Taylor’s University
School of Computer Science & IT

Operating Systems, ITS60503

Assignment 1

Venantius Kumar S.

Administrative

This assignment will be marked out of 100 marks and is worth 15% of the marks for this course.

Warning on Plagiarism

Students are reminded that this assignment is to be attempted individually. Plagiarism of any form will be reported to the Head of School for action. If it is determined that plagiarism or other academic misconduct has occurred, serious consequences can result. This can include a result of zero marks being recorded for this assessment. A permanent record is also made, and further academic misconduct can result in disciplinary action. See the school and university rules on plagiarism for more details. You may also wish to remember that the lecturer in this subject also lectures in system administration which includes teaching methods of searching and comparing files and their contents.
This is not a group assignment, thus every student needs to work on their own submission.
The example files used to check your assignment will be different from the ones presented on this specification sheet.
The assignment is due on Tuesday, 20th October 2015 at 23:59 hrs.

Objectives

The labs and assignments in this subject essentially form a second subject devoted to Unix systems programming. This first assignment will give students the opportunity to investigate how pipes work in Unix.

Additionally, students will be using system level functions for parts of the I/O rather than the C language functions. Doing I/O with the system functions requires a different approach than the one that is used with C.

The twin concepts of processes and programs are central to an understanding of how Unix (and for that matter *any* operating system) manages multiple concurrent tasks. Actually writing code that creates and terminates processes is one of the best ways to clearly understand the distinctions between programs and processes.

Finally there is an introduction to basic signals as a means of communicating between Unix processes.

Introduction

This section provides a general description of the way that your assignment is expected to work. It is not the specification of the detailed requirements.

Your completed assignment will be run by typing the name of the parent program at a command line. Since the directory where you are building your assignment is unlikely to be in the PATH, you will probably need to preface the name with "./" to tell the shell that the program is in the current directory.

Basic Specifications

The assignment requires you to write code in the C programming language and a makefile which builds two executable files.

The assignment requires the writing of code for two programs which will be launched to create a total of four interrelated processes.

The parent process will be the controlling process of the process group. In addition to creating two of the child processes and linking their I/O using pipes, the parent will read a message file that will be transmitted through the pipes to the child processes. The format of the message file is strictly fixed to make reading as easy as possible. Fancy I/O in C is *not* the point of the assignment.

The exact format of the message lines in the file is:

N<tab|<Up to 127 ASCII characters in the message| where N is the child number and the text string is guaranteed to not exceed 127 bytes. This allows for the safe use of a **`char buffer[128]`** buffer since there is room for the terminating null byte.

1. The parent process(1) will fork to create one new process(2).
2. Process(2) will fork a child, process(3).
3. After each process creation, the code will create an unnamed pipe from stdout of process(1) to stdin of process(2), and from stdout of process(2) to stdin of process(3).
4. Following the fork of process(3), process(1) will fork another child process(4). Piped messages will need to be used to ensure the correct order of events.
5. The code will also create a named pipe from process(3) to process(4).
6. Process(4) will also have a pipe back to the parent (process(1)).
7. The parent and each child process will produce a log of messages sent and received. A log entry will be: timestamp full_message from or to pipe <either keep or forward| The timestamp is the long long value returned by the gethrtime() and the full message can simply be displayed and then keep or forward depending on the disposition
8. The parent will read messages one at a time from the message file and pass them to the first child process.
9. Each child process, as it receives a message will read the number and determine if it is the intended recipient of the message. It will then log the message as described above and then either keep the message or pass it to the next child process in line. (Note that a message to "child 4" is to be passed through all three child processes and then back to the parent.)

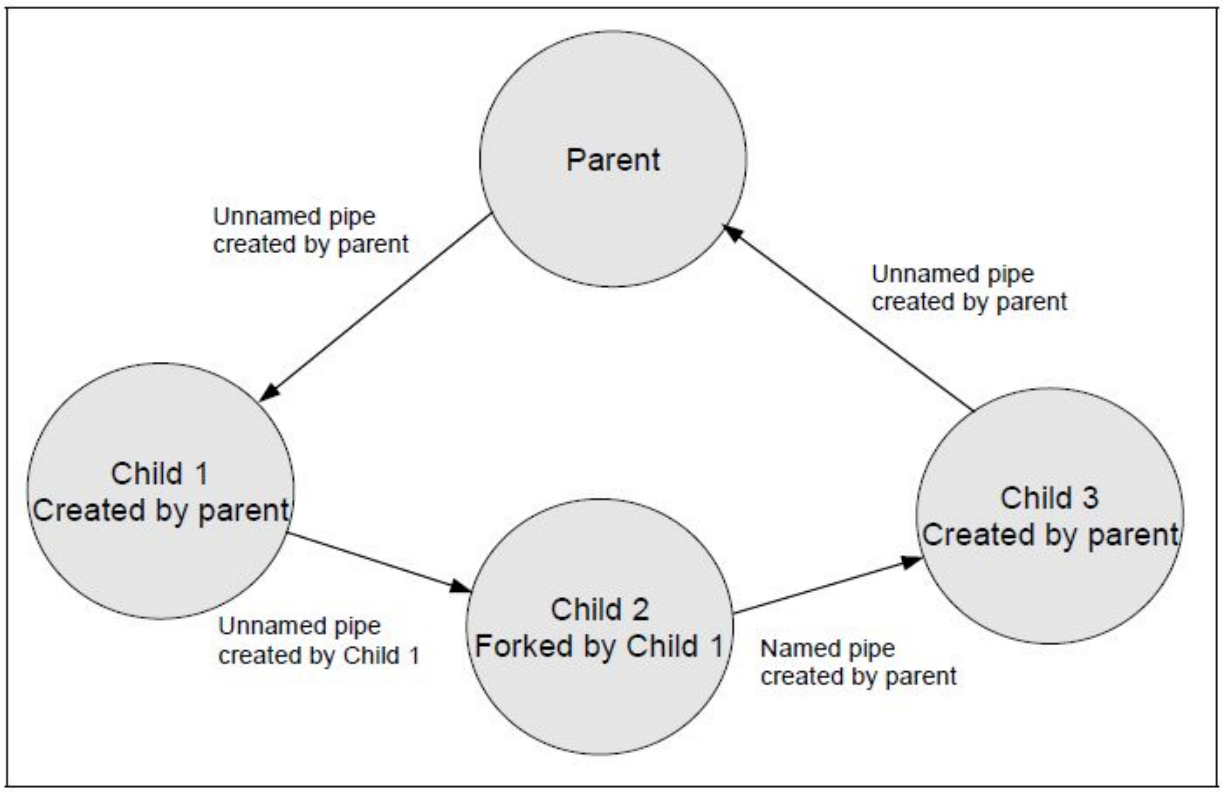

Figure 1: Diagram of the inter-process communications

Overall the relationship between the processes in the system looks something like this:

Submission Information

• This assignment is worth 15% of your final mark.

• This assignment is due on 6th October 2015, and must be submitted before 23:59.

• Submission is may be done by uploading a zipped folder of your source codes along with necessary makefile.

Important Announcement Regarding Requesting Extensions.

With the exception of dire circumstances, no extension requests will be considered within 5 working days of the submission date.

“Dire Circumstances”, are problems such as an injury or serious illness for you or a close relative, *etc*. Standard university special consideration requirements apply. You will be required to supply evidence of your situation either to the lecturer or to one of the Teaching and Learning advisors.

A student requesting an extension at a date near the announced deadline may also be required to submit the portion of the work that has already been completed.