**Name**: Luv Ahuja

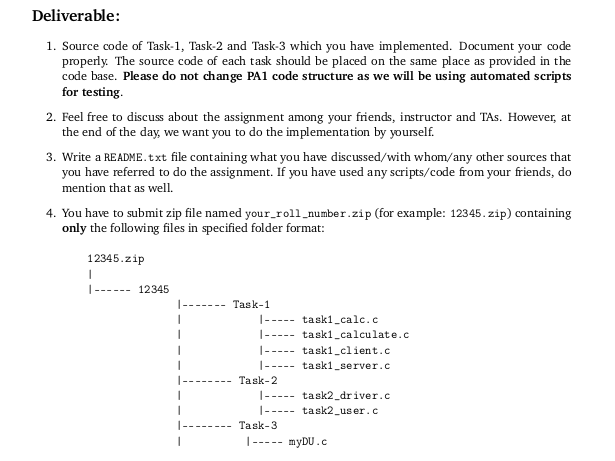
**Subject**: CS962 - Operating System Principle

**Assignment No**: 1

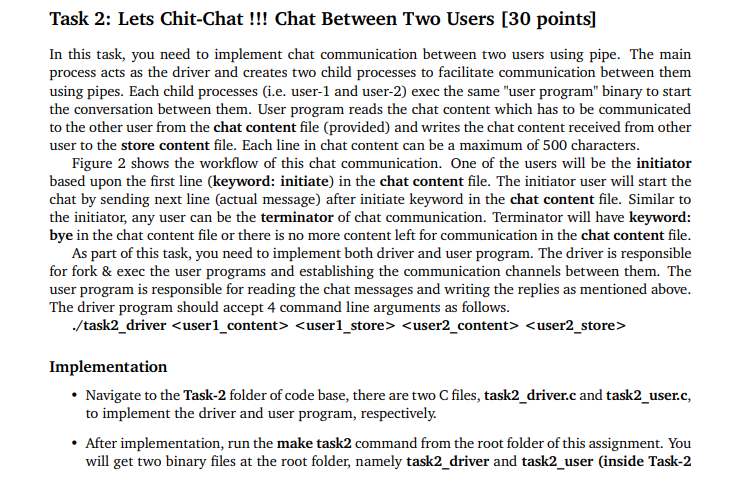
**Roll Number** : 23157028

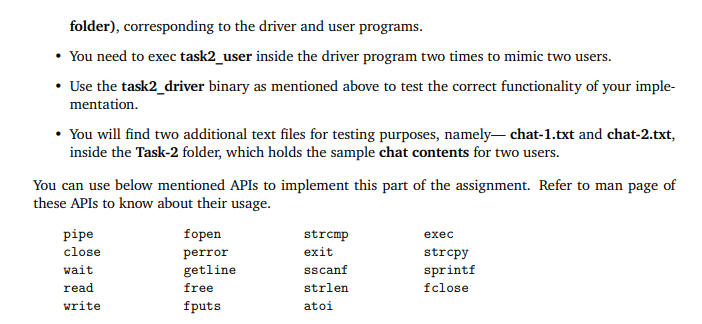
**Email ID**: [luvahuja23@iitk.ac.in](mailto:luvahuja23@iitk.ac.in)

**Deliverables**



**Task-2**





**Code files**

** **

**Resources**

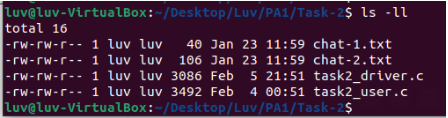
Usage of the exec1(), fork(), pipe() system calls after understanding from below stack overflow

1. [https://stackoverflow.com/questions/51636128/how-to-use-exec-system-call-to-return-the-square-of-a-number-and-store-it-to-a](https://stackoverflow.com/questions/51636128/how-to-use-exec-system-call-to-return-the-square-of-a-number-and-store-it-to-a%20)
2. <https://stackoverflow.com/questions/16203196/in-linux-calling-system-from-a-forked-process-with-pipe?rq=2>

