FIT2100 Assignment #2 (Part B) Interprocess Communication with C Programming Week 12 Semester 2 2017

Dr Jojo Wong
Lecturer, Faculty of IT.
Email: Jojo.Wong@monash.edu

October 2, 2017

Revision Status

$Id: FIT2100-Assignment-03.tex, Version 1.0 2017/10/02 12:50 Jojo $

Contents

1 Introduction

This final assignment is due on **22nd October 2017 (Sunday) by 5:00pm**. It is worth **15% of the total unit marks**. A **penalty of 10% per day** will apply for late submission. Refer to the FIT2100 Unit Guide for the policy on extension or special consideration.

Note that this is **an individual assignment** and **must be your own work**. Please pay attention to Section 3.2 of this document on the University's policies on the *Academic Integrity, Plagiarism and Collusion*.

This assignment consists of **three implementation tasks**. Each task should be implemented separately as individual C programs. All the program files and any supporting documents should be compressed into one single file for submission. (The submission details are given in Section 3.)

Assessment: Each of the three tasks has an equal weightage of marks.

2 Interprocess Communication

Interprocess communication is a Unix facility to enable two or more distinct processes to communicate with each other within the same computer system as well as across a network system.

As presented in Practical 5 (Week 10), there are a number of mechanisms that can be deployed to support interprocess communication as a **client-server** model. Under the setting of this model, one process is referred to as the *server* whose responsibility is to fulfil or satisfy the requests put forward to it by the other process, referred to as the *client*.

A client-server model

In this assignment, you are required to implement a client/server model in Unix by using three mechanisms: **named pipes (FIFO)**, **message queues**, and **sockets**. For the implementation of each interprocess communication mechanism, a server program and a client program need to be implemented and execute as a pair.

- **The role as a client:** The client sends a list of Unix commands to the server using each of the three interprocess communication mechanisms.
- **The role as a server:** Upon receiving the Unix commands from the client through a specific interprocess communication mechanism, the server interprets and executes each of the Unix commands accordingly.

Running your program with an input file

For each pair of the client/server programs implemented for each task below, the client program should take an input file as a command line input or as an input redirection. You may name the input file as "`commands.txt`", for example.

The input file should consist of a list of Unix commands with each command as a separate line in the input file. For commands which require arguments for execution, the arguments should be presented on the same line as the command itself with space separated.

Note: All the programs should be implemented on the Unix/Linux operating system with the C programming language. To execute, the server program should be started first in the background before the client program is executed.

2.1 Task 1: Named Pipes (FIFO)

In the first task, you will implement the *named pipe* mechanism, which is also known as the *first-in-first-out (FIFO) device*. The server program should create a new named pipe to be shared with the client program, such that the server can receive and execute the Unix commands sent by the client through the same named pipe.

Note: Name both the server and client programs for Task 1 as "`yourstudentid_pipeserver.c`" and "`yourstudentid_pipeclient.c`" respectively.

2.2 Task 2: Message Queues

For the second task, you will implement the *message queue* mechanism. The server program should create a new message queue and then waits for the Unix commands to be sent to it from the client program through the message queue. Upon receiving the Unix commands, the server will execute each of them accordingly.

The client program should send the Unix commands as messages of `Type 1` and send a `Type 2` message to inform the server that the end of the command list has been reached — no more Unix commands for execution.

Note: Name both the server and client programs for Task 2 as "`yourstudentid_mqserver.c`" and "`yourstudentid_mqclient.c`" respectively.

2.3 Task 3: Unix Sockets

The final interprocess communication mechanism that you will implement is based on the Unix *sockets*. The server program should create a new socket and name it with a Unix domain name (pathname), such that the client program will know which socket to connect to for communication. The server will then listen to the incoming connection request from the client program.

The client program, on the other hand, should attempt to connect to the known socket setup by the server program. Upon successfully connected, the client will send the Unix commands to the server through that socket. As in Task 1 (Section 2.1) and Task 2 (Section 2.2), the server will execute each of the Unix commands received from the client through the socket.

Note: Name both the server and client programs for Task 3 as "`yourstudentid_sktserver.c`" and "`yourstudentid_sktclient.c`" respectively.

2.4 Sample Output

The following screen shot demonstrates the execution of a pair of client and server programs communicating through a named pipe. In this sample output, each Unix command from the client is printed on the terminal before the server executes the command and displays the output of the command execution.

```
$ ./pipeserver &
$ ./pipeclient < commands.txt
client: echo FIT2100
server: FIT2100
client: date
server: Sun 1 Oct 2017 17:37:44 AEDT
client: who am i
server: jojowong ttys000 Oct 1 17:37
$
```

(Note: "$" indicates the shell command prompt.)

2.5 Important Notes

Commenting your code is essential as part of the assessment criteria (refer to Section 2.6). You should also include comments at the beginning of your program file, which specify your name, your Student ID, the start date and the last modified date of the program, as well as with a high-level description of the program. In-line comments within the program are also part of the required documentation.

2.6 Marking Criteria

The assessment of this assignment will be based on the following marking criteria. The same marking criteria will be applied on each of the implementation tasks (Task 1 to Task 3):

- 60% for working program;
- 20% for code architecture (algorithms, data structures, use of procedures and libraries, etc.);
- 10% for coding style (clarity in variable names, function names, etc.);
- 10% for documentation (both program comments and user documentation).

3 Submission

There will be NO hard copy submission required for this assignment. You are required to submit your assignment as a `.zip` file named with your Student ID. For example, if your Student ID is `12345678`, you would submit a zipped file named `12345678_A2B.zip`. Note that marks will be deducted if this requirement is not strictly complied with.

Your submission is via the assignment submission link on the FIT2100 Moodle site by the deadline specified in Section 1, i.e. **22nd October 2017 (Sunday) by 5:00pm**.

3.1 Deliverables

Your submission should contain the following documents:

- A completed the assignment submission statement for online submission via the FIT2100 Moodle site.
- An user documentation (not more than 3 pages) in PDF format with clear and complete instructions on how to run your programs. (Note that your programs must at least run on the Unix/Linux system in the University's computer labs. Any submission that does not compile or run accordingly will receive no marks.)
- Electronic copies of ALL your files that are needed to run your programs.

Marks will deducted for any of these requirements that are not strictly complied with.

3.2 Academic Integrity: Plagiarism and Collusion

Plagiarism means to take and use another person's ideas and or manner of expressing them and to pass them off as your own by failing to give appropriate acknowledgement. This includes materials sourced from the Internet, staff, other students, and from published and unpublished works.

Collusion means unauthorised collaboration on assessable work (written, oral, or practical) with other people. This occurs when you present group work as your own or as the work of another person. Collusion may be with another Monash student or with people or students external to the University. This applies to work assessed by Monash or another university.

It is your responsibility to make yourself familiar with the University's policies and procedures in the event of suspected breaches of academic integrity. (Note: Students will be asked to attend an interview should such a situation is detected.)

The University's policies are available at: `http://www.monash.edu/students/academic/policies/academic-integrity`