

Name: Luv Ahuja

Subject: CS962 - Operating System Principle

Assignment No: 1

Roll Number : 23157028

Email ID: luvahuja23@iitk.ac.in

Deliverables

Deliverable:

1. Source code of Task-1, Task-2 and Task-3 which you have implemented. Document your code properly. The source code of each task should be placed on the same place as provided in the code base. **Please do not change PA1 code structure as we will be using automated scripts for testing.**
2. Feel free to discuss about the assignment among your friends, instructor and TAs. However, at the end of the day, we want you to do the implementation by yourself.
3. Write a README.txt file containing what you have discussed/with whom/any other sources that you have referred to do the assignment. If you have used any scripts/code from your friends, do mention that as well.
4. You have to submit zip file named `your_roll_number.zip` (for example: `12345.zip`) containing **only** the following files in specified folder format:

```
12345.zip
|
|----- 12345
|           |----- Task-1
|           |           |----- task1_calc.c
|           |           |----- task1_calculate.c
|           |           |----- task1_client.c
|           |           |----- task1_server.c
|           |----- Task-2
|           |           |----- task2_driver.c
|           |           |----- task2_user.c
|           |----- Task-3
|           |           |----- myDU.c
```

Task-2

Task 2: Lets Chit-Chat !!! Chat Between Two Users [30 points]

In this task, you need to implement chat communication between two users using pipe. The main process acts as the driver and creates two child processes to facilitate communication between them using pipes. Each child processes (i.e. user-1 and user-2) exec the same "user program" binary to start the conversation between them. User program reads the chat content which has to be communicated to the other user from the **chat content** file (provided) and writes the chat content received from other user to the **store content** file. Each line in chat content can be a maximum of 500 characters.

Figure 2 shows the workflow of this chat communication. One of the users will be the **initiator** based upon the first line (**keyword: initiate**) in the **chat content** file. The initiator user will start the chat by sending next line (actual message) after initiate keyword in the **chat content** file. Similar to the initiator, any user can be the **terminator** of chat communication. Terminator will have **keyword: bye** in the chat content file or there is no more content left for communication in the **chat content** file.

As part of this task, you need to implement both driver and user program. The driver is responsible for fork & exec the user programs and establishing the communication channels between them. The user program is responsible for reading the chat messages and writing the replies as mentioned above. The driver program should accept 4 command line arguments as follows.

```
./task2_driver <user1_content> <user1_store> <user2_content> <user2_store>
```

Implementation

- Navigate to the **Task-2** folder of code base, there are two C files, **task2_driver.c** and **task2_user.c**, to implement the driver and user program, respectively.
- After implementation, run the **make task2** command from the root folder of this assignment. You will get two binary files at the root folder, namely **task2_driver** and **task2_user** (inside **Task-2**

folder), corresponding to the driver and user programs.

- You need to exec **task2_user** inside the driver program two times to mimic two users.
- Use the **task2_driver** binary as mentioned above to test the correct functionality of your implementation.
- You will find two additional text files for testing purposes, namely— **chat-1.txt** and **chat-2.txt**, inside the **Task-2** folder, which holds the sample **chat contents** for two users.

You can use below mentioned APIs to implement this part of the assignment. Refer to man page of these APIs to know about their usage.

pipe	fopen	strcmp	exec
close	perror	exit	strcpy
wait	getline	sscanf	sprintf
read	free	strlen	fclose
write	fputs	atoi	

Code files



task2_user.c



task2_driver.c

Resources

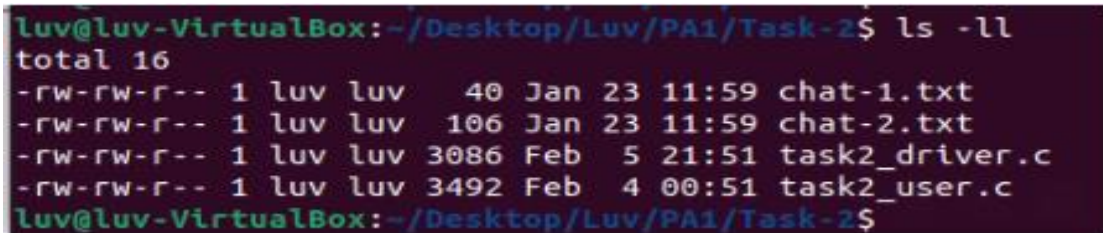
Usage of the `exec1()`, `fork()`, `pipe()` system calls after understanding from below stack overflow

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

Implementation steps & Test-case Results

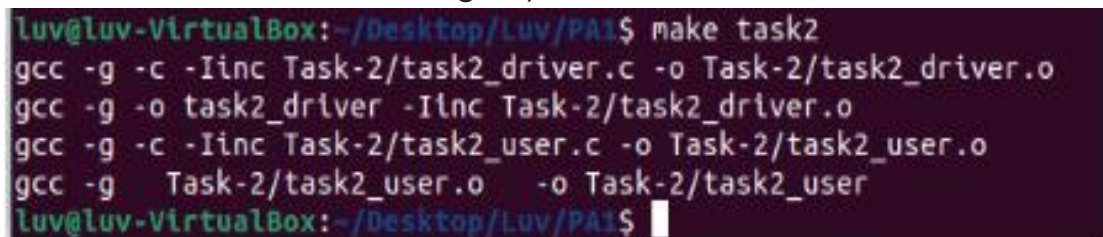
Steps by Steps of implementation of the **task2_user.c** file and **task2_driver.c** within code base structure execute and test it using `test_task2.sh` script