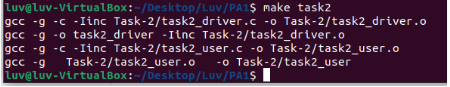
**Implementation steps & Test-case Results**

Steps by Steps of implementation of the **task2\_user.c** file and **task2\_driver.c** within code base structure execute and test it using test\_task2.sh script

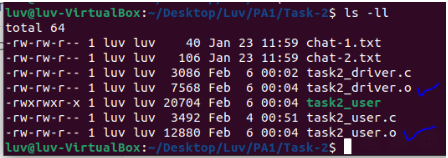
1. Place the attached files **task2\_user.c** and **task2\_driver.c** into PA/Task-2 folder



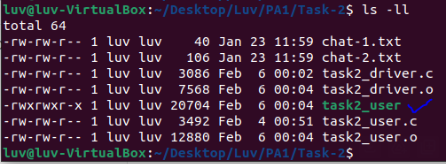
1. Run the command from **PA/** folder "**make task2**" - this will compile the code (if you make any changes in the files then you have to run make command once again).



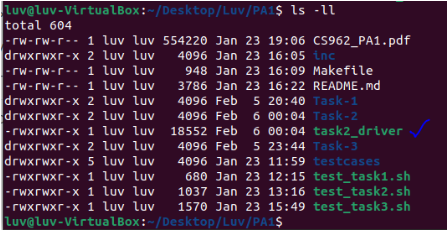
1. After step 2, below are behaviours are observed after compiling both files
2. This will create **task2\_user.o** and **task2\_driver.o** within **PA/Task-2/** folder.



1. This will create compiled file: **task2\_user** within **PA/TASK-2/** folder.



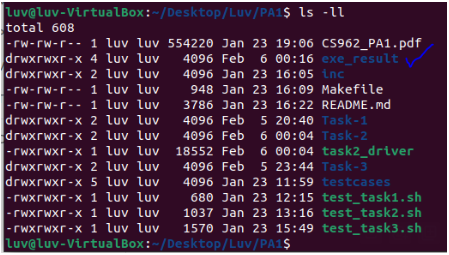
1. This will create compiled file: **task2\_driver** within **PA/** folder



1. Now, from **/Luv/PA1/** folder, we will run/execute the test-case: **test\_task2.sh** script which has been provided Prof/TA for checking if code file are running against the test case for TASK-2 using command "**sh test\_task2.sh**"



1. After running test-case script for task2 validation in step 4, there will be **exe\_result** folder gets created within **Luv/PA1** folder



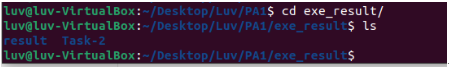
1. Folder **exe\_result** created in step 4, has 2 folders further - **result** and **Task-2**. Same has been attached for evaluation purpose.

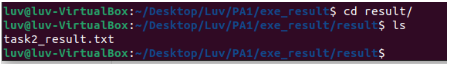


1. Task-2 Folder shows 3 folders are created - **t1, t2, t3** - each has 2 files - store-1.txt and store2.txt generated when we run the script **'test\_task2.sh'** taking **task2\_driver** compile file.

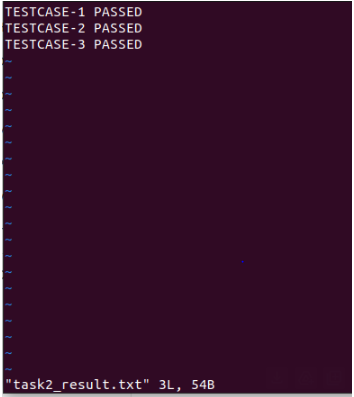


1. **result** folder has **task2\_result.txt** file which shows the status of the test case run after validating the files above generated (**store1.txt and store2.txt**).

