CS 221 Fundamental of Operating Systems

Fall 2024/2025

PROJECT

Multi-threaded Shell

Project Overview:

The purpose of this project is to implement a basic shell that can execute simple commands using system calls. Students will gain experience with fundamental operating system concepts like process creation (`fork()`), process synchronization (`wait()`), and multithreading using `pthread`. The project is designed to be completed in three phases, with increasing complexity in each phase.

Objective:

1. Understand the basics of **process creation** and execution in an Operating System using `fork()`, `execvp()` and `wait()`.
2. Manage command history and develop **custom shell commands**.
3. Introduce **threading** with `pthread` and **synchronization** using mutexes.

Course Learning Outcome:

The project contributes to CLO2.1, 2.2 and CLO3.1.

Group Formation:

This is a group project of three students maximum. Group registration is to be done during the first lab on the project.

Successful Projects:

A successful project must achieve the following objectives:

1. A fully working program
2. Satisfy all requirements and

3. Well understood by all members.

Only such project deserves full mark.

Assessment:

The project contributes 10% to your final grade (including weekly project progress). The assessment is according to the following distribution:

1. CLO2.1 (2 marks) – A well written report and in depth understanding of the project.
2. CLO2.2 (5 marks) – Professionally developed and well understood multi-threaded program, which includes efficient code, testing for correctness for various different cases etc.
3. CLO3.1 (3 marks) – communication skills and team work will be assessed based on the weekly progress, quality of the report and the discussion.

Deliverables:

The deliverables are as follow:

1. Project report: You must include a screen capture images of the output along with a discussion on the results. You should also include the important part of your code with description. **Please include a table of group members and the tasks carried out by each**.
2. Source code: Good programming practices must be reflected in the code, such as proper naming of variable, use of comments and indentation.
3. Project demo and Question & Answer: The demonstration must involve all the group members. Failing to be present during the agreed demo time will result in an F for the whole project. Individual marks will be given based on the ability to answer the questions satisfactorily.
4. For phase1 and phase2, only source codes to be submitted.

Deadlines:

- Forming Teams: End of week 11, Saturday, 9th November 2024 by 11:59 pm. Send me a private chat informing me who is in the team.
- Phase1 Submission: End of week 13, Saturday, 23rd November 2024 by 11:59 pm (only C files)
- Phase2 Submission: End of week 14, Saturday, 30th November 2024 by 11:59 pm (only C files)
- Project submission: End of week 15, Saturday, 7th December 2024 by 11:59 pm. Please submit one zip file containing all your files.
- Presentation and demo: During your lab hours of week 16.
- Note: 1.5 marks will be deducted per day for late submission.

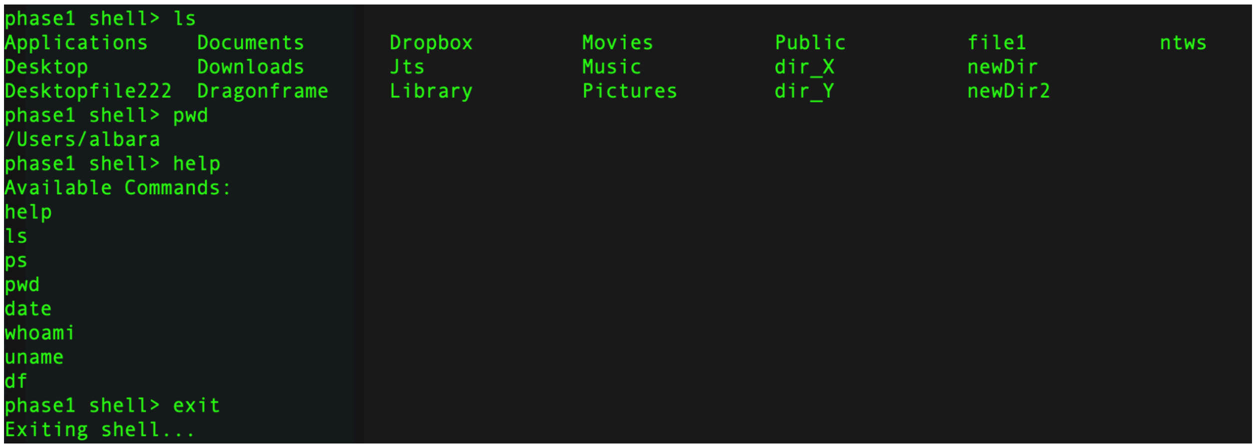
Phase 1: Basic Shell Implementation

Objective: In this phase, students will implement a simple shell that can execute basic commands like ls, pwd, and ps.