1. Place the attached files **task2_user.c** and **task2_driver.c** into **PA/Task-2** folder



```
luv@luv-VirtualBox:~/Desktop/Luv/PA1/Task-2$ ls -ll
total 16
-rw-rw-r-- 1 luv luv  40 Jan 23 11:59 chat-1.txt
-rw-rw-r-- 1 luv luv 106 Jan 23 11:59 chat-2.txt
-rw-rw-r-- 1 luv luv 3086 Feb  5 21:51 task2_driver.c
-rw-rw-r-- 1 luv luv 3492 Feb  4 00:51 task2_user.c
luv@luv-VirtualBox:~/Desktop/Luv/PA1/Task-2$
```

2. 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).



```
luv@luv-VirtualBox:~/Desktop/Luv/PA1$ make task2
gcc -g -c -Iinc Task-2/task2_driver.c -o Task-2/task2_driver.o
gcc -g -o task2_driver -Iinc Task-2/task2_driver.o
gcc -g -c -Iinc Task-2/task2_user.c -o Task-2/task2_user.o
gcc -g Task-2/task2_user.o -o Task-2/task2_user
luv@luv-VirtualBox:~/Desktop/Luv/PA1$
```

3. After step 2, below are behaviours are observed after compiling both files
 - (a) This will create **task2_user.o** and **task2_driver.o** within **PA/Task-2/** folder.

```

luv@luv-VirtualBox:~/Desktop/Luv/PA1/Task-2$ ls -ll
total 64
-rw-rw-r-- 1 luv luv 40 Jan 23 11:59 chat-1.txt
-rw-rw-r-- 1 luv luv 106 Jan 23 11:59 chat-2.txt
-rw-rw-r-- 1 luv luv 3086 Feb 6 00:02 task2_driver.c
-rw-rw-r-- 1 luv luv 7568 Feb 6 00:04 task2_driver.o
-rwxrwxr-x 1 luv luv 20704 Feb 6 00:04 task2_user
-rw-rw-r-- 1 luv luv 3492 Feb 4 00:51 task2_user.c
-rw-rw-r-- 1 luv luv 12880 Feb 6 00:04 task2_user.o
luv@luv-VirtualBox:~/Desktop/Luv/PA1/Task-2$

```

- (b) This will create compiled file: **task2_user** within **PA/TASK-2/** folder.

```

luv@luv-VirtualBox:~/Desktop/Luv/PA1/Task-2$ ls -ll
total 64
-rw-rw-r-- 1 luv luv 40 Jan 23 11:59 chat-1.txt
-rw-rw-r-- 1 luv luv 106 Jan 23 11:59 chat-2.txt
-rw-rw-r-- 1 luv luv 3086 Feb 6 00:02 task2_driver.c
-rw-rw-r-- 1 luv luv 7568 Feb 6 00:04 task2_driver.o
-rwxrwxr-x 1 luv luv 20704 Feb 6 00:04 task2_user
-rw-rw-r-- 1 luv luv 3492 Feb 4 00:51 task2_user.c
-rw-rw-r-- 1 luv luv 12880 Feb 6 00:04 task2_user.o
luv@luv-VirtualBox:~/Desktop/Luv/PA1/Task-2$

```

- (c) This will create compiled file: **task2_driver** within **PA/** folder

```

luv@luv-VirtualBox:~/Desktop/Luv/PA1$ ls -ll
total 604
-rw-rw-r-- 1 luv luv 554220 Jan 23 19:06 CS962_PA1.pdf
drwxrwxr-x 2 luv luv 4096 Jan 23 16:05 inc
-rw-rw-r-- 1 luv luv 948 Jan 23 16:09 Makefile
-rw-rw-r-- 1 luv luv 3786 Jan 23 16:22 README.md
drwxrwxr-x 2 luv luv 4096 Feb 5 20:40 Task-1
drwxrwxr-x 2 luv luv 4096 Feb 6 00:04 Task-2
-rwxrwxr-x 1 luv luv 18552 Feb 6 00:04 task2_driver
drwxrwxr-x 2 luv luv 4096 Feb 5 23:44 Task-3
drwxrwxr-x 5 luv luv 4096 Jan 23 11:59 testcases
-rwxrwxr-x 1 luv luv 680 Jan 23 12:15 test_task1.sh
-rwxrwxr-x 1 luv luv 1037 Jan 23 13:16 test_task2.sh
-rwxrwxr-x 1 luv luv 1570 Jan 23 15:49 test_task3.sh
luv@luv-VirtualBox:~/Desktop/Luv/PA1$

