COMP3230 Principles of Operating Systems

Programming Assignment One

Due date: Oct. 22, 2023, at 23:59

Total 11 points

(version: 1.0)

Programming Exercise – Implement a job submission program

Objectives

1. An assessment task related to ILO4 [Practicability] – "demonstrate knowledge in applying system software and tools available in a modern operating system for software development".
2. A learning activity related to ILO 2a.
3. The goals of this programming assignment are:
   - to have hands-on practice in designing and developing a shell program, which involves the creation, management, and coordination of multiple processes;
   - to learn how to use various important Unix system functions
     - to perform process creation and program execution;
     - to support interaction between processes by using signals and pipes; and
     - to get the processes' running statistics by reading the /proc file system.

Task

Shell program, commonly known as a command interpreter, is a program that acts as the user interface to the Operating System and allows the system to understand your requests. For example, when you enter the "**ls -la**" command, the shell executes the system utility program called **ls** for you. The Shell program can be used interactively or in batch processing.

You are going to implement a C program that acts as an ***interactive job submission program,*** named ***JCshell***. This program supports the following features (full details will be covered in the Specification section):

1. The program accepts a single command or a job that consists of a sequence of commands linked together with pipes (|) and executes the corresponding command(s) with the given argument list(s).
2. The program can locate and execute any valid program (i.e., compiled programs) by giving an absolute path (starting with /) or a relative path (starting with ./ or ../) or by searching directories under the $PATH environment variable.
3. The program should be terminated by the built-in *exit* command but it cannot be terminated by the Cltr-c key or the SIGINT signal.
4. After the submitted command/job terminated, the program prints the running statistics of all terminated command(s) and waits for the next command/job from the user.

We suggest you divide the implementation of the program into three stages:

- Stage 1 – Create the first version of the JCshell program, which (i) accepts an input line (contains a command and its arguments) from the user, (ii) creates a child process to execute the command, (iii) waits for the child process to terminate, (iv) check the exit status of the child process, and (v) prints the running statistics and the exit status of the terminated child process. If the process is terminated

by a signal, it should be reflected in the output. After the child process terminated, the JCshell program prints the shell prompt and waits for the next command from the user.

- Stage 2 - Modify previous version of the JCshell program to allow it to (i) handle the SIGINT signal correctly, (ii) use the SIGUSR1 signal to control the child process, and (iii) terminate the JCshell program when the user enters the *exit* command.
- Stage 3 - Modify previous version of the JCshell program to allow it to accept no more than 5 commands with/without arguments and the commands are separated by the '|' symbol. The JCshell program should wait for all commands to complete first before printing the running statistics of each terminated command (according to their order of termination).

Specification

The behavior of the JCshell program:

- Like a traditional shell program, when the JCshell process is ready to accept input, it displays a prompt and waits for input from the user. The prompt should consist of the process ID of the JCshell.

  ```
  ## JCshell [60] ##
  ```

  After accepting a line of input from the user, it parses the input line to extract the command name(s) and the associated argument list(s), and then creates the child process(es) to execute the command(s). We can assume that the command line is upper bound by 1024 characters with 30 strings at maximum (including the | symbols).

  When the child process(es) is/are running, JCshell should wait for the child process(es) to terminate before displaying the prompt for accepting the next input from the user.

- The JCshell process should be able to locate and execute any program that can be found by
  - the absolute path specified in the command line, e.g.

    ```
    ## JCshell [65] ##   /home/tmchan/a.out
    ```

  - the relative path specified in the command line, e.g.

    ```
    ## JCshell [65] ##   ./a.out
    ```

  - searching the directories listed in the environment variable $PATH, e.g.

    ```
    ## JCshell [65] ##   gcc
    ```

    where *gcc* is in the directory /usr/bin, and
    $PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

  Please refer to Programming Lab 1 to learn how to locate and execute a valid command or program.

- If the target program cannot be located or executed, JCshell displays an error message:

  ```
  ## JCshell [65] ##   simp
  JCshell: 'simp': No such file or directory

  ## JCshell [65] ##   /bin/simp
  JCshell: '/bin/simp': No such file or directory
  ```

- **|** symbol - if the **|** (pipe) symbols appear in between commands in each input line, JCshell tries to create multiple child processes and "connects" the standard output of the command before **|** (pipe) and links it to the standard input of the command after **|** (pipe). That is, each command (except the

first one) reads the previous command's output. We can assume that the user will not enter more than 4 pipes (i.e., 5 commands) in an input command line. For example, when the user types:

```
## JCshell [261] ##  cat cpu-mechanisms.txt | grep trap | wc -w
456
```

We can assume that it is incorrect to have the | sign as the first or last character on the input line. Also, it is incorrect to allow two | signs without a command in between. For example,

```
## JCshell [261] ##  cat cpu-mechanisms.txt | | grep trap
JCshell: should not have two | symbols without in-between command

## JCshell [261] ##  cat cpu-mechanisms.txt || grep trap
JCshell: should not have two | symbols without in-between command
```

Please refer to Programming Lab 2 to learn how to use *pipe()* and *dup2()* system functions to set up a pipe between two processes.

