CS307 Assignment-1 Report

Instructor: Dr. Aditya Nigam

Submitted By-

1. Ashutosh Sharma - B18010
2. Anuj Goel - B18161
3. Om Pandey - B18182

Q1. Creating a shell.

Commands to run the Program:

In the root directory type:

```
make
```

Then, if you have batch file type:

```
./shell <batch-file-name>
```

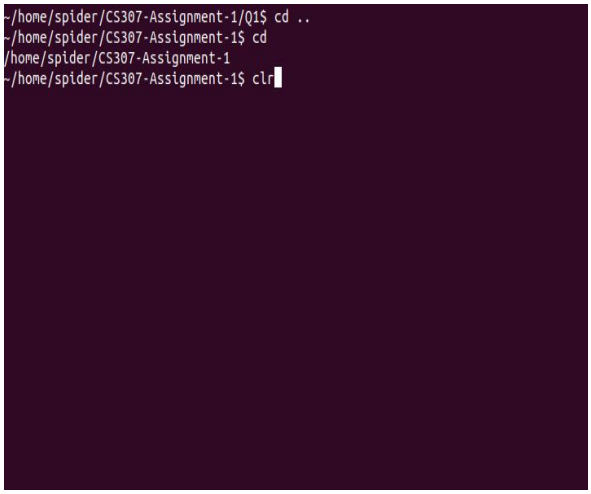
Otherwise, for user prompt shell type:

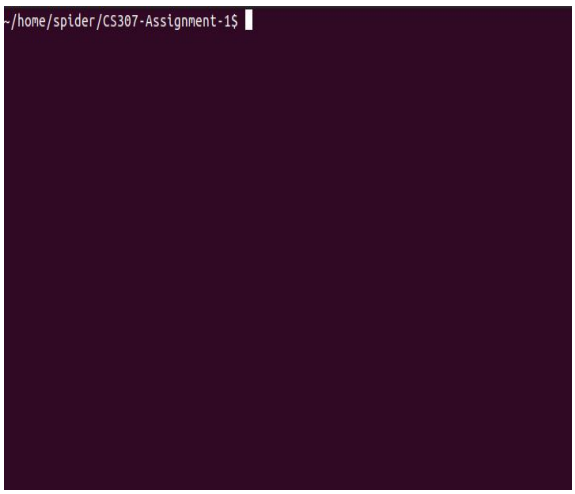
```
./shell
```

The shell supports following commands-

1. clr: To clear the screen.

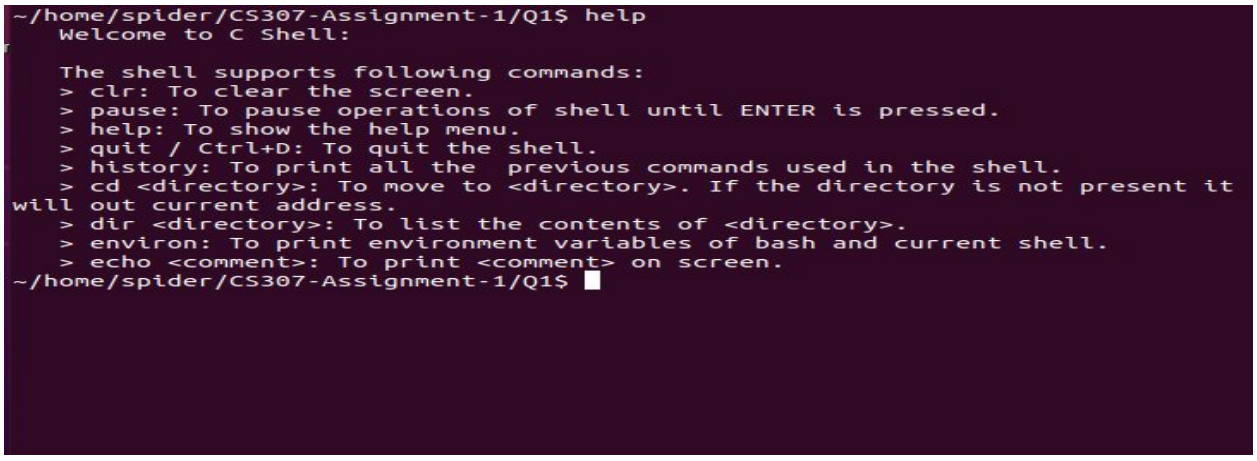
Before using and after using clr





2. pause: To pause operations of the shell until ENTER is pressed.
System will be paused until ENTER is pressed.

