OS Assignment 3

IPC

Document v1.1 (02/11/2021,20:10)

Objective

To learn IPC using:

- Shared files
- Pipes (named, unnamed)
- Message queues
- Sockets
- Signals

On Linux.

Requirements for all designs

Terminal output format

Each message will have the prefix format given below:
[PID:(Parent or Child)][HH:MM:SS.NNN][USER_NAME]
N is milliseconds (see)($ date +'%H:%M:%S:%3N').
If the message is a report message (not a chat message), it will be enclosed by parentheses.
Example:
[1342:P][16:04:57:464][Server] (Server has started.)
[1342:P][16:04:57:512][Server] (Creating child.)
[1342:P][16:04:57:964][Server] (Child has been created.)

Actions to be reported to terminal

- When the process state changes, report it:
  - Created
  - Going to sleep
  - Woke up
  - Exiting
- In each cycle of while or for loops:
  - E.g: [1342:C][16:04:57:964][Ahmet] (Trying to read message. For loop i is 5)
- Trying to send or receive message/signal etc.
  - E.g: [1342:C][16:04:57:964][Ahmet] (Trying to send message:”Hello Kevser, How are you?” to PID:4352.)
- Others
  - E.g: [1342:C][16:04:58:845][Kasim] (Client name is set to “Kasim”)

Client names

Names of the clients will be asked from the users.
E.g.:
[1342:C][16:04:57:964][] (Client has been started.)

[1342:C][16:04:58:845][] Please enter your name:
Kasim
[1342:C][16:04:58:845][Kasim] (Client name is set to “Kasim”)

Source and Executable files locations

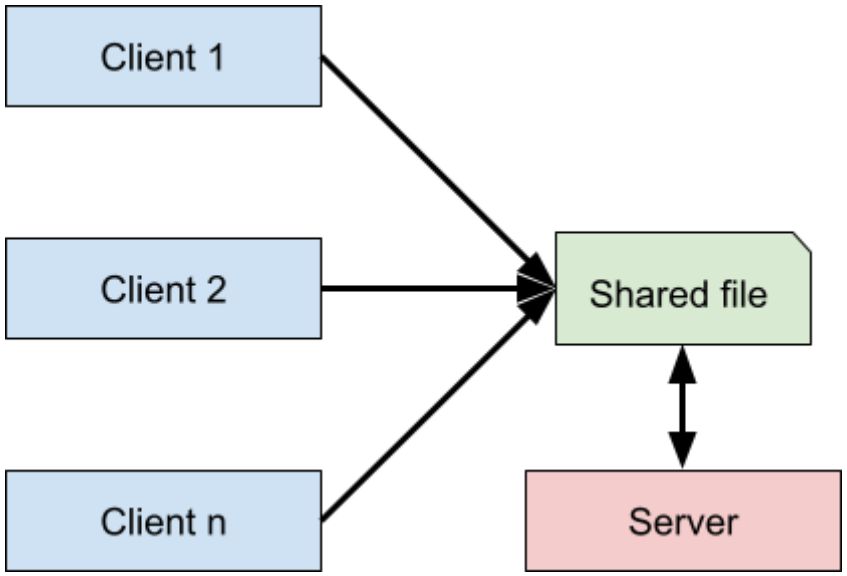
- Each answer must be in its own folder.
- You need to create your own Makefiles for the answers.
- The main Makefile in the main folder should compile all of your codes

Coding

- The code must be well documented
- The code style must be coherent.