Requirements:

1. Create a shell that continuously displays a prompt and waits for user input.
2. Use `fork()` to create a child process for executing each command.
3. Use `execvp()` to run commands (ls, ps, pwd, whoami, etc.).
4. Parent process should use `wait()` to wait for the child to finish before allowing the next command.
5. Handle the `exit` command to terminate the shell.
6. Print error message if command is not defined in your program.

Example Output:



```
phase1 shell> ls
Applications    Documents       Dropbox         Movies          Public          file1           ntws
Desktop         Downloads       Jts             Music           dir_X           newDir
Desktopfile222  Dragonframe     Library         Pictures        dir_Y           newDir2
phase1 shell> pwd
/Users/albara
phase1 shell> help
Available Commands:
help
ls
ps
pwd
date
whoami
uname
df
phase1 shell> exit
Exiting shell...
```

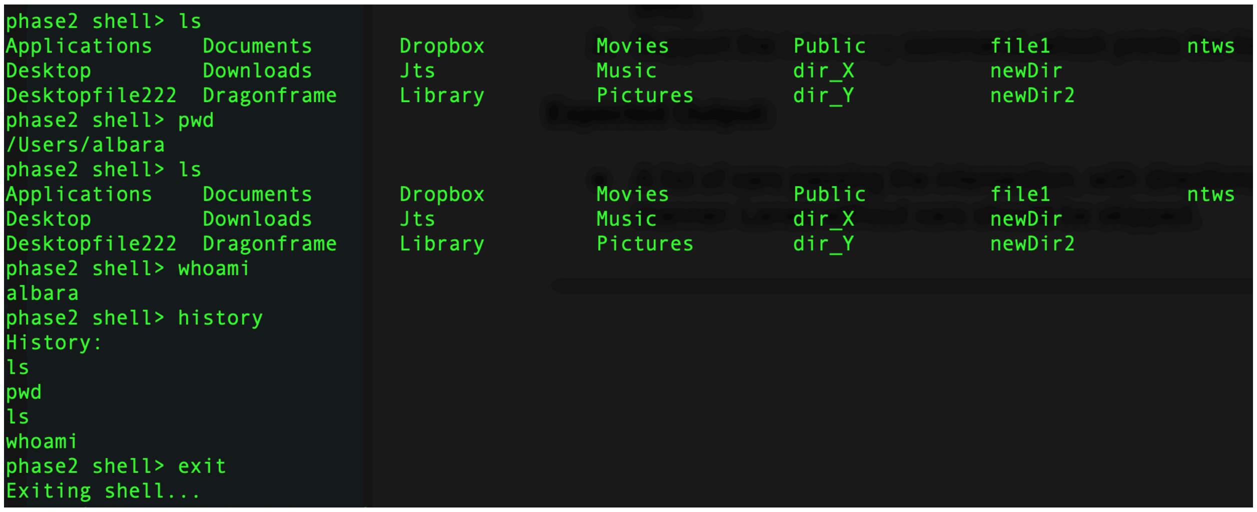
Phase 2: Adding Command History

Objective: Extend the shell from Phase 1 by implementing a **history** command that displays all previously entered commands.

Requirements:

1. Implement a history feature that saves user commands in an array (up to a predefined limit).
2. Support the `history` command, which prints the list of all commands entered so far.

Example Output:



```
phase2 shell> ls
Applications    Documents       Dropbox         Movies          Public          file1           ntws
Desktop         Downloads       Jts             Music           dir_X           newDir
Desktopfile222  Dragonframe     Library         Pictures        dir_Y           newDir2
phase2 shell> pwd
/Users/albara
phase2 shell> ls
Applications    Documents       Dropbox         Movies          Public          file1           ntws
Desktop         Downloads       Jts             Music           dir_X           newDir
Desktopfile222  Dragonframe     Library         Pictures        dir_Y           newDir2
phase2 shell> whoami
albara
phase2 shell> history
History:
ls
pwd
ls
whoami
phase2 shell> exit
Exiting shell...
```

