Programming Assignment 1
(Use of System Calls)
CS962: Operating System Principles
2022 - Quarter 1
eMasters in Cyber Security
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur
Due Date: 15th February 2022, 11:55pm

This programming assignment makes you acquainted with the working principle of system calls. You need to establish a communication channel between two processes to achieve different functionalities as described in each task. You can download/clone the code base and all other relevant files for PA1 from the following link `https://github.com/skmtr1/emasters-os-2022-PA1.git`. Template code for tasks are placed under the root folder in respective folders—Task-1, Task-2 and Task-3. Please go through the `README` file to understand the code base structure and build process of the assignment.

Task 1: Lets Do Some Calculation !!!! Client-Server Calculator [40 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 client's queries one by one and send back the answer as a response.

In this task, the client-server pair is created by fork where child process acts as the server and the parent 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 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

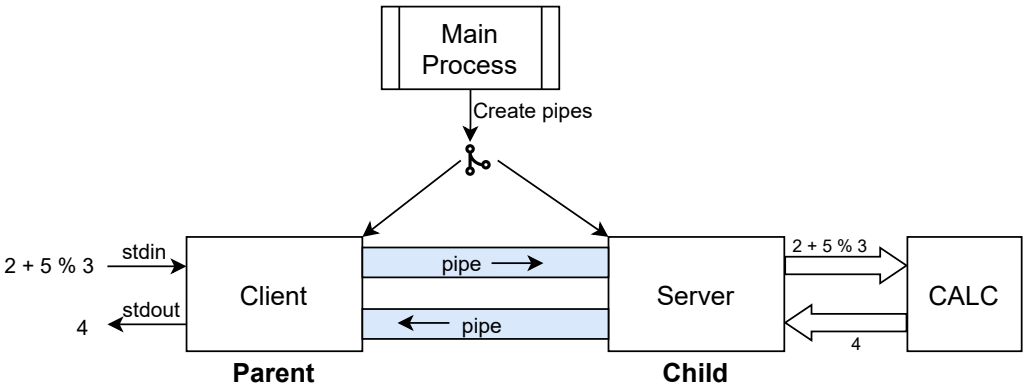

Figure 1: Workflow of client-server based calculator.

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 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.

```
pipe          write         strcmp
close         sprintf       sscanf
wait          strtok        exit
read          strcmp
```

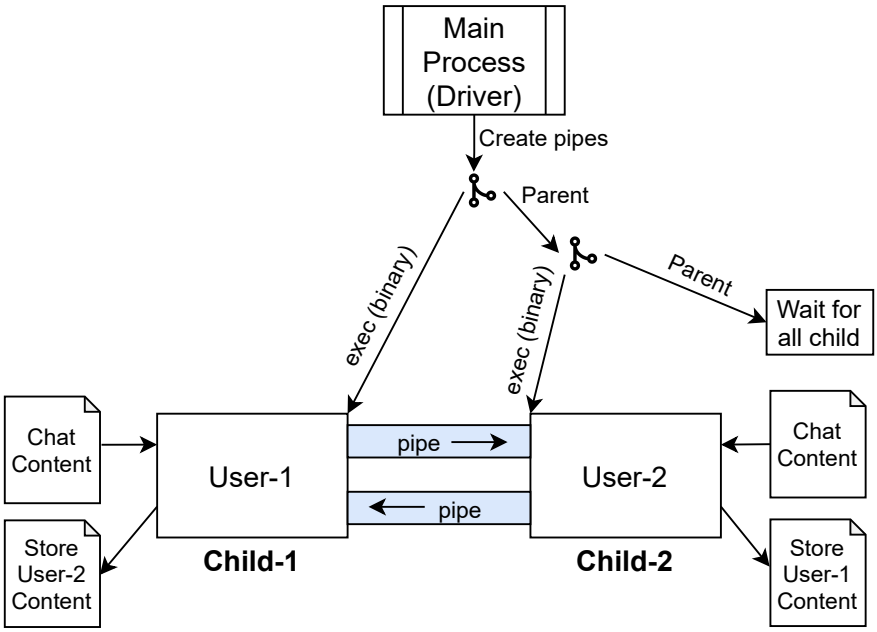

Figure 2: Flow of establishing chat communication between two user.

