Systems Programming Midterm Project Report
Mohammad Ashraf Yawar 161044123
April 16, 2022

- HOW TO RUN AND TEST THE PROGRAM WITH DIFFERENT INPUTS?

- You can find the instructions in README.txt in order to run and test the program.

```
-HOW TO RUN THE PROGRAM:
> Run below commands in order:

alias vg='valgrind --leak-check=full -v --track-origins=yes --log-file=vg_logfile.out'

make

*****************************************************************************************************************************************************

- HOW TO TEST THE PROGRAM WITH DIFFERENT INPUTS:

-on terminal1 paste below code:

vg ./serverY -s requests -o logs.log –p 5 -r 5 -t 2

-on terminal2 paste below code:
vg ./client -s requests -o data.csv

*****************************************************************************************************************************************************
```

Implemented Concepts:

- File read, write, lock using syscalls.
- Signal handaling, parent child signal relations.
- Multiple child process, fork, exec family.
- Make files.
- pipes, fifos and their relations and implementations.
- daemon process, log_file generator, dyamic matrix parsing.

Not Impelmented Concepts:
- shared memory

Design Explanation:

- I separated my program into functions using make file where each function does a specific task:

```
void fcntl_syscall_error_print();

void open_syscall_error_print();

void close_syscall_error_print();

void lseek_syscall_error_print();

void mkstemp_syscall_error_print();

void unlink_syscall_error_print();

void read_syscall_error_print();

void write_syscall_error_print();

float frobeniusNorm(float matrix[3][3]);

int power (int x, int y);
```

- All the System-Calls and their possible return error values are checked with detailed errno checks.

- the program start by setting up the serverY and then client's can send requests to the serverY throuth pipes and fifos.

- clients takes a matrix of nxn and send's it to the serverY throuth already handshaked fifo so that serverY can get the request.

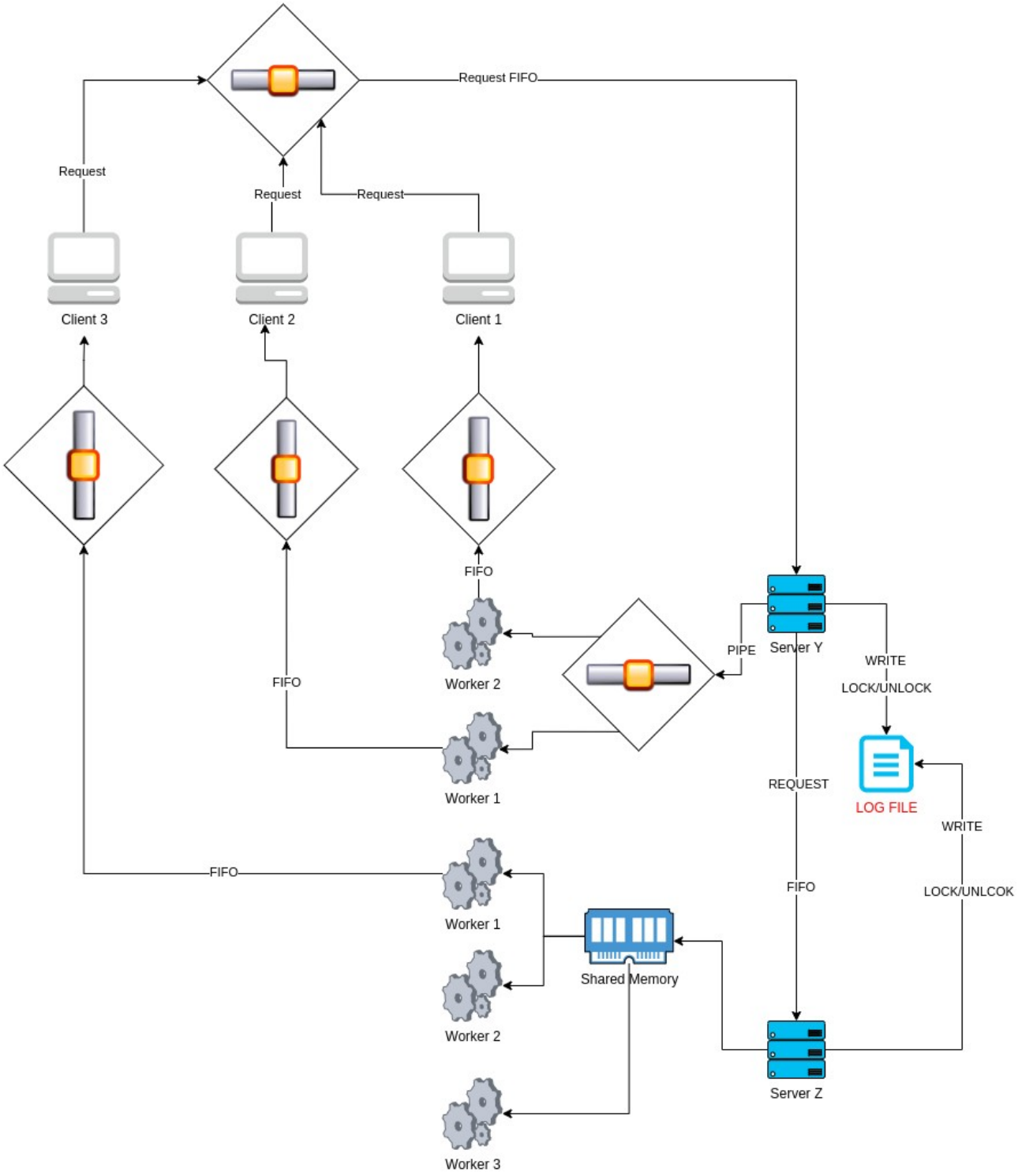
- once before everything serverY instantiates the serverZ and creates's it's children processes, and the then waits for incoming requests, when ever a request arrives, it forwards it's request to it's workers, in my case I was able to implement single worker process mechanism.