Chat Room

First Design: Server + Clients , Shared File

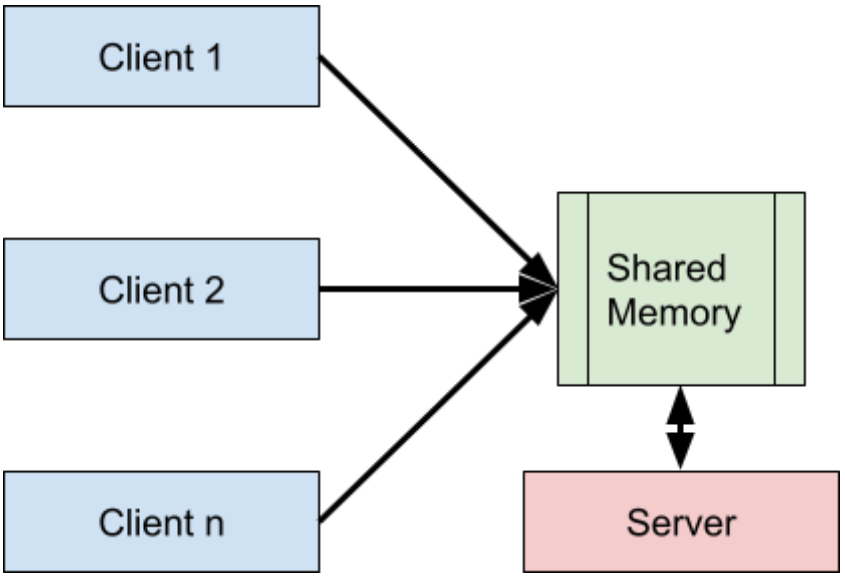


- There will be a shared file. All communications will be carried out on this file.
- Clients can write to the file, server can read and write to the file.
- We can create as many client as we want. There will be only one server.
- Chat outputs will only be visible on the server’s terminal. Client terminals will be used for only inputs.
- **The file will be locked and unlocked before and after read/write actions at all times!**

A possible test scenario that TA will carry out

- Open three different terminals.
    - Terminal 1: Run the server
    - Terminal 2: run the client and give a name when asked
    - Terminal 3: run the client and give a name when asked
- Enter some text to client 1 (terminal2) and check if the message is shown in the chat room (server’s terminal)
- Do the same to client 2.
- The code will be checked to see if the file is properly locked and unlocked.

Second Design: Server + Clients , Shared Memory

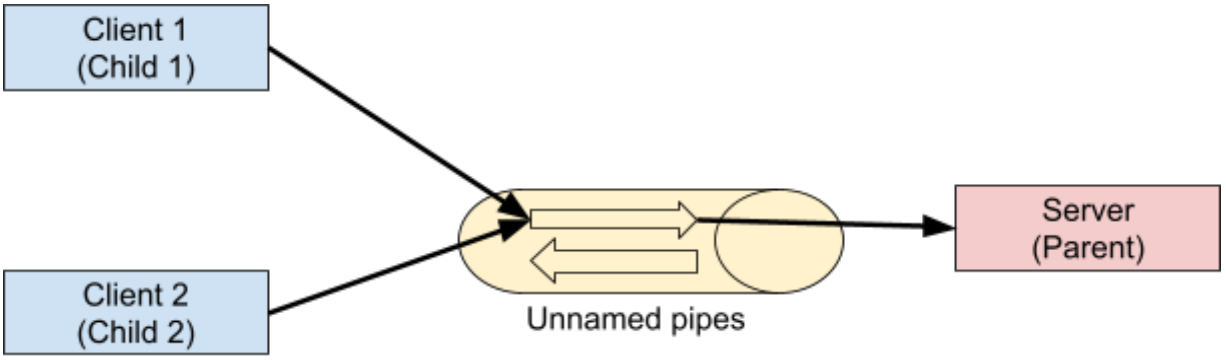


- There will be a shared memory. All communications will be carried out on this memory.
- Clients can write to the memory, server can read and write to the memory.
- We can create as many client as we want. There will be only one server.
- Chat outputs will only be visible on the server's terminal. Client terminals will be used for only inputs.
- **The memory will be locked and unlocked using semaphores before and after read/write actions at all times!**
- Typical scenario:
    - There will be two semaphores: one for reader (server) one for writing permissions.
    - Server opens the shared memory and waits for a client to write something into the memory (waits on a semaphore (read_semaphore which is initially 0))
    - Client
        - gets the message from the user,
        - waits on a write_semaphor (initially 1),
        - Write the message on the memory,
        - Releases write_semaphore,
        - Increments read semaphore,
    - Server
        - will wake up upon the read_semaphore being 1,
        - Read the memory
        - Puts the message on the chatroom (server terminal)

A possible test scenario that TA will carry out

- Open three different terminals.
    - Terminal 1: Run the server
    - Terminal 2: run the client and give a name when asked
    - Terminal 3: run the client and give a name when asked
- Enter some text to client 1 (terminal2) and check if the message is shown in the chat room (server's terminal)
- Do the same to client 2.
- The code will be checked to see if the memory is properly locked and unlocked, read and write semaphores are used properly.

