Computer Engineering Department – ITU
CE301L: Operating Systems Lab

Course Instructor: Dr. Rehan Ahmed	**Dated:**
Lab Engineer: Muhammad Umair Shoaib	**Semester: Fall 2023**
Batch: BSCE21	

Lab 6 – OS Shell Utility: Implementation of Parallel and Path-Based Commands and Batch Mode

Name	**Roll number**	**Total (out of 35)**

Checked on: ______________________________

Signature: ________________________________

6 OS Shell Utility: Implementation of Parallel and Path-Based Commands and Batch Mode

Introduction

In this lab, we will add a few new features to the shell that we had built in the last lab, like parallel commands, the execution of path-based commands and batch mode execution.

Objectives

This lab aims to achieve the following key objectives:

- Implement path-based commands
- Use fork, exec and wait system calls to create processes and execute commands
- Learn to parse user-entered commands
- Understand how scripts are run by actual shells by implementing such a feature in our custom shell program

Theory and background

6.1 Shell specifications for this lab

In this lab, we will build upon our *itush* shell and add to it several new features. We will enable our shell to take more than one commands in parallel by using the "&" operator. Another feature that we will add is the option to define paths from where command executables can then be run. Lastly, our shell will now be able to take in batch files and execute commands written inside the batch file.

6.1.1 Parallel commands

Your shell will also allow the user to launch parallel commands. This is accomplished with the *ampersand (&)* operator as follows:

```
itush> cmd1 & cmd2 args1 args2 & cmd3 args1
```

In this case, instead of running `cmd1` and then waiting for it to finish, your shell should run `cmd1`, `cmd2`, and `cmd3` (each with whatever arguments the user has passed to it) in parallel, before waiting for any of them to complete.

Then, after starting all such processes, you must make sure to use `wait()` (or `waitpid`) to wait for them to complete. After all processes are done, return control to the user as usual.

6.1.2 Paths and Linux command execution

In your itush shell, if the user types `ls -la /tmp`, your shell should run the program `ls` residing inside `/bin` directory with the given arguments `-la` and `/tmp`.

Before execution of a command on your shell, the user must specify a *path variable* using the built-in `path` command implemented in last lab to describe the set of directories to search for executables; the set of directories that comprise the path are sometimes called the *search path* of the shell. The path variable contains the list of all directories to search, in order, when the user types a command.

Note that the shell itself does not implement `ls` or other commands (except the built-in commands implemented in the previous lab: `path`, `exit` and `cd`). All it does is find those executables in one of the directories specified by `path` and create a new *process* to run them.

Your initial shell path should contain one directory: `/bin`

Note that most shells allow you to specify a binary specifically without using a search path, using either *absolute paths* or *relative paths*. For example, a user could type the absolute path `/bin/ls` and execute the `ls` binary without a search path being needed. A user could also specify a relative path which starts with the current working directory and specifies the executable directly, e.g., `./main`. For your itush shell, you do not have to worry about these features.

6.1.2.1 The access() system call

To check if a particular file exists in a directory and is executable, consider the `access()` system call. For example, when the user types `ls`, and path is set to include both `/bin` and `/usr/bin`, try the following:

```
access(“/bin/ls”, X_OK);
```

If that fails, try `/usr/bin/ls`. If that fails too, it is an error.

6.1.3 Batch and interactive modes

According to the specifications provided in the previous lab, at the moment, our shell itush is basically just an interactive loop; it repeatedly prints a prompt `itush>`, parses the input, executes the command specified on that line of input, and waits for the command to finish. This is repeated until the user types exit.

The mode above is called *interactive mode*, and allows the user to type commands directly. For this lab, we will enable our shell to support a *batch mode* as well, in which the shell reads input from a *batch file* and executes commands from therein.

Here is how you run the shell with a batch file named `batch.txt`:

```
$ ./itush batch.txt
```

One difference between batch and interactive modes is that, in interactive mode, a prompt is printed (`itush>`). In batch mode, no prompt should be printed.

Lab tasks

6.2 Task 1: Parallel commands [Marks: 7]

Modify your program to enable it to take parallel commands as described in section 6.1.1.

Show working to lab engineer and paste code and screenshots showing the execution of the built-in commands in a parallel manner. [7]

6.3 Task 2: Implementation of path-based commands [Marks: 7]

Add program logic to your `itush.c` file to implement path-based commands as described in Section 6.1.2. Your program should also be able to run parallel path-based commands using the `fork` and `exec` commands.

Show your work to the lab engineer and in your report, paste code and screenshots showing the execution of path-based commands as a separate single command and in parallel with other commands using the & symbol. [7]

6.4 Task 3: Implementation of batch mode [Marks: 6]

Modify your program to include the batch mode. The batch mode should run commands from a batch file as described in Section 6.1.3.

Show your work to the lab engineer and in your report, paste code and screenshots showing successful execution of commands contained inside a batch file. [6]

6.5 Analysis [Marks: 5]

1. Describe your approach for implementation of the batch-mode of the itush shell. [3]

2. What is the use of the `access` system call? [2]

The following table will be filled by the lab engineer during grading:

Task	**Max marks**	**Obtained marks**	**Remarks if any**
1	7		
2	7		
3	6		
Analysis	5		

Assessment rubric for Lab 6

Performance	**CLO**	**>80% complete (4 – 5)**	**40 – 80% complete (2 – 3)**	**<40% complete (0 – 1)**	**Marks**
1. Realization of experiment	1	Conceptually understands the topic under study and develops the experimental setup accordingly	Needs guidance to understand the purpose of the experiment and to develop the required setup	Incapable of understanding the purpose of the experiment and consequently fails to develop the required setup	
2. Conducting experiment	1	Sets up the required software, writes and executes bash/C/C++ programs according to the requirement of task and examines the program output carefully	Needs assistance in setting up the software, makes minor errors in writing codes according to task requirements	Unable to set up the software and to write and execute program according to task requirements	
3. Data collection	1	Interprets program output and completes data collection as required in the lab task, ensures that the data is entered in the lab report according to the specified instructions	Completes data collection with minor errors and enters data in lab report with slight deviations from provided guidelines	Fails at observing output states of experimental setup and collecting data, unable to fill the lab report properly	
4. Data analysis	1	Analyzes the data obtained from experiment thoroughly and accurately verifies it with theoretical understanding, accounts for any discrepancy in data from theory with sound explanation, where asked	Analyzes data with minor error and correlates it with theoretical values reasonably. Attempts to account for any discrepancy in data from theory	Unable to establish a relationship between practical and theoretical results and lacks in-depth understanding to justify the results or to explain any discrepancy in data	
5. Computer use	1	Possesses sufficient hands-on ability to work with Linux OS, GNU toolchain and other relevant software	Is able to work on Linux OS with GNU toolchain or other relevant software with some assistance	Cannot operate the Linux OS and other software without significant assistance	
6. Teamwork	3	Actively engages and cooperates with other group members in an effective manner	Cooperates with other group members in a reasonable manner	Distracts or discourages other group members from conducting the experiments	
7. Lab safety and disciplinary roles	3	Observes lab safety rules; and adheres to the lab disciplinary guidelines aptly	Observes safety rules and disciplinary guidelines with minor deviations	Disregards lab safety and disciplinary rules	
				Total (out of 35)	