

# 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 handling, parent child signal relations.
- Multiple child process, fork, exec family.
- Make files.
- pipes, fifos and their relations and implementations.
- daemon process, log\_file generator, dynamic matrix parsing.

#### Not Implemented 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 through pipes and fifos.

- clients takes a matrix of nxn and send's it to the serverY through 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.

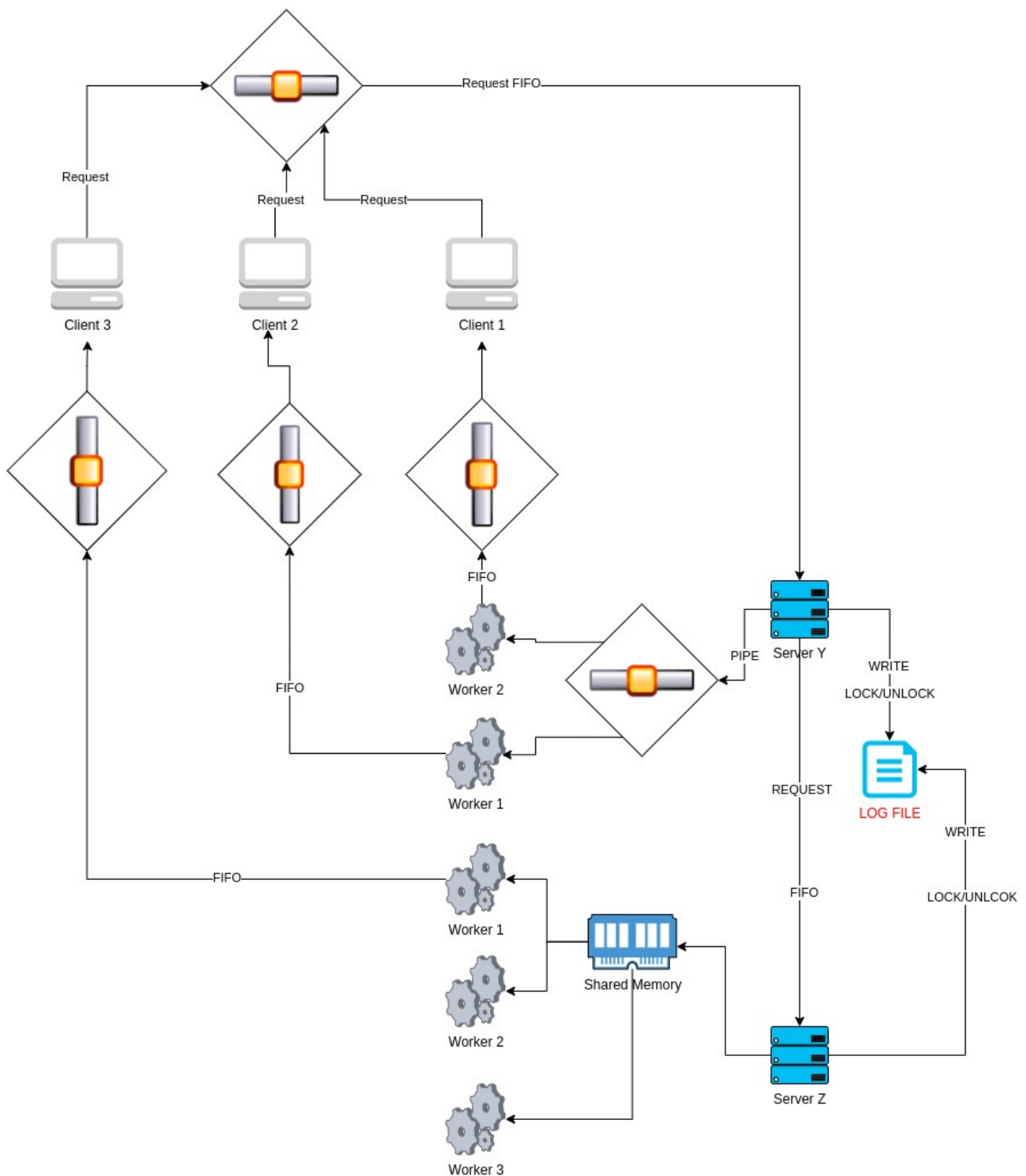
- 

- 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:



- server y and server z 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:

```
ashraf@ashraf: ~/Desktop/SEMESTER 10/2 SYSTEMS PROGRAMMING/HomeWor...
ashraf@ashraf:~/Desktop/SEMESTER 10/2 SYSTEMS PROGRAMMING/HomeWorks/Mldtern/source_code$ vg ./client -s r
requests -o data.csv
[Sat Apr 16 12:09:56] Client PID#166987 (data.csv) is submitting a 4x4 matrix
[Sat Apr 16 12:09:56] Client PID#166987: the matrix is invertable, total time 0.408000 seconds, goodbye.
ashraf@ashraf:~/Desktop/SEMESTER 10/2 SYSTEMS PROGRAMMING/HomeWorks/Mldtern/source_code$ vg ./client -s r
requests -o data.csv
[Sat Apr 16 12:09:59] Client PID#167016 (data.csv) is submitting a 4x4 matrix
[Sat Apr 16 12:09:59] Client PID#167016: the matrix is invertable, total time 0.408000 seconds, goodbye.
ashraf@ashraf:~/Desktop/SEMESTER 10/2 SYSTEMS PROGRAMMING/HomeWorks/Mldtern/source_code$ vg ./client -s r
requests -o data.csv
[Sat Apr 16 12:10:01] Client PID#167027 (data.csv) is submitting a 4x4 matrix
[Sat Apr 16 12:10:01] Client PID#167027: the matrix is invertable, total time 0.408000 seconds, goodbye.
ashraf@ashraf:~/Desktop/SEMESTER 10/2 SYSTEMS PROGRAMMING/HomeWorks/Mldtern/source_code$ vg ./client -s r
requests -o data.csv
[Sat Apr 16 12:10:03] Client PID#167039 (data.csv) is submitting a 4x4 matrix
[Sat Apr 16 12:10:03] Client PID#167039: the matrix is invertable, total time 0.408000 seconds, goodbye.
ashraf@ashraf:~/Desktop/SEMESTER 10/2 SYSTEMS PROGRAMMING/HomeWorks/Mldtern/source_code$ vg ./client -s r
requests -o data.csv
[Sat Apr 16 12:10:05] Client PID#167048 (data.csv) is submitting a 4x4 matrix
[Sat Apr 16 12:10:05] Client PID#167048: the matrix is invertable, total time 0.408000 seconds, goodbye.
ashraf@ashraf:~/Desktop/SEMESTER 10/2 SYSTEMS PROGRAMMING/HomeWorks/Mldtern/source_code$ vg ./client -s r
requests -o data.csv
[Sat Apr 16 12:10:07] Client PID#167058 (data.csv) is submitting a 4x4 matrix
[Sat Apr 16 12:10:07] Client PID#167058: the matrix is invertable, total time 0.408000 seconds, goodbye.
ashraf@ashraf:~/Desktop/SEMESTER 10/2 SYSTEMS PROGRAMMING/HomeWorks/Mldtern/source_code$ |

ashraf@ashraf:~/Desktop/SEMESTER 10/2 SYSTEMS PROGRAMMING/HomeWor...
ashraf@ashraf:~/Desktop/SEMESTER 10/2 SYSTEMS PROGRAMMING/HomeWorks/Mldtern/source_code$ make clean
rm -rf *.o response.* requests yloZfifPath serverY serverZ client vg_logfile.out
ashraf@ashraf:~/Desktop/SEMESTER 10/2 SYSTEMS PROGRAMMING/HomeWorks/Mldtern/source_code$ make
gcc -c -Wall serverY.c -ln
gcc serverY.o -o serverY -ln
gcc -c -Wall serverZ.c -ln
gcc serverZ.o -o serverZ -ln
gcc -c -Wall client.c -ln
gcc client.o -o client -ln
ashraf@ashraf:~/Desktop/SEMESTER 10/2 SYSTEMS PROGRAMMING/HomeWorks/Mldtern/source_code$ vg ./serverY -s
requests -o logs.log -p 5 -r 5 -t 2
```

## log file result for above run test

```
File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS client.c x serverY.c x serverZ.c x logs.log x README.txt x data.csv x
source_code o
  +/ become_daemon.c o
  +/ become_daemon.h o
  client o
  +/ client.c o
  data.csv o
  logs.log o
  +/ Makefile o
  requests o
  serverY o
  +/ serverY.c o
  serverZ o
  +/ serverZ.c o
  vg_logfile.out o
  yloZfifPath o

1 [Sat Apr 16 12:09:53] Server Y (logs.log, poolSize = 5, sleepDuration = 2) Started
2 [Sat Apr 16 12:09:53] [Sat Apr 16 12:09:53] Worker PID#166989 is handling Client PID#0, matrix size[Sat Apr 16 12:09:53] z:Worker PID#166988 is handling Client PID#0, matrix size 0x0, pool busy 1/5
3 size 4x4, pool busy 1/5
4 [Sat Apr 16 12:09:53] Worker PID#166989 is handling Client PID#167016, matrix size 4x4, pool busy 1/5
5 [Sat Apr 16 12:09:53] Worker PID#166989 is handling Client PID#167027, matrix size 4x4, pool busy 1/5
6 [Sat Apr 16 12:09:53] Worker PID#166989 is handling Client PID#167039, matrix size 4x4, pool busy 1/5
7 [Sat Apr 16 12:09:53] Worker PID#166989 is handling Client PID#167048, matrix size 4x4, pool busy 1/5
8 [Sat Apr 16 12:09:53] Worker PID#166989 is handling Client PID#167058, matrix size 4x4, pool busy 1/5
9 [Sat Apr 16 12:09:53] SIGINT recieved(SERVERY), terminating Z and exiting server Y. Total requests handled: 0, 0 invertable, 0 not. 0 requests were forwarded.
10 [Sat Apr 16 12:09:53] Worker PID#166989 is handling Client PID#167058, matrix size 4x4, pool busy 1/5
11
```

```

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

sig_atomic_t sigintcaught = 0;

void sighandler(){
    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 = &sighandler; /* 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 checked for sufficient 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 accordingly, free up spaces, close files and exit elegantly. I used the same method for server, serverz and client side.