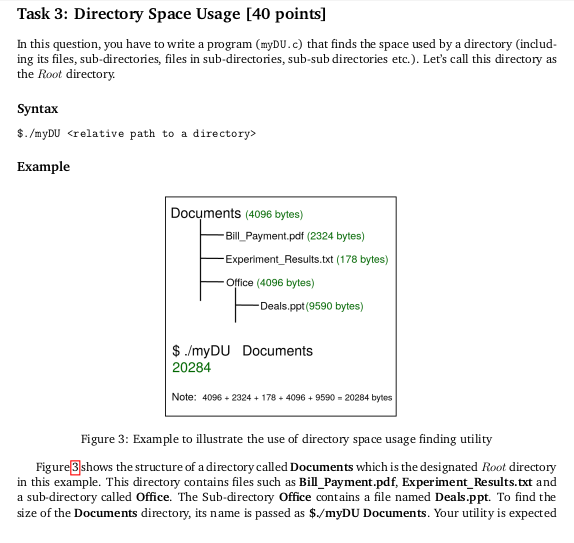


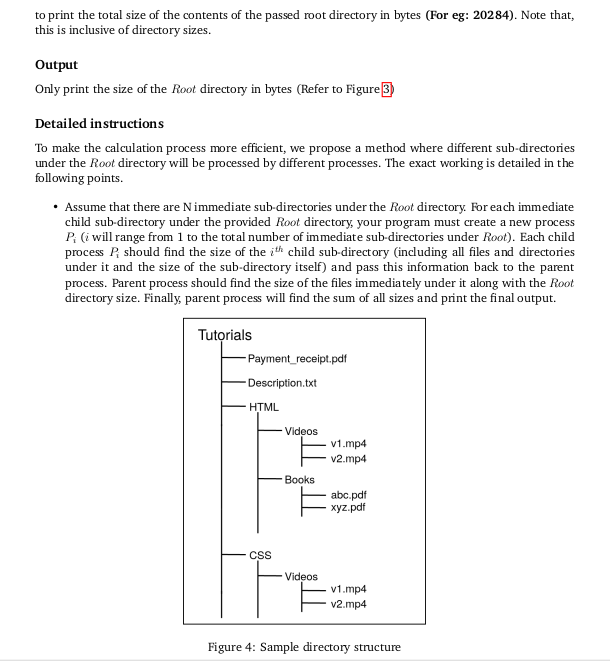


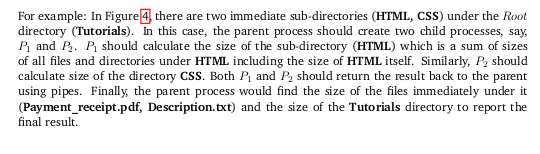


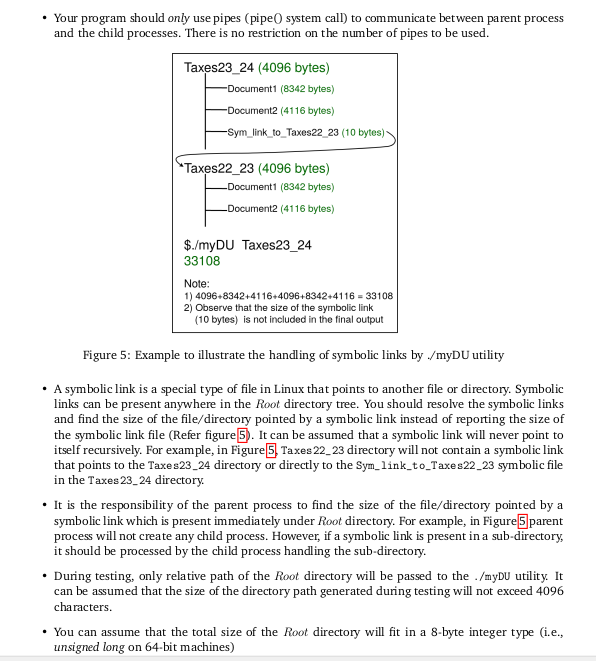


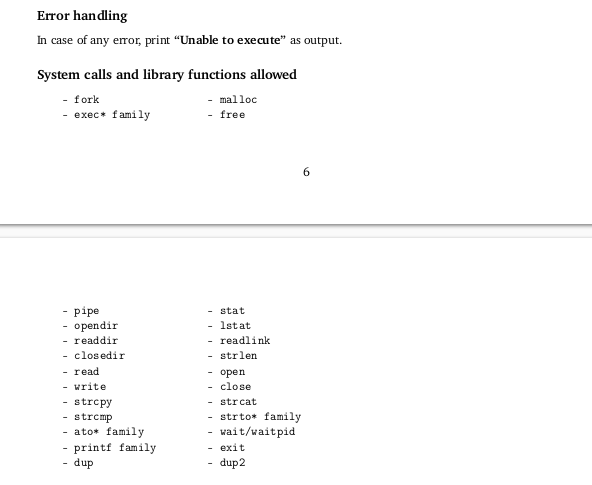
**Task-3**











**Code files**

****

**Resources**

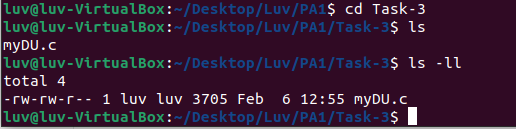
1. Calculating the sum of sizes of files and directories within parent directory and file handling logic based on file type has been introduced.
2. Introduced the logic of handing symbolic links path (directory path) with target path and further reading it using readlink ()
3. For each directory or symbolic link, it create a pipe, forks a child process for calculating the size of the files in the directory and read the size through pipe into parent process.

* https://stackoverflow.com/questions/52737688/cannot-get-the-size-of-a-file-from-its-directory-listing?rq=3
* <https://stackoverflow.com/questions/3984948/how-to-figure-out-if-a-file-is-a-link>
