

Author:  
Hacı Hasan Savan  
1901042704

Design:

This is the design that we saw in lecture, but I added an extra fifo to handle easily. The first fifo takes command results and the other one is for sending command to server.

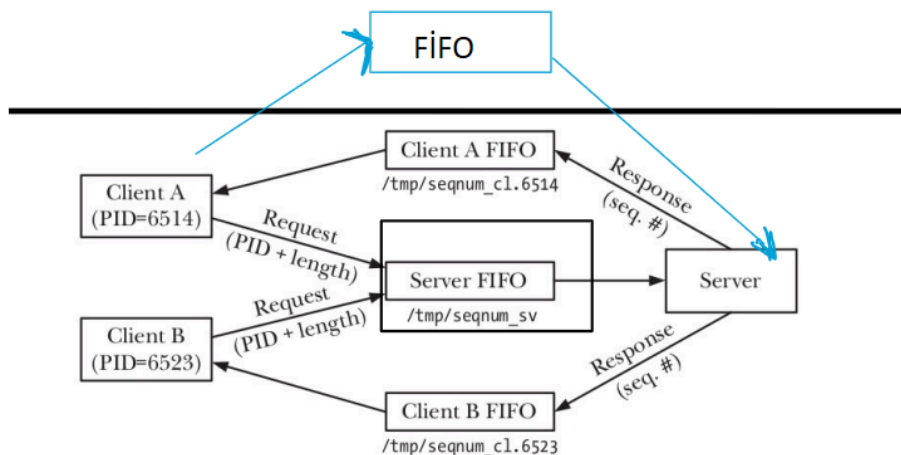


Figure 44-6: Using FIFOs in a single-server, multiple-client application

The file system server-client program is structured around a main process that oversees multiple subprocesses. This main process is responsible for managing client connections and allocating subprocesses to serve them. It acts as the parent process to all other subprocesses, handling tasks such as adding new clients to a queue and monitoring the number of active clients.

Client connection requests are received through a FIFO (named pipe) with the format "serverFifo". The main process places these requests into a waiting queue for processing. If a client requests to "connect", it is added to the queue. If a client requests to "tryconnect", the server checks if there is space available for the client. If there is no space, the client terminates.

Meanwhile, subprocesses (subservers) are continuously created and terminated as needed to serve individual clients. These subservers handle the actual communication and file operations with the clients.

System workflow is like that:

**serverSide:**

create serverFifo  
create signal handlers  
create semaphores

wait for connections

Determine what is request  
Do the command's job  
Send the result via cl\_pid (in child)  
go to "wait for connections"

**clientSide:**

send connection req to server  
open clientFifo (cl\_pid) to take responses  
open clientFifoReq (cl\_req\_pid) to send command  
Send Command to server  
wait for the command result and write  
.  
.  
go to "Send Command to server"

**Synchronization:**

To maintain synchronization during connections and file operations in the server-client system, named semaphores are employed. These semaphores ensure exclusive access to critical sections of code, preventing simultaneous access by multiple processes.

When connecting to the server, a semaphore called "`serverFifoSemaphore`" is utilized. This semaphore is initialized with a value of 1, indicating availability. Before writing a connection request to the server FIFO, the client waits on "`serverFifoSemaphore`" to lock it. Once acquired, the client proceeds with the write operation and releases the semaphore afterward.

Communication between clients and subservers occurs via blocking FIFOs, ensuring sequential message exchange and preventing conflicts. This synchronous communication model enhances system reliability by maintaining data integrity. On the client side we have no semaphore; they are all just on the server side.

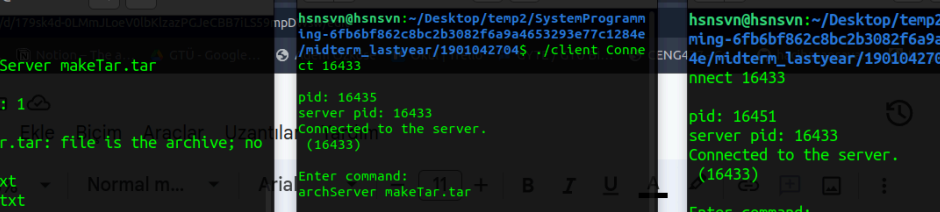
When adding logs, a named semaphore is used to synchronize access to the log file. Before writing, the function waits on the semaphore associated with the log file. After writing, the semaphore is released to allow other processes access.