```

4. 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**"


```

luv@luv-VirtualBox:~/Desktop/Luv/PA1$ ls -ll
total 604
-rw-rw-r-- 1 luv luv 554220 Jan 23 19:06 CS962_PA1.pdf
drwxrwxr-x 2 luv luv 4096 Jan 23 16:05 inc
-rw-rw-r-- 1 luv luv 948 Jan 23 16:09 Makefile
-rw-rw-r-- 1 luv luv 3786 Jan 23 16:22 README.md
drwxrwxr-x 2 luv luv 4096 Feb 5 20:40 Task-1
drwxrwxr-x 2 luv luv 4096 Feb 6 00:04 Task-2
-rwxrwxr-x 1 luv luv 18552 Feb 6 00:04 task2_driver
drwxrwxr-x 2 luv luv 4096 Feb 5 23:44 Task-3
drwxrwxr-x 5 luv luv 4096 Jan 23 11:59 testcases
-rwxrwxr-x 1 luv luv 680 Jan 23 12:15 test_task1.sh
-rwxrwxr-x 1 luv luv 1037 Jan 23 13:16 test_task2.sh
-rwxrwxr-x 1 luv luv 1570 Jan 23 15:49 test_task3.sh
luv@luv-VirtualBox:~/Desktop/Luv/PA1$ sh test_task2.sh
luv@luv-VirtualBox:~/Desktop/Luv/PA1$

```

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

```

luv@luv-VirtualBox:~/Desktop/Luv/PA1$ ls -ll
total 608
-rw-rw-r-- 1 luv luv 554220 Jan 23 19:06 CS962_PA1.pdf
drwxrwxr-x 4 luv luv 4096 Feb 6 00:16 exe_result ✓
drwxrwxr-x 2 luv luv 4096 Jan 23 16:05 inc
-rw-rw-r-- 1 luv luv 948 Jan 23 16:09 Makefile
-rw-rw-r-- 1 luv luv 3786 Jan 23 16:22 README.md
drwxrwxr-x 2 luv luv 4096 Feb 5 20:40 Task-1
drwxrwxr-x 2 luv luv 4096 Feb 6 00:04 Task-2
-rwxrwxr-x 1 luv luv 18552 Feb 6 00:04 task2_driver
drwxrwxr-x 2 luv luv 4096 Feb 5 23:44 Task-3
drwxrwxr-x 5 luv luv 4096 Jan 23 11:59 testcases
-rwxrwxr-x 1 luv luv 680 Jan 23 12:15 test_task1.sh
-rwxrwxr-x 1 luv luv 1037 Jan 23 13:16 test_task2.sh
-rwxrwxr-x 1 luv luv 1570 Jan 23 15:49 test_task3.sh
luv@luv-VirtualBox:~/Desktop/Luv/PA1$

```

6. Folder **exe_result** created in step 4, has 2 folders further - **result** and **Task-2**. Same has been attached for evaluation purpose.



exe_result.zip

- (a) 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.

Task-3

Task 3: Directory Space Usage [40 points]

In this question, you have to write a program (`myDU.c`) that finds the space used by a directory (including its files, sub-directories, files in sub-directories, sub-sub directories etc.). Let's call this directory as the *Root* directory.

Syntax

```
$/myDU <relative path to a directory>
```

Example

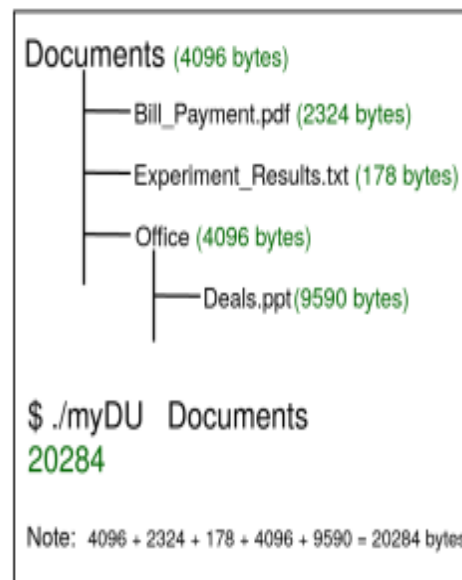


Figure 3: Example to illustrate the use of directory space usage finding utility

Figure 3 shows the structure of a directory called **Documents** which is the designated *Root* directory in this example. This directory contains files such as **Bill_Payment.pdf**, **Experiment_Results.txt** and a sub-directory called **Office**. The Sub-directory **Office** contains a file named **Deals.ppt**. To find the size of the **Documents** directory, its name is passed as `$/myDU Documents`. Your utility is expected

to print the total size of the contents of the passed root directory in bytes (**For eg: 20284**). Note that, this is inclusive of directory sizes.

Output

Only print the size of the *Root* directory in bytes (Refer to Figure 3)

Detailed instructions

To make the calculation process more efficient, we propose a method where different sub-directories under the *Root* directory will be processed by different processes. The exact working is detailed in the following points.

- Assume that there are N immediate sub-directories under the *Root* directory. For each immediate child sub-directory under the provided *Root* directory, your program must create a new process P_i (i will range from 1 to the total number of immediate sub-directories under *Root*). Each child process P_i should find the size of the i^{th} child sub-directory (including all files and directories under it and the size of the sub-directory itself) and pass this information back to the parent process. Parent process should find the size of the files immediately under it along with the *Root* directory size. Finally, parent process will find the sum of all sizes and print the final output.