* <https://stackoverflow.com/questions/15465436/git-how-to-handle-symlinks>
* <https://stackoverflow.com/questions/39719479/unable-to-use-readlink-for-proc-self-exe-for-a-c-program>
* https://stackoverflow.com/questions/16343122/implement-a-pipe-in-c

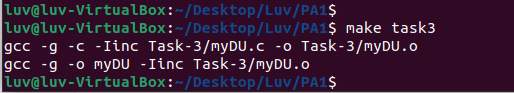
**Implementation steps & Test-case Results**

Steps by Steps of implementation of the **myDU.c** file within code base structure execute and test it using test\_task3.sh script

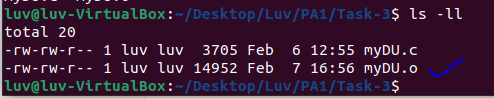
1. Place the attached files **myDU.c** into PA/Task-3 folder



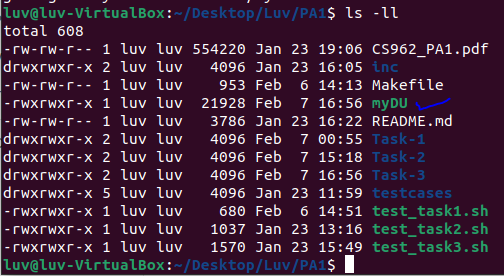
1. Run the command from **PA/** folder "**make task3**" - this will compile the code (if you make any changes in the files then you have to run make command once again).



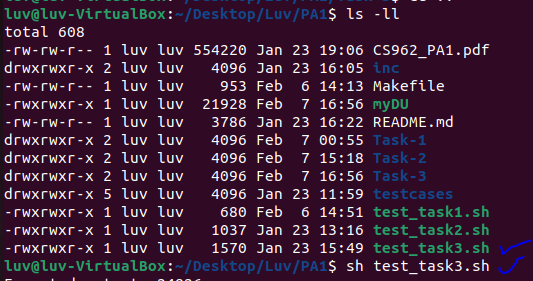
1. After step 2, below are behaviours are observed after compiling both files
2. This will create **myDU.o** within **PA/Task-2/** folder.