Third Design: Server + Clients , Unnamed Pipes

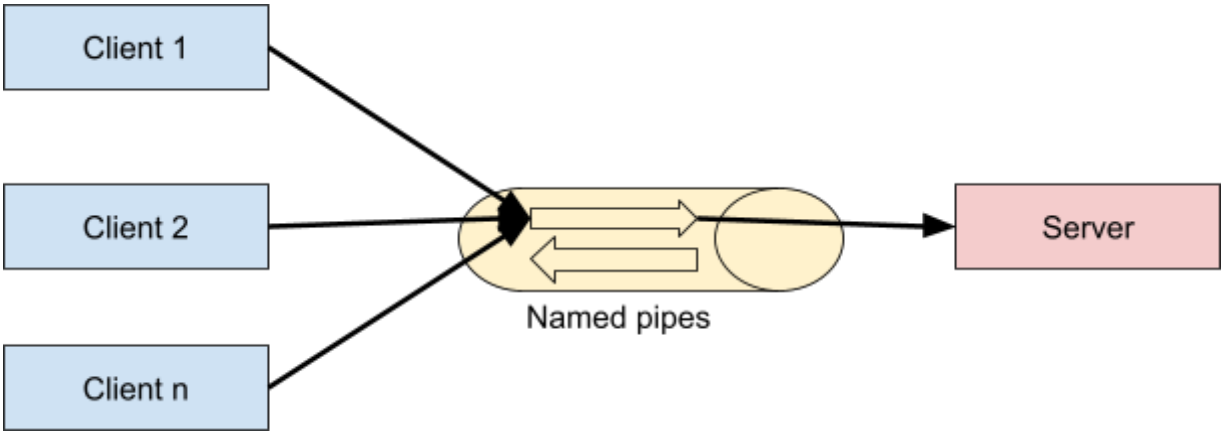


- There will be a unnamed pipeline from clients to the server. All communications will be carried out on this pipeline.
- Clients can write to the pipeline, server can read from the pipeline.
- The server will create two children (clients). So, there will be only two clients. Each client will have its own terminal window (Upon the question from Olamilekan, I have realized that having two separate terminals is a bit triky. You can create a new terminal from each child and communicate. Or, if you find a better solution, please share your idea with us in the discussions. Thanks).
- Chat outputs will only be visible on the server's terminal. Client terminals will be used for only inputs.
- **The pipe will be locked and unlocked using semaphores before and after read/write actions at all times!**
- Typical scenario:
    - There will be two semaphores: one for reader (server) one for writing permissions.
    - Server
        - Creates two childs (with separate terminals)
        - opens the shared memory and waits for a client to write something into the pipe (waits on a semaphore (read_semaphore which is initially 0))
    - Client
        - gets the message from the user,
        - waits on a write_semaphor (initially 1),
        - Write the message to the pipe,
        - Releases write_semaphore,
        - Increments read semaphore,
    - Server
        - will wake up upon the read_semaphore being 1,
        - Read the pipe
        - Puts the message on the chatroom (server terminal)

A possible test scenario that TA will carry out

- Open a terminal.
    - Terminal 1: Run the server. It will create two more terminals with children running on them
    - Terminal 2 (created by client 1): give a name when asked
    - Terminal 3 (created by client 2): give a name when asked
- Enter some text to client 1 (terminal2) and check if the message is shown in the chat room (server's terminal)
- Do the same to client 2.
- The code will be checked to see if the pipe is properly locked and unlocked, read and write semaphores are used properly.

Fourth Design: Server + Clients , Named Pipes



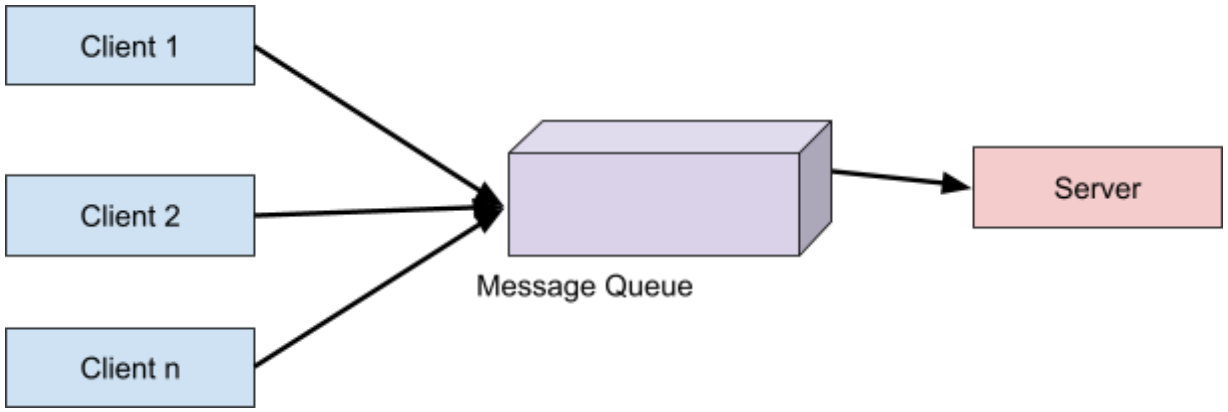
This will be similarly to unnamed pipes question. However, this time there will not be a parent-child relation. We can create as many clients as we want. Server will not create children.

