CSE 344 MIDTERM PROJECT:

REPORT:

-Serdil Anıl Ünlü

-1801042672

CLIENT.C FILE:

First of all I have two files named client.c and server.c. These files communicate with each other via fifo structure. In order to access the matrix information from the file, I got the information of the matrix saved in the .csv file in the client.c file. In addition , to calculate the size, I took the number of lines in the matrix as a basis.

```c
int findDimension(char *argv[]){
  int index=0;
  int line=1;
  int fileRead=0;
  int readByte;


  int input = open(argv[4],O_RDONLY);

  char buffer[1000];

  if(input < 0){
    perror("open");
    exit(-1);
  }
  do{

    readByte = read(input, buffer,1000);

    if(readByte == -1)
        perror("read while:-1");

    if(readByte == 0)
        fileRead = 1;
    for(int i=0;i<readByte;i++){
        if(buffer[i] == '\n'){
            ++line;
        }
    }
  }while(readByte != 0);


    return line;
    close(input);

}
```

-This function finds the size of the matrix written to the file. Here , every time I reach the end of the line, I increase the size by one.

HOW I CALCULATE TIMESTAMP?

For the time calculation, I used the gettimeofday() structure , defined two values called end and start, and calculated the time difference between the two values with a function I defined globally.

```
float time_diff(struct timeval *start, struct timeval *end)
{
    return (end->tv_sec - start->tv_sec) + 1e-6*(end->tv_usec - start->tv_usec);
}
```

```
struct timeval start;
struct timeval end;

gettimeofday(&start, NULL);
```

```
gettimeofday(&end,NULL);
printf("Total time spent: %f sec  goodbye.\n",time_diff(&start,&end));
```

HOW DID I ESTABLISH THE FIFO CONNECTION BETWEEN THE TWO FILES AND TRANSFER THE INFORMATION TO EACH OTHER?

```
int client_to_server;

int server_to_client;
char return_fifo[64];
char *myfifo2 = "/tmp/fifo2";
char str[2] = "0";
sprintf(return_fifo, "cl_%d", getpid());
printf("Client PID#%s, ",return_fifo);
printf("is submitting a %d*%d matrix:\n",findDimension(argv),findDimension(argv));
mkfifo(return_fifo, 0666);
client_to_server = open(argv[2], O_WRONLY);
server_to_client = open(myfifo2, O_RDONLY);
int buffer[1];
array[0] = findDimension(argv);
array[1] = getpid();

write(client_to_server,array,sizeof(int)*(dimension*dimension +2));
sleep(5);
int ret_fifo = open(return_fifo, O_RDONLY);
read(ret_fifo, buffer, sizeof(int));
```

-As seen in the example above , I also used findDimension() and getpid() structures to send information via fifo and saved this information to the array. I added both the matrix size and the client id (to provide communication between files) to the array , and finally the elements of the matrix, because I will process the elements of the matrix in the server file and print them back to the client file.

```
int ret_fifo = open(return_fifo, O_RDONLY);
read(ret_fifo, buffer, sizeof(int));

if(buffer[0] == 0){
    printf("the matrix is not invertible\n");
}

else{
    printf("the matrix is invertible\n");
}
```

After the information I sent in the image above is processed in the server file , I receive it again. I have already sent this information to the log file, and I also print this information on the terminal for the client.

SERVER:C FILE:

When the matrix information comes from the client.c file, I calculate whether the incoming matrix is invertible with the help of my 3 functions in the server file. In My 3 functions are as follows:

```
void cofactor(int **matris,int **temp,int p,int q,int n){

    int i=0;
    int j=0;

    for(int row=0;row<n;row++){
        for(int column =0;column<n;column++){
            if(row != p && column != q){

                temp[i][j++] = matris[row][column];
                if(j == n-1){ //Row is filled, so increase row index and reset column index
                    j=0;
                    i++;
                }

            }

        }
    }
}
```

```c
int findDeterminant(int **matris,int n){
    int result=0;


    if(n==1)
        return matris[0][0];

    int **temp; //record cofactors
    temp = (int**)malloc(sizeof(int*) * n);
    for(int i = 0; i < n; ++i)
        temp[i] = (int*)malloc(sizeof(int)*n);
    int sign =1;  //To store sign multiplier

    for(int f =0;f<n;f++){
        cofactor(matris,temp,0,f,n);
        result += sign * matris[0][f] * findDeterminant(temp, n - 1);

        sign = -sign;
    }

    free(temp);
    return result;
}
```

```c
bool checkInvertible(int **matris,int n){
    //printf("N boolean value: %d\n",n);
    if(findDeterminant(matris,n) != 0)
        return true;

    else
        return false;
}
```

HOW DO I PROVIDE THE RELATIONSHIP BETWEEN PARENT AND CHILD FOR serverY in server.c file ?

```c
  int **pd = (int**)malloc(atoi(argv[6])*sizeof(int*));
  int poolSize;
  int *available =(int*)malloc(sizeof(int)*poolSize);
  for(int i=0;i<atoi(argv[6]);i++){
      pd[i] = (int*)malloc(2*sizeof(int));
      pipe(pd[i]);
```

Here I used pipe to solve this problem.

```
while(1){ //parent
    client_to_server = open(argv[2], O_RDONLY);
    read(client_to_server,buf,sizeof(int) * 1024);
```

```
    write(pd[avRec][1],buf,sizeof(int)*(buf[0]*buf[0]+2));
```

As seen above, the parent first receives the information from the client file and transfers it to the child via the available worker.

```
    t = clock();
    read(pd[index][0],buffer,sizeof(int) * 1024);
```

```
    invertible = checkInvertible(mat,dim);
    int req_fifo = open(cl_file, O_WRONLY);
    int buf_test[1];
    buf_test[0] = invertible;
    clock_t t2;
    t2 = clock();
    write(req_fifo, buf_test, sizeof(int));
```

```
fprintf(fptr,"%s %s %s %s %s %f\n","Timestamp", "is", "for", "serverY", "child", "Reading s",time_taken);
fprintf(fptr,"%s %s %s %s %s %f\n","Timestamp", "is", "for", "serverY", "child", "Writing s",time_taken2);
fprintf(fptr,"%s %s %s %s %s %d %s %s %s %d*%d","Worker","PID","IS","HANDLED","CLIENT","PID#",buffer[1],"MATRIX","SIZE", "IS", buffer[0],buffer[0]);
```

After I get information from the child parent, after processing that matrix through my matrix functions , I send the operation results back to the client file and also print these results to the log file.

SO HOW DO I FIGURE OUT IF THE WORKER IS AVAILABLE OR NOT?

I find whether the worker is available or not thanks to SIGUSR1 signal. In the for loop , I return the size of the pool and use the kill command.

```
    int avRec;
    int avcount =0;
    sigset_t set;
    sigemptyset(&set);
    for(int i=0;i<poolSize;i++){
        kill(available[i],SIGUSR1);
        sigsuspend(&set);
        sleep(2);
            if(av==1){
                avRec=i;
                avcount++;
            }
    }
```

HOW I USE SIGINT SIGNAL TO EXIT GRACEFULLY?

Actually, I could not use this signal in the server.c file, but I can run it in the client file withoutany problems. Even if you try to exit suddenly, the program may end after the transaction is completed in the terminal, and I do this thanks to sigaction. I call the SIGINT signal in sigaction and it solves the problem. In addition, I provide control at control points thanks t my handler function that I created globally at certain points and my sigusr1_count int value.

```
void handler(int signal_number){

    ++sigusr1_count;
}
```

```
sig_atomic_t sigusr1_count = 0;
```

```
if(sigusr1_count == 1){
    write(1, "SIGINT signal is caught, exiting gracefully...\n", 44);
    free(input);
    free(output);
    return -1;
}
```