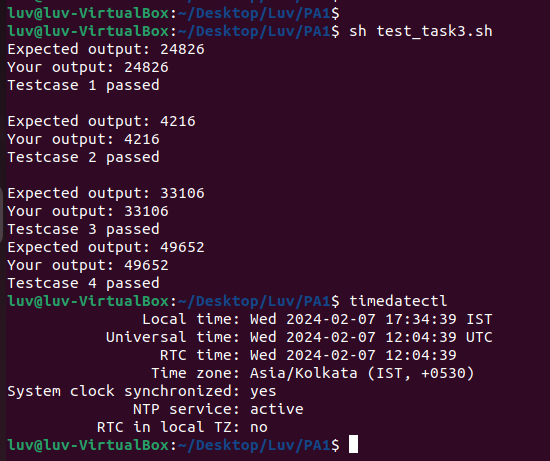
1. This will create compiled file: **myDU** within **PA/** folder



1. Now, from **/Luv/PA1/** folder, we will run/execute the test-case: **test\_task3.sh** script which has been provided Prof/TA for checking if code file are running against the test case for TASK-3 using command "**sh test\_task3.sh**"

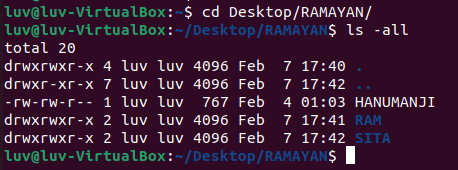


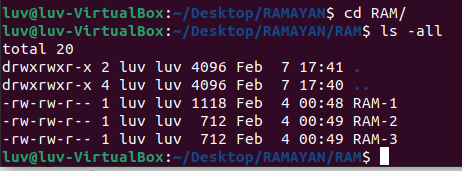
1. After running test-case script for task3 validation in step 4, there will be expected output and your output total size results of test-case within TASK-3 folder.

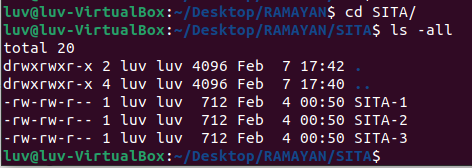


1. For Task-3, I have built my own additional test-case which reside on path **Desktop/RAMAYAN/** - **Parent Folder structure be like below:**

* **RAMAYAN** (parent folder)
  + **RAM** (immediate sub-folder/directory within parent)
* **RAM-1** (text file within sub-folder/directory)
* **RAM-2** (text file within sub-folder/directory)
* **RAM-3** (text file within sub-folder/directory)
  + **SITA** (immediate sub-folder/directory within parent)
* **SITA-1** (text file within sub-folder/directory)
* **SITA-2** (text file within sub-folder/directory)
* **SITA-3** (text file within sub-folder/directory)
  + **HANUMANJI** (immediate file within parent)