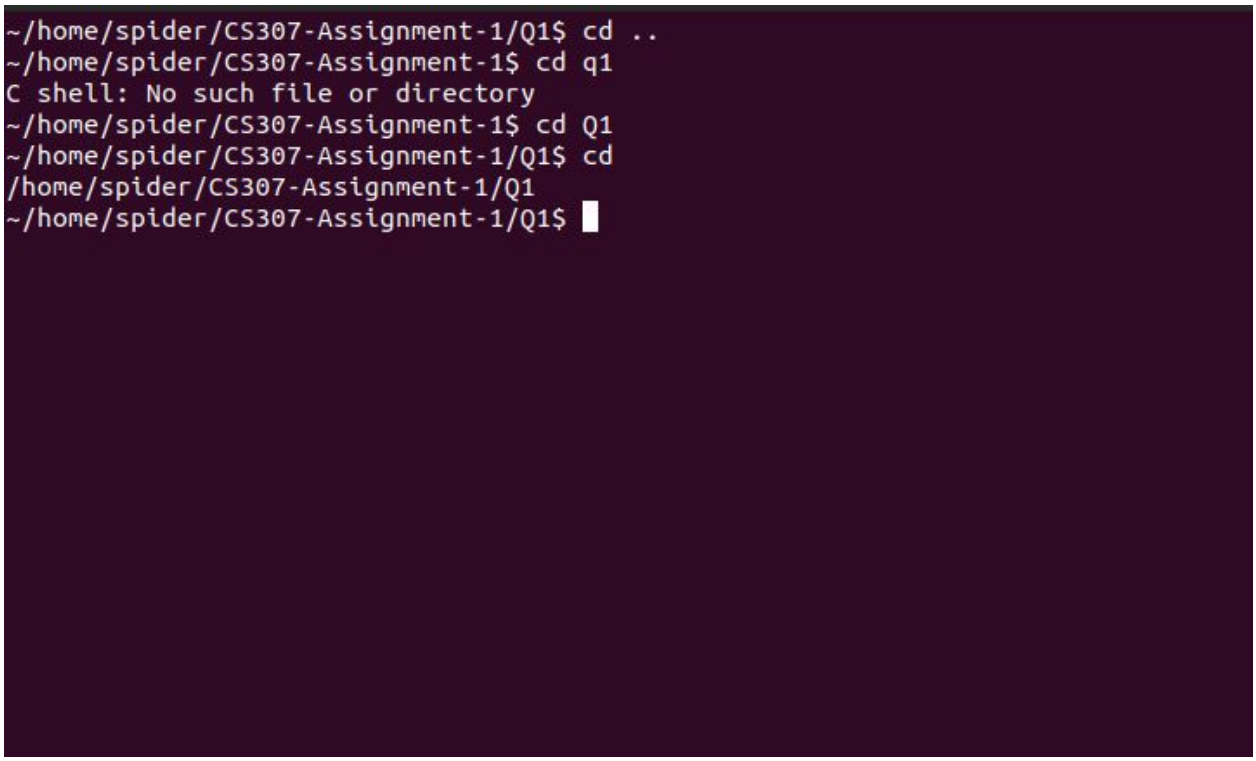
3. help: To show the help menu.



4.quit / Ctrl+D: To quit the shell.

5. history: To print all the previous commands used in the shell.
The history will be saved even after the shell is restarted because it is stored in an external history.txt file

6. cd <directory>: To move to <directory>. If the directory is not present it will out current address.
It will move to directory if it is present otherwise reports an error. Also, if only cd entered it tell current directory address and when moving to another directory it changes pwd environment variable of the current shell also.



7. dir <directory>: To list the contents of <directory>.
It will list all the contents of current directory.