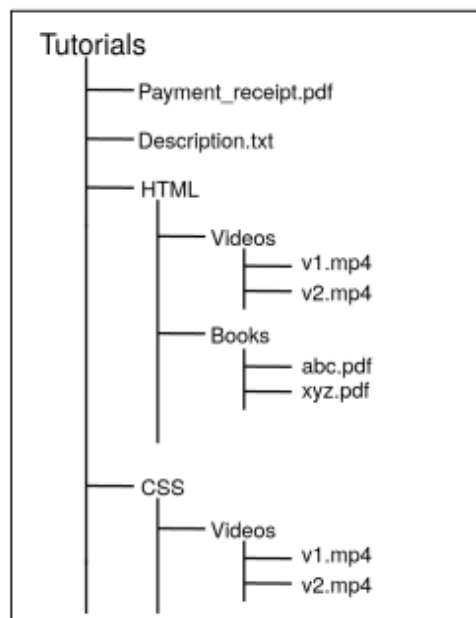


Figure 4: Sample directory structure

For example: In Figure 4, there are two immediate sub-directories (**HTML**, **CSS**) under the *Root* directory (**Tutorials**). In this case, the parent process should create two child processes, say, P_1 and P_2 . P_1 should calculate the size of the sub-directory (**HTML**) which is a sum of sizes of all files and directories under **HTML** including the size of **HTML** itself. Similarly, P_2 should calculate size of the directory **CSS**. Both P_1 and P_2 should return the result back to the parent using pipes. Finally, the parent process would find the size of the files immediately under it (**Payment_receipt.pdf**, **Description.txt**) and the size of the **Tutorials** directory to report the final result.

- Your program should *only* use pipes (pipe() system call) to communicate between parent process and the child processes. There is no restriction on the number of pipes to be used.

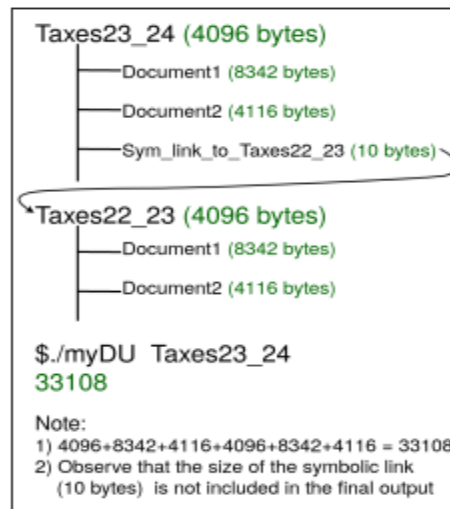


Figure 5: Example to illustrate the handling of symbolic links by ./myDU utility

- A symbolic link is a special type of file in Linux that points to another file or directory. Symbolic links can be present anywhere in the *Root* directory tree. You should resolve the symbolic links and find the size of the file/directory pointed by a symbolic link instead of reporting the size of the symbolic link file (Refer figure 5). It can be assumed that a symbolic link will never point to itself recursively. For example, in Figure 5, Taxes22_23 directory will not contain a symbolic link that points to the Taxes23_24 directory or directly to the Sym_link_to_Taxes22_23 symbolic file in the Taxes23_24 directory.
- It is the responsibility of the parent process to find the size of the file/directory pointed by a symbolic link which is present immediately under *Root* directory. For example, in Figure 5 parent process will not create any child process. However, if a symbolic link is present in a sub-directory, it should be processed by the child process handling the sub-directory.
- During testing, only relative path of the *Root* directory will be passed to the ./myDU utility. It can be assumed that the size of the directory path generated during testing will not exceed 4096 characters.
- You can assume that the total size of the *Root* directory will fit in a 8-byte integer type (i.e., *unsigned long* on 64-bit machines)

Error handling

In case of any error, print "Unable to execute" as output.

System calls and library functions allowed

- | | |
|-----------------|----------------|
| - fork | - malloc |
| - exec* family | - free |
| - pipe | - stat |
| - opendir | - lstat |
| - readdir | - readlink |
| - closedir | - strlen |
| - read | - open |
| - write | - close |
| - strcpy | - strcat |
| - strcmp | - strt* family |
| - ato* family | - wait/waitpid |
| - printf family | - exit |
| - dup | - dup2 |

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

```
luv@luv-VirtualBox:~/Desktop/Luv/PA1$ cd Task-3
luv@luv-VirtualBox:~/Desktop/Luv/PA1/Task-3$ ls
myDU.c
luv@luv-VirtualBox:~/Desktop/Luv/PA1/Task-3$ ls -ll
total 4
-rw-rw-r-- 1 luv luv 3705 Feb  6 12:55 myDU.c
luv@luv-VirtualBox:~/Desktop/Luv/PA1/Task-3$
```

2. 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).

```
luv@luv-VirtualBox:~/Desktop/Luv/PA1$  
luv@luv-VirtualBox:~/Desktop/Luv/PA1$ make task3  
gcc -g -c -Iinc Task-3/myDU.c -o Task-3/myDU.o  
gcc -g -o myDU -Iinc Task-3/myDU.o  
luv@luv-VirtualBox:~/Desktop/Luv/PA1$
```

3. After step 2, below are behaviours are observed after compiling both files

- (a) This will create **myDU.o** within **PA/Task-2/** folder.

```
luv@luv-VirtualBox:~/Desktop/Luv/PA1/Task-3$ ls -ll  
total 20  
-rw-rw-r-- 1 luv luv 3705 Feb 6 12:55 myDU.c  
-rw-rw-r-- 1 luv luv 14952 Feb 7 16:56 myDU.o ✓  
luv@luv-VirtualBox:~/Desktop/Luv/PA1/Task-3$
```