A possible test scenario that TA will carry out

- Open three different terminals.
    - Terminal 1: Run the server
    - Terminal 2: run a client and give a name when asked
    - Terminal 3: run a client and give a name when asked
- Enter some text to client 1 (terminal2) and check if the message is shown in the chat room (server's terminal)
- Do the same to client 2.
- The code will be checked to see if the pipe is properly locked and unlocked, read and write semaphores are used properly. There must be no parent-child relation

---

Fifth Design: Server + Clients , Message Queues



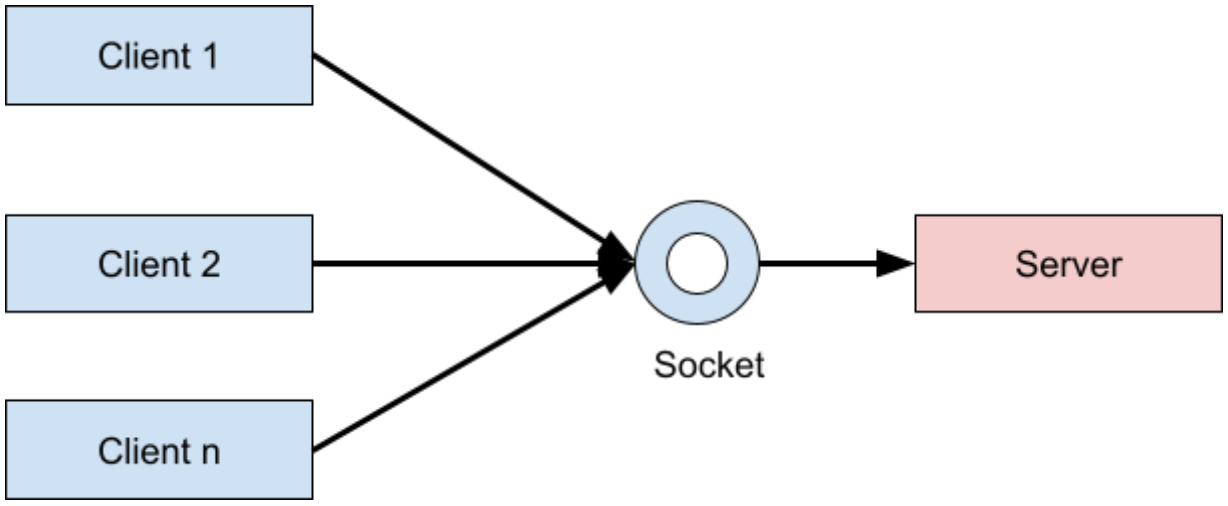
This will be similar to the named pipes question. We can create as many clients as we want. Server will not create children.

- There will be a single semaphore, only for the writers. This time reader, server, would not need to be signalled because it will make a blocking read call on the message queue.

A possible test scenario that TA will carry out

- Open three different terminals.
    - Terminal 1: Run the server
    - Terminal 2: run a client and give a name when asked
    - Terminal 3: run a client and give a name when asked
- Enter some text to client 1 (terminal2) and check if the message is shown in the chat room (server's terminal)
- Do the same to client 2.
- The code will be checked to see if the message queue is properly locked and unlocked, write semaphores are used properly. There must be no parent-child relation.

