Problem 2 Solution Description:

My program has 6 files (3 p1 and 3 p2) and the description is as follows:

Generating Random Strings (Common for all P1):

I have taken a 2d character array named buffer of size 50 * 10 and I am generating random strings of size 4.

Using nested loops on the range (i -> 0 to 49) and (j-> 0 to 3), I have assigned a random character value using the character present at a random index of the following string:
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"

Finally, the last character (i.e 4th index) in each string is assigned as '\0' to mark the end of the string.

Message Queues:

For passing the strings using message queues I have defined a struct mesg_buffer which is similar to the default struct msgbuf but the struct mesg_buffer contains a long mess_type and inside contains another structure data's variable that is the info that needs to be passed by the program.Struct info contains a string and int variable which corresponds to the random strings and its id respectively.

```
struct mesg_buffer {
    long mesg_type;
    struct data{
        char mesg_text[MAX];
        int id;
    } info;
} message1,message2;
```

Inside p1:

P1 first generates the random strings as explained above, Then using the function ftok() it generates a unique id which is further passed on to the function msgget() which returns an integer stored inside msgid().

Then there is a while loop with the variable i that starts from 0 and ends whenever i >= 50, inside the loop there is a another loop with variable j which goes from j= i -> i+5 and sends the message using the function msgsnd() with arguments as msgid, message (of type struct mesg_buffer) and sizeof(message).

message is assigned with the value of string and id to be passed to p2.

The mesg_type inside struct is initialized as 1.

After sending 5 messages it waits for the message by p2 which contains the highest id received by p2 using the function msgrcv() with arguments as msgid, message (of type struct mesg_buffer) , sizeof(message) and mesg_type = 2.

After reading the highest id it updates the value of i = max_id + 1

And the process continues till all the strings are sent.

Inside p2:

p2 first uses the function ftok() which generates a unique id which is further passed on to the function msgget() which returns an integer stored inside msgid().

Then there is a while loop with the variable i that starts from 0 and ends whenever i >= 50, inside the loop there is a another loop with variable j which goes from j= i -> i+5 and reads the message by p1 which contains the string and its id using the using the function msgrcv() with arguments as msgid, message (of type struct mesg_buffer) , sizeof(message) and mesg_type = 1.

Simultaneously it calculates the max id received inside a local variable.

After reading a batch of 5 strings it sends the message using the function msgsnd() with arguments as msgid, message (of type struct mesg_buffer) and sizeof(message).

message is assigned with the value of max_id passed on to p1.

The mesg_type inside struct is initialized as 2.

And the process continues till all the strings are received.

FIFOs:

For passing the strings using message queues I have defined a struct data which contains two fields, a string and int variable which corresponds to the random strings and its id respectively.

```
struct data{

    char mesg_text[MAX];

    int id;

} info;
```

In this set of programs, we have to FIFOs to achieve IPC.

Inside p1:

mkfifo() system call is used to create a FIFO file with the specified pathname and mode (0666)

It then generates a random array of strings as described above.

Then there is a while loop with the variable i that starts from 0 and ends whenever i >= 50.

Which opens the FIFO using open() system call in only write-only mode and further inside the loop there is an another loop with variable j which goes from j = i -> i+5 and sends the message first using the write() system call with the fd returned by open() system call as the argument and the message that contains string and id.

Now, it closes the FIFO using the close() system call.

To receive the message from p1 it again opens the FIFO using open() system call but in read-only mode.

Now, it waits for the message by p2 which contains the highest id received by p2 using the read() system call.

After reading the highest id it updates the value of i = max_id + 1

And the process continues till all the strings are sent.

Inside p2:

Similar to p1 mkfifo() system call is used to create a FIFO file with the specified pathname and mode (0666)

Then there is a while loop with the variable i that starts from 0 and ends whenever i >= 50,

Which opens the FIFO using open() system call in only read-only mode and further inside the loop there is an another loop with variable j which goes from j= i -> i+5 and reads the message by p1 which contains the string and its id using the read() system call with the fd returned by open() system call as the argument; Simultaneously it calculates the max id received inside a local variable.

Now, it closes the FIFO using the close() system call.

To send the message to p1 it again opens the FIFO using open() system call but in write-only mode.

Now it sends the max_id received using write system call();

And the process continues till all the strings are recieved.

Socket Programming:

For passing the strings using message queues I have defined a struct data which contains two fields, a string and int variable which corresponds to the random strings and its id respectively.

```
struct data{

    char mesg_text[MAX];

    int id;

} info;
```

In this set of programs, we have to use UNIX DOMAIN SOCKETS to achieve IPC.

Inside p1:

P2 is acting as the client socket .

Similar to the server it intializes an strict sockaddr_un with same parameters.

After which it uses the connect() system call to initiate the connection with the server socket;

It then generates a random array of strings as described above.

Then there is a while loop with the variable i that starts from 0 and ends whenever i >= 50, inside the loop there is a another loop with variable j which goes from j= i -> i+5 and sends the message using the write() system call with the data-socket as the argument and the message that contains string and id.

After sending 5 messages it waits for the message by p2 which contains the highest id received by p2 using the read() system call with arguments as data_socket.

After reading the highest id it updates the value of i = max_id + 1

And the process continues till all the strings are sent.

Finally the socket is closed by the client using close().

Inside p2:

P2 is acting as the server socket , it initializes the socket using the system call socket() with arguements AF_UNIX and SOCK_STREAM

Variable of type struct sockaddr_un is intialized which have two values sa_family_t which is initialized as AF_UNIX and other variable contains the pathname for a socket i.e. SOCKET in this case.

Now, the bind() system call is used to assign address to the socket intialized before.

listen() system call is used so that it will accept the connection request from p1 program which is acting as a client server.

accept() system call is finally made to accept the client server request and it returns an integer stores inside variable data_socket.

Then there is a while loop with the variable i that starts from 0 and ends whenever i >= 50, inside the loop there is a another loop with variable j which goes from j= i -> i+5 and reads the message by p1 which contains the string and its id using the using the read() system call with data_socket as its argument; Simultaneously it calculates the max id received inside a local variable.

After reading a batch of 5 strings it sends the message using the write() system call with arguments as data_socket and max_id.

After all the strings are received finally the socket is closed using close() function.

Nakul Thureja

2020528