- (b) This will create compiled file: **myDU** within **PA/** folder

```
luv@luv-VirtualBox:~/Desktop/Luv/PA1$ ls -ll  
total 608  
-rw-rw-r-- 1 luv luv 554220 Jan 23 19:06 CS962_PA1.pdf  
drwxrwxr-x 2 luv luv 4096 Jan 23 16:05 inc  
-rw-rw-r-- 1 luv luv 953 Feb 6 14:13 Makefile  
-rwxrwxr-x 1 luv luv 21928 Feb 7 16:56 myDU ✓  
-rw-rw-r-- 1 luv luv 3786 Jan 23 16:22 README.md  
drwxrwxr-x 2 luv luv 4096 Feb 7 00:55 Task-1  
drwxrwxr-x 2 luv luv 4096 Feb 7 15:18 Task-2  
drwxrwxr-x 2 luv luv 4096 Feb 7 16:56 Task-3  
drwxrwxr-x 5 luv luv 4096 Jan 23 11:59 testcases  
-rwxrwxr-x 1 luv luv 680 Feb 6 14:51 test_task1.sh  
-rwxrwxr-x 1 luv luv 1037 Jan 23 13:16 test_task2.sh  
-rwxrwxr-x 1 luv luv 1570 Jan 23 15:49 test_task3.sh  
luv@luv-VirtualBox:~/Desktop/Luv/PA1$
```

4. 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**"

```

luv@luv-VirtualBox:~/Desktop/Luv/PA1$ ls -ll
total 608
-rw-rw-r-- 1 luv luv 554220 Jan 23 19:06 CS962_PA1.pdf
drwxrwxr-x 2 luv luv 4096 Jan 23 16:05 inc
-rw-rw-r-- 1 luv luv 953 Feb 6 14:13 Makefile
-rwxrwxr-x 1 luv luv 21928 Feb 7 16:56 myDU
-rw-rw-r-- 1 luv luv 3786 Jan 23 16:22 README.md
drwxrwxr-x 2 luv luv 4096 Feb 7 00:55 Task-1
drwxrwxr-x 2 luv luv 4096 Feb 7 15:18 Task-2
drwxrwxr-x 2 luv luv 4096 Feb 7 16:56 Task-3
drwxrwxr-x 5 luv luv 4096 Jan 23 11:59 testcases
-rwxrwxr-x 1 luv luv 680 Feb 6 14:51 test_task1.sh
-rwxrwxr-x 1 luv luv 1037 Jan 23 13:16 test_task2.sh
-rwxrwxr-x 1 luv luv 1570 Jan 23 15:49 test_task3.sh
luv@luv-VirtualBox:~/Desktop/Luv/PA1$ sh test_task3.sh

```

5. 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.

```

luv@luv-VirtualBox:~/Desktop/Luv/PA1$
luv@luv-VirtualBox:~/Desktop/Luv/PA1$ sh test_task3.sh
Expected output: 24826
Your output: 24826
Testcase 1 passed

Expected output: 4216
Your output: 4216
Testcase 2 passed

Expected output: 33106
Your output: 33106
Testcase 3 passed
Expected output: 49652
Your output: 49652
Testcase 4 passed
luv@luv-VirtualBox:~/Desktop/Luv/PA1$ timedatectl
          Local time: Wed 2024-02-07 17:34:39 IST
        Universal time: Wed 2024-02-07 12:04:39 UTC
              RTC time: Wed 2024-02-07 12:04:39
            Time zone: Asia/Kolkata (IST, +0530)
System clock synchronized: yes
              NTP service: active
          RTC in local TZ: no
luv@luv-VirtualBox:~/Desktop/Luv/PA1$

```


6. 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)

```
luv@luv-VirtualBox:~$ cd Desktop/RAMAYAN/
luv@luv-VirtualBox:~/Desktop/RAMAYAN$ ls -all
total 20
drwxrwxr-x 4 luv luv 4096 Feb  7 17:40 .
drwxr-xr-x 7 luv luv 4096 Feb  7 17:42 ..
-rw-rw-r-- 1 luv luv 767 Feb  4 01:03 HANUMANJI
drwxrwxr-x 2 luv luv 4096 Feb  7 17:41 RAM
drwxrwxr-x 2 luv luv 4096 Feb  7 17:42 SITA
luv@luv-VirtualBox:~/Desktop/RAMAYAN$
```

```
luv@luv-VirtualBox:~/Desktop/RAMAYAN$ cd RAM/
luv@luv-VirtualBox:~/Desktop/RAMAYAN/RAM$ ls -all
total 20
drwxrwxr-x 2 luv luv 4096 Feb  7 17:41 .
drwxrwxr-x 4 luv luv 4096 Feb  7 17:40 ..
-rw-rw-r-- 1 luv luv 1118 Feb  4 00:48 RAM-1
-rw-rw-r-- 1 luv luv 712 Feb  4 00:49 RAM-2
-rw-rw-r-- 1 luv luv 712 Feb  4 00:49 RAM-3
luv@luv-VirtualBox:~/Desktop/RAMAYAN/RAM$
```

```
luv@luv-VirtualBox:~/Desktop/RAMAYAN$ cd SITA/
luv@luv-VirtualBox:~/Desktop/RAMAYAN/SITA$ ls -all
total 20
drwxrwxr-x 2 luv luv 4096 Feb  7 17:42 .
drwxrwxr-x 4 luv luv 4096 Feb  7 17:40 ..
-rw-rw-r-- 1 luv luv 712 Feb  4 00:50 SITA-1
-rw-rw-r-- 1 luv luv 712 Feb  4 00:50 SITA-2
-rw-rw-r-- 1 luv luv 712 Feb  4 00:50 SITA-3
luv@luv-VirtualBox:~/Desktop/RAMAYAN/SITA$
```

