**MEHMET ÖNDER-131044076**

**Midterm Projesinin Çözüm Aşamaları**

## TimeServer:

Program temel olarak her açılan client için bir adet fork oluştur. Client (seeWhat) açıldığı ilk zaman parametre olarak gelen mainfifo ya kendi pidsini yazar. Main Server(Ana timeServer) bu yazılan pid değerini okur ve bu özel olarak sürekli bir matris göndermek için bir fork oluşturur

```
do
{
/* signal handler olustur*/
    flag=0;
    sigprocmask(SIG_BLOCK, &sa.sa_mask, NULL);
    SentMatrixToClient(clientpid,matrixSize,logpipe);
    sigprocmask(SIG_UNBLOCK, &sa.sa_mask, NULL);
    pause();

}while(flag==1);
```

.Daha sonra mainServer parametrede girilen zaman aralıklarıyla sürekli olarak yeni process gelip gelmediğini kontrol eder.Eğer yeni client gelmişse tekrar fork üretir ve kendi işini bu şekilde tekrarlar.

```
while(1){

    sleep(waitingtime);
    while(read(mainFifoDes,&clientpid,sizeof(pid_t))>0){
        cliendArr[cliendArrSize++]=clientpid;
        clock_t endtime = clock();
        pid_t tempPid=fork();
        if(tempPid==0){
            close(mainFifoDes);
            struct sigaction sa;
            sa.sa_handler = sighand;
            sigemptyset(&sa.sa_mask);
            sigaddset(&sa.sa_mask, SIGUSR1);

            if (sigaction(SIGUSR1, &sa, NULL) == -1) {
                fprintf(stderr, "Cannot set sigaction...\n");
                exit(1);
            }
            do
            {
            /* signal handler olustur*/
                flag=0;
                sigprocmask(SIG_BLOCK, &sa.sa_mask, NULL);
                SentMatrixToClient(clientpid,matrixSize,logpipe);
                sigprocmask(SIG_UNBLOCK, &sa.sa_mask, NULL);
                pause();

            }while(flag==1);
            exit(0);
        }

    }
}
```

Client için oluşturulmuş özel fork içinde SIGUSR1 sinyali handle edilmiştir. Bu sinyal client(Seewhat )gönderilen son matrisin üzerindeki yapılacak tüm işlemler bittiğinde seewhat tarafından yeni bir matris istemek için gönderilir.

```
struct sigaction sa;
sa.sa_handler = sighand;
sigemptyset(&sa.sa_mask);
sigaddset(&sa.sa_mask, SIGUSR1);

if (sigaction(SIGUSR1, &sa, NULL) == -1) {
    fprintf(stderr, "Cannot set sigaction...\n");
    exit(1);
}
do
{
/* signal handler olustur*/
    flag=0;
    sigprocmask(SIG_BLOCK, &sa.sa_mask, NULL);
    SentMatrixToClient(clientpid,matrixSize,logpipe);
    sigprocmask(SIG_UNBLOCK, &sa.sa_mask, NULL);
    pause();

}while(flag==1);
```

Sinyal geldiği zaman yeni bir matris oluşacağı zaman başka bir sinyalın araya girmemesini önlemek amacıyla matrisin üretim fonksiyonları gelecek sinyallere blocklanmıştır. Matris üretiliren nxn den 2nx2n bir matris üretimi yapılır bunun nedeni matrisin nxn tarafının tersi alınacağı zaman determinantı sıfır olmasını önlemek içindir. Ve bu üretilen 2nx2n matris tersi alınabilen bir matristir. Matrisin log'a yazdırmak için pipe ile bir struct yapısı yollanmıştır.

```
clock_t begin=clock();
while(determinantResult==0){
    float tempNmatrix[MATRIXMAXSIZE][MATRIXMAXSIZE];
    j=0;
    for (i = 0; i < matrix4nsize; ++i){

        matrix[i]= rand()% 15 ;

        if(i%matrixSize==0){
            if(j==matrixSize)
            {
                float tempdeter=determinant(tempNmatrix,matrixSize);
                if(tempdeter==0)
                    i-=matrixSize;
                j=0;
            }else{
                tempNmatrix[i%matrixSize][j]=matrix[i];
                j++;
            }
        }

    }
    for ( i = 0; i < matrix2nsize; ++i)
    {
        for ( j = 0; j < matrix2nsize; ++j)
        {
            detarr[i][j]=matrix[i*matrix2nsize+j];

        }
    }
    for ( i = matrix2nsize; i < MATRIXMAXSIZE; ++i)
    {
        for (j = matrix2nsize; j < MATRIXMAXSIZE; ++j)
        {
            detarr[i][j]=0;
        }
    }

    determinantResult =determinant(detarr,matrix2nsize);
}
clock_t end=clock();
```

