Gebze Technical University
Computer Engineering
CSE344-Midterm
Project

Mustafa Gurler
171044034

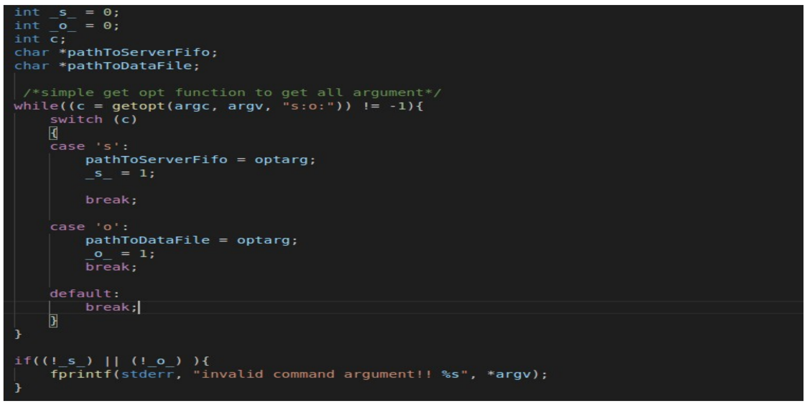
1 Problem Definition

In this project, there is multiple clients, one server Y and one server Z has been created. Clients who are sending request to ServerY through by FIFO, ServerY forwards the requests to serverY workers through PIPE. If workers are not available new server which is serverZ instantiated, ServerY forwards the all requests to serverZ through PIPE. ServerZ communicate with its workers through shared memory.

2 Design and Solution

At the beginning I created clientX file. It has a only two job reading and writing. It does not have a lot of tricky part. ServerY has a lot of options , a lot of choices that needs to be connected. And ServerZ needs to solve the problem that called producer consumer fashioned.

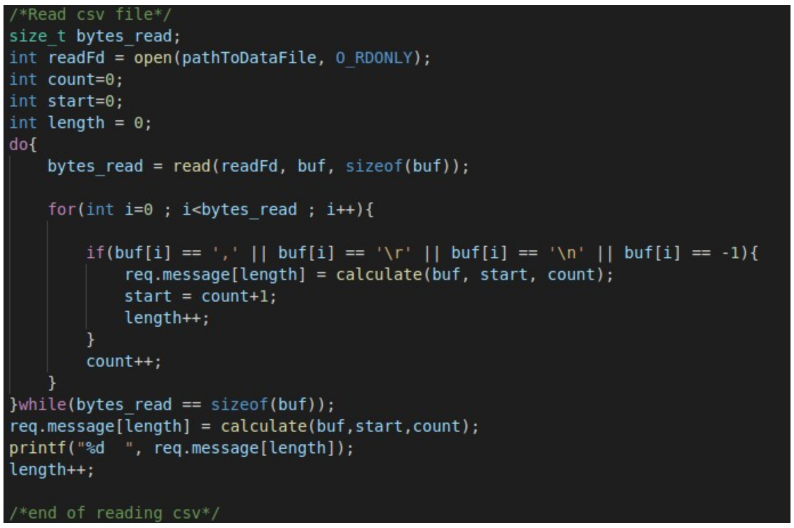
2.1 Reading arguments



```c
int _s_ = 0;
int _o_ = 0;
int c;
char *pathToServerFifo;
char *pathToDataFile;

 /*simple get opt function to get all argument*/
while((c = getopt(argc, argv, "s:o:")) != -1){
    switch (c)
    {
    case 's':
        pathToServerFifo = optarg;
        _s_ = 1;

        break;

    case 'o':
        pathToDataFile = optarg;
        _o_ = 1;
        break;

    default:
        break;
    }
}

if((!_s_) || (!_o_) ){
    fprintf(stderr, "invalid command argument!! %s", *argv);
}
```

I used just simple getopt function. I got all the requirements

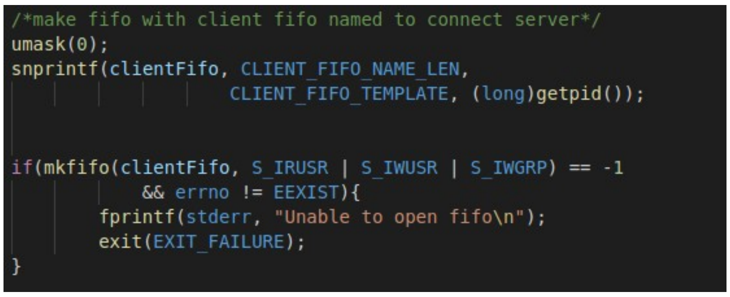
2.2 Reading csv file



```c
/*Read csv file*/
size_t bytes_read;
int readFd = open(pathToDataFile, O_RDONLY);
int count=0;
int start=0;
int length = 0;
do{
    bytes_read = read(readFd, buf, sizeof(buf));

    for(int i=0 ; i<bytes_read ; i++){

        if(buf[i] == ',' || buf[i] == '\r' || buf[i] == '\n' || buf[i] == -1){
            req.message[length] = calculate(buf, start, count);
            start = count+1;
            length++;
        }
        count++;
    }
}while(bytes_read == sizeof(buf));
req.message[length] = calculate(buf,start,count);
printf("%d  ", req.message[length]);
length++;

/*end of reading csv*/
```

I used my own function to get first number in the character array. It reads until ‘,’ comma and returns the number.
I know that matrix only can be square matrix and that is why I checked the not square matrix error and return failure.
Length was counted and fixed size array has been created.