8. environ: To print environment variables of bash and current shell.
It will list environment variables of bash shell as well as current shell.

```
~/home/spider/CS307-Assignment-1$ environ
Bash environment variables:

SHELL=/bin/bash
SESSION_MANAGER=local/spider-Lenovo-ideapad-330-15IKB:@/tmp/.ICE-unix/2361,unix/
spider-Lenovo-ideapad-330-15IKB:/tmp/.ICE-unix/2361
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
GTK_IM_MODULE=ibus
LANGUAGE=en_IN:en
QT4_IM_MODULE=ibus
MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path
GNOME_SHELL_SESSION_MODE=ubuntu
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
XMODIFIERS=@im=ibus
DESKTOP_SESSION=ubuntu
SSH_AGENT_PID=2254
GTK_MODULES=gail:atk-bridge
DBUS_STARTER_BUS_TYPE=session
PWD=/home/spider/CS307-Assignment-1/Q1
LOGNAME=spider
```

9. echo <comment>: To print <comment> on screen.
It will print the output which is entered after echo command.

```
~/home/spider/CS307-Assignment-1$ echo ashutosh
ashutosh
~/home/spider/CS307-Assignment-1$
```

10. Current shell Environment:

```
C shell environment variables

ENVIRONMENT: shell=/home/spider/CS307-Assignment-1/Q1/shell.c=myshell
PWD: /home/spider/CS307-Assignment-1
~/home/spider/CS307-Assignment-1$
```

11. Taking commands from batch file:

```
spider@spider:~/CS307-Assignment-1/Q1$ ./shell command.txt
   Welcome to C Shell:

   The shell supports following commands:
   > clr: To clear the screen.
   > pause: To pause operations of shell until ENTER is pressed.
   > help: To show the help menu.
   > quit / Ctrl+D: To quit the shell.
   > history: To print all the  previous commands used in the shell.
   > cd <directory>: To move to <directory>. If the directory is not present it
will out current address.
   > dir <directory>: To list the contents of <directory>.
   > environ: To print environment variables of bash and current shell.
   > echo <comment>: To print <comment> on screen.
C shell: No such file or directory
Bash environment variables:

SHELL=/bin/bash
SESSION_MANAGER=local/spider-Lenovo-ideapad-330-15IKB:@/tmp/.ICE-unix/2361,unix/
spider-Lenovo-ideapad-330-15IKB:/tmp/.ICE-unix/2361
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
XDG_MENU_PREFIX=gnome-
```

When content of command.txt files are:

```
1 help
2 cd test
3 environ
4 quit
```

References for Shell:

1. Sriram Sir notes and shell skeleton.
2. https://brennan.io/2015/01/16/write-a-shell-in-c/

Q2 Dining students

Commands to run the program:

Run the commands

In the root directory type:

```
make
```

```
./prog
```

This will generate a text file(log file),file_name, which contains the logs in the required format.

Findings:

After running code 5 times for a simulation time of 30 minutes each, it was found that:

- Initially each student was in waiting state (waiting for spoons)
- On an average, after 17 entries in the log, everyone finishes eating at least once
- The student who eats last is different on each run
- Deadlock was never encountered

```
--------------------------------------------
S0: Waiting for spoons.
S1: Both spoons acquired and eating.
S2: Waiting for spoons.
S3: Both spoons acquired and eating.
S4: Waiting for spoons.
--------------------------------------------
S0: One spoon acquired.
S1: Both spoons acquired and eating.
S2: Waiting for spoons.
S3: Both spoons acquired and eating.
S4: Waiting for spoons.
--------------------------------------------
S0: One spoon acquired.
S1: Both spoons acquired and eating.
S2: Waiting for spoons.
S3: Thinking.
S4: Waiting for spoons.
--------------------------------------------
S0: One spoon acquired.
S1: Both spoons acquired and eating.
S2: Waiting for spoons.
S3: Thinking.
S4: One spoon acquired.
--------------------------------------------
S0: Both spoons acquired and eating.
S1: Thinking.
S2: One spoon acquired.
S3: Thinking.
S4: One spoon acquired.
--------------------------------------------
```

Sample Log File snapshot

Analysis:

- Mutual Exclusion: Algorithm ensured mutual exclusion by using mutex utility of POSIX thread library. The array of spoons was declared as a mutex object and was locked before acquiring by the threads. This ensured that each spoon(resource) was not shared by more than one student(threads) at a particular time.
- Deadlock Prevention: A deadlock is only possible if all the students pick up one spoon and wait for the other. The algorithm prevents deadlock by imposing the condition that odd numbered students will pick spoon to their right first, followed by one on their left. On the other hand even numbered students picked spoon to their left first. This way, it is impossible for all students to pick up all the spoons together, thus preventing deadlock.
- Preventing starvation: starvation happens when one of the students is in waiting state for spoons for a long time. The algorithm discussed above also prevents starvation by ensuring no student is in waiting state for long as all students release lock on both the spoons after eating. Also, the locks are acquired on fcfs basis so the algorithm is unbiased. This is also evident from the fact that all students eat at least once in 17 entries.
- Fair Allocation: Mutex locks are fcfs so the student who releases the lock can not acquire it immediately. The logs also show that the order in which students eat differs every time the program runs, which shows that the allocation of resources is fair.