During file operations, semaphores regulate access to critical code sections, ensuring only one process executes a particular operation at a time. Shared memory structures track file access and manage coordination between processes, enhancing system reliability and performance.

operations:

```
hsnsvn@hsnsvn... Code hsnsvn@hsnsvn...
ming-6fb6bf862c8bc2b3082f6a9a4653293e77c128
4e/mldterm_lastyear/19010427045 ./server he do
Server Started PID 16159...
Waiting for clients... while (1) {
    Client PID 16189 connected as client1
    Client PID 16200 connected as client2
    Client PID 16213 connected as client3
    ComingReq: 108 // printf("BUFResult:\n");
    pId: 16189 // fflush(stdout);
    seqLen: 0 // if (readBy == clientFd, &req, sizeof(req))
    comment: list // printf("%d\n", req.a.out);
    connectOption: 1 // fflush(stdout);
    ComingReq: 113 // if (readBy == Enter command:
    pId: 16200 // help, list, readF, writeT, upload, downlo
    seqLen: 0 // d, quit, killServer
    comment: help Result: list basklar
    connectOption: 1 Available comments are:
    printReq(&req); // help, list, readF, writeT, upload, downlo
    struct response resp; // d, quit, killServer
    char tokens[MAX_TOKENS][M // Enter command:
    int tokenCount = 0; //
    tokenCount = parseComment //
    Enter command:
    printf("ComingReq: \n");
    printReq(&req);
    struct response resp;
```

