Programming project# 1
Feb 17 2017

Due: Tuesday, Feb 28, at 11:55PM
Total points: 100 programming project points

Programs containing compilation errors will receive failing grades. Those containing run-time errors will incur a substantial penalty. You should make a serious effort to complete all programs on time. Programming assignments are individual. You should complete them with your own effort. If you need help, you should come to my office hours or contact me by email. You may discuss concepts with others in the class, but not specific program code**.** The total points of this programming project are 100. However, the 100 points are NOT same as 100 points in exams, quizzes, or quizzes. **If you are not sure about the grading policy, refer to the course syllabus.**

NOTE: Each program should be placed in its own directory. Also, you need to include a Makefile in the same directory as the source code to allow the instructor to compile your program automatically. Similarly, your half-page write-up in plain text format should be placed in the same directory. You will submit a single "ZIP" file that includes all the subdirectories where your programs and documentation are located. The name of ZIP file should be program1_LastName_FirstNameInitial.zip. (**If filename is incorrectly made, you will lose 5 points automatically**). The single ZIP file should contain C/C++ source files and Makefiles in the following subdirectories appropriately: 1-a, 2-a, 3-a, 4-a, 5 and 6

- You might want to use the following Unix command to create a single zip file:
  % zip –r **program1_*yourlastname*.zip** ./1 ./2 ./3 ./4 ./5 ./6

You are strongly recommended to use the online book titled "Advanced Linux Programming" as a reference.

Warm-up exercises (typing existing code!)

[1] (5 points) Creating a separate process using the Linux/Unix fork() system call (Figure 3.9 "Creating a separate process using the UNIX fork() system call" in the textbook)

Create a subdirectory called "1". Change to the subdirectory. **Type** in the C program in Figure 3.9 and compile/run/test the program on IT Linux system. Explain in detail how APIs (fork(), execlp(),wait()) work in a separate file.

Deliverables: source code, Makefile, written document **(in plain text; at least 100 words in total**)

[2] (5 points) C program illustrating POSIX shared-memory API

Create a subdirectory called "2". Change to the subdirectory. **Type** in the C programs in Figure 3.17 (Producer process illustrating POSIX shared-memory API) and Figure 3.18 (Consumer process illustrating

POSIX shared-memory API) and compile/run/test the program on IT Linux system. Explain in detail how APIs (shm_open(),ftruncate(),mmap(),shm_unlink()) work in a separate text file.

- Typo corrections in Figure 3.17 and 3.18:
  - include <stdlib.h> instead of <stlib.h>.
  - O_RDRW should be replaced by O_RDWR
  - include <sys/mman.h> before <sys/shm.h>
- Compilation:
  - Compile the code with "-lrt" option. More details can be found by running "man shm_open".

Deliverables: source code, Makefile, written document (**in plain text; at least 200 words in total**)

[3] (5 points) Ordinary pipes in Linux/Unix (Figures 3.25 and 3.26 in the textbook)
Create a subdirectory called "3". Change to the subdirectory. **Type** in the C program in Figures 3.25/3.26 and compile/run/test the program under a Linux operating system. Explain in detail how APIs (pipe(), write(), close()) work in a separate file.

Deliverables: source code, Makefile, written document (in plain text; at least 100 words)

[4] (5 points) pid values (Fig 3.34 in the textbook)
Create a subdirectory called "4". Change to the subdirectory. **Type** in the C program in Fig 3.34 and compile/run/test the program under a Linux operating system. **Record** the values of pid at lines A,B,C, and D. Explain in detail how APIs (getpid() and fprintf) works in a separate file.
Deliverables: source code, Makefile, written document (**in plain text; at least 50 words**)

Programming

[5] (50 points) Create a subdirectory called "5". Change to the subdirectory.

Do the programming project named "Project1 -Unix Shell and History Feature" in pages 157-159 of the textbook. In addition to all features of the shell described in the textbook, your shell also needs to support "pipe" and "redirection of standard OUTPUT" features. The prompt of the shell should be "yourLastname_osh>". The following is the summary of the key features that your shell needs to support:

(1) Running a single executable file in foreground (15 points)
e.g., yourLastname_osh> ls –l

(2) Running a single executable file in background (5 points). Once the given program has been executed in background more, your shell needs to print the pid of the child process along with the name of the program and command-line arguments.

e.g.,
yourLastname_osh> sleep 60 &
[1]+ Running (Pid: 12345) sleep 60 &
yourLastname_osh >

(3) Support pipe feature between two executable programs (10 points)
e.g., yourLastname_osh> ls –lR / | more

(4) Support redirection of standard OUTPUT (10 points)

e.g., yourLastname_osh> ls –lR /etc > ./output1.txt
e.g., yourLastname_osh> cat /etc/hosts > ./output1.txt

(5) History feature (10 points)
   a. *!!* command
   b. !*integer* command
   c. *history* command

You need to provide a half-page write-up describing your source code.

Deliverables: source code, Makefile, half-page written document (**in plain text**)

[6] (30 points) Create a subdirectory called "6". Change to the subdirectory.

In this programming assignment, you will write a *simple* file sharing application which is comprised of two separate C programs (i.e., shm_fileuploader and shm_filedownloader), which allow users to exchange a *single* file (maximum size: 10MB) via a shared memory segment. The detailed operations done in each program are presented in the following:

- shm_fileuploader
  - create a *new* shared memory object that will be used as a storage to share the content of a single file. The name of the shared object should be "{yourLinuxID}_filesharing" ({yourLinuxID} should be replaced by your actual Linux user account)
  - Configure the size of the shared memory object. The size should be 10MB.
  - Memory map the shared memory object
  - **The structure called *filesharing_struct* is laid out (i.e., typecast) on the shared memory**. The structure has the following fields
    - Flag (in char data type) indicating whether file has been uploaded.
    - size (in int data type) indicating the size of the file that has been uploaded
    - character array in which the content of a file is stored
  - ask the user to type in the name of the file that will be shared (e.g., "/bin/ls")
  - The content of the specified file is read from the file system and then copied to the appropriate location in the shared memory. The flag and size fields in the shared memory are updated accordingly

- shm_filedownloader
  - Open the shared memory object using "{yourLinuxID}_filesharing" as the name of the object
  - Memory map the shared memory object. The size should 10MB.
  - access the shared memory to find out the following information:
    - whether the content of a file is available on the shared memory
    - size of the file
  - If there is a file already in the shared memory, download the file to your local directory (i.e., store the store in your local directory). You may choose any name for the file. Otherwise, print out a message indicating that there is no file on the shared memory.

Remove shared memory objects that you have created by removing special files under "/dev/shm" directory from time to time.

The instructor will provide more detailed information about the simple file sharing program in the class.

You need to provide a half-page write-up describing your source code.

Deliverables: source code, Makefile, half-page written document (**in plain text)**