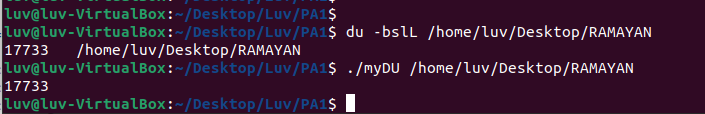




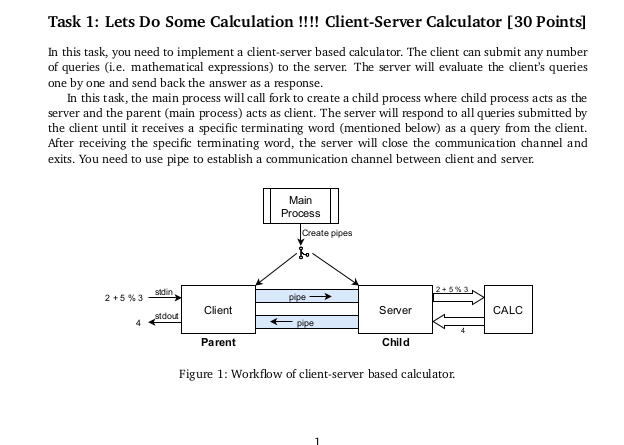


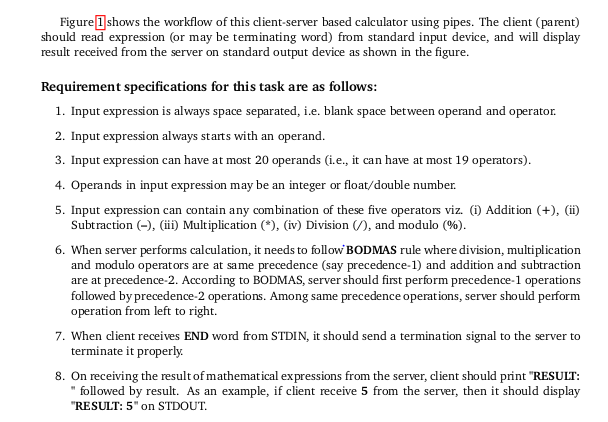
**Test-case Results**

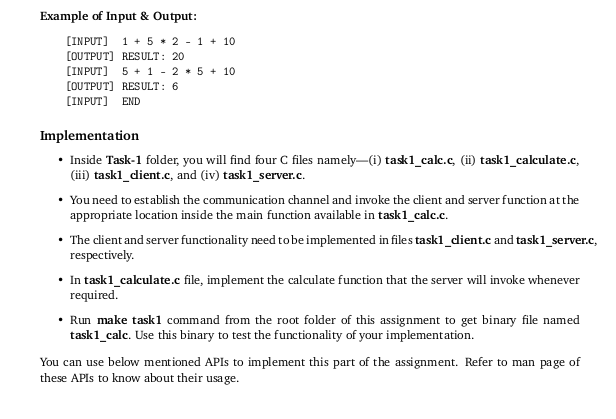
When run the command “**du –bslL**” taken up from the test\_task3.sh script on path - /home/luv/Desktop/RAMAYAN to calculate the total size, it comes out to be 17733 and when run command **./myDU** script on the same directory it comes out to be same (i.e 17733). PFB



**Task-1**







**Code files**

****

**Methodology & Resources/Discussion**

**Client.c**

1. Client function use 2 file descriptor (**readfd**, **writefd**) which helps in storing user input value within **inputBuffer** and also declares **result\_buf** to store response from server.
2. User input read by client function from **stdin** using **fgets** line by line.
3. User input string gets written to **writefd**’s connected pipe.
4. It formulate server “**RESULT**:” and print output.

**Server.c**

1. Server function use 2 file descriptor (**readfd**, **writefd**)
2. Read the expression/user input sent by client from the pipe connected to readfd
3. Calls the calculate function to evaluate the mathematical expression received from client.
4. formats the evaluated results as string and send back to client through the write pipe

**Calculate.c**

1. Extracting operands from the buffers and storing into **buf\_operands.**
2. Extracting operations from the buffers and storing into **buf\_operation**.
3. Evaluating precedent 1 and 2 into using the values of operands and operations store into buffers.

**Calc.c**

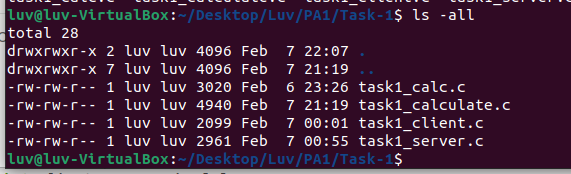
1. Create 2 pipes (**clienttoServerPipe** and **servertoClientPipe**)
2. Uses fork() to create a child process (server) from the parent process (Client).
3. Child Process : Reads -> **clienttoServerPipe** and Write🡪 **ServertoClientPipe**
4. Call Server () to handle server side functionality
5. Parent Process: Reads -> **ServertoClientPipe** and write -> **clienttoServerPipe**
6. Call Client () to handle client side functionality
7. Calls the server() in the child process and client () in the parent process.