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

```
luv@luv-VirtualBox:~/Desktop/Luv/PA1$  
luv@luv-VirtualBox:~/Desktop/Luv/PA1$ du -bsLL /home/luv/Desktop/RAMAYAN  
17733    /home/luv/Desktop/RAMAYAN  
luv@luv-VirtualBox:~/Desktop/Luv/PA1$ ./myDU /home/luv/Desktop/RAMAYAN  
17733  
luv@luv-VirtualBox:~/Desktop/Luv/PA1$
```

Task-1

Task 1: Lets Do Some Calculation !!!! Client-Server Calculator [30 Points]

In this task, you need to implement a client-server based calculator. The client can submit any number of queries (i.e. mathematical expressions) to the server. The server will evaluate the client's queries one by one and send back the answer as a response.

In this task, the main process will call fork to create a child process where child process acts as the server and the parent (main process) acts as client. The server will respond to all queries submitted by the client until it receives a specific terminating word (mentioned below) as a query from the client. After receiving the specific terminating word, the server will close the communication channel and exits. You need to use pipe to establish a communication channel between client and server.

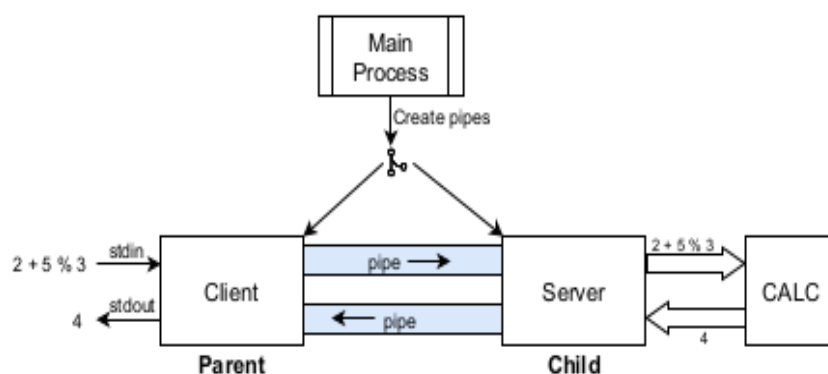


Figure 1: Workflow of client-server based calculator.

Figure 1 shows the workflow of this client-server based calculator using pipes. The client (parent) should read expression (or may be terminating word) from standard input device, and will display result received from the server on standard output device as shown in the figure.

Requirement specifications for this task are as follows:

1. Input expression is always space separated, i.e. blank space between operand and operator.
2. Input expression always starts with an operand.
3. Input expression can have at most 20 operands (i.e., it can have at most 19 operators).
4. Operands in input expression may be an integer or float/double number.
5. Input expression can contain any combination of these five operators viz. (i) Addition (+), (ii) Subtraction (-), (iii) Multiplication (*), (iv) Division (/), and modulo (%).
6. When server performs calculation, it needs to follow **BODMAS** rule where division, multiplication and modulo operators are at same precedence (say precedence-1) and addition and subtraction are at precedence-2. According to BODMAS, server should first perform precedence-1 operations followed by precedence-2 operations. Among same precedence operations, server should perform operation from left to right.
7. When client receives **END** word from STDIN, it should send a termination signal to the server to terminate it properly.
8. On receiving the result of mathematical expressions from the server, client should print "**RESULT:** " followed by result. As an example, if client receive 5 from the server, then it should display "**RESULT: 5**" on STDOUT.

Example of Input & Output:

```
[INPUT] 1 + 5 * 2 - 1 + 10
[OUTPUT] RESULT: 20
[INPUT] 5 + 1 - 2 * 5 + 10
[OUTPUT] RESULT: 6
[INPUT] END
```

Implementation

- Inside **Task-1** folder, you will find four C files namely—(i) **task1_calc.c**, (ii) **task1_calculate.c**, (iii) **task1_client.c**, and (iv) **task1_server.c**.
- You need to establish the communication channel and invoke the client and server function at the appropriate location inside the main function available in **task1_calc.c**.
- The client and server functionality need to be implemented in files **task1_client.c** and **task1_server.c**, respectively.
- In **task1_calculate.c** file, implement the calculate function that the server will invoke whenever required.
- Run **make task1** command from the root folder of this assignment to get binary file named **task1_calc**. Use this binary to test the functionality of your implementation.

You can use below mentioned APIs to implement this part of the assignment. Refer to man page of these APIs to know about their usage.