Bu sayede kolay şekilde her bir client için oluşmuş sub processler bilgilerini pipe içerisine yazmaktadır. Subprocess oluşan matrisi gönderebilmek için client tarafından oluşturulan clientpidsi fifo ismi olan fifo dosyasına bu matrisi yazar ve client bu fifo üzerinden yeni matrisini alma işlemine devam eder.

```c
int logpipe[2];
if(pipe(logpipe)<0){
    perror("pipe çalışmadı");
}

pid_t logfork=fork();
if(logfork==0){
    LogInfType templog;
    logpipe[1];
    logfile=fopen("log/timeserver.log","w");
    while(1){
        while(read(logpipe[0],&templog,sizeof(LogInfType))>0){
            fprintf(logfile, "clientpid=%d  createdTime=%lf determinant=%lf\n",templog.clientpid,templog.elastime ,templog.determinant);
        }

    }


}
while(1){
```

Burada önemli olan nokta yeni bir cilent oluştuğunda pidsini sadece bir kere mainfifoya yazmaktadır. Fifo yapısı queue yapısına benzediği çin yazılan pid için bir kere subprocess oluşur. Artık mainprocess'in oluşan seeWhat ile bir işi kalmamış durumdadır. Subprocess ile sinyal ve fifo yardımı ile ile iletişim kurup sürürlüğünü devam ettirmektedir.

```c
clock_t end=clock();
LogInfType templog;
templog.determinant=determinantResult;
float tempclock=(float)(end-begin)/CLOCKS_PER_SEC;
templog.clientpid=getpid();
templog.elastime=tempclock;
close(logpipe[0]);
write(logpipe[1],&templog,sizeof(LogInfType));
pid_t serverpid=getpid();
write(writeClientFifodes,&parentpid,sizeof(pid_t));
write(writeClientFifodes,&matrix2nsize,sizeof(int));
write(writeClientFifodes,&serverpid,sizeof(pid_t));
write(writeClientFifodes,matrix,sizeof(float)*matrix2nsize*matrix2nsize);
close(writeClientFifodes);
free(matrix);
```

TimeServer-CRTL+C sinyali:

Bu sinyal geldiğinde tüm sub processlere kill(0) ile sıgint komutu gönderilir. Eğer subprocess bir matris üretim yapıyorsa ,malloc kullanıldığı için blocklanır. Üretmiyorsa açık olan mainfifo yu kapama işlemi yapar ve kendini sonlandırır. MainProcess daha sonra tüm clientların(seewhat)  oluğu pid arrayını gezerek tüm pidlere kill sinyalı gönderir. Ve Showresult eğer açıksa bir adet Show resultin temp olarak oluşturduğu dosyadan showresult pid değerini okur. Ve son olarak bu programada kill sinyali gönderir ve tüm açık dosyaları kapatıp exit ile çıkış yapar.

```c
void killhander(int signo){
    if(signo==SIGINT && parentpid==getpid()){
        kill(0,SIGINT);
        fprintf(stderr, "CTRL+C signal was taken\n");
        int i;
        for ( i = 0; i < cliendArrSize; ++i)

        {
            kill(cliendArr[i],SIGINT);
        }
        close(mainFifoDes);
        FILE* tempread=fopen("showresultpid.temp","r");
        int tempid=0;
        if(tempread!=NULL){
            fscanf(tempread,"%d",&tempid);
            fclose(tempread);
            kill(tempid,SIGINT);
        }
        unlink("serverpid.temp");
        unlink("showresultpid.temp");
        unlink("showresult.fifo");
        unlink(mainFifoName);
        exit(0);

    }
    else if(signo==SIGINT && parentpid!=getpid()){
        fclose(logfile);
        exit(0);
    }



}
```

