Design Document: Mini Project One

Courtney Gosselin - 60186160
Benjamin Tisserand - 37615168
Alex Qin - 35732156

COSC 315 - Operating System
Dr. Apurva Narayan
February 5, 2020

Table of Contents

1.0 Introduction: **2**

2.0 Prioritization/Planning: **2**

3.0 Contributions: **3**

4.0 Code Outline: **3**

5.0 Code structure/Folder layout: **4**

6.0 System Outline Diagrams: **5**
- 6.1 Parent process diagram 5
- 6.2 Child Diagram 6

7.0 Code output examples: **7**
- 7.1 Output for the parallel: 7
- 7.2 Output for the parallel on time out: 7
- 7.3 Output for the sequential: 8
- 7.4 Output for the sequential on time out: 8

1.0 Introduction:

The C program ‘closh’ works by taking the inputs program, count, run type (parallel or sequential), and timeout. For the program input, we created a test program ‘HelloWorld’, which can be used as the argument with the syntax “./HelloWorld”. However, hostname, pwd, echo, etc are all options that could be used as the program name being inputted as well. The next input - count - decides how many of the named processes are going to be created, accepting an integer from 1 to 9. The following input asks the user for a character, ‘s’ or ‘p’, to choose if the created processes are to run in sequence or in parallel respectively. Finally, it asks if you would like to define a timeout value. The inputted integer 0-9 denotes the seconds after which the processes will time out and be terminated; an inputted value of 0 results in no timeout. The whole program is to take in a process and run it for a specific count either sequentially or concurrently in parallel and to do this for a specific time 1-9 seconds or until it is finished.

2.0 Prioritization/Planning:

The following is a breakdown of the project into a list of measurable and timed checkpoints

- Product funtionality:
    - Use fork and exec (Jan,24th)
    - Make the parallel and sequential execution differently (Jan,28th)
    - Process timeouts (Jan,31st)
    - Manage code structures and put comments on (Feb,1st)
    - Design document (Feb,3rd)
    - README.md (Feb,4th)

These tasks were outlined in a scrum methodology style using the github projects section: https://github.com/COSC315-Group/MiniProject/projects/1

3.0 Contributions:

Benjamin:

- Coding and Logic Design
- Planning
- Documentation (Design document)
- Document Editing

Courtney:

- Base code
- Scrum master
- Documentation (Design document, README)
- Planning

Alex:

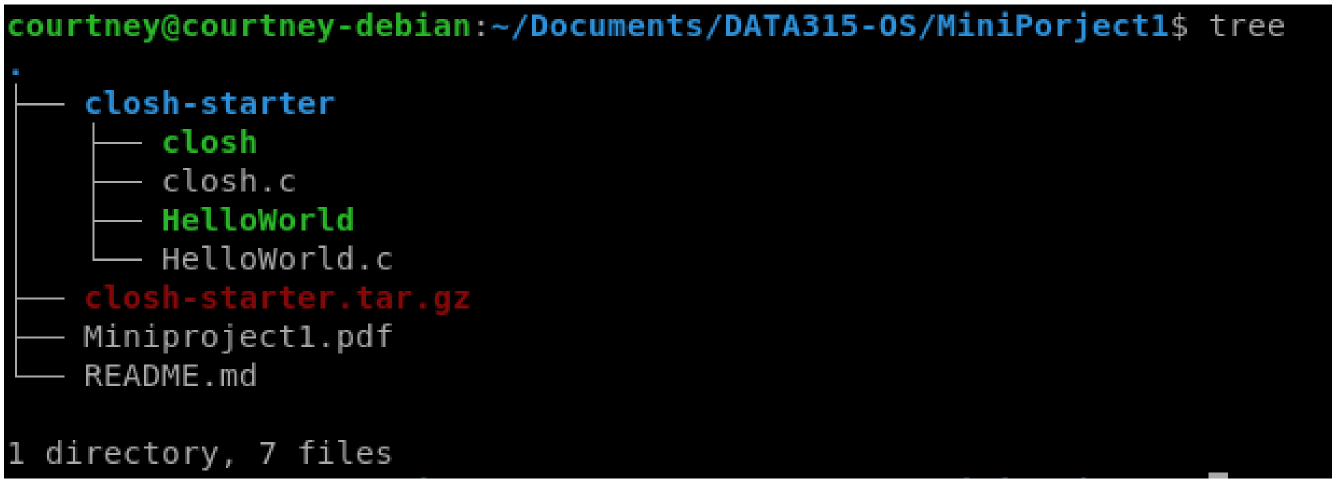
- Planning
- Feedback and Testing
- Documentation (Design document)

