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

Sistemas Operativos - 2015/2016
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, FIFOs, sockets, and redirection. Write a C program that emulates the following shell command line:
"`cat /etc/passwd /etc/passwd | cut -d: -f 1 | sort | uniq >file.txt ; less <file.txt`". The program should include the use of the `cat`, `cut`, `sort`, `uniq`, and `less` commands, the implementation of the three 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 "`cat /etc/passwd /etc/passwd | cut -d: -f 1 | sort | uniq >file.txt`" sequence of pipelines has terminated. You may also need the `wait()` or the `waitpid()` system calls (cf. `man 2 wait`). On the other hand, the processes in a pipeline "`|`" run concurrently.

To implement each of the 3 pipelines, your program should use two processes (note that some processes are 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 (investigate the `dup()` and `dup2()` functions). Do not forget to close the extra file descriptors, such that the each program of each pipeline starts with exactly 3 open file descriptors, as expected. You should also appropriately implement the output redirection of the `uniq` command and the input redirection of the `less` command, and run all the commands `cat`, `cut`, `sort`, `uniq`, and `less` in combination, as specified in the above command line. The IPC mechanisms used to implement each of the 3 pipelines should be as follows:

- The **pipeline 1** (the pipeline between the "`cat`" and "`cut`" commands) should be implemented using a *pipe* (`pipe()`) as the IPC mechanism;
- The **pipeline 2** (the pipeline between the "`cut`" and "`sort`" commands) should be implemented using a *named pipe* (FIFO) (`mknod()`, `mkfifo()`) as the IPC mechanism;
- The **pipeline 3** (the pipeline between the "`sort`" and "`uniq`" 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.

Note 1: The `system()` function should not be employed 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 mechanisms should be used instead of a *pipe* to implement 2 of the 3 pipelines. Hint to organise the solution of the problem: the parent creates the IPC mechanisms, creates 4 child processes (to run "`cat`", "`cut`", "`sort`", and "`uniq`"), and runs the "`less`".

Problem 2. Write a program that sleeps forever until the user interrupts it twice with a `Ctrl-C`, and then exits. Once the first interrupt is received, tell the user: "`Interrupt again to exit.`". The first interrupt should be forgotten 3 seconds after it has occurred. Additionally, the program should block the `SIGQUIT` signal, and ignore the `SIGTSTP` signal. The program should start by printing "`Interrupt twice with Ctrl-C to quit.`" on the screen.

Problem 3 When executing the following program, a possible *output* is "`P P C `". Please explain this result?

```
#include <stdio.h>
#include <unistd.h>

int main(int argc, char *argv[]){
    printf("P ");
    if(fork()==0)
        printf("C ");
    return(0);
}
```

Problem 4 Develop/write a simple program that exemplifies the use of the `read()`, `write()`, and `lseek()` system calls applied perform input/output to disk file(s). Explain the developed program in a succinct manner.

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-so16.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$^{\mathrm{TM}}$-2008. 2008. [Online]. Available: `http://www.opengroup.org/onlinepubs/9699919799/` .