TimeServer SIGUSR1 sinyali:

Bu sinyali her bir sub processe ait client programı gönderir .Subprocess ile client arasında özel olarak kurulmuş olan fifo üzerinden subprocessin pidsi gönderilir clientin işlemleri bittiğinde bu sinyalı göndererek yeni bir matris ister.

```c
struct sigaction sa;
sa.sa_handler = sighand;
sigemptyset(&sa.sa_mask);
sigaddset(&sa.sa_mask, SIGUSR1);

if (sigaction(SIGUSR1, &sa, NULL) == -1) {
    fprintf(stderr, "Cannot set sigaction...\n");
    exit(1);
}
do
{
/* signal handler olustur*/
    flag=0;
    sigprocmask(SIG_BLOCK, &sa.sa_mask, NULL);
    SentMatrixToClient(clientpid,matrixSize,logpipe);
    sigprocmask(SIG_UNBLOCK, &sa.sa_mask, NULL);
    pause();

}while(flag==1);
```

## SeeWhat :

Bu program client görevi görür. Oluştuğu zaman ilk olarak mainfifo'ya kendi pidsini yazar ve özel bir subprocess beklemek üzere sıraya girer.Subprocess oluştuğunda subprocessin pidsini alip ,sürekli olarak matris bekleme sürecine girer. Eğer matris gelmişse matrisin **, s**hifted inverse matris, 2d convolution matris değerlerini hesaplamak için 2 adet fork oluşturur.

```c
pid_t fork1=fork();
if(fork1==0){
    float shiftedMatrix[MATRIXMAXSIZE][MATRIXMAXSIZE];
    clock_t begintime = clock();
    CalculateShiftedMatrix(arr,shiftedMatrix,matrixSize);
    clock_t endTime = clock();
    double diftimee=(double)(endTime-begintime)/CLOCKS_PER_SEC;
    MatrixArrayType temp;
    for ( i = 0; i < matrixSize; ++i)
    {
        for ( j = 0; j < matrixSize; ++j)
        {
            temp.arr[i][j]=shiftedMatrix[i][j];
            //fprintf(stderr, "%lf,", shiftedMatrix[i][j]);
        }
        //fprintf(stderr, "\n");
    }
    temp.type='S';
    temp.elapsedtime=diftimee;
    close(pipearr[0]);
    write(pipearr[1],&temp,sizeof(MatrixArrayType));
    close(pipearr[1]);
    exit(0);
}
```

```c
pid_t fork2=fork();
if(fork2==0){
    float convuliton[MATRIXMAXSIZE][MATRIXMAXSIZE];
    clock_t begintime = clock();
    ConvolutionCalculate(arr,convuliton,matrixSize,matrixSize);
    clock_t endTime = clock();
    double diftimee=(double)(endTime-begintime)/CLOCKS_PER_SEC;
    MatrixArrayType temp;
    for ( i = 0; i < matrixSize; ++i)
    {
        for ( j = 0; j < matrixSize; ++j)
        {
            temp.arr[i][j]=convuliton[i][j];
        }
    }

    temp.type='C';
    temp.elapsedtime=diftimee;
    close(pipearr[0]);
    write(pipearr[1],&temp,sizeof(MatrixArrayType));
    close(pipearr[1]);


    exit(0);
}
while(wait(NULL)!=-1 || errno==EINTR);
```

