respectively), and runs the "`less`" command.

Explain the above shell command line. Submit the text of the explanation as a comment at the end of the developed C program.

Problem 2. FIFOs. Make a program equivalent to the program of Problem 1 and complete it similarly to what is requested in Problem 1 (see above), but with only the following difference: modify the program, so that Pipeline 2 is implemented using a *named pipe* (FIFO) (`mknod()`, `mkfifo()`) instead of a pipe.

Problem 3. Signals. Write two programs (thus, involving two `main()` functions), named Agent and Controller, with the following definitions. On startup, the agent program creates the file "`agent.pid`" and writes its PID into the file. The agent then reads the contents of a file called "`text.in`", prints it to the console, and finally sleeps indefinitely. The agent should be sensitive to two signal types: `SIGHUP` and `SIGTERM`. When the agent receives a `SIGHUP`, it reads the contents of the "`text.in`" file and prints it on the console. When it receives a `SIGTERM`, it prints "`Process terminating...`", and then exits.

The Controller program, on startup, checks for a running agent by fetching the agent's PID from the file "`agent.pid`" and verifying it by sending it the signal number 0. If the controller cannot not find

an agent, it prints "`Error: no agent found.`" and then exits; otherwise, it prints "`Agent found.`" and continues. Then, in an infinite loop, the controller prints "`Choose a signal to send [1: HUP; 15: TERM]: `" and then waits for user input. When the user enters his/her choice, the controller sends the selected signal to the agent. If the user chooses `SIGTERM`, then the controller should also terminate after sending the signal. If the user presses `Ctrl+C`, then the controller should send a `SIGTERM` to the agent, and then exit.

Problem 4. Write a program that writes the integer "`i+1`" into element "`i`" of a table of `MAXBUF` integers (for every element of the table). `MAXBUF` should be initially "`#define`"d as 10 in the source code of the program. Then, using only one `write()` operation, the program should write the entire table of integers in binary format into an initially truncated file, named "`filetable.bin`". In the next step the program should create a child process, and then print the message "`The parent process is terminating.`", and then exit. The child process should separately read, in binary format, from the file each integer in the same order as the integers are stored in the file, and print each such integer to the standard output. In the final step of the program, the child process should wait for its parent process to terminate, and then print to the standard output the message "`The child process is terminating.`", and then terminate. All the operations on the "`filetable.bin`" file should be performed using system calls (e.g. `open()`, `write()`, `read()`, etc).

Note: The `init` process has a PID equal to 1. Since the Linux version 3.4, there is the possibility that the process that adopts orphan processes is not the `init` process. This information should be taken into account when designing in the program, a method for the child process to detect that the parent process has terminated.

Report and Material to Deliver

A succinct report should be delivered to the professor in PDF format. The report should contain the following information in the first page: name of the course ("disciplina"), title and number of the practical assignment, name of the students, number of the class ("turma"), number of the group. It should be submitted through the Nónio system (`https://inforestudante.uc.pt/`) area of the "Sistemas Operativos" course in a single file in `zip` format that should contain all the relevant files that have been involved in the realisation of the practical assignment. The name of the submitted `zip` file should be "`tXgYrZ-so20.zip`", where "`X`", "`Y`" e "`Z`" are characters that represent the numbers of the class ("turma"), group, and practical assignment, respectively.

References

[Araújo, 2014] Rui Araújo. *Operating Systems*. DEEC-FCTUC, 2014.

[Kernighan e Ritchie, 1988] Brian W. Kernighan e Dennis M. Ritchie. *The C Programming Language*. Prentice-Hall, Inc, Englewood Cliffs, New Jersey, USA, Second ed., 1988.

[Marshall, 1999] A. Dave Marshall. *Programming in C: UNIX System Calls and Subroutines Using C*. Cardiff University, UK, 1999. [Online]. Available: "`http://www.cs.cf.ac.uk/Dave/C/`".

[Robbins e Robbins, 2003] Kay A. Robbins e Steven Robbins. *Unix Systems Programming: Communication Concurrency, and Threads*. Prentice–Hall, Inc, Upper Saddle River, NJ, USA, 2003.

[Stevens, 1990] W. Richard Stevens. *UNIX Network Programming*. Prentice-Hall, Inc, Englewood Cliffs, New Jersey, USA, First ed., 1990.

[The IEEE and The Open Group, 2008] The IEEE and The Open Group. [POSIX Specification] The Open Group Base Specifications Issue 7; IEEE Std 1003.1^{TM}-2008. 2008. [Online]. Available: `http://www.opengroup.org/onlinepubs/9699919799/` .