French-Azerbaijani University

Operating Systems & Architecture

Evaluation 1 : IPC System V

Lecturers and Exercise Session Professors :

Konul Aliyeva — Pierre Parrend — Pierre David

April 18, 2020

1 Exercise 1

1.1 Overview

Firstly, my code consist of two C programs where one play the role of *server* and the other as a *client*.In order to make this system work, `server.c` must be run first and stay like so.

1.2 `server.c` Functions

- `new_queue()` - creates a new message queue according to the key generated with invoked *ftok()* function.
- `dequeue()` - simply deletes the message queue with *queue_id* sent as argument.
- `reader()` - It receives messages infinitely until it does the same with a message byte of 0 which simply means **EOF**.To do these, it takes *queue_id* to know where to read the message and of course array of messages to later sort them in `handle()` function with respect to their priorities.
- `handle()` - This is just a function which takes these processed messages[] array and process them further to get printed by priorities (*e.g first prints messages by priority 1, then 2 and 3 at the end.*

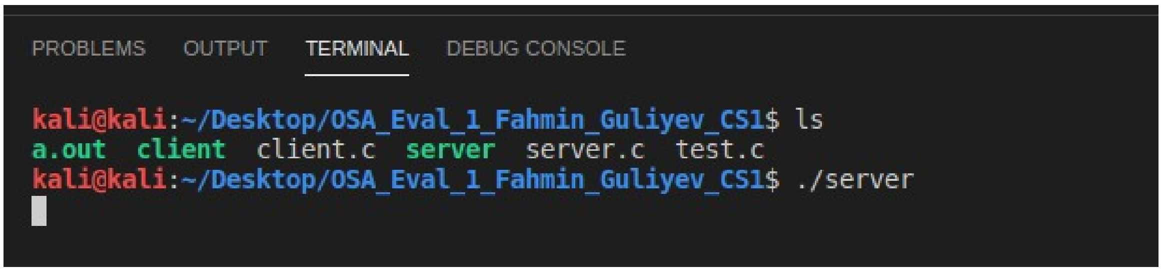
1.3 `client.c` Functions

- `getQueue()` - It finds and accesses the message queue according to our generated key (with *ftok()*) and returns the *queue_id* after the message is gotten.
- `writer()` - Takes a *queue_id* as argument to understand where to write the messages and keeps raeding from user or **stdin** with `read_line()` function until the user sends an EOF signal or simply presses Ctrl+D buttons simultaneously.After user does this, again it sends an empty message with the size of 0 bytes.
- `read_line()` takes an address of text sent with a char array to assign the user input and another address to express the priority chosen by user. "n" is the maximum number of bytes to be read from the user. If there is no input by user, function returns **EOF**.`fgets()` - takes the new line character as the input as well so we have to change this to null terminator to tell that it's the end of the string.

1.4 Demo - Running the Code

So in this program Client sends messages and Server takes these messages and process (handle) them in a way that all the messages get printed back in server with respect to their priority (*mentioned above*).

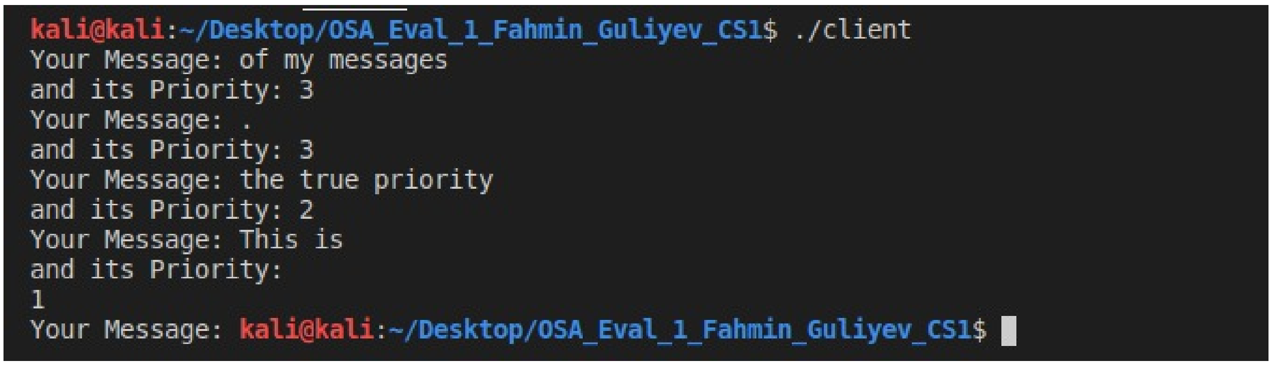
- First i compile and now run the `server.c` program.



```
kali@kali:~/Desktop/OSA_Eval_1_Fahmin_Guliyev_CS1$ ls
a.out  client  client.c  server  server.c  test.c
kali@kali:~/Desktop/OSA_Eval_1_Fahmin_Guliyev_CS1$ ./server
█
```