Q3 Matrix Multiplication using threading

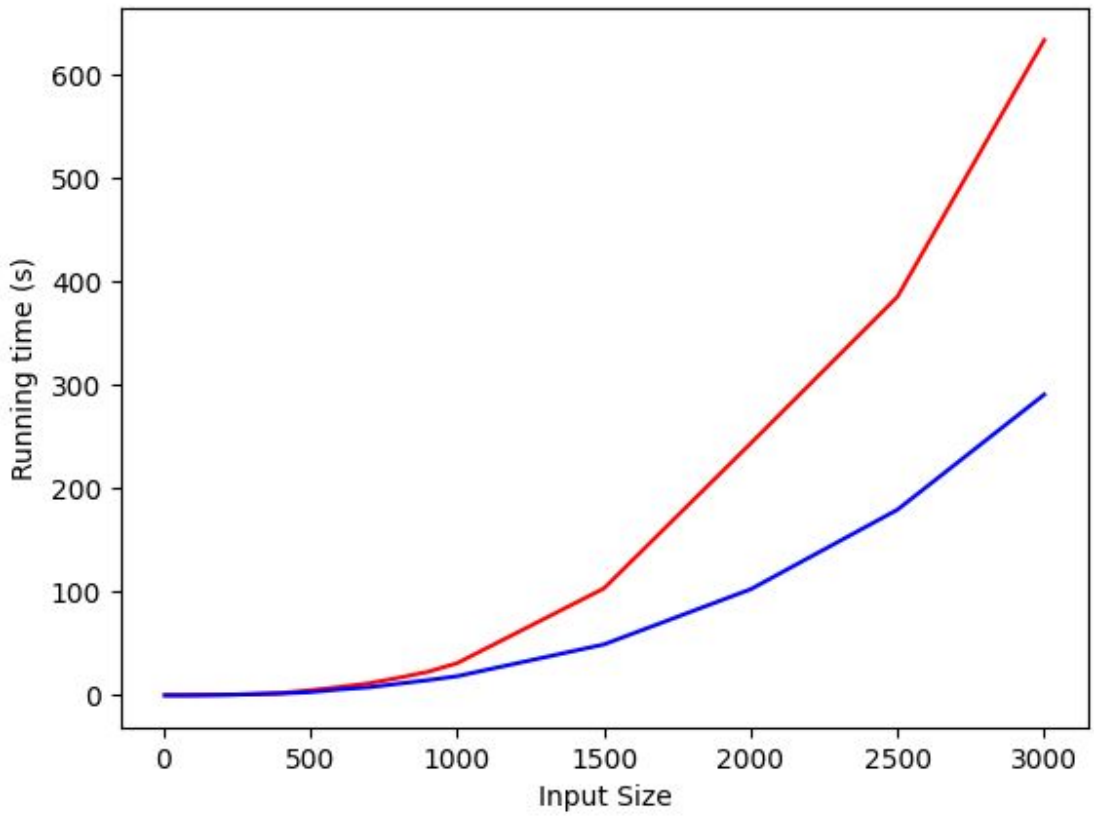
Commands to run the program:
To run the program, first run make followed by ./simp [n] for program without threading and ./base [n] for program with threading. Here n is the input size to be entered as a command line argument.
To measure time, use “time ./base [n]” or “time ./simp [n]”

```
time ./base 100
```

The output will be resultant matrix which is multiplication of randomly generated square matrix of size n x n. The time will also get displayed if the above command is used.
Sample Output
This was output for n = 10

