CMPT-300 Day: Assignment 1

1 The Deadline

Due at 4:30pm, Oct. 3rd 2003, Friday.

2 Where to Submit HW1

Please drop your answers for Q1 to Q5 into the assignment drop-in box outside the room ASB 9838. That box is labeled with "CMPT 300 D, S. Zhou".
Submit your answers for Q6 and Q7 through online submission server.

3 Questions

In this homework, we have **7** problems counting for **36** marks. You need to answer all of these questions. Collaboration is strictly prohibited.

3.1 (2 marks)

What are the differences and commonalities between a trap and an interrupt?

3.2 (4 marks)

Suppose you have a multi-user computer system. Which of the following instructions should be allowed only in the kernel mode? Use 1-2 sentences to justify your answers as why.

- Read data from disk;
- Disable all interrupts;
- Read the time-of-day clock;
- Set the time-of-day clock

3.3 (4 marks)

Suppose you already have the program of process P that does not have any child process. Please write a small program in pseudo-code for P so that P can be the root node of the following process hierarchy.

```
    |--- C1 ----C11
    |      |
P---       |-----C12
    |
    |--- C2
```

`Notice: P has two children C1 and C2, and C1 has C11 and C12 as its children.`

3.4 (4 marks)

Please explain the following two statements

- Threads are wanted where the threads block often.
- The protection among the threads within the same process is impossible and unnecessary.

3.5 (2 marks)

The register set and stack are private to each thread. Please explain why they are not shared among the threads.

3.6 (8 marks)

This is a programming-based question. Your program must be written in C/C++ on UNIX or LINUX platform.

In this task, your program will first create two pipes. Then your program will create a child process. Now suppose that your program is called *parent*. The task is described as follows.

The parent wants to communicate with the child. It initializes the conversation and sends the first message to the child. After the child gets the message, it prints it out on the screen. After that, the child replies to the parent by sending a new message. After the parent gets it, the parent prints out the message it received too, and then sends out the second message. The process goes on like this until all the messages are sent and received.

In order for the parent to send a message to the child, the parent will write to one of the two pipes created and the child will be listening to that pipe. Correspondingly, in order to send a message to the parent from the child, the child will write to the second pipe created by parent and the parent in turn will be listening to it.

The following is a set of sample conversation messages:
Message sent from the parent: Hello, my child.
Message replied by the child: Hello, my father.

Message replied by the parent: How are you?
Message replied by the child: I am fine. And how are you?
Message replied by the parent: I am fine too. Good-bye.
Message replied by the child: Good-bye.

You could just use these sample messages in your program. The purpose of this question is to let you feel that the pipes are a shared resource between the parent and child processes. So do not make the program complex. You might want to use the command *man* to take a look at the manual for the system call *pipe*. Please refer to page 41 in textbook for a brief introduction of pipe.

Requirements:

- Document your code with header comments at the top of each source code and header file and before each function.
- Create a Makefile that builds the executable program when the command *make* is typed on the command line.
- Submit your source code and header files with the Makefile through online submission server at *https://onara.cs.sfu.ca/*. Do **NOT** submit the executable or object (.o) files. So make sure your Makefile works correctly.

3.7 (12 marks)

Please build a simple shell similar to "csh" using C **or** C++ language on UNIX **or** LINUX system. Please refer to page 41 in textbook for a simple introduction of shell.

When you type a command in your shell, it creates a child process using *fork()*, then sets the program that child will run using one of *execv(), execve(), execvp()*. If you specify arguments, it passes those arguments on to the child. If you use the "&" to run the program in the background, then your shell does not wait, but immediately prints another prompt; but without "&", your shell program waits for the child to terminate before continuing.

Requirements:

- Prompt the user to type in commands. The prompt is '%'.
- When the user types a command with arguments, executes the command with those arguments. Should assume that the program resides in one of the directories in the current PATH environment variable. Continues to prompt the user for commands until the user types the command *myexit*, at which time your shell terminates.
- If the user types an ampersand (&) at the end of the command, the shell prompt should

return immediately. Otherwise, the shell prompt should not return until the child process terminates.

- You need **NOT** to implement the commands typed by user (e.g., *ls, cp*) by yourself. What you need is to pass all the commands to the UNIX by calling one of *execv(), execve(), execvp()*. The only exception are the commands *cd* and *myexit*. You need to write codes to handle these commands.
- Your shell need **NOT** support pipe, input/output redirection or shell script. Also it can ignore the CTRL+C and CTRL+Z keys.
- You must **NOT** use the C library function "*system()*". You will get zero mark for this question if you do so.
- Create a Makefile that builds the executable shell when the command *make* is typed on the command line.
- Document your code with header comments at the top of each source code and header file and before each function.
- Submit your source code and header files with the Makefile through online submission server at *https://onara.cs.sfu.ca/*. Do **NOT** submit the executable or object (.o) files. So make sure your Makefile works correctly.
- This assignment is designed to get you familiar with UNIX/LINUX and C/C++.