**Engaged in collaborative discussion with Mr Udayan Singh, a fellow participant within our 2023 cohort, regarding problem statement illustrated and possible potential methodologies for TASK-1**

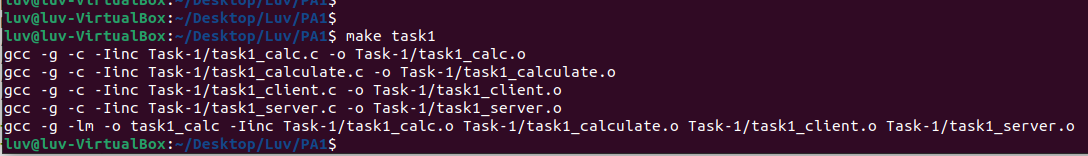
**Implementation steps & Test-case Results**

Steps by Steps of implementation of the **client.c , server.c , calculate.c , calc.c** file within code base structure execute and test it using test\_task1.sh script

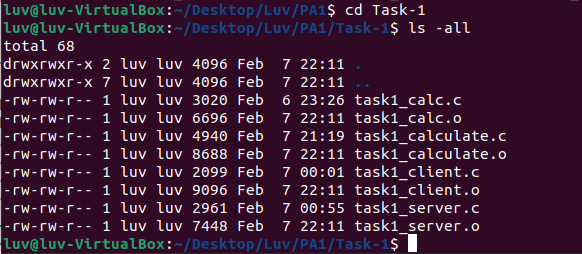
1. Place the attached files **client.c , server.c , calculate.c , calc.c** into PA/Task-1 folder



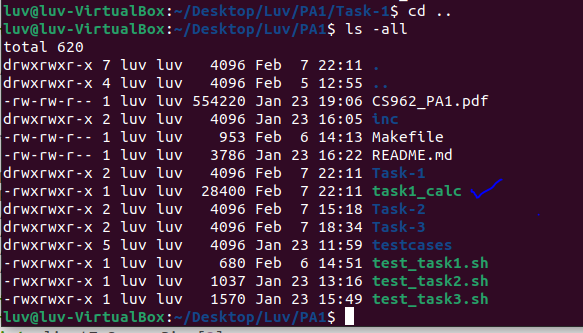
1. Run the command from **PA/** folder "**make task1**" - this will compile the code (if you make any changes in the files then you have to run make command once again).



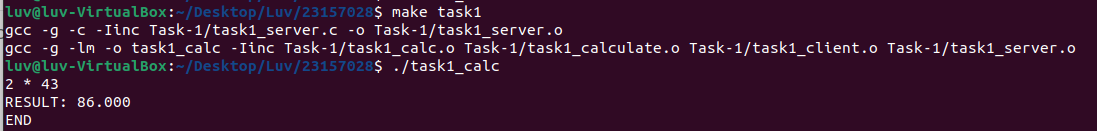
1. After step 2, below are behaviours are observed after compiling both files
2. This will create **client.o , server.o , calculate.o , calc.o** within **PA/Task-3/** folder.



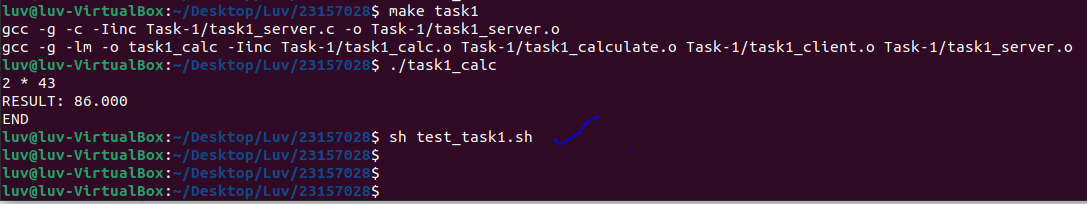
1. This will create compiled file: **task1\_calc** within **PA/** folder