HOW I DID RUN THE DAEMON PROCESS?

```
int becomeDaemon(int flags)
{
    int maxfd,fd;

    switch(fork()){
        case -1:
            return -1;
        case 0:
            break;

        default: _exit(EXIT_SUCCESS);
    }

    if (setsid() == -1)
        return -1;

    switch (fork()) {
        case -1:
            return -1;
        case 0:
            break;
        default: _exit(EXIT_SUCCESS);
    }


    if (!(flags & BD_NO_UMASK0))
        umask(0);
    if (!(flags & BD_NO_CHDIR))
        chdir("/");

    if(!(flags & BD_NO_CLOSE_FILES)){
        maxfd = sysconf(_SC_OPEN_MAX);
        if(maxfd == -1)
            maxfd= BD_MAX_CLOSE;

        for(fd=0;fd<maxfd;fd++){
            close(fd);
        }
```

```
        default: _exit(EXIT_SUCCESS);
    }


    if (!(flags & BD_NO_UMASK0))
        umask(0);
    if (!(flags & BD_NO_CHDIR))
        chdir("/");

    if(!(flags & BD_NO_CLOSE_FILES)){
        maxfd = sysconf(_SC_OPEN_MAX);
        if(maxfd == -1)
            maxfd= BD_MAX_CLOSE;

        for(fd=0;fd<maxfd;fd++){
            close(fd);
        }
    }


    if (!(flags & BD_NO_REOPEN_STD_FDS)) {
        close(STDIN_FILENO);

        fd = open("/dev/null", O_RDWR);
        if (fd != STDIN_FILENO)
        return -1;

        if (dup2(STDIN_FILENO, STDOUT_FILENO) != STDOUT_FILENO)
        return -1;

        if (dup2(STDIN_FILENO, STDERR_FILENO) != STDERR_FILENO)
        return -1;
    }
    return 0;

}
```

-First of all I created the above function for the daemon process

```
void handler(int signal_number){

    if(av==0){
        kill(getppid(), SIGUSR1);
    }
    else{
        kill(getppid(),SIGUSR2);
    }
}

int flag=0;

void SIGINTHandler(){

    flag=1;
}
```

-Then I created the SIGINTHandler function, which is also my helper function and I set the flag in it as 1, and I also assigned the number 0 as a global value to the same flag.

```
void prevent(){
    unlink("serverYY.file");

}
```

Also, thanks to the prevent function, I create a temporary serverYY file and this file prevents a lot of daemons from forming behind.

```c
struct sigaction sa6;
memset(&sa6,0,sizeof(sa6));
sa6.sa_handler=SIGINTHandler;
sigaction(SIGINT,&sa6,NULL);
int one_instance = open("serverYY.file",O_CREAT|O_EXCL );
if(one_instance==-1){
    printf("Server Y is already running.CANNOT RUN ANOTHER SERVER...\n");
    exit(EXIT_SUCCESS);
}

else{
    atexit(prevent);
}

becomeDaemon(BD_NO_CHDIR | BD_NO_CLOSE_FILES);
```

And finally I define sigaction at the beginning of Main and then I call my becomeDaemon function.

WHICH REQUIREMENTS I ACHIEVED AND FAILED?

I received and read the data in the client.c file and successfully transferred it to the server file via fifo. After getting the information from the client file to the server file, I successfully exchanged information between the child and the parent , thanks to the pipe inside the server file. Also, I processed that information with child and after processing, I sent the processed information to both the client file and the log file without any problems.I do the daemon process successfully , for this the SIGINThandler global function, the becomeDaemon() function also works with the flag parameter,and whent he program runs, the serverYY file is created temporarily and prevents the processes from occurring at the same time. For Z process, I used shared memory. Shared memory causes child and parent to use the same memory space.

However, After processing the matrix in the server file , the matrix always returns as non-invertible or sometimes invertible so its about function but I manage the communication successfully. It’s just a problem with the functions themselves. Also another place where I am wrong is that when printing the workers, namely

the children, to the log file, instead of pressing one by one , it shows that all the workers are full, and prints itself to the log file only once.