Code files



task1_dient.c



task1_server.c



task1_calculate.c



task1_calc.c

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

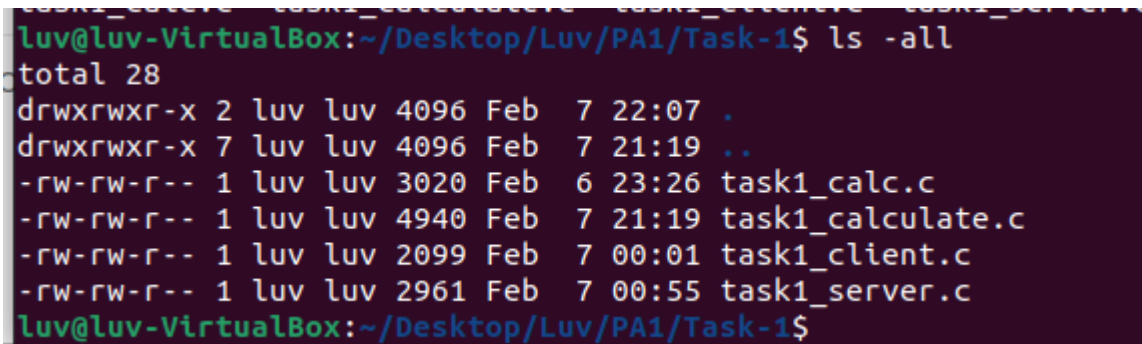
1. Create 2 pipes (**clienttoServerPipe** and **servertoClientPipe**)
2. Uses fork() to create a child process (server) from the parent process (Client).
3. Child Process : Reads -> **clienttoServerPipe** and Write → **ServertoClientPipe**
4. Call Server () to handle server side functionality
5. Parent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



```
luv@luv-VirtualBox:~/Desktop/Luv/PA1/Task-1$ ls -all
total 28
drwxrwxr-x 2 luv luv 4096 Feb  7 22:07 .
drwxrwxr-x 7 luv luv 4096 Feb  7 21:19 ..
-rw-rw-r-- 1 luv luv 3020 Feb  6 23:26 task1_calc.c
-rw-rw-r-- 1 luv luv 4940 Feb  7 21:19 task1_calculate.c
-rw-rw-r-- 1 luv luv 2099 Feb  7 00:01 task1_client.c
-rw-rw-r-- 1 luv luv 2961 Feb  7 00:55 task1_server.c
luv@luv-VirtualBox:~/Desktop/Luv/PA1/Task-1$
```

2. 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).

```

luv@luv-VirtualBox:~/Desktop/Luv/PA1$
luv@luv-VirtualBox:~/Desktop/Luv/PA1$
luv@luv-VirtualBox:~/Desktop/Luv/PA1$ make task1
gcc -g -c -Iinc Task-1/task1_calc.c -o Task-1/task1_calc.o
gcc -g -c -Iinc Task-1/task1_calculate.c -o Task-1/task1_calculate.o
gcc -g -c -Iinc Task-1/task1_client.c -o Task-1/task1_client.o
gcc -g -c -Iinc Task-1/task1_server.c -o Task-1/task1_server.o
gcc -g -lm -o task1_calc -Iinc Task-1/task1_calc.o Task-1/task1_calculate.o Task-1/task1_client.o Task-1/task1_server.o
luv@luv-VirtualBox:~/Desktop/Luv/PA1$

```

3. After step 2, below are behaviours are observed after compiling both files

(c) This will create **client.o** , **server.o** , **calculate.o** , **calc.o** within **PA/Task-3/** folder.

```

luv@luv-VirtualBox:~/Desktop/Luv/PA1$ cd Task-1
luv@luv-VirtualBox:~/Desktop/Luv/PA1/Task-1$ ls -all
total 68
drwxrwxr-x 2 luv luv 4096 Feb  7 22:11 .
drwxrwxr-x 7 luv luv 4096 Feb  7 22:11 ..
-rw-rw-r-- 1 luv luv 3020 Feb  6 23:26 task1_calc.c
-rw-rw-r-- 1 luv luv 6696 Feb  7 22:11 task1_calc.o
-rw-rw-r-- 1 luv luv 4940 Feb  7 21:19 task1_calculate.c
-rw-rw-r-- 1 luv luv 8688 Feb  7 22:11 task1_calculate.o
-rw-rw-r-- 1 luv luv 2099 Feb  7 00:01 task1_client.c
-rw-rw-r-- 1 luv luv 9096 Feb  7 22:11 task1_client.o
-rw-rw-r-- 1 luv luv 2961 Feb  7 00:55 task1_server.c
-rw-rw-r-- 1 luv luv 7448 Feb  7 22:11 task1_server.o
luv@luv-VirtualBox:~/Desktop/Luv/PA1/Task-1$

```

(d) This will create compiled file: **task1_calc** within **PA/** folder

```

luv@luv-VirtualBox:~/Desktop/Luv/PA1/Task-1$ cd ..
luv@luv-VirtualBox:~/Desktop/Luv/PA1$ ls -all
total 620
drwxrwxr-x 7 luv luv 4096 Feb  7 22:11 .
drwxrwxr-x 4 luv luv 4096 Feb  5 12:55 ..
-rw-rw-r-- 1 luv luv 554220 Jan 23 19:06 CS962_PA1.pdf
drwxrwxr-x 2 luv luv 4096 Jan 23 16:05 inc
-rw-rw-r-- 1 luv luv 953 Feb  6 14:13 Makefile
-rw-rw-r-- 1 luv luv 3786 Jan 23 16:22 README.md
drwxrwxr-x 2 luv luv 4096 Feb  7 22:11 Task-1
-rwxrwxr-x 1 luv luv 28400 Feb  7 22:11 task1_calc ✓
drwxrwxr-x 2 luv luv 4096 Feb  7 15:18 Task-2
drwxrwxr-x 2 luv luv 4096 Feb  7 18:34 Task-3
drwxrwxr-x 5 luv luv 4096 Jan 23 11:59 testcases
-rwxrwxr-x 1 luv luv 680 Feb  6 14:51 test_task1.sh
-rwxrwxr-x 1 luv luv 1037 Jan 23 13:16 test_task2.sh
-rwxrwxr-x 1 luv luv 1570 Jan 23 15:49 test_task3.sh
luv@luv-VirtualBox:~/Desktop/Luv/PA1$