- JCshell always waits for all commands to complete before printing the running statistics of all terminated child process(es). For example, when the user runs the ***ls*** command:

```
## JCshell [60] ## ls
JCshell  JCshell.c  Makefile  loop.c  loopever  loopever.c  segfault
segfault.c  tloop

(PID)61 (CMD)ls (STATE)Z (EXCODE)0 (PPID)60 (USER)0.00 (SYS)0.00 (REAL)0.00
(VCTX)8 (NVCTX)0
```

All information about the statistics of a process can be found in the stat and status files under the /proc/{process id} directory. **It is a requirement of this assignment to make use of the /proc filesystem to extract the running statistics of a process.** Please refer to Programming Lab 1 to learn how to retrieve those process statistics from the proc filesystem. In addition, you can find more detailed description of the /proc filesystem at https://man7.org/linux/man-pages/man5/proc.5.html.

Here is a summary of the information in the output.

PID	The process ID of the terminated process
CMD	The filename of the command – obtain from /proc/{pid}/stat or /proc/{pid}/status
STATE	The process state (for our application, it is always in Zombie (Z) state) – obtain from /proc/{pid}/stat or /proc/{pid}/status
EXCODE	The exit code of the terminated process – obtain from /proc/{pid}/stat or from waitpid()
EXSIG	The name of the signal that caused the process to terminate
PPID	The process's parent ID – obtain from /proc/{pid}/stat or /proc/{pid}/status
USER	The amount of time (in seconds) the process was in user mode – obtain from /proc/{pid}/stat
SYS	The amount of time (in seconds) the process was in kernel mode – obtain from /proc/{pid}/stat
REAL	The amount of time (in seconds) the process was in the system – obtain from /proc/{pid}/stat and /proc/uptime
VCTX	The number of voluntary context switches experienced by the process – obtain from /proc/{pid}/status
NVCTX	The number of non-voluntary context switches experienced by the process – obtain from /proc/{pid}/status

When a process is terminated because of receiving a signal, it should be reflected in the output.

```
## JCshell [261] ##  ./segfault

(PID)309 (CMD)segfault (STATE)Z (EXSIG)Segmentation fault (PPID)261
(USER)0.30 (SYS)0.00 (REAL)0.31 (VCTX)15 (NVCTX)1
```

If a process is terminated by a signal, JCshell should display the type of signal that caused the program to terminate, e.g., if detects SIGINT, prints "Interrupt"; if detects SIGKILL, prints "Killed". Please refer to Programming Lab 2 to learn how to handle signals and display appropriate messages.

When executing a job (with multiple commands), JCshell should wait for all processes to terminate before displaying the statistics. The order of the output should reflect the termination order of those child processes. Here is an example with three commands joined by two pipes:

```
## JCshell [261] ##  cat cpu-mechanisms.txt | grep trap | wc -w
456
(PID)305 (CMD)cat (STATE)Z (EXCODE)0 (PPID)261 (USER)0.00 (SYS)0.00
(REAL)0.00 (VCTX)11 (NVCTX)0
(PID)306 (CMD)grep (STATE)Z (EXCODE)0 (PPID)261 (USER)0.00 (SYS)0.00
(REAL)0.00 (VCTX)4 (NVCTX)0
(PID)307 (CMD)wc (STATE)Z (EXCODE)0 (PPID)261 (USER)0.00 (SYS)0.00
(REAL)0.00 (VCTX)4 (NVCTX)0
```

Here is another example that shows some processes experienced error situation.

```
## JCshell [261] ##  ./tloop 10 | ./tloop 5 | ./tloop 2
Time is up: 2 seconds
Program terminated.
(PID)330 (CMD)tloop (STATE)Z (EXCODE)2 (PPID)261 (USER)2.00 (SYS)0.00
(REAL)2.00 (VCTX)13 (NVCTX)2
(PID)329 (CMD)tloop (STATE)Z (EXSIG)Broken pipe (PPID)261 (USER)5.00
(SYS)0.00 (REAL)5.00 (VCTX)11 (NVCTX)2
(PID)328 (CMD)tloop (STATE)Z (EXSIG)Broken pipe (PPID)261 (USER)10.00
(SYS)0.00 (REAL)10.00 (VCTX)14 (NVCTX)9
```

- The JCshell process should be able to handle the SIGINT and SIGUSR1 signals. The corresponding signal handlers should be implemented.