-
-below is the design decision for this project:

-worker waits for the request to be send to it's pipe and when ever a request arrives, it sleeps for some specific seconds and parses the string and finds out whether the matrix is invert-able or not and sends the result the client's fifo directly.

- I have taken care of consideration all the system programming rules :).

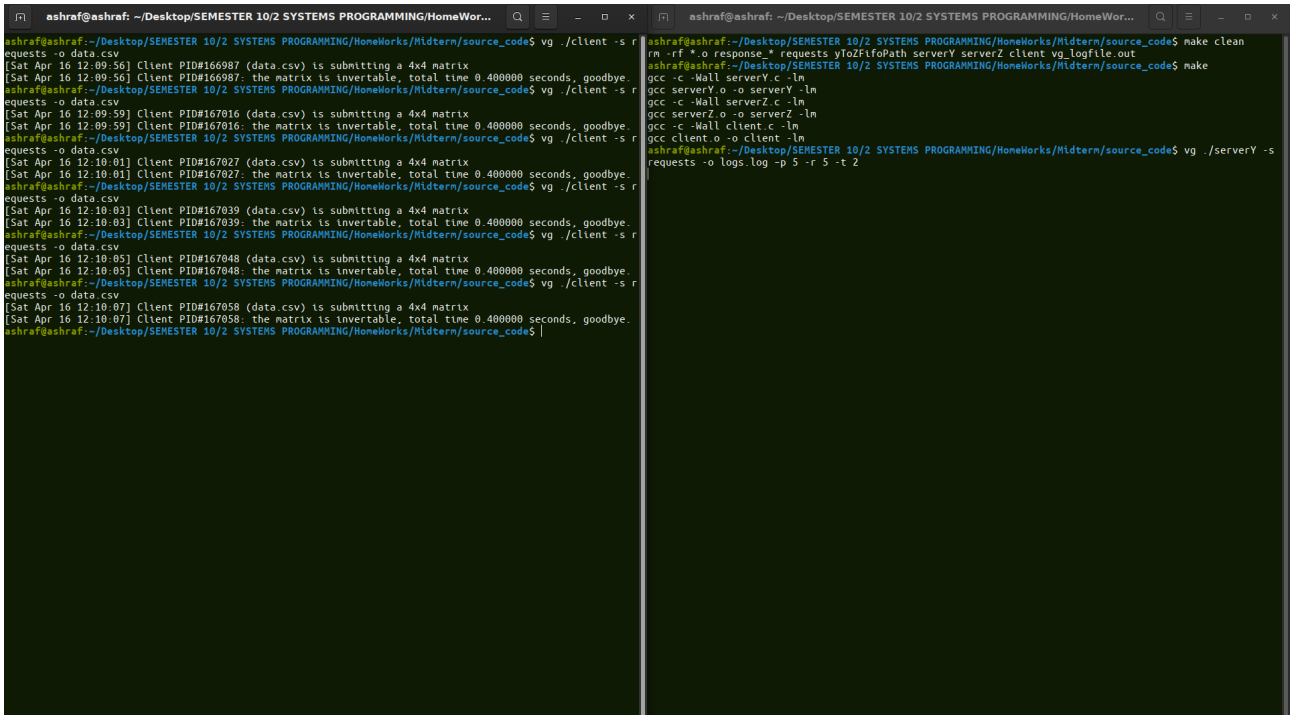
- below is the over all project and design decision of mine:



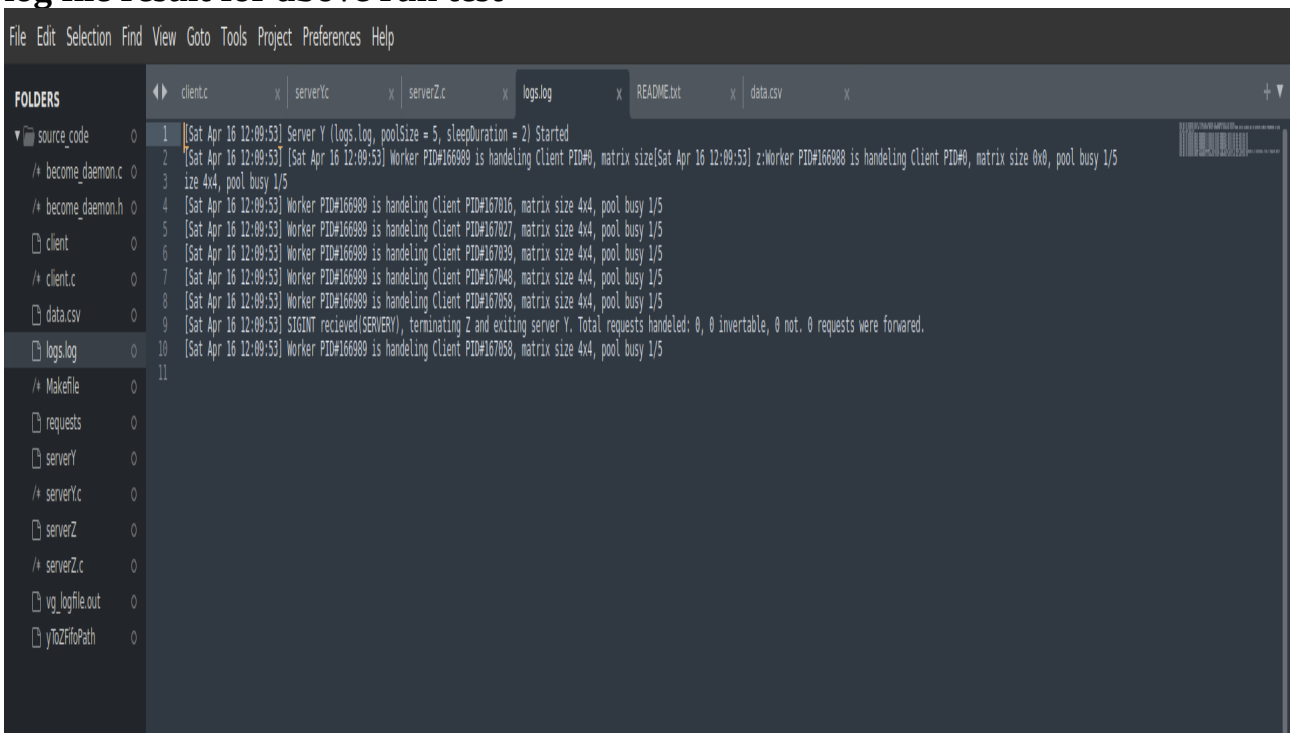
- servery and serverz write their content into the logs.log file and while writing each of the lock the file and then unlock so that other files can also use it.
- I have used lock mechanism in every and each one the read write calls for logs.log file.

- server z is just like server y but it talks through shared memory with it’s workers, I haven’t been able implement this part.

SCREEN SHOTS FROM THE PROGRAMS:



log file result for above run test



```c
struct request{
    pid_t pid;
    int matrixLen;
    char payLoad[BUFFER_SIZE];
};
struct response{
    int isInvertable;
};

sig_atomic_t sigintcaught = 0;

void siginthandler(){
    int esaved = errno;
    sigintcaught = 1;
    errno = esaved;
}

int main(int argc, char **argv){

    // int printBufferPointerLen = 1000;

    char* serverFifoPath = NULL;
    char* dataFilePath = NULL;
    char buf[BUFFER_LIMIT];
    // char* printBufferPointer = (char*) malloc(printBufferPointerLen * sizeof(char));

    int serverFd = 0, clientFd = 0,bytesread = 0,dataFilePathFd = 0;
    struct request req;
    struct response resp;
    struct sigaction newact;
    time_t t;   // not a primitive datatype
    time(&t);

    newact.sa_handler = &siginthandler; /* set the new handler */
    newact.sa_flags = 0;
    sigaction(SIGINT,&newact,NULL);
    if ((sigemptyset(&newact.sa_mask) == -1) || (sigaction(SIGINT,&newact, NULL) == -1)){
        perror("Failed to install SIGINT signal handler");
        _exit(EXIT_FAILURE);
    }

    if (sigintcaught == 1){
        _exit(EXIT_FAILURE);
    }

    // check if the user has entered sufficient arguments.
    if (argc < 5){
        perror("No Sufficient Parameters To Execute The Edit !!!\n");
        perror("Usage: vg ./client -s pathToServerFifo -o pathToDataFile\n");
        return 1;
    }else if (argc > 5){
        perror("Too Much Parameters To Execute The Edit !!!\n");
        perror("Usage: vg ./client -s pathToServerFifo -o pathToDataFile\n");
        return 1;

    }

    //file paths read from terminal
    serverFifoPath = argv[2];
    dataFilePath = argv[4];
    dataFilePathFd = open(dataFilePath,O_RDONLY);
```

- here above, I used structs , they act like json object in http requests.

- I have chcked for suffecient arguments from the user and proved the usage.

- I used signal handler as setting a flag, when ever we receive SIGINT the signal handler will run and set the flag to 1, I check the flag in every important segments of the code and act accrodingly, free up spaces, close files and exit elegantly. I used the same method for servery, serverz and client side.