```

4. 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.

```
luv@luv-VirtualBox:~/Desktop/Luv/23157028$ make task1
gcc -g -c -Iinc Task-1/task1_server.c -o Task-1/task1_server.o
gcc -g -lm -o task1_calc -Iinc Task-1/task1_calc.o Task-1/task1_calculate.o Task-1/task1_client.o Task-1/task1_server.o
luv@luv-VirtualBox:~/Desktop/Luv/23157028$ ./task1_calc
2 * 43
RESULT: 86.000
END
```

5. 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**"

```
luv@luv-VirtualBox:~/Desktop/Luv/23157028$ make task1
gcc -g -c -Iinc Task-1/task1_server.c -o Task-1/task1_server.o
gcc -g -lm -o task1_calc -Iinc Task-1/task1_calc.o Task-1/task1_calculate.o Task-1/task1_client.o Task-1/task1_server.o
luv@luv-VirtualBox:~/Desktop/Luv/23157028$ ./task1_calc
2 * 43
RESULT: 86.000
END
luv@luv-VirtualBox:~/Desktop/Luv/23157028$ sh test_task1.sh
luv@luv-VirtualBox:~/Desktop/Luv/23157028$
luv@luv-VirtualBox:~/Desktop/Luv/23157028$
luv@luv-VirtualBox:~/Desktop/Luv/23157028$
```

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

```
luv@luv-VirtualBox:~/Desktop/Luv/23157028$ ls -ll
total 612
-rw-rw-r-- 1 luv luv 554220 Jan 23 19:06 CS962_PA1.pdf
drwxrwxr-x 4 luv luv 4096 Feb 8 21:17 exe_result
drwxrwxr-x 2 luv luv 4096 Jan 23 16:05 inc
-rw-rw-r-- 1 luv luv 953 Feb 6 14:13 Makefile
-rw-rw-r-- 1 luv luv 3786 Jan 23 16:22 README.md
drwxrwxr-x 2 luv luv 4096 Feb 8 21:19 Task-1
-rwxrwxr-x 1 luv luv 23888 Feb 8 21:19 task1_calc
drwxrwxr-x 2 luv luv 4096 Feb 7 15:18 Task-2
drwxrwxr-x 2 luv luv 4096 Feb 7 18:34 Task-3
drwxrwxr-x 5 luv luv 4096 Jan 23 11:59 testcases
-rwxrwxr-x 1 luv luv 680 Feb 6 14:51 test_task1.sh
-rwxrwxr-x 1 luv luv 1037 Jan 23 13:16 test_task2.sh
-rwxrwxr-x 1 luv luv 1570 Jan 23 15:49 test_task3.sh
luv@luv-VirtualBox:~/Desktop/Luv/23157028$
```

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



exe_result.zip

```
luv@luv-VirtualBox:~/Desktop/Luv/23157028/exe_result$ ls -all
total 16
drwxrwxr-x 4 luv luv 4096 Feb  8 21:17 .
drwxrwxr-x 8 luv luv 4096 Feb  8 21:35 ..
drwxrwxr-x 2 luv luv 4096 Feb  8 21:17 result
drwxrwxr-x 2 luv luv 4096 Feb  8 21:17 Task-1
```

- (a) 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.

```
luv@luv-VirtualBox:~/Desktop/Luv/23157028$ cd exe_result/
luv@luv-VirtualBox:~/Desktop/Luv/23157028/exe_result$ ls
result Task-1
luv@luv-VirtualBox:~/Desktop/Luv/23157028/exe_result$ cd Task-1/
luv@luv-VirtualBox:~/Desktop/Luv/23157028/exe_result/Task-1$ ls -all
total 20
drwxrwxr-x 2 luv luv 4096 Feb  8 21:17 .
drwxrwxr-x 4 luv luv 4096 Feb  8 21:17 ..
-rw-rw-r-- 1 luv luv  30 Feb  8 21:19 r1.csv
-rw-rw-r-- 1 luv luv  29 Feb  8 21:19 r2.csv
-rw-rw-r-- 1 luv luv  82 Feb  8 21:19 r3.csv
luv@luv-VirtualBox:~/Desktop/Luv/23157028/exe_result/Task-1$
```



r1.csv



r2.csv



r3.csv

- (b) **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**).

```
luv@luv-VirtualBox:~/Desktop/Luv/23157028/exe_result$ cd result/
luv@luv-VirtualBox:~/Desktop/Luv/23157028/exe_result/result$ ls -all
total 12
drwxrwxr-x 2 luv luv 4096 Feb  8 21:17 .
drwxrwxr-x 4 luv luv 4096 Feb  8 21:17 ..
-rw-rw-r-- 1 luv luv  54 Feb  8 21:19 task1_result.txt
luv@luv-VirtualBox:~/Desktop/Luv/23157028/exe_result/result$
```

```
luv@luv-VirtualBox:~/Desktop/Luv/23157028/exe_result/result$ vim task1_result.txt
luv@luv-VirtualBox:~/Desktop/Luv/23157028/exe_result/result$
luv@luv-VirtualBox:~/Desktop/Luv/23157028/exe_result/result$
luv@luv-VirtualBox:~/Desktop/Luv/23157028/exe_result/result$
```