killServer



Three screenshots of a Kali Linux terminal window showing the execution of a CTF challenge. The terminal is titled 'hsnsvn@hsnsvn...'. The first screenshot shows the user running 'makeTar.tar' and receiving a 'kill signal from client (16451)'. The second screenshot shows the user running 'killServer' and receiving a 'Killed...' message. The third screenshot shows the user running 'killServer' and receiving a 'Killed...' message.

```

hsnsvn@hsnsvn:~/Desktop/temp2/SystemProgram
pid: 16435
seqLen: 0
comment: archServer makeTar.tar

connectOption: 1
./
tar: ./makeTar.tar: file is the archive; not
t dumped
./aTempFile.txt
./server_log.txt
./client
./deneme.txt
./a.out
Archive created successfully.
ComIngReq:
pid: 16451
seqLen: 0
comment: killServer

connectOption: 1
kill signal from client (16451)..
terminating..
hsnsvn@hsnsvn:~/Desktop/temp2/SystemProgram
ming-6fb6bf862c8bc2b3082f6a9a4653293e77c128
4e/midterm_lastyear/1901042704$

hsnsvn@hsnsvn:~/Desktop/temp2/SystemProgram
ling-6fb6bf862c8bc2b3082f6a9a4653293e77c1284e
/midterm_lastyear/1901042704$ ./client Conne
ct 16433

pid: 16435
server pid: 16433
Connected to the server.
(16433)

Enter command:
archServer makeTar.tar

Result:

Enter command:
Killed...
hsnsvn@hsnsvn:~/Desktop/temp2/SystemProgram
ling-6fb6bf862c8bc2b3082f6a9a4653293e77c1284e
/midterm_lastyear/1901042704$

hsnsvn@hsnsvn:~/Desktop/temp2/SystemProgram
ming-6fb6bf862c8bc2b3082f6a9a4653293e77c128
4e/midterm_lastyear/1901042704$ ./client Co
nnect 16433

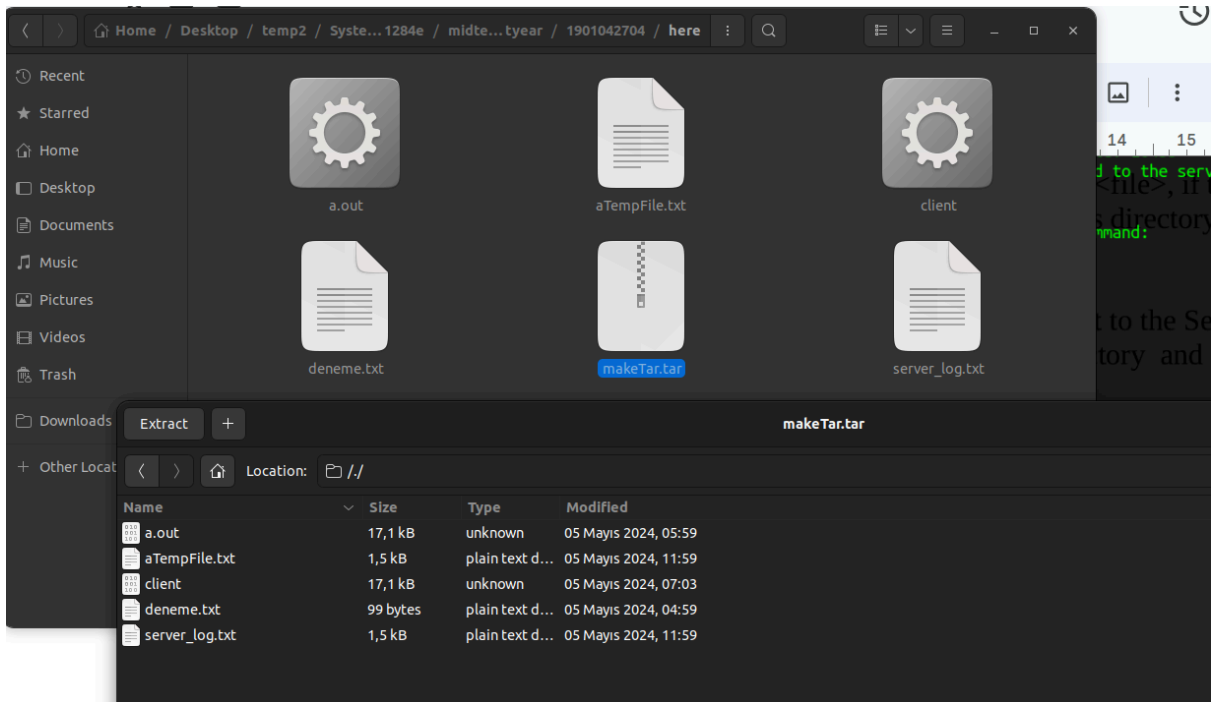
pid: 16451
server pid: 16433
Connected to the server.
(16433)

Enter command:
killServer
Killed...
hsnsvn@hsnsvn:~/Desktop/temp2/SystemProgram
ming-6fb6bf862c8bc2b3082f6a9a4653293e77c128
4e/midterm_lastyear/1901042704$

```

SIGINT





When server is full the newcomers will wait:

```
hsnsvn@hsnsvn:~/Desktop/temp2/SystemProgram
ming-6fb6bf862c8bc2b3082f6a9a4653293e77c128
4e/midterm_lastyear/1901042704$ ./server he
re 2
Server Started PID 16433...HandleClientw
Waiting for clients...
105 {
106 // flush stdout;
107 // client's atomis process burda yeni knut gelmesini bekler
108 int readByt = read(reqClientFd, &req, sizeof(req));
109 // printf("BUF:003\n");
110 // fflush(stdout);
111 if (readByt == -1)
112 {
113 printf(stderr, "Error reading request(Command): discarding\n");
114 continue; // Either partial read or error
115 }
116 printf("ComingReq: \n");
117 printReq(&req);
118 struct response resp;
119 char tokens[MAX_TOKENS][
120 int tokenCount = 0;
121 tokenCount = parseComm
122 req.comment, tokens); // parse command
123 // printf("tokenCount: %d\n", tokenCount);
124 // printf("tokens: %s\n", tokens[0]);
125 // printf("tokens: %s\n", tokens[1]);
126 // printf("tokens: %s\n", tokens[2]);
127
128
129
130
hsnsvn@hsnsvn:~/Desktop/temp2/SystemProgram
ming-6fb6bf862c8bc2b3082f6a9a4653293e77c128
4e/midterm_lastyear/1901042704$ ./client Conne
ct 16433
pid: 16435
server pid: 16433
Connected to the server.
(16433)
Enter command:
hsnsvn@hsnsvn:~/Desktop/temp2/SystemProgram
ming-6fb6bf862c8bc2b3082f6a9a4653293e77c128
4e/midterm_lastyear/1901042704$ ./client Conne
ct 16433
pid: 16451
server pid: 16433
Connected to the server.
(16433)
Enter command:
hsnsvn@hsnsvn:~/Desktop/temp2/SystemProgram
ming-6fb6bf862c8bc2b3082f6a9a4653293e77c128
4e/midterm_lastyear/1901042704$ ./client Conne
ct 16433
pid: 16461
server pid: 16433
Connected to the server.
(16433)
Enter command:
```

