GIT Department of Computer Engineering

CSE 344 – Spring 2022

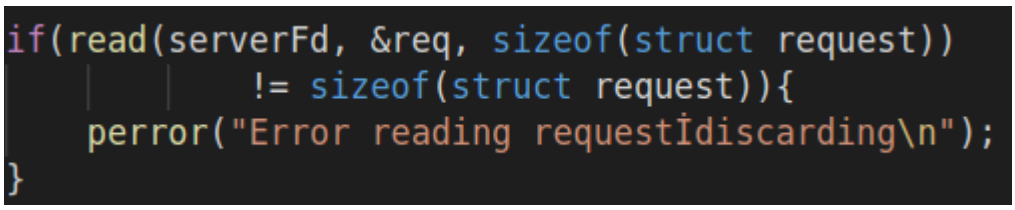
Midterm Project #Report

Okan Torun

1801042662

Communication between Client and Server

In this project, what is requested from us is to deal with the desired situations by establishing a relationship between the clients and the servers. First, I stored the desired properties of the matrix by reading the data in the desired file in the client and handled the matrix situations in the faulty situations. Then I put the matrix and the client's pid into the request struct that I created in the fifo_seqnum. Then I create the client's fifo with mkfifo() so that the client can get data from the fifo. Then I created the fd with the server's fifo address and sent the address of the request I created to this fd address, and I took the first step to deliver the data to the server.Read More Then after I make the necessary fifo expansion in serverY, I read the necessary struct data from the serverFd address in the parent.



Communication between ServerY and its children

After obtaining the arguments in ServerY, I fork as many as the p argument. After this process, I keep the children circulating in the endless loop, so serverY is constantly waiting for a client. Before this operation, I created a pipe array and created a read and write end with the pipe function.Then I sent the data with the write tip of the pipe array.If the child is the read function, we read the data from the parent with the pipe's reading tip, and I calculate the desired matrix operations with the data we read and send the result to the client according to the result.

```
write(filedes[1], &req,  sizeof(struct request));
```

Semaphore

I keep the pool size of Y with the semaphore I created in serverY and increase it every time the data comes in and reduce it after the process is finished. So I check serveri's bussy event.

```
sem_post(sema);
sem_getvalue(sema, &turn2);
```

Communication between ServerY and ServerZ

When ServerY is opened, I perform a fork operation for serverZ and as a result I execve. Before this operation, I create another pipe and use it for communication with serverZ. I put the read and write ends of this pipe with enVar variable a sprintf and send it to serverZ with execv. As a result, if all the children in serverY are busy, I transfer the request to serverZ with the writing tip of this pipe.Later, when the server is opened, I transfer the data from Y to the pipe I created with atoi, and when I want to read data here, I use this pie tip.

Communication between ServerZ and their children

The communication difference in ServerZ is sharedMemory. First, I provide shared memory space in struct type, and then I created a struct to keep the pid and status of the children. Then I create poolSize2 as many children and set their status to 0, which means that these children are suitable. is coming.Here, the key point is SIGSTOP and SIGCONT. With these two signals, I provide communication between parent and child. I initialize the data I read from Y

with shared memory to the child available in the parent with SIGCONT, and the child does the operation with the data it can access and the result is sent to the client. sending it back.

```
filedesZ[0]=atoi(argve[0]);
filedesZ[1]=atoi(argve[1]);
```

```
struct request* shmem = (struct request *)create_shared_memory(128);
```

```
read(filedesZ[0], &req, sizeof(struct request));
memcpy(shmem, &req, sizeof(struct request));
```

```
kill(objChildsZ[i].childsPid, SIGCONT);
objChildsZ[i].status = 1;
```

Daemon Process and Double Instantiation

I created the become_daemon.h library daemon process state and when serverY is opened, it opens in daemon process mode. Then, to prevent double instantiation, I try to write to a file as serverY is opened, if it cannot write to this file, it means that serverY has not been opened before and I am creating this file. I kill the new server that is trying to open.

After running in daemon process mode, if ServerY receives a kill command, I also killed Server Z.

Just type "killall serverY -2" into the terminal.

TEST CASES

1)

```
okan@okan-ABRA-A5-V16-4:~/Desktop$ ./serverY -s pathServerFifo -o pathToLogFile -p 1 -r 2 -t 5
okan@okan-ABRA-A5-V16-4:~/Desktop$
```

```
AcceptedClient PID6217 (path) is submitting is 3x3 matrix
Client PID6217: the matrix is non-invertible, total time 0.013500 seconds, goodbye.
```

```
Server Y (pathToLogFile , p=1, t=5) started
Instantiated server Z
Tue Apr 19 03:16:18 2022
 Worker PID#9145 is handling client PID#9144, matrix size 3x3, pool busy 1/1
Tue Apr 19 03:16:18 2022
 Forwarding request of client PID#9139, to serverZ, matrix size 1x1, pool busy 1/1
Tue Apr 19 03:16:18 2022
 Worker PID#9142 is handling client PID#9146, matrix size 5x5, pool busy 1/2
Tue Apr 19 03:16:19 2022
 Forwarding request of client PID#9139, to serverZ, matrix size 1x1, pool busy 1/1
Tue Apr 19 03:16:19 2022
 Worker PID#9143 is handling client PID#9147, matrix size 5x5, pool busy 2/2
Tue Apr 19 03:16:23 2022
 Worker PID#9145 is handling client PID#9148, matrix size 3x3, pool busy 1/1
Tue Apr 19 03:16:31 2022
 SIGINT received, terminating Z and exiting server Y. Total requests handled: 4, 2 invertible, 2 not. 2 requests were forwarded
```

2)

```
okan@okan-ABRA-A5-V16-4:~/Desktop$ ./serverY -s pathServerFifo -o pathToLogFile
-p 2 -r 2 -t 5
```

```
Server Y (pathToLogFile , p=2, t=5) started
Instantiated server Z
Tue Apr 19 03:14:12 2022
 Worker PID#9094 is handling client PID#9093, matrix size 3x3, pool busy 1/2
Tue Apr 19 03:14:12 2022
 Worker PID#9095 is handling client PID#9096, matrix size 3x3, pool busy 2/2
Tue Apr 19 03:14:13 2022
 Forwarding request of client PID#9088, to serverZ, matrix size 1x1, pool busy 2/2
Tue Apr 19 03:14:13 2022
 Worker PID#9090 is handling client PID#9097, matrix size 5x5, pool busy 1/2
Tue Apr 19 03:14:13 2022
 Forwarding request of client PID#9088, to serverZ, matrix size 1x1, pool busy 2/2
Tue Apr 19 03:14:13 2022
 Worker PID#9091 is handling client PID#9098, matrix size 5x5, pool busy 2/2
Tue Apr 19 03:14:20 2022
 SIGINT received, terminating Z and exiting server Y. Total requests handled: 4, 2 invertible, 2 not. 2 requests were forwarded
```