  – **SIGINT** signal: The JCshell process and its child processes are required to respond to the SIGINT signal (generated by pressing Ctrl-c or kill command) according to the following guidelines:

  The JCshell process should not be terminated by SIGINT. When the user presses Ctrl-c while the JCshell process is waiting for input from the user, JCshell should react with a new prompt.

  ```
  ## JCshell [60] ##  ^C
  ## JCshell [60] ##  ^C
  ## JCshell [60] ##  ^C
  ## JCshell [60] ##
  ```

  When the user presses Ctrl-c while the JCshell process is waiting for its child process(es) to terminate, the JCshell process should not be terminated, while the child process(es) should respond to the SIGINT signal according to the predefined behavior of the child command(s). For

example, if the command was set to handle SIGINT without termination, the process should continue running; otherwise, the process should terminate.

```
## JCshell [112] ## ./forever
^CReceives SIGINT!! IGNORE IT :)
Receives SIGINT!! IGNORE IT :)
^CReceives SIGINT!! IGNORE IT :)
^C

## JCshell [112] ## ./tloop 10
^C
(PID)113 (CMD)tloop (STATE)Z (EXSIG)Interrupt (PPID)112 (USER)3.01 (SYS)0.00
(REAL)3.02 (VCTX)16 (NVCTX)2
```

– **SIGUSR1** signal: This signal is available to users to define their own activity or event. With the JCshell, all child processes would not run the target commands immediately after creation. We want them to wait for the signal (SIGUSR1) sent by the JCshell before executing the commands. This mechanism guarantees all control structures used by the JCshell for managing processes have been updated before executing the child processes.

- built-in command: ***exit*** – If the user enters the ***exit*** command, JCshell should terminate, and the standard shell prompt reappears. For example, if the user types

```
## JCshell [76] ## exit
JCshell: Terminated
root@ff860ab794d9:/home/c3230# ps
  PID TTY      STAT   TIME COMMAND
    1 pts/0    Ss     0:00 bash
   79 pts/0    R+     0:00 ps
root@ff860ab794d9:/home/c3230#
```

If the ***exit*** command has other arguments, JCshell would not treat it as a valid request and would not terminate. In addition, if the ***exit*** command does not appear as the first word in the command line, JCshell would not treat it as the ***exit*** command.

```
## JCshell [96] ##  not exit
JCshell: 'not': No such file or directory
## JCshell [96] ##  exit now
JCshell: "exit" with other arguments!!!
```

Resources

You are provided with the following file.

- `utility.zip` – this file contains three C files (forever.c, tloop.c, and segfault.c) that can be used for testing the JCshell program.

Documentation

1. At the head of the submitted source code, state clearly the
   - Student name and No.:
   - Development platform:
   - Remark – Describe how much you have completed
2. Inline comments (try to be detailed so that your code can be understood by others easily)

Computer platform to use

For this assignment, you are expected to develop and test your programs on the workbench2 Linux platform. Your programs must be written in C and successfully compiled with gcc. It would be nice if you develop and test your program on your own machine (WSL2 or Ubuntu docker image). After fully testing locally, upload the program to the workbench2 server for the final test.

Submission

Submit your program to the Programming # One submission page on the course's Moodle website. Name your program with this format: JCshell_StudentNumber.c (replace StudentNumber with your HKU student number). As the Moodle site may reject source code submission, please compress your program to the zip or tgz format before uploading.

Grading Criteria

1. Your submission will be tested on workbench2. Make sure that your program can be compiled *without any errors*. Otherwise, we have no way to test your submission and you will get a zero mark.
2. As the tutor will check your source code, please write your program with good readability (i.e., with good code conventions and sufficient comments) so that you will not lose marks due to possible confusion.

Documentation	• Include necessary comments to clearly indicate the logic of the program (-1 point if not include comments). • Include the required student's info at the beginning of the program (-0.5 points if is missing).	
Correctness of the program (11 points)	Process creation and execution – (3 points)	• Should be able to print "`## JCshell ##` " and accept user's input • Should be able to execute a command entered by the user • Can locate and execute a command with a full path, with a relative path, or under the standard PATH • Can execute a command with any number of arguments • Can handle error situations correctly, e.g., incorrect filename, incorrect path, etc. • Should wait for the command to complete before accepting the next command • Should accept the next command for execution • Should remove all zombie processes
	Process creation and execution – use of '|' (3 points)	• Correct use of | symbol; can report improper use of | symbol • Can execute multiple commands (at most five) with any number of arguments that are connected by the pipes
	Print process's running statistics (2.5 points)	• The system can print the process's running statistics in the correct format • The system should print the information after the command/job completed • For a job with multiple commands, the system should output the statistics according to the termination order • The program should retrieve the data from the /proc filesystem (if not, -1 point).

	Use of signals (2 points)	SIGINT signal (1 point) • Correct behavior of the JCshell process and the child processes in handling the SIGINT signal SIGUSR1 signal (1 point) • All child processes should wait for the SIGUSR1 signal before executing the target commands.
	Built-in command: exit (0.5 points)	• Correct use of the *exit* command; can report improper usage

Plagiarism

Plagiarism is a very serious offense. Students should understand what constitutes plagiarism, the consequences of committing an offense of plagiarism, and how to avoid it. **Please note that we may request you to explain to us how your program is functioning as well as we may also make use of software tools to detect software plagiarism.**