Sixth Design: Server + Clients , Sockets



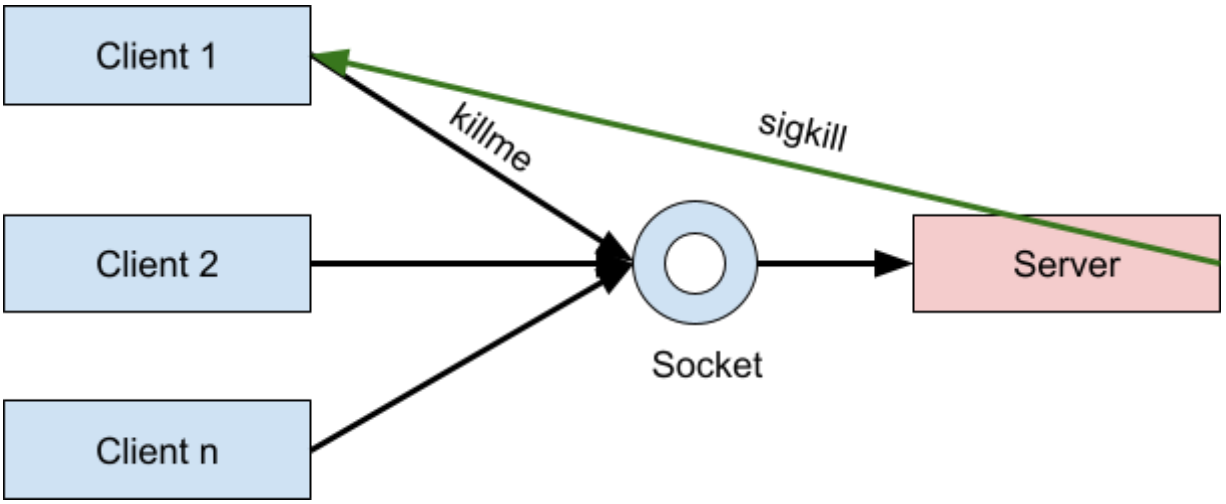
There will be single socket. So the writes will need to use a semaphore.

A possible test scenario that TA will carry out

- Open three different terminals.

    - Terminal 1: Run the server
    - Terminal 2: run a client and give a name when asked
    - Terminal 3: run a client and give a name when asked
- Enter some text to client 1 (terminal2) and check if the message is shown in the chat room (server's terminal)
- Do the same to client 2.
- The code will be checked to see if the socket is properly established, write semaphores are used properly. There must be no parent-child relation.

Seventh Design: Server + Clients , Sockets + Signals



This will be same as the previous (sixth design) except the server should be able to kill the clients. If a client sends a message "killme", the server will send a kill signal to the client and the client will be terminated.

A possible test scenario that TA will carry out

- Open three different terminals.
    - Terminal 1: Run the server
    - Terminal 2: run a client and give a name when asked
    - Terminal 3: run a client and give a name when asked
- Enter some text to client 1 (terminal2) and check if the message is shown in the chat room (server's terminal)
- Do the same to client 2.
- Enter "killme" to client 1 and wait to see if it is terminated by the server.
- The code will be checked to see if the socket is properly established, write semaphores are used properly. There must be no parent-child relation.

Eighth Design: Server and clients, sockets , threads

If not indicated below, you can use any tool we used in this assignment.

- Server and clients will communicate over sockets. However, this time, each client will communicate through different socket. To do so,
    - Server will create a main thread which listens to a predefined port, e.g., 8000,
    - When a client arrives, server will create a new port, i.e., 8501, and a new thread to communicate with the client over 8501. It will let the client know that it will communicate over 8501 from now on.
    - The server waits for a new client on 8000.
- Client connects to the predefined port, i.e. 8000,
- Client receives a new communication port provided by the server, ie. 8501.
- Client communicates through the new port, ie. 8501.

Your Design: Clients only

This is your turn to design a chat room. If not indicated below, you are free to use any tool we used in this assignment.

- There will not be server.
- We should be able to create as many clients as we want
- Each client will use their own terminal for both input and output (Possibly using multiple threads)

Ultimate Design:

Requirements:

- There will be a server and multiple clients.
- Client and server terminals can be used for both input and output.
- Clients and server terminals can run commands.
- Server commands and samples:
    - createroom
      E.g.,
      $ createroom foods
    - kill clientt_name
      E.g., the following kills the client with name Ahmet
      $ kill Ahmet
- Client commands:
    - joinroom
      E.g,
      $ joinroom foods
- Clients can use the terminal for input and see the chat at the same time.
- Clients can only see the messages to the chat room they are in.

Resources

https://opensource.com/sites/default/files/gated-content/inter-process_communication_in_linux.pdf
https://medium.com/@cloudchef/linux-process-states-and-signals-a967d18fab64