152115017 INTRODUCTION TO OPERATING SYSTEMS(A)
Instructor: Research Assistant Dr. Zuhal Can

Project 1: Using Unix processes

The purpose of this project is to lo learn concurrent execution of processes and effectively use Unix system calls for process control and management, especially, fork, exec, wait, pipe and kill system call API. This project consists of several code files as defined below.

Project explanation

1. Submit a README file that lists the files you have submitted along with a one sentence explanation. Call it Prj1README.

2. MakeCopy.c : Write a C program that makes a new copy of an existing file using system calls for file manipulation. The names of the two files and copy block sizes are to be specified as command line arguments. Open the source file in read only mode and destination file in read/write mode.

3. ForkCopy.c : Write a C program that creates a new process to copy the files using the MyCopy. This program should spawn a new process using fork system call. Then use execl to execute MyCopy program. The source and destination file names presented as command-line arguments should be passed to execl as system call arguments. The main process waits for completion of copy operation using wait system call.

4. PipeCopy.c : Write a C program that forks two processes one for reading from a file (source file) and the other for writing (destination file) into. These two programs communicate using pipe system call. Once again the program accomplishes copying files, the names of which are specified as command-line arguments.

5. Use CompareTime.c for time to compare the three versions of the file copy programs as specified above and write an explanation about the results in Prj1README. You can either get the results through PreTest program or linux terminal with the command below. ./ComapareTime <ProgramToTest> <src> <dst> <bufferSize>

6. MyShell.c : Write a shell-like program that illustrates how UNIX spawns processes. This simple program will provide its own prompt to the user, read the command from the input and execute the command. It is sufficient to handle just ``argument- less'' commands, such as ls and date.

7. MoreShell.c :Make the mini-shell (from the previous part) a little more powerful by allowing arguments to the commands. For example, it should be able to execute commands such as more filename and ls –l ./tmp etc. (MoreShell.c MoreShell)

8. DupShell.c : Add to the mini-shell ability to execute command lines with commands connected by pipes. Use dup system call to redirect IO. Example: ls -l | wc . (DupShell.c DupShell).

Submission Notes:
1. Add comments to explain your program in each code file. State clearly the purpose of each program at the start of the program. (-5 points, if comments are insufficient).

2. For pre-evaluation, run Pretest.c program using the command below, and, submit output in Output.txt file.
   gcc -o pt Pretest.c
3. Use the given names for files as given in Project explanation above. Compress the files below into Prj1_"Your id number".tar file. Submit through platindys.

   Prj1README
   Prj1header.h
   MakeCopy.c
   ForkCopy.c
   PipeCopy.c
   CompareTime.c
   MyShell.c
   MoreShell.c
   DupShell.c
   Output.txt

4. Due date: 22.03.2020, submit before midnight.

Late policy

For every day the assignment is late after the due date, you will lose 4 points from your assignment score. Assignments will not be accepted after they are five days late.

Additional notes

Use “>” character for shell indicator. Use “q” character to quit your shell programs.