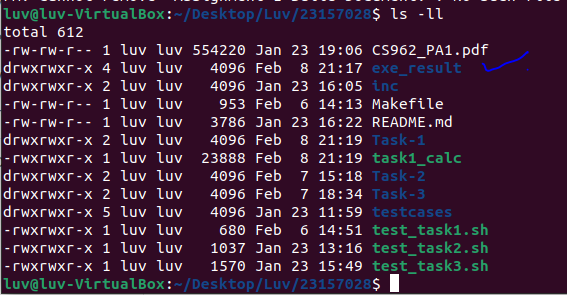
1. Now, from **/Luv/23157028** folder, we will run/execute the compiled file **task1\_calc** using command. .**/task1\_calc** script for checking if code file are running with random test-case supplied for validating the TASK-1.



1. Now, from /**Luv/23157028** folder, we will run/execute the test-case: **test\_task1.sh** script which has been provided Prof/TA for checking if code file are running against the test case for TASK-1 using command "**sh test\_task1.sh**"

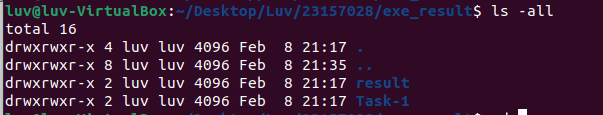


1. After running test-case script for task1 validation in step 5, there will be **exe\_result** folder gets created within **Luv/23157028** folder

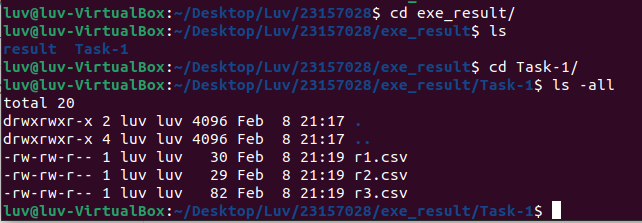


1. Folder **exe\_result** created in step 6, has 2 folders further - **result** and **Task-1**. Same has been attached for evaluation purpose.



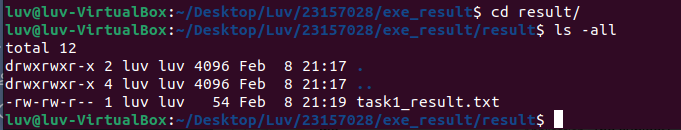


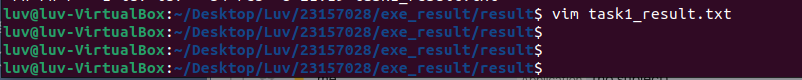
1. Task-1 Folder shows 3 excel files which are created – **r1.csv, r2.csv, r3.csv** generated when we run the script **'test\_task1.sh'** taking **task1\_calc** compile file.

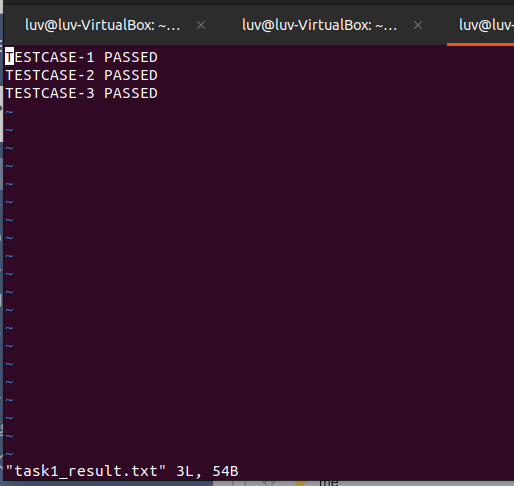




1. **result** folder has **task1\_result.txt** file which shows the status of the test case run after validating the files above generated (**r1.csv , r2.csv , r3.csv**).







****