CSE 344 SYSTEMS PROGRAMMING MIDTERM REPORT
1801042609
YAKUP TALHA YOLCU

My structs

```c
#define SERVER_FIFO ""
                                /* Well-known name for server's FIFO */
#define CLIENT_FIFO_TEMPLATE "seqnum_cl.%ld"
                                /* Template for building client FIFO name */
#define CLIENT_FIFO_NAME_LEN (sizeof(CLIENT_FIFO_TEMPLATE) + 20)
                                /* Space required for client FIFO pathname
                                   (+20 as a generous allowance for the PID) */

struct request {                /* Request (client --> server) */
    pid_t pid;                  /* PID of client */
    int matrix_entry;
    int row;
};

struct response {               /* Response (server --> client) */
    int invertible;
};

struct worker {
    int client_id;
    int row_count;
    int pid;
    int available;
    int pipe_read_fd;
    int fifo_write_fd;
    int pipe_write_fd;
};

struct worker_z {
    int client_id;
    int row_count;
    int pid;
    int available;
```

```c
typedef struct shared_memory {
    char queue[MAX_BLKSIZE][MAX_BLKSIZE];
    int cursor;
    int last;
    int pool_size;
    int working_worker;
    int handled;
    sem_t empty;
    sem_t full;
    sem_t lock;
}shared_memory;
```

CLIENT:
./client -s serverfifo -o input2.csv
In client code I first read the cmd arguments and depend on them I determined the fifofile, datafile. I prepared request and response structs. Before I write to the fifo, I read the data file to learn matrix size. Then I write to the common server fifo the matrix entries,pid of client and matrix size. Then Client waits for the response at its unique fifo with blocking at read statement.

ServerY
./serverY -s serverfifo -o logfile -p 2 -r 3 -t 5
ServerY first sets its signal masks. I use sigint, sigusr1, sigusr2 signals to communicate with childs. I made daemon serverY. And it is also can't be instantiated 2 times. I prepare request and response structs. With respect to cmd arguments, I determined the serverfifo, logfile, sleep time, pool size of y, pool size of z. I created a pipe to communicate through Z.

I created Z with fork() and I sent serverfifo, logfile, pool number z, sleep time as cmd arguments to Z with execv.

I open the log file and duplicated the stdout and then I close the stdout. I directed output and errors to the logfile. I created worker structs. I created pipes to communicate through childs. I made fork and execute run_worker function. This function is function of child. It is in infinite loop unless it gets SIGINT signal. I keep the childs process ids in the worker struct. If I get signal from child says that I am available, then serverY writes sth to the pipe and waits for new request. If there is no available child, then serverY forwards the request to the serverZ with again writing to the pipe.

Childs reads the pipe and makes the operation on the matrix. Sends response to the client. serverY has written them to the pipe:
-row count
-matris entries
-pipe file descriptor
-client pid
-client file descriptor

Child reads them and makes operation, sends response to the client and returns the ready state. It is blocked on read pipe.
If childs becomes available it sends a sigusr1 signal to its parent to know the parent child is available anymore.

ServerZ
ServerZ is created automatically.

Firstly it reads the cmd arguments and it switches its stdout to log file.
It creates shared memory segment.
It initializes the semaphores.
It reads the pipe with serverY.
It writes it to the shared memory.

Child waits on semaphores to read data on the shared memory. If it reads it, it makes operation and send response to the client. It makes sure to know parent that matrix was invertible or not by writing sth to the results2.txt