Lab #4: Shell Lab

Prof. Jae W. Lee (jaewlee@snu.ac.kr)
Department of Computer Science and Engineering
Seoul National University

TA (snu-arc-sysprog-ta@googlegroups.com)

Before get started

- **Make sure your submission is correctly formatted**

  -
    ```
    {student_id}.tar
    ├── {student_id}.pdf
    └── src
        └── csapsh.c
    ```
- **Few previous submissions had following problems**
  - Do not modify file extension (.zip -> .tar)
    - Renaming to .zip does not compress your file!!!!!!!
  - Build Failures
    - Source code is cut in the middle of your code (cause build failure)
  - Report is not a pdf (its name is *.pdf, but its type is tar)
    - you can check type by `file ${student_id}.pdf` command’s output

Contents

Important Dates

- 14 May. - Lab Hand-out Session (Today!)
- 21 May. - Live Q&A Session (1)
- 28 May. - Live Q&A Session (2)
- 03 Jun. 23:59 - Submission Deadline

- Questions about the lab will be conducted through github issue. Feel free to post questions (except your code!)
- Live Q&A Session is totally optional
  (But highly recommended for this lab)

Before the Presentation:

- **All content on this slide is sourced from README file.**

- **For detailed information, please refer to README.**
  - https://github.com/SNU-ARC/2024_spring_sysprog_Lab4/blob/main/README.md

Goal of this Lab(1/2)

- **We implement a tiny shell program named `csapsh`.**

```
$ ./csapsh
csapsh> sleep 10 &
[1] (26925) { 26925 } Running sleep 10
csapsh> sleep 20 &
[2] (26932) { 26932 } Running sleep 20
csapsh> jobs
[2] (26932) { 26932 } Running sleep 20
[1] (26925) { 26925 } Running sleep 10
csapsh> ls tools | grep my | grep spin
myspin
myspin.c
csapsh> quit
```

Goal of this Lab(2/2)

- **You will learn**
  - how to run other processes through `fork()` and `exec()`
  - how to redirect input/output
  - how to create a pipe between two child processes
  - how to send a group of processes to the background/foreground
  - that proper signal handling is quite tricky

Environment setup(1/3)

- **You can get skeleton code and test bench from git repo**

```
git clone https://github.com/SNU-ARC/2024_spring_sysprog_Lab4.git
```

Environment setup(2/3)<Optional>

- **If you want to keep your own repository, you should keep the lab's visibility to private. Otherwise, others would see your work.**

- **Changing visibility**
  - After cloning the repository, you should change the push remote URL to your own repository.

1. **Create an empty repository that you're going to manage (again, keep it private)**
2. **Copy the url of that repository**
3. **On your terminal in the cloned directory, type**
   ```
   git remote set-url --push origin <repo_url>
   ```
4. **Check with git remote -v if the push URL has changed to yours while the fetch URL remains the same (this repo)**

Environment setup(3/3)

- **The handout contains the following files and directories.**

dir	file	description
src	csapsh.c	Skeleton for csapsh.c. Implement your solution by editing this file.
	jobcontrol.c/h	Implementation of job control APIs (add, delete, list, ...). Do not modify!
	parser.c/h	Implementation of command line parser. Do not modify!
references	csapsh	Reference implementation
tools	Makefile	Makefile to build example binaries and run tests.
	myint.c	Example binary that prints counter every second and send SIGINT.
	myprod.c	Example binary that prints counter every second.
	myspin.c	Example binary that prints a string every second.
	mysplit.c	Example binary that forks a child process and prints a string every second.
	mystop.c	Example binary that prints counter every second and send SIGTSTP.
	sdriver.pl	Driver script to test csapsh with traces.
traces	trace01-2X.txt	Traces of command line used to test your implementation
.	Makefile	Makefile driver program to compile csapsh
	README.md	

Overview of Unix Shell(1/4)

- **A shell is an interactive command-line interpreter.**
  - Runs programs on behalf of the user.
  - Repeatedly prints a **prompt**
  - Wait a command line on ***stdin*** and then carries out some action.

- **Command line (CL)**
  - A sequence of ASCII text words delimited by whitespace.
    - The first word is a built-in command or the pathname of an executable file.
    - The remaining words are command-line arguments.
  - Shell executes the built-in commands in the **current process**.
  - Shell forks a **child process** directed by the pathname of an executable program.
    - A process and its child processes are known collectively as a ***job***.

Overview of Unix Shell(2/4)

- **Background & Foreground**
  - If the command line ends with an ampersand "&", then the job runs in the background.
    - The shell does not wait for the job to terminate before printing the prompt and awaits the next command line.
    - An arbitrary number of jobs can run in the background.

```
csapsh> ./myspin 100 &
```

  - Otherwise, the job runs in the foreground.
    - The shell waits for the job to terminate before awaiting the next command line.
    - At most one job can run in the foreground.