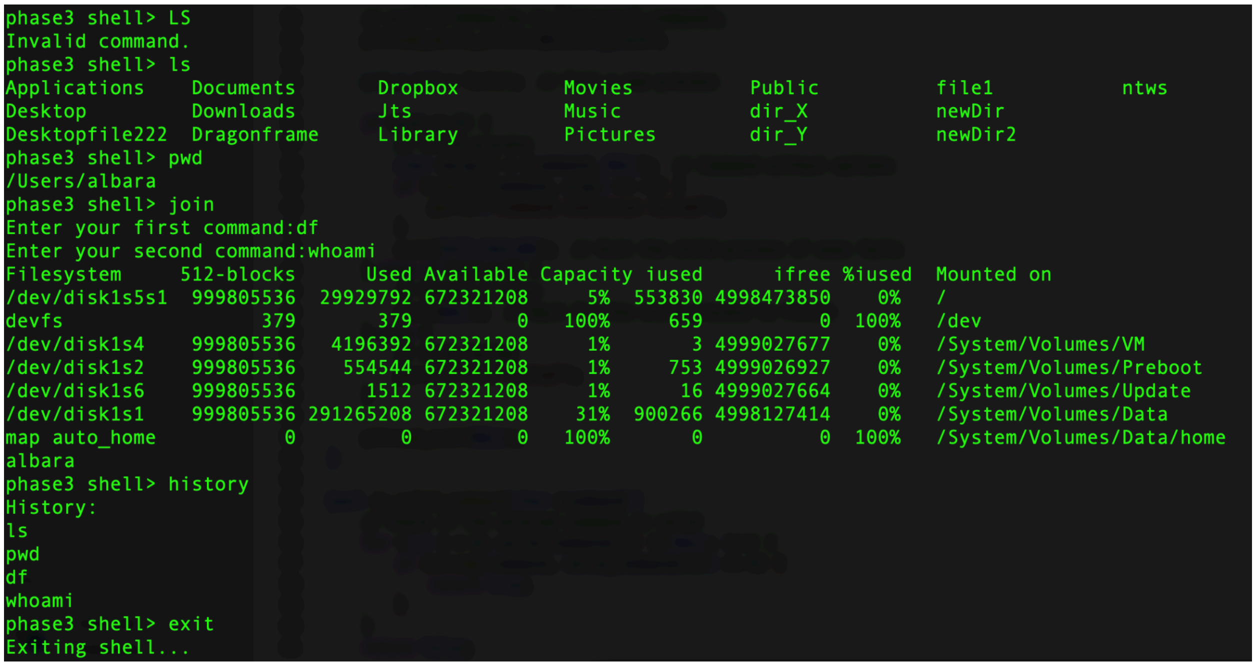
Phase 3: Implementing Threads and Synchronization (Final Submission)

Objective: Add multithreading to the shell, allowing certain commands to run in parallel, and ensure safe access to shared resources using mutexes.

Requirements:

1. Introduce a **`join`** command that takes two user commands and runs them concurrently in separate threads.
2. Use `pthread_create()` to create threads for running these commands.
3. **Synchronize** access to shared resources (such as command history) using a **mutex**.
4. Use `pthread_join()` to wait for both threads to finish execution before proceeding.

Example Output:



```
phase3 shell> LS
Invalid command.
phase3 shell> ls
Applications    Documents       Dropbox         Movies          Public          file1           ntws
Desktop         Downloads       Jts             Music           dir_X           newDir
Desktopfile222  Dragonframe     Library         Pictures        dir_Y           newDir2
phase3 shell> pwd
/Users/albara
phase3 shell> join
Enter your first command:df
Enter your second command:whoami
Filesystem     512-blocks      Used Available Capacity iused      ifree %iused  Mounted on
/dev/disk1s5s1  999805536  29929792 672321208     5%  553830 4998473850    0%   /
devfs                 379       379         0   100%     659          0  100%   /dev
/dev/disk1s4    999805536   4196392 672321208     1%       3 4999027677    0%   /System/Volumes/VM
/dev/disk1s2    999805536    554544 672321208     1%     753 4999026927    0%   /System/Volumes/Preboot
/dev/disk1s6    999805536      1512 672321208     1%      16 4999027664    0%   /System/Volumes/Update
/dev/disk1s1    999805536 291265208 672321208    31%  900266 4998127414    0%   /System/Volumes/Data
map auto_home           0         0         0   100%       0          0  100%   /System/Volumes/Data/home
albara
phase3 shell> history
History:
ls
pwd
df
whoami
phase3 shell> exit
Exiting shell...
```

Hints:

- **Reading a string from the user:**
  - Assume you have a string to store the command of `MAX_COMMAND_LENGTH` length:
    - `char command[MAX_COMMAND_LENGTH];`
  - You can use `fgets` function to read the string from the user:
    - `fgets(command, MAX_COMMAND_LENGTH, stdin);`
  - You should remove the new line character `"\n"` at the end of the string, and add the terminating character `'\0'`
    - `command[strcspn(command, "\n")] = '\0';`
- **Compare a string variable with a string:**
  - To check if the string variable equals some value, you can use the function `strcmp()`
    - the condition (`strcmp(s, "value") == 0`) is true if s = “value”.
- To create an array of strings, you can use array of char*
  - `char *string_array[] = {"value1","value2", "value3"];`