CSE344 Midterm Report

Yeşim Yalçın-200104004094

1-) My Solution Approach

A-) ClientX

In clientX, I firstly check if the command line arguments are valid. After checking that and finding out the necessary file inputs, I create the client FIFO file. This fifo file has a basic template which will be used through processes.
The Template: InsertClientPIDHereFIFI.fifo

Afterwards I open data file and extract the matrix information.
I check if the matrix is a square matrix. If not, I do not accept the matrix.
Also the matrix cannot have size 1.

After extracting the matrix information, I try to open the server fifo. If the serverY was not initialized yet, the process will terminate here. If it was initialized then opening the server fifo in clientX wakes up serverY and the communication starts.

After opening the server fifo, I arrange the input and write it into the server fifo. Then try to read from the client fifo. However, this operation will be blocked until one of the worker processes actually write the response into the client fifo.

After the clientX receives it's response, it outputs it and then exits.

I also assign signal handlers for SIGINT and SIGPIPE.

B-) ServerY

ServerY is supposed to be a daemon process therefore it should not be initialized more than once at the same time. To achieve this, I have a semaphore. This semaphore will be waited on when a new serverY is initialized. If there is already a serverY running, then the new serverY process will terminate. Only the first one will remain. This semaphore is posted if the actual serverY exits.

In serverY, I again firstly check the command line arguments and find out the necessary file inputs, poolSizes etc. Also check if they are valid.

Afterwards, I make the serverY a daemon. The error outputs of serverY is directed into stderr till the log file is opened. After the log file is opened, all the outputs and errors are directed to the log file.

I create the server fifo if it does not exist then open it in read mode. If a clientX also opens the server fifo, serverY will wake up. After that I also open a dummy writer for the server fifo. This is needed not to see eof if all clients are terminated.

Afterwards, I initialize serverZ. I initialize serverY workers, then wait for a request coming from the server fifo.

If a request comes, I direct it to one of serverY workers through their pipes. (All workers have individual pipes) if not all workers are busy. If they are all busy, then I write them into serverZ’s pipe. onto a shared memory segment. This shared memory segment is like a request queue. It holds an array of requests, current request’s index (the request that needs to be handled next) and the current request amount in the queue. If the queue is full, it is refreshed using these indexes.

I have a signal handler for SIGINT. If a SIGINT signal comes, serverY sends this signal to all it’s workers and serverY. Wait for them to properly exit then serverY exits as well.

C-) ServerY Worker

ServerY worker waits for a request to arrive to their pipe. Until then, they are blocked. If a request comes, it starts processing it. After processing and creating a response, it sleeps for t seconds then writes the response to the client’s fifo. After that it lets serverY know that it is free now by sending it a SIGTRMIN signal. If serverY receives that signal, it marks the worker which sent it as free. I specially used this signal because it is possible to queue it. If it was not possible, then there could be lost signals if more than one worker woke up at the same time.

D-) ServerZ

ServerZ waits till a request comes into their pipe. When a request comes, the request is placed onto a shared memory segment. This shared memory segment is like a request queue. It holds an array of requests, current request’s index (the request that needs to be handled next) and the current request amount in the queue. If the queue is full, it is refreshed using these indexes. This shared memory segment is protected via semaphores. I use 3 semaphores for this purpose. 1 for the critical region, 1 to see if the queue is empty, 1 to see if the queue is full. ServerZ will be blocked if there is no requests coming also if the queue is full till there is space.

I have a signal handler for SIGINT. If a SIGINT signal comes, serverZ sends this signal to all it’s workers. Wait fort hem to properly exit then serverZ exits as well.

E-) ServerZ Worker

ServerZ workers will be blocked until a request comes to the queue in the shared memory segment. If a request arrives, one of the workers will handle it and then write to the client’s fifo after sleeping for t seconds.

F-) Extra Notes:

I also have one more semaphore for a shared memory segment. This segment stores the amount of invertible matrices calculated. Thanks to this, we can print out SIGINT messages.

Calculation of if the matrix is invertible is done by calculating the determinant of the matrices. While calculating I have utilized the cofactor method. I calculated necessary cofactors and summed them up in a recursive way.

Communication of client-serverY through fifos: Done
Communication of serverY-serverY workers through pipes: Done
Communication of workers-clients through fifos: Done
Communication of serverY-serverZ through pipes: Done
Communication of serverY-serverY workers through shared memory: Done
Locking log file everytime a process writes on it: Done via fcntl
Locking and synchronizing the shared memory: Done via 3 semaphores
Not letting more than 1 instance of serverY: Done via 1 semaphore
Daemonizing serverY: Done
Letting serverY know a worker became free: Done via signals
Letting serverY and serverZ know how many invertible matrices calculated: Done via shared memory
Handling SIGINT and cleaning up children: Done
Calculating if the matrix is invertible: Done via calculating determinant

SIGINT can be handled regardless of the state of serverY. Even if it is blocked etc it will be handled. However, if there are clients waiting for responses, before terminating the responses are given. This is like this because if the SIGINT comes when serverY is blocked on a read fifo call. ServerY will wait till all of the clients terminate and close their fifos then become unblocked then terminate. SIGINT handler closes the dummy write fd of the server fifo. However, it cannot close the write fds of clients. I did not want to directly exit from the signal handler thinking it would be a bad design. Therefore I made my design like this.