Problem 1 - Creating a Shell

Commands

We have created a shell and have implemented the following commands:

1. clr
    - Function: Clear the screen
    - Implementation:
2. pause
    - Function: Pause operations of the shell until ‘Enter’ is pressed.
    - Implementation: Initially all the keyboard interrupts are ignored using SIG_IGN and then only enter is identified using cin.get(), then SIG_DFL is implemented which takes the default action for the upcoming signals.
3. help
    - Function:Display User Manual which includes syntax and description of every valid command in myshell.
    - Implementation: Prints the User Manual using cout statement.
4. quit / Ctrl+D
    - Function: Terminate execution of the shell program
    - Implementation: The program is quit using exit(0), which is a jump statement to terminate execution of the current process with relevant exit codes.
5. history
    - Function: Display the list of previously executed commands, even on shell restart
    - Implementation: To facilitate persistent history, the program stores the command history in a file present in the directory of the executable. The history file path is stored as a local environment variable.
6. cd DIRECTORY
    - Function: Change the current default directory to DIRECTORY. If the DIRECTORY argument is
      not present, report the current directory. If the directory doesn’t exist, “Invalid

Directory” error is reported. This command changes the PWD environment variable for the current shell.

- Implementation: If there are no arguments, displayPWD function is used where the pwd environment variable is taken and the current directory is displayed. Error is reported in case of non existent directory or more arguments than required. In the general case, the environment variable, PWD, is set to the given directory(which is given as the argument).

7. dir DIRECTORY
    - Function: List all the contents of the directory DIRECTORY
    - Implementation: The absolute path of the specified directory is derived using realpath function.
      Dirent.h is used to iterate over the directory contents and print the file/folder names. In case of invalid user input, the error is caught and reported accordingly.
8. environ
    - Function: List all the environment strings of the current shell and the bash shell
    - Implementation: The local environment variables are stored in a map in the program’s memory. The map is simply iterated to list the variables.
      The environment strings of the bash shell are accessed using `extern char** environ`.
9. echo COMMENT
    - Function: Displays comment on the display followed by a new line. Multiple spaces/tabs
      are reduced to a single space.
    - Implementation: The given arguments are displayed using cout and finally followed by a new line.
10. Executable path
    - Function: The shell environment should contain shell==myshell where =myshell is full path for the shell executable
    - Implementation: Each C++ program execution supplies filename as argv[0] parameter. As it might be absolute or relative path, the absolute path is

obtained using realpath function. It is then stored as an environment variable.

11. Batchfile execution
    - Function: The shell must be able to take its command line input from a file. That is, if the command line is invoked with a command line argument: myshell then is assumed to contain a set of command lines for the shell to process
    - Implementation: If the shell is invoked with a commandline parameter, it is assumed to be batchfile. The batchfile is read line by line and executed either as a pre defined shell command or a program execution.
12. Bonus - Executable execution
    - Function: Assume that the full path to the executable is provided as an input, run the program
    - Implementation: If the command line input does not match with any of the pre-defined commands, it is assumed to be the path to an executable followed by arguments. Fork system call is used to create a child process to isolate executable execution. The program is run using the execvp system call, any errors are displayed in the console. In case the executable supplied does not exits, the prompt prints "invalid executable".