logfile:

```
hsnsvn@hsnsvn: ~/Desktop/temp2/SystemProgram...
hsnsvn@hsnsvn:~/Desktop/temp2/SystemProgram...
ming-6fb6bf862c8bc2b3082f6a9a4653293e77c1284e/midterm_lastyear/1901042704$ ./server here 2
Server Started PID 16857...
Waiting for clients...

Client PID 16874 connected as client1
Client PID 16884 connected as client2
Client connection attempt (PID: 16895, server full)
^Chsnsvn@hsnsvn:~/Desktop/temp2/SystemProgram...
ming-6fb6bf862c8bc2b3082f6a9a4653293e77c1284e/midterm_lastyear/1901042704$ ./server here 2
Server Started PID 16996...
Waiting for clients...

Client PID 17004 connected as client1
Client PID 17017 connected as client2
Client connection attempt (PID: 17022, server full)

hsnsvn@hsnsvn:~/Desktop/temp2/SystemProgram...
hsnsvn@hsnsvn:~/Desktop/temp2/SystemProgram...
ming-6fb6bf862c8bc2b3082f6a9a4653293e77c1284e/midterm_lastyear/1901042704$ ./client Connect 16857
pid: 16874
server pid: 16857
Connected to the server.
(16857)

Enter command:
Killed...
hsnsvn@hsnsvn:~/Desktop/temp2/SystemProgram...
ming-6fb6bf862c8bc2b3082f6a9a4653293e77c1284e/midterm_lastyear/1901042704$ ./client Connect 16996
pid: 17004
server pid: 16996
Connected to the server.
(16996)

Enter command:
killServer

hsnsvn@hsnsvn:~/Desktop/temp2/Sy...
hsnsvn@hsnsvn:~/Desktop/t...
hsnsvn@hsnsvn:~/Desktop/t...
hsnsvn@hsnsvn:~/Desktop/temp2/SystemProgramming-6fb6bf862c8bc2b3082f6a9a4653293e77c1284e/midterm_lastyear/1901042704$ ./client Connect 16857
pid: 16895
^Killed...
hsnsvn@hsnsvn:~/Desktop/temp2/SystemProgramming-6fb6bf862c8bc2b3082f6a9a4653293e77c1284e/midterm_lastyear/1901042704$ ./client Connect 16996
pid: 17022
```

server\_log.txt

```
~/Desktop/temp2/SystemProgramming-6fb6bf86...3e77c1284e/midterm_lastyear/1901042704/here
1 Client connected (PID: 17004)
2 Client connected (PID: 17017)
3 Client connection attempt (PID: 17022, server full)
```

it will also write the operations:

```
server_log.txt
~/Desktop/temp2/SystemProgramming-6fb6bf86...3e77c1284e/midterm_lastyear/1901042704/here
1 Client connected (PID: 17004)
2 Client connected (PID: 17017)
3 Client connection attempt (PID: 17022, server full)
4 Client17017 used list Command (PID: 17018)
5 Client17004 used help Command (PID: 17005)
6 Client17004 send killServer (PID: 17005)
```