CS421 Introduction to Operating System
Fall 99
Project #1
Implementing Concurrency using processes

Bina Ramamurthy

September 11, 1999

1 Objective

To familiarize the students with:

- Programming in C/C++.
- Unix system/library calls, especially *fork, pipe, exec, wait, kill* and *exit.*
- Command line argument handling.
- Sharing file descriptors among parent and child processes.

2 Problem Statement

This project is to be developed in several small steps to help you understand the concepts better.

1. (5 points) (prj1v1.cc) Implement a C or C++ program that takes two file names as command line arguments, and copies the contents of the first file into the second. Use **read** and **write** system calls. Write a function **rdwrt** to do the reading and writing. Let the main function call this function. Do this as a normal, regular, single-threaded and single process program.
2. (10 points) (prj1v2.cc) Implement the same file copying, but this time a child process is forked and it does the file copying while the parent keeps outputting the period ('.') character at regular intervals. You may use *sleep* library call, if needed, to suspend the main function.
3. (10 points) (prjv3.cc) Instead of the parent staying idle, if the file descriptors are shared then both the parent and the child can be working to accomplish the copying. Let the child as well as the parent do the reading and writing function. (Both can call the function rdwrt written above in version 1.) Make sure that the order of forking a child and the opening of the files for reading and writing are such that the file descriptors are shared.
4. (10 points) (prj1v4.cc) Consider a very large file copying. Instead of just one child, fork 5 child processes that share the file descriptors among them and the parent. Measure and compare the times for version 2, version 3 and version 4.
5. (10 points) (prj1v5.cc) Fork two child processes, let one of them do the reading and the other do the writing, communication among the two carried out by pipe IPC mechanism.
6. (5 points) (prj1v6.cc) Fork a child process that uses the *exec* system call and Unix command to copy *cp* to copy the files.

3 Implementation Details

1. In general, the execution of any of the programs above will is carried out by specifying the executable program name followed by the command line arguments.

2. See the man pages for more details about specific system or library calls and commands: UNIX fork(2), pipe(2), execve(2), execl(3), execlp(3), cat(1), wait(2) etc.

3. One of the dangers of learning about forking processes is leaving unwanted processes active and wasting system time. Make sure each process terminates cleanly when processing is completed. Parent process should wait until the child processes complete, print a message and then quit. Do not leave behind any processes. See kill(2), ps(1), ps(1b), sps(1).

4. Your program should be robust. If any of the calls fail, it should print error message and exit with appropriate error code. Always check for failure when invoking a system or library call. By convention, most UNIX calls return a value of negative one (-1) in case of an error (but check the `RETURN VALUES` section of each man page for details), and you can use this information for a graceful exit. Use cerr, perror(3), or strerror(3) library routines to generate/print appropriate error messages.

   For example, something like the following (but with a real syscall name, and a real bad value check):

   ```
   if (syscall(args) == BAD_VALUE) {
       perror(``syscall'');
       exit(1);
   }
   ```

4 Material to be Submitted

1. Submit online the **source code each of the programs**. Use meaningful names for the file so that the contents of the file is obvious.

2. Only internal documentation is needed. Please state clearly the purpose of each program at the start of the program. Add comments to explain your program. (-5 points, if insufficient.)

3. Record the timings and timing comparison at the beginning of version 5 program.

4. Test runs: It is very important that you show that your program works for all possible inputs. Submit online **a single typescript** file clearly showing the working of all the programs for correct input as well as graceful exit on error input.

5. Submit a README file describing which program file corresponds to which part of the project. And include any special notes, instructions, etc.

6. You are required to submit a **Makefile** for you project. It must be set up so that just typing *make*, once, in your submission directory, will correctly compile all versions of the programs and get the executables ready.

5 Due Date

`9/25/99` before midnight.