Fundamental of Operating System (COIS3320H-A)

Bin Guo, Winter 2024  
  
Assignment 1 [40 points]

|  |  |
| --- | --- |
| **Name** |  |
| **ID** |  |

|  |  |
| --- | --- |
| **Group Partner’s Name** |  |
| **Group Partner’s ID** |  |

**Note**: In your C programs, you should follow good programming style, which includes providing instructive comments and well-indented code.

**Part 1: Short Answer Questions [10 points, each questions has 2 points]**

**Your answers normally have 50 words. Less than 50 words will not get marks.**

1. List at least five Operating Systems you know.
2. What is the difference between the kernel mode and the user mode for the Linux?
3. What is the system-call? Give an example of API in OS that use the system-call.
4. What is cache? Why the CPU has cache?
5. What is the difference between the Static Linking and Dynamic Linking when compiling the code.

**Part 2: Programming Questions**

1. **This question involves designing a kernel module (based on Lab 2). [10 points]**

The Linux kernel keeps track of the global variable jiffies, which maintains the number of timer interrupts that have occurred since the system was booted. The jiffies variable is declared in the file <linux/jiffies.h>.

Design a kernel module that creates a /proc file named /proc/jiffies that reports the current value of jiffies when the /proc/jiffies file is read, such as with the command

cat /proc/jiffies

Be sure to remove /proc/jiffies when the module is removed.

This question should be completed using the Linux virtual machine you installed as part of Lab1.

**Deliverables**:

1. **jiffies.c** - You are to provide your solution as a single C program named jiffies.c that contains the entire solution for question 1.

It is important for you to name your file jiffies.c as the TA grading this question has a Makefile using this name to test your code.

2. **q1output.txt** - You are also to provide the output of dmesg command in a text file called q1output.txt. The output should show that the kernel module was loaded into the kernel, the current value of jiffies, and final that the kernel module was removed.

Note: It is possible that some version of Linux systems (like MacOS) does not support jiffies file. In this case, you should connect to our TA for help (to find a correct Linux system).

Note: If you still cannot find a proper Linux OS, it is acceptable that your code including all the details and expected results without running (you can still get marks). Only such kernel module requires specific Linux OS, and all the other labs or assignments can run on common Linux OS.

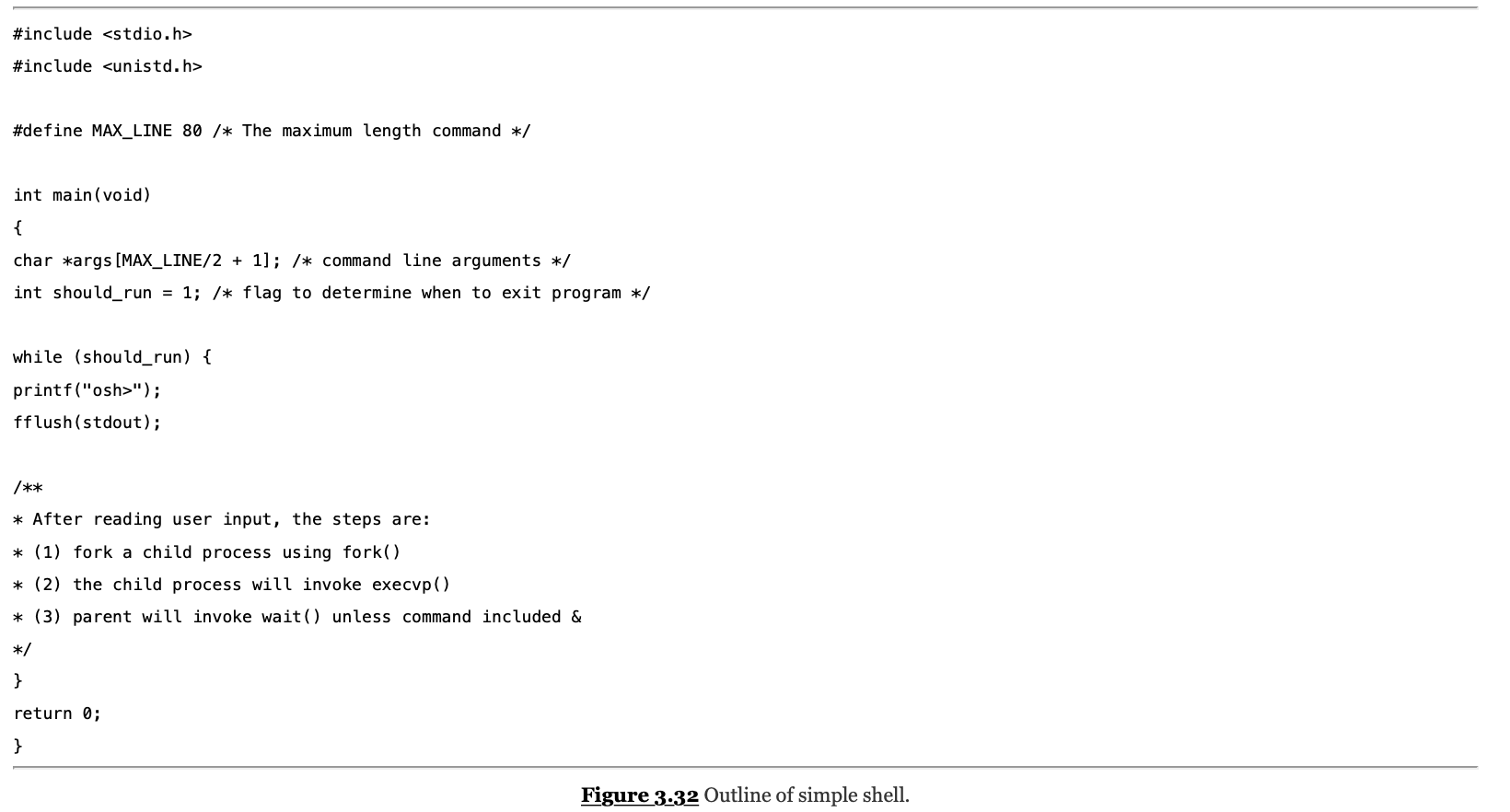
1. **UNIX Shell and History Feature [20 points]**