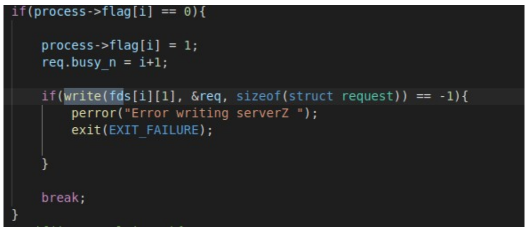
2.3 Client-ServerY Communication



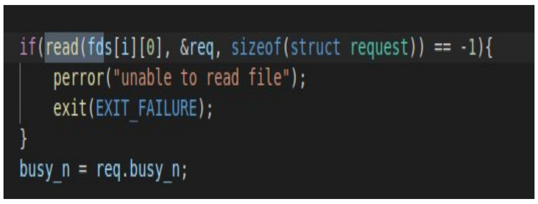
```c
/*make fifo with client fifo named to connect server*/
umask(0);
snprintf(clientFifo, CLIENT_FIFO_NAME_LEN,
                    CLIENT_FIFO_TEMPLATE, (long)getpid());


if(mkfifo(clientFifo, S_IRUSR | S_IWUSR | S_IWGRP) == -1
            && errno != EEXIST){
        fprintf(stderr, "Unable to open fifo\n");
        exit(EXIT_FAILURE);
}
```

Creating new requests has been added to request structure, responses also have been added the response structure. They have been communicated with each other via FIFO structure. I created one FIFO for sending request, also one FIFO for getting response. Getting response were tricky because we were dealing with multiple client. We don't know which one is going to use if we use one named FIFO for every client-server communication. So I create my FIFO name adding PID of client server and sending request with PID made my job easier.

2.3.1 ServerY-Workers Communication



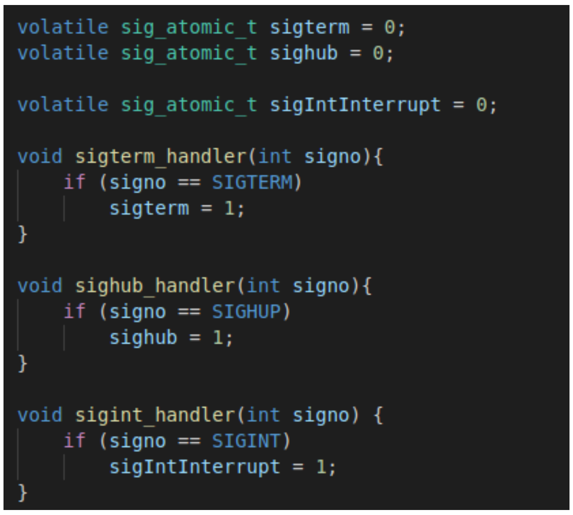
```c
if(process->flag[i] == 0){

    process->flag[i] = 1;
    req.busy_n = i+1;

    if(write(fds[i][1], &req, sizeof(struct request)) == -1){
        perror("Error writing serverZ ");
        exit(EXIT_FAILURE);

    }

    break;
}
```



```c
if(read(fds[i][0], &req, sizeof(struct request)) == -1){
    perror("unable to read file");
    exit(EXIT_FAILURE);
}
busy_n = req.busy_n;
```

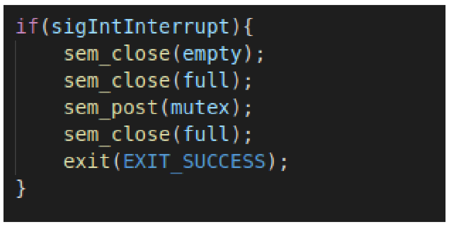
The assignment says that we need to use pipe, I followed the orders and used pipe. Message are forwarded via pipe. Pipes are created before fork and on way opened.

3 SIGINT



```c
volatile sig_atomic_t sigterm = 0;
volatile sig_atomic_t sighub = 0;

volatile sig_atomic_t sigIntInterrupt = 0;

void sigterm_handler(int signo){
    if (signo == SIGTERM)
        sigterm = 1;
}

void sighub_handler(int signo){
    if (signo == SIGHUP)
        sighub = 1;
}

void sigint_handler(int signo) {
    if (signo == SIGINT)
        sigIntInterrupt = 1;
}
```

When signal is received sigIntInterrupt is set to 1 and inside the server processes terminated.


```
if(sigIntInterrupt){
    sem_close(empty);
    sem_close(full);
    sem_post(mutex);
    sem_close(full);
    exit(EXIT_SUCCESS);
}
```

4.ServerY-ServerZ Communication

It has been accomplished by using pipe ,I managed communiction similar to serverY workers. Generally I wrote the all the steps for communication between serverY and serverZ. Also I wrote shared memory between serverZ and serverZ workers.

5. Which requirements I achieved?

I generally achieved the structure of midterm SIGINT Part and Daemon. They are working but have some mistakes. The sent code has been accomplished properly except these parts.