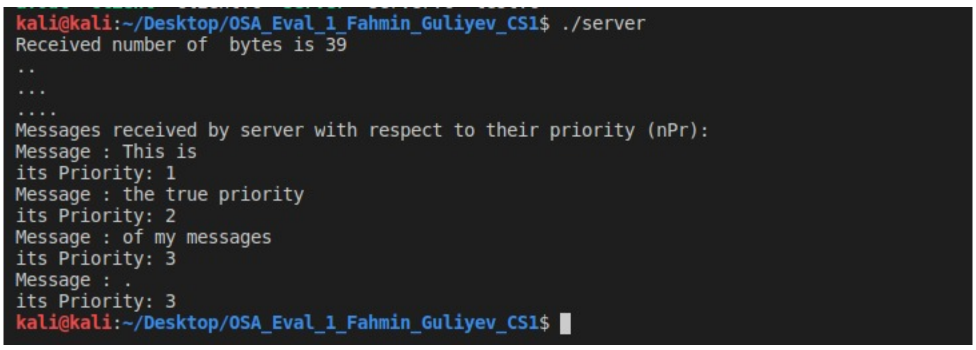
  As you see there is a blinking cursor which indicates that the server is running at the moment.

- Now i compile and run the `client.c` program to start sending messages.



```
kali@kali:~/Desktop/OSA_Eval_1_Fahmin_Guliyev_CS1$ ./client
Your Message: of my messages
and its Priority: 3
Your Message: .
and its Priority: 3
Your Message: the true priority
and its Priority: 2
Your Message: This is
and its Priority:
1
Your Message: kali@kali:~/Desktop/OSA_Eval_1_Fahmin_Guliyev_CS1$ █
```

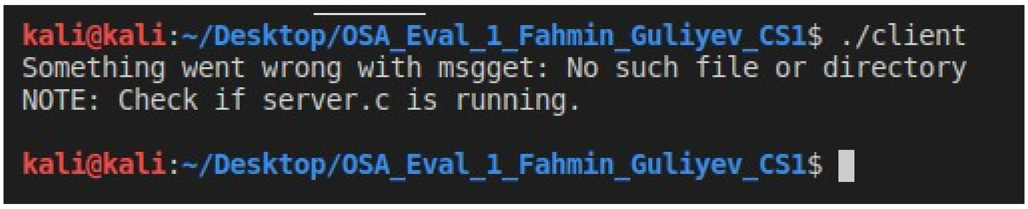
- After the messages are sent, let's take a look at the console where `server.c` is still running at.



```
kali@kali:~/Desktop/OSA_Eval_1_Fahmin_Guliyev_CS1$ ./server
Received number of  bytes is 39
..
...
....
Messages received by server with respect to their priority (nPr):
Message : This is
its Priority: 1
Message : the true priority
its Priority: 2
Message : of my messages
its Priority: 3
Message : .
its Priority: 3
kali@kali:~/Desktop/OSA_Eval_1_Fahmin_Guliyev_CS1$ █
```

As we see the program is working and server didn't just return the sent messages the same way client sent them, it handled them with respect to their priorities and printed back in an ordered way.

- Additionally, if we start(run) the `client.c` program without having `server.c` already running,



```
kali@kali:~/Desktop/OSA_Eval_1_Fahmin_Guliyev_CS1$ ./client
Something went wrong with msgget: No such file or directory
NOTE: Check if server.c is running.

kali@kali:~/Desktop/OSA_Eval_1_Fahmin_Guliyev_CS1$ 
```

It'll just say that something's wrong or the server may not be running.

2 Exercise 2

2.1 First Part

```
        struct listelem *head;

        void insert_list (struct listelem *e)
        {
            e->next = head;
            head = e;
        }
```

The problem here is that when 2 thread tries to add a new element to the head, the 2 processes will conflict due to a very very small delay of one of these 2 threads.If we take a look at the below example:

- Before any operations, let's assume an integer 20 points to the 30 and head points to the 20 as the list has 2 items only.
- Then set the program to execute both threads at the same time, telling one to "let head point to the new integer of 30" and another the same thing with a different number, like **50**.
- In this scenario, due to that small delay i mentioned earlier, if let's say first thread did the job a bit late than the second, the *head* will be pointing to **30**.
- That turns out second thread's job is not considered here and it counts as a loss.

2.2 Second Part

```
        double bank_account;
        void receive (double amount) {
            bank_account += amount ;
        }
```

The same prooblem occurs here as well, when 2 threads tries to access the *bank_account* variable, they both will see the value as-is.But when they increment the account by the amount, whichever thread did the job last, the *bank_account* will be incremented by that *amount* which this very thread incremented with.So in short, it is dangerous to have 2 threads do this job because they will do it asynchronously which is not wished.