```
csapsh> ./myspin 100
```

Overview of Unix Shell(3/4)

- **Unix shells support the notion of job control.**
  - Allows users to move jobs back and forth between background and foreground.
  - Allows users to change the state of the processes (**running**, **stopped**, or **terminated**) in a job.

- **Signal command**
  - Ctrl-C : causes a **SIGINT** signal to be delivered to each process in the foreground job.
    - The default action for SIGINT is to terminate the process.
  - Ctrl-Z : causes a **SIGTSTP** signal to be delivered to each process in the foreground job.
    - The default action for SIGTSTP is to place a process in the stopped state, where it remains until it is awakened by the receipt of a **SIGCONT** signal.

Overview of Unix Shell(4/4)

- **Examples of built-in commands supporting job control.**
  - `jobs` : List the running and stopped background jobs.
  - `bg <job>` : Change a stopped background job to a running background job.
  - `fg <job>` : Change a stopped or running background job to a running foreground job.
  - `kill <job>`: Terminate a job.

The csapsh Specification(1/3)

- **Prompt string "`csapsh> `".**
- **Command line should consist of a name and optional arguments.**
  - name : built-in command or the path of an executable file.
- **Signal handling**
  - Ctrl-C : cause a SIGINT
  - Ctrl-Z : cause a SIGTSTP
  - Signals should be sent to the current foreground job, as well as any descendants of that job.
- **Foreground & background job.**
  - Command of background job ends with ampersand "&"
- **Multiple jobs in a single command line.**
  - Separated by "&"
  - At most one job (the last one) can run in the foreground

The csapsh Specification(2/3)

- **A process ID (PID), a process group ID (PGID), and a job ID (JID)**
  - Assigned by `csapsh`.
  - JIDs should be denoted on the command line by the prefix '%' (e.g. "%5")
  - PGIDs should be denoted on the command line by the prefix '@' (e.g. "@5")
- **`csapsh` should reap all of its zombie children.**
- **`csapsh` should support the following built-in commands.**
  - **`quit`** : terminates the shell.
  - **`jobs`** : lists all background jobs.
  - **`bg <PID or JID>`** : restarts <PID or JID> by sending it a **SIGCONT** signal, and then runs it in background.
  - **`fg <PID or JID>`** : restarts <PID or JID> by sending it a **SIGCONT** signal, and then runs it in foreground.

The csapsh Specification(3/3)

- I/O file redirection(<, >).

```
csapsh> ls > file
csapsh> cat < file
```

- csapsh should support pipe.

```
csapsh> ls | grep "CSAP" | sort > /tmp/result.txt
```

- csapsh does not support I/O file redirection and pipe for built-in commands. Also, built-in commands does not run in the background.

Code & Test(1/4)

- **Function list of what do you need to implement in this lab with approximate number of lines in our reference solution code.**
  - **`eval`** : Main routine that parses and interprets the command line. [180 lines]
  - **`builtin_cmd`** : Recognizes and interprets the built-in commands. [<10 lines]

    `quit`, **`fg`**, **`bg`** and **`jobs`**.
  - **`do_bgfg`** : Implements the bg and fg built-in commands. [65 lines]
  - **`waitfg`** : Waits for a foreground job to complete. [~10 lines]
  - **`sigchld_handler`** : Catches SIGCHILD signals. [75 lines]
  - **`sigint_handler`** : Catches SIGINT(ctrl-c) signals. [~10 lines]
  - **`sigtstp_handler`** : Catches SIGTSTP(ctrl-z) signals. [~10 lines]
- **To run your shell, type csapsh to the command line.**

```
$ ./csapsh
csapsh> [type commands to your shell here]
```

Code & Test(2/4)

- **Reference Solution**
  - `reference/csapsh` is the reference solution for the shell.
  - Your shell should emit output that is identical to the reference solution.
    - Except for PIDs, of course, which change from run to run.
- **Shell driver**
  - `sdriver.pl` executes a shell as a child process, sends it commands and signals as directed by a trace file, and captures and displays the output from the shell.

```
$ ./sdriver.pl -h
Usage: ./sdriver.pl [-hv] -t <trace> -s <shellprog> -a <args>
Options:
    -h             Print this message
    -v             Be more verbose
    -t <trace>     Trace file
    -s <shell>     Shell program to test
    -a <args>      Shell arguments
    -g             Generate output for autograder
```

Code & Test(3/4)

- **2X trace files (trace{01-2X}.txt) provided**
  - From very simple tests to more complicated tests.
- **To compare your result with the reference shell using trace driver**

```
$ ./sdriver.pl –t trace01.txt –s ../csapsh –a "-p"
$ make test01
```

- **To compare your result with the reference shell using trace driver**

```
$ ./sdriver.pl –t trace01.txt –s ../reference/csapsh –a "-p"
$ make rtest01
```

Code & Test(4/4)

- **Example**

```
$ ./sdriver.pl -t trace10.txt -s ../csapsh -a "-p"
$ make test10
./sdriver.pl -t trace10.txt -s ../csapsh -a "-p"
#
# trace10.txt - Process fg builtin command.
#
csapsh> ./myspin 4 &
[1] (29391) ./myspin 4 &
csapsh> fg %1
Job [1] (29391) stopped by signal 20
csapsh> jobs
[1] (29391) Stopped ./myspin 4 &
csapsh> fg %1
csapsh> jobs
```

Grading Policy

- **Test Bench: 75 %**
  - 25 given trace files (3 points each)
- **Report** : **25 % (should include following contents)**
  - Description of your implementation
  - Difficulties and thoughts during the implementation of this lab
- **For late submission:**
  - A deduction of 20%p per 24 hours

Submission(via eTL)

- **Write-up**
  - Briefly describe your implementation.
  - Filename: [student_id].pdf (example: 2024-12345.pdf)
  - **Please** submit it in **pdf** format. Other formats are not accepted.
- **Compress your source code and write-up into a single file**
  - Compress **csaph.c** and your report with following command
  - `$ tar -cvf 2024-12345.tar src/csapsh.c 2024-12345.pdf`
  - Filename should be [student_id].tar (example: 2024-12345.tar).
  - **Please** submit it in **tar** format. Other formats are not accepted.
  - Refer README.md for submission instructions.
- **Submission deadline: by 23:59 on June 3, 2024**

Questions?