This question consists of designing a C program to serve as a shell interface that accepts user commands and then executes each command in a separate process. A shell interface gives the user a prompt, after which the next command is entered. The example below illustrates the prompt osh> and the user’s next command:

cat prog.c. The UNIX/Linux cat command displays the contents of the file prog.c on the terminal using the UNIX/Linux cat command and your program needs to do the same.

osh> cat prog.c

The above can be achieved by running your shell interface as a parent process. Every time a command is entered, you create a child process by using fork(), which then executes the user’s command using one of the system calls in the exec() family (as described in Chapter 3). A C program that provides the general operations of a command-line shell can be seen below.



The main() function presents the prompt osh-> and outlines the steps to be taken after input from the user has been read. The main() function continually loops as long

as should\_run equals 1; when the user enters exit at the prompt, your program will set should\_run to 0 and terminate.

This question is organized into two parts:

**First Part: [10 points] Creating the child process and executing the command in the child**

Your shell interface needs to handle the following two cases.

1. **Parent waits while the child process executes.**

In this case, the parent process first reads what the user enters on the command line (in this case, cat prog.c), and then creates a separate child process that executes the command. Unless otherwise specified, the parent process waits for the child to exit before continuing. This is similar in functionality to the new process creation illustrated in Lab 3.

1. **Parent executes in the background or concurrently while the child process executes (similar to UNIX/Linux)**

To distinguish this case from the first one, add an ampersand (&) at the end of the command. Thus, if we rewrite the above command as osh> cat prog.c & the parent and child processes will run concurrently.

**Second Part:** [**10 points] Modifying the shell to allow a history feature**

In this part your shell interface program should provide a ***history*** feature that allows the user to access the most recently entered commands. The user will be able to access up to 5 commands by using the feature. The commands will be consecutively numbered starting at 1, and the numbering will continue past 5. For example, if the user has entered 35 commands, the 5 most recent commands will be numbered 31 to 35. The user will be able to list the command history by entering the command

osh> history

As an example, assume that the history consists of the commands (from most to least recent): ls -l, top, ps, who, date. The command history should output:

5 ls -l

4 top

3 ps

2 who

1 date

Your program should support two techniques for retrieving commands from the command history:

1. Whentheuserenters**!!**,themostrecentcommandinthehistoryisexecuted.In the example above, if the user entered the command:

Osh> !!

The ‘ls -l’ command should be executed and echoed on user’s screen. The command should also be placed in the history buffer as the next command.

1. Whentheuserentersasingle**!**followedbyaninteger*N*,the*Nth*commandin the history is executed. In the example above, if the user entered the command:

Osh> ! 3

The ‘ps’ command should be executed and echoed on the user’s screen. The command should also be placed in the history buffer as the next command.

**Error handling:**

The program should also manage basic error handling. For example, if there are no commands in the history, entering !! should result in a message “No commands in history.” Also, if there is no command corresponding to the number entered with the single !, the program should output "No such command in history."

**Deliverables**:  
  
**shell.c** - You are to provide your solution as a single C program named shell.c that contains your solution for this question.

**Testing**:

You should design test cases in order to test the correctness of your program. The test cases should include the widely used Linux cmd, which may have one or multiple parameters. For example:

* ls -l
* ps -aux
* which python
* gcc -o …
* ….