Task 2: Lets Chit-Chat !!! Chat Between Two Users [40 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.

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 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**, 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.
- We will push the sample output file (correct **store content** files) in some time to the github repo, You can use these output files to verify your **store content** files.

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`	`sscanf`	`strcpy`
`wait`	`getline`	`sscanf`	`sprintf`
`read`	`free`	`strlen`	`fclose`
`write`	`fputs`	`atoi`	`exit`

Task 3: Now Secure it !!!! Securing the Chat Communication [20 points]

Now you will secure the communication between users that you have implemented in task-2 by encrypting the chat using secure key. Here one of the users will be **initiator** as in task-2 based upon the first line (keyword: initiate) in the **chat content** file. The **initiator** will have secure key as the second line in the **chat content** file. The **initiator** will send this secure key as first message in the chat communication. The other user (non initiator) will have secure key as first line in her corresponding **chat content** file and will send this as first message to initiator after receiving key from the initiator. All further communication between these two users will be encrypted by their on key. i.e. initiator will send all his messages from third line on wards in **chat content** file by encrypting using his key, same way non-initiator will send all her messages encrypted using her key. Secure key will have two components **key_a** and **key_b**. Both these components of key can hold any value between **0** to **25**. The format of line which contains secure key inside the **chat content** file is **KEY key_a Key_b**.

On receiving encrypted message, the users should write messages both encrypted and decrypted version of the messages to their respective **store content** files.

Same as task-2, 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.

./task3_driver <user1_content> <user1_store> <user2_content> <user2_store>

Implementation

- Navigate to the **Task-3** folder of code base, you will find two C files, **task3_driver.c** and **task3_user.c**, to implement the driver and user program, respectively.
- Run the **make task3** command from the root folder of this assignment. You will get two binary files at the root folder, **task3_driver** and **task3_user**, corresponding to the driver and user program.
- Exec **task3_user** inside the driver program two times to mimic two users same as in task-2.
- Use the **task3_driver** binary as mentioned above to test the correct functionality of your implementation.
- For testing purposes, there are two additional text files, **chat-1.txt** and **chat-2.txt** inside the **Task-3** folder, these hold the sample **chat contents** for two users.

- We will push the sample output file (correct **store content** files) in some time to the github repo, You can use these output files to verify your **store content** files.
- For the encryption and decryption process, you may use the **decryptMessage(char msg[], KEYS key, size_t length)** and **encryptMessage(char msg[], KEYS key, size_t length)** custom-built function which is available in crypt.c file inside the Task-3 folder itself.

More information on encrypt/decrypt functions (provided):
These functions take three arguments—(i) **msg:** a message to encrypt/decrypt, (ii) **KEYS key:** key to be used for encryption/decryption process, and (iii) **length:** the length of the message. These functions will take encrypted/decrypted messages on the msg buffer and return the decrypted/encrypted message on the same buffer. Here, **KEYS** is a C structure that holds both key components to be used for the encryption and decryption process. The declaration and usage of the **KEYS** structure are as follows:

```
/* Declaration */
typedef struct{
    int key_a;
    int key_b;
} KEYS;

/* usage */
KEYS key;       //declaration of KEYS type variable.
key.key_a = 2;  // Setting the first component key_a of key
key.key_b = 10; // Setting the second component key_b of key
```

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	sscanf	strcpy
wait	getline	sscanf	sprintf
read	free	strlen	fclose
write	fputs	atoi	exit

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. Submit the tarball/zip file of your assignment named as your roll number. You can use makefile provided in the code base to prepare tarball. The syntax for preparing tar file using makefile is as follows:
   make prepare-submit RNO=your_roll_number
   We will accept your submission via emasters portal only, and any other submission mode is strictly prohibited such as submitting via email.

All the best. Looking forward to the submissions !!.....PS: Start Early.