Problem 2 - Dining Students

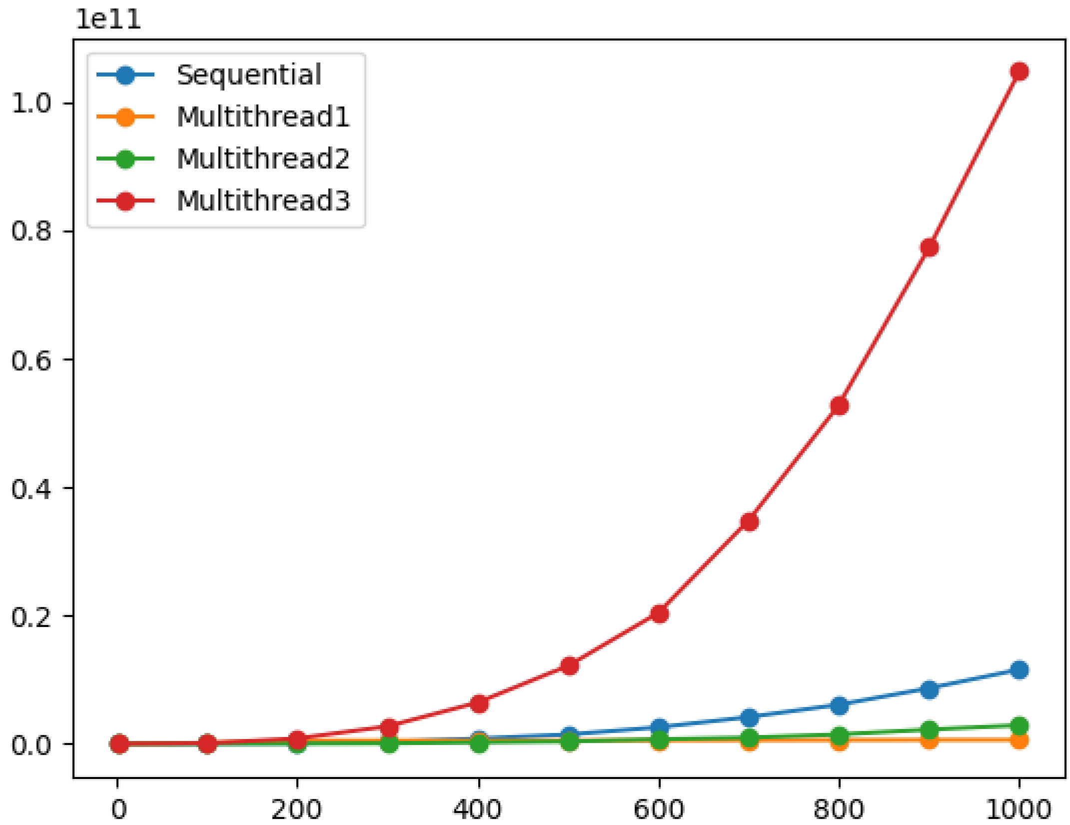
- For 5 students and 5 spoons case, each person has a left and right spoon and when simulating, we had to make sure that only person could use a given spoon at a time. To do this, we have used locks and condition variables and as per the results, 2 students can eat parallelly.
- We created 5 threads which have a conditional variable and wait on the canTakeSpoons function returning true. Further when sleeping the lock is given up so that other threads can run. It is locked again when operating on global variables like the student state.
- We chose to skip the "One Spoon Acquired" state because taking both spoons when they free is optimal than taking a spoon and waiting for other which can result in a deadlock when each student takes one spoon and waits for someone to give up a spoon.

- When a student is done eating and goes to thinking state, he gives up both spoons and goes to the waiting pool after the thinking period is over.
- When notifying other threads after completion, the scheduler schedules the thread that has waited the most and a particular student doesn't starve. We can confirm this from the state change log where against each student the cycles completed is printed and in most cases all students would have completed the same number of cycles.

Problem 3 – Matrix Multiplication

- We are comparing the sequential program against 3 variants of parallelised matrix multiplication algorithms.
- A `mythreads.h` library has been included, which is basically a wrapper to some standard `pthread.h` library functions. The wrappers check for exception safety.
- **Variant 1** -
  - We create n^2 threads, each one computing one of the output element of the final product matrix.
- **Variant 2** -
  - We create n threads, each one computing one row final product matrix.
- **Variant 3** -
  - We create 4 threads and use locks for computing the final product matrix.
  - Divide the first matrix into 4 quadrants. perform the sequential method with each quadrant with the 2nd matrix while holding a lock on the result of a partuicular element of the product matrix.
  - The lock is needed as the other thread is also performing an addition on a shared variable result.

- Plots



- Observations
    - The sequential algorithm is slower than parallel algorithms (`multithread2` and `multithread3`)
    - `Multithread1` uses n^2 threads and thus after a certain n , the program exceeds kernel thread limits, and throws errors for thread creation.
    - `Multithread1` uses n threads and thus it is the fastest.
    - `Multithread3` uses 4 threads and because there is a critical section too, locks are required and thus it is slower than `multithread2`.