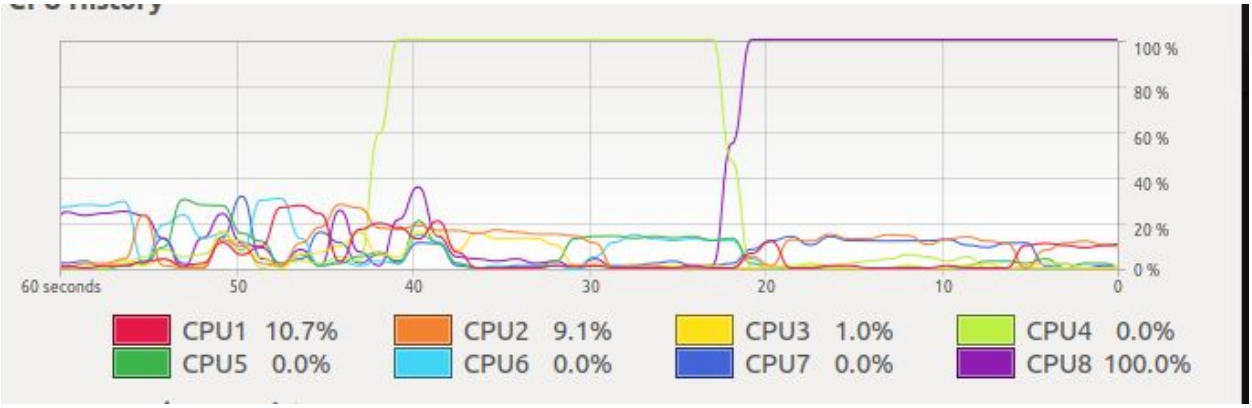
```
tom@Keyzer-Soze:~/Desktop/Threading$ time ./base 10
42150  43105  44060  45015  45970  46925  47880  48835  49790  50745
37650  38505  39360  40215  41070  41925  42780  43635  44490  45345
33150  33905  34660  35415  36170  36925  37680  38435  39190  39945
28650  29305  29960  30615  31270  31925  32580  33235  33890  34545
24150  24705  25260  25815  26370  26925  27480  28035  28590  29145
19650  20105  20560  21015  21470  21925  22380  22835  23290  23745
15150  15505  15860  16215  16570  16925  17280  17635  17990  18345
10650  10905  11160  11415  11670  11925  12180  12435  12690  12945
6150  6305  6460  6615  6770  6925  7080  7235  7390  7545
1650  1705  1760  1815  1870  1925  1980  2035  2090  2145

real    0m0.005s
user    0m0.001s
sys     0m0.005s
```

Results
Time was recorded for 21 values of n ranging from 5 to 3000. The resultant csv file and the python code used to generate the given curve is also attached with the code.

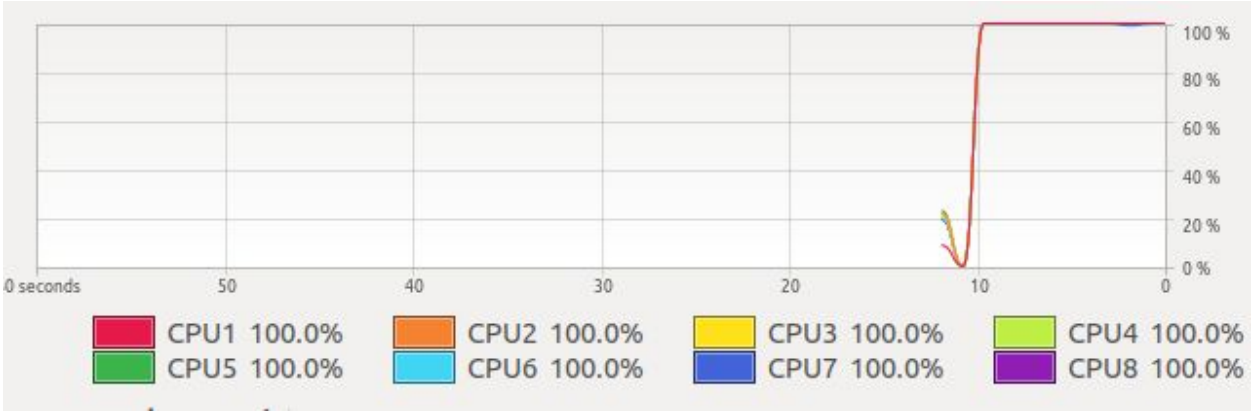


Inferences

From the given graph, it is evident that for values around 100-400, the advantage of using multithreading is not very noteworthy as the amount of computations is minimal. For values less than 80, it is even observed that the code without multithreading performs better as the overhead of making threads is more than the relaxation in computation. But as the value of n increases, the gap between the time taken by the programs starts widening as 8 CPUs (i have octa-core processor) divide work among themselves which was done by 1 CPU when multithreading was not used. The snippets from system monitor of ubuntu make the last point more clear.



This is when multithreading is not used. It can be seen that only 1 CPU is being used at 100% at a moment.



This is when multithreading is used. It can be seen that all the CPUs are working at 100%, which reduces the time of computation.