4.0 Code Outline:

In the “closh.c” file: after the starter code runs, we initialize time and child variables by using the types time_t and pid_t. Next, we loop through the fork command based on the number entered in count. Inside the loop, we first set up a “if statement” to check to see if the process is a child. If so, we use “break” to leave the loop. Inside that “if statement” is a second “if statement”: If we are in parallel mode with a timeout value, initiate the “parallelTimeoutAlarm” method. The second “if statement” ends here. (“parallelTimeoutAlarm” method is what we create to timeout processes that are running in parallel. Use “if statement” to check if it is parent, if so monitor the timeout until the process is done or times out) Then print the process id by using the function getpid() and break the loop. The first “if statement” ends here. After that, set an “else if” statement to check If running in sequential mode, use a while loop that monitors childs status (First set the basetime to now, then use “WNOHANG” which prevents the waitpid function from hanging on this while condition ensure our while loop runs while waiting for the child. Inside the while loop we use “if statement”: if we have exceeded the basetime by the time out, kill the child then print the specific process ID which has time out and been terminated. After that break the while loop.) Then the "else if" statement ends here. Once the big for loop has finished we only run the program on the children by using "if statement" to check childID, then use "execvp" to replace the current process with the given program.

To test our program, we created and compiled another program 'HelloWorld.c'. This test program is very barebones; it calls sleep(3), then outputs "Hello World!". This is to ensure we get output from our closh, as well as properly testing parallel vs sequential and timeouts using the sleep delay.

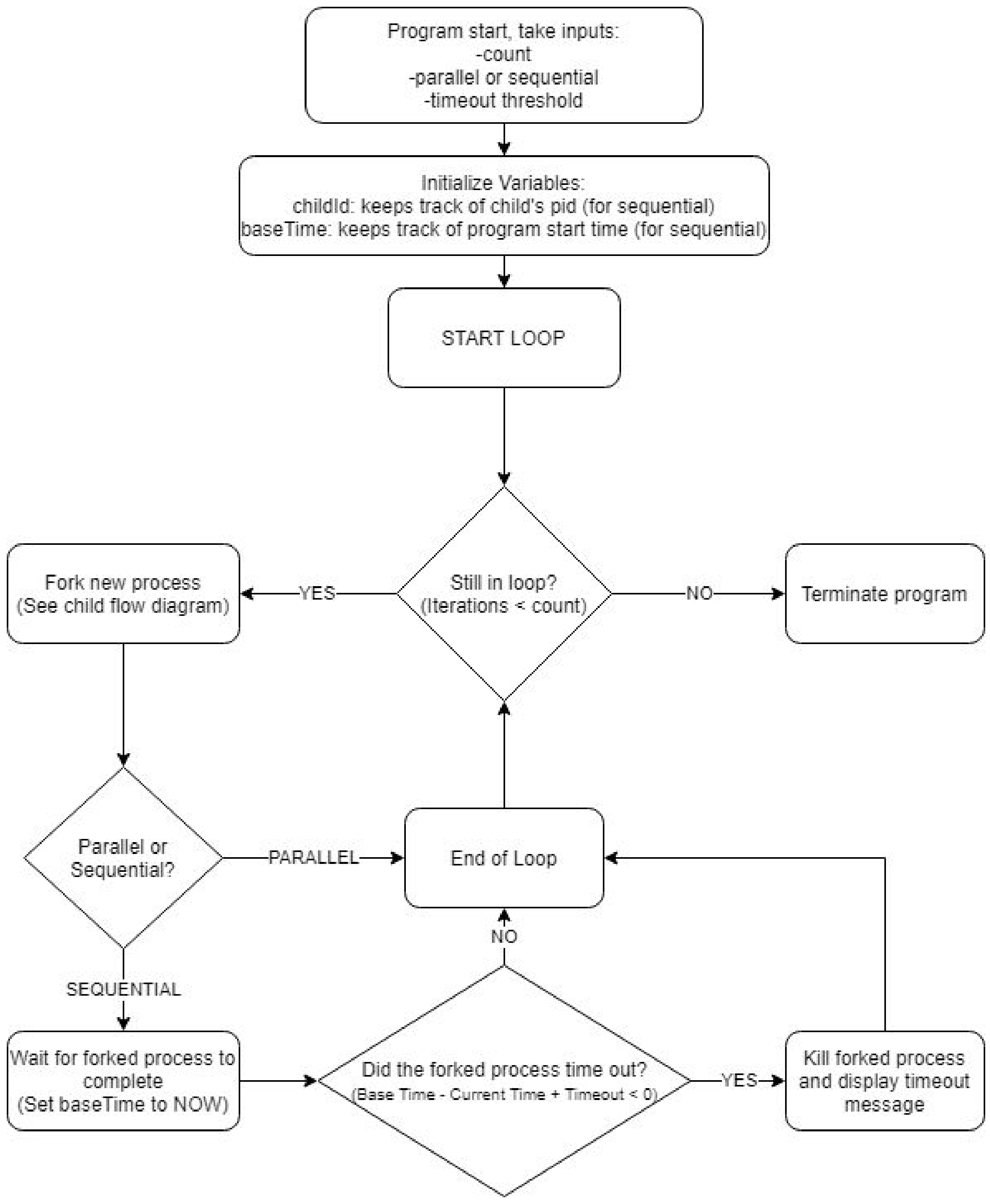
5.0 Code structure/Folder layout:



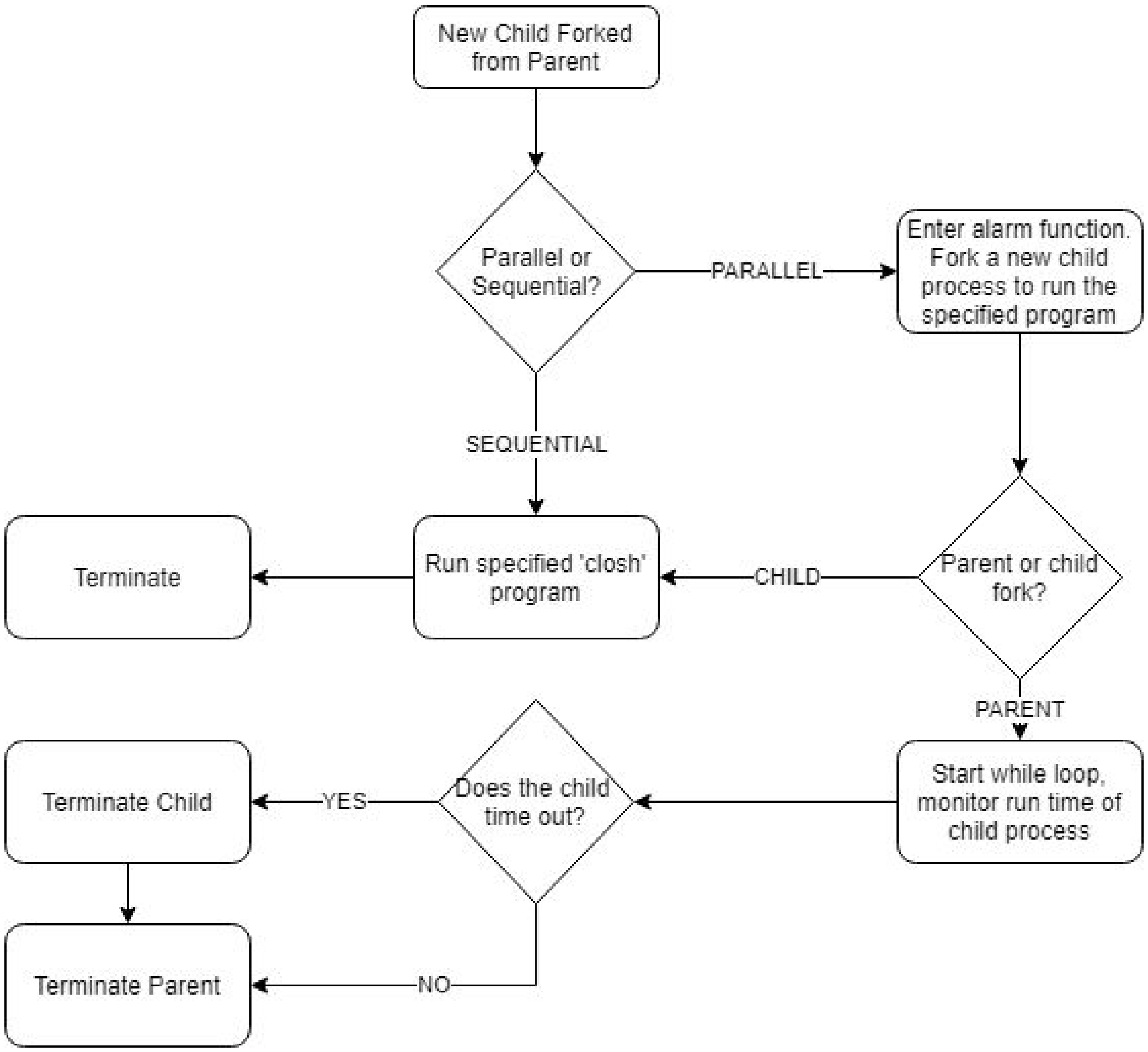
```
courtney@courtney-debian:~/Documents/DATA315-OS/MiniPorject1$ tree
.
├── closh-starter
│   ├── closh
│   ├── closh.c
│   ├── HelloWorld
│   └── HelloWorld.c
├── closh-starter.tar.gz
├── Miniproject1.pdf
└── README.md

1 directory, 7 files
```

6.0 System Outline Diagrams:

6.1 Parent process diagram


Program start, take inputs:
-count
-parallel or sequential
-timeout threshold
Initialize Variables:
childId: keeps track of child's pid (for sequential)
baseTime: keeps track of program start time (for sequential)
START LOOP
Still in loop? (Iterations < count)
YES
NO
Fork new process (See child flow diagram)
Terminate program
Parallel or Sequential?
PARALLEL
End of Loop
SEQUENTIAL
NO
Wait for forked process to complete (Set baseTime to NOW)
Did the forked process time out? (Base Time - Current Time + Timeout < 0)
YES
Kill forked process and display timeout message

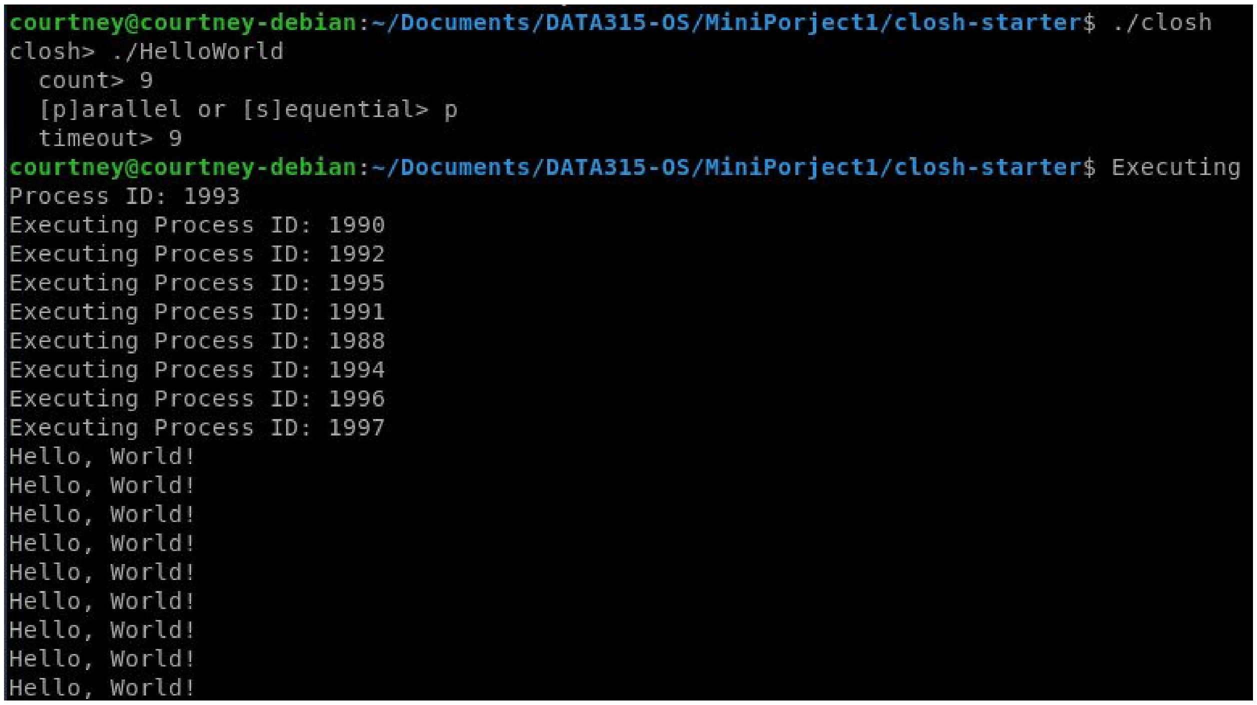
6.2 Child Diagram


New Child Forked from Parent
Parallel or Sequential?
PARALLEL
Enter alarm function. Fork a new child process to run the specified program
SEQUENTIAL
Terminate
Run specified 'closh' program
CHILD
Parent or child fork?
PARENT
Terminate Child
YES
Does the child time out?
Start while loop, monitor run time of child process
Terminate Parent
NO

We can see from these flow diagrams that parallel and sequential executions are handled very differently in this program, especially when it comes to tracking the runtime against the timeout value. In sequential execution, the root parent of closh is the process that tracks timeouts in its children, since it only needs to watch one executing child at a time. In parallel, we could not find a simple solution to allow the parent to watch all children separately, so we decided that each created process in parallel should run inside the loop in ‘parallelTimeoutAlarm’. This function creates yet another fork for each child that returns from this function and continues execution. The parent left inside this function is the process that watches for timeouts.

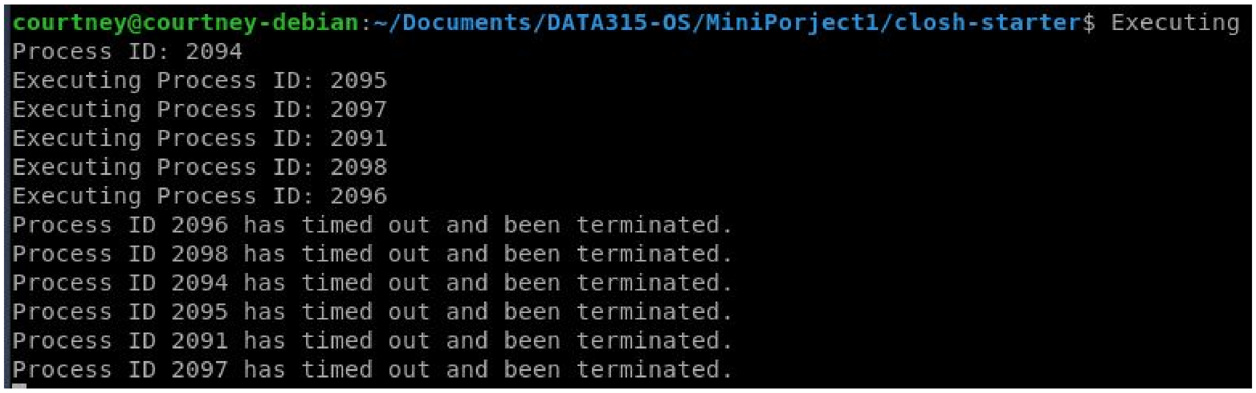
7.0 Code output examples:

7.1 Output for the parallel:



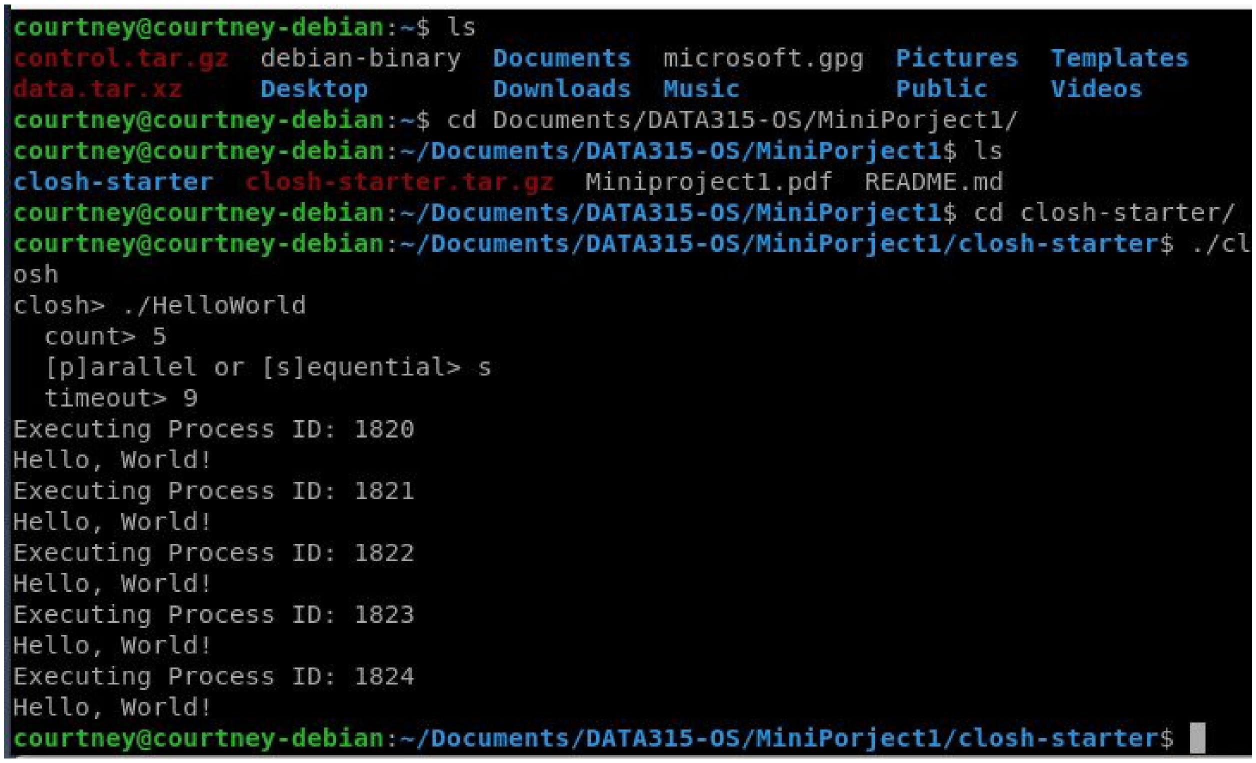
```
courtney@courtney-debian:~/Documents/DATA315-OS/MiniPorject1/closh-starter$ ./closh
closh> ./HelloWorld
  count> 9
  [p]arallel or [s]equential> p
  timeout> 9
courtney@courtney-debian:~/Documents/DATA315-OS/MiniPorject1/closh-starter$ Executing
Process ID: 1993
Executing Process ID: 1990
Executing Process ID: 1992
Executing Process ID: 1995
Executing Process ID: 1991
Executing Process ID: 1988
Executing Process ID: 1994
Executing Process ID: 1996
Executing Process ID: 1997
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
```

7.2 Output for the parallel on time out:



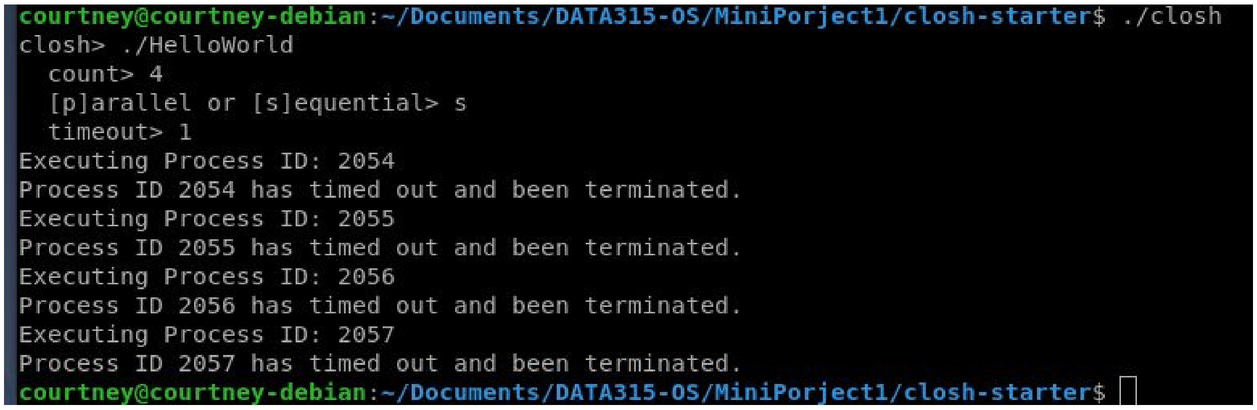
```
courtney@courtney-debian:~/Documents/DATA315-OS/MiniPorject1/closh-starter$ Executing
Process ID: 2094
Executing Process ID: 2095
Executing Process ID: 2097
Executing Process ID: 2091
Executing Process ID: 2098
Executing Process ID: 2096
Process ID 2096 has timed out and been terminated.
Process ID 2098 has timed out and been terminated.
Process ID 2094 has timed out and been terminated.
Process ID 2095 has timed out and been terminated.
Process ID 2091 has timed out and been terminated.
Process ID 2097 has timed out and been terminated.
```

Output for the sequential:



```
courtney@courtney-debian:~$ ls
control.tar.gz  debian-binary  Documents  microsoft.gpg  Pictures  Templates
data.tar.xz     Desktop        Downloads  Music          Public    Videos
courtney@courtney-debian:~$ cd Documents/DATA315-OS/MiniPorject1/
courtney@courtney-debian:~/Documents/DATA315-OS/MiniPorject1$ ls
closh-starter  closh-starter.tar.gz  Miniproject1.pdf  README.md
courtney@courtney-debian:~/Documents/DATA315-OS/MiniPorject1$ cd closh-starter/
courtney@courtney-debian:~/Documents/DATA315-OS/MiniPorject1/closh-starter$ ./cl
osh
closh> ./HelloWorld
  count> 5
  [p]arallel or [s]equential> s
  timeout> 9
Executing Process ID: 1820
Hello, World!
Executing Process ID: 1821
Hello, World!
Executing Process ID: 1822
Hello, World!
Executing Process ID: 1823
Hello, World!
Executing Process ID: 1824
Hello, World!
courtney@courtney-debian:~/Documents/DATA315-OS/MiniPorject1/closh-starter$ 
```

Output for the sequential on time out:



```
courtney@courtney-debian:~/Documents/DATA315-OS/MiniPorject1/closh-starter$ ./closh
closh> ./HelloWorld
  count> 4
  [p]arallel or [s]equential> s
  timeout> 1
Executing Process ID: 2054
Process ID 2054 has timed out and been terminated.
Executing Process ID: 2055
Process ID 2055 has timed out and been terminated.
Executing Process ID: 2056
Process ID 2056 has timed out and been terminated.
Executing Process ID: 2057
Process ID 2057 has timed out and been terminated.
courtney@courtney-debian:~/Documents/DATA315-OS/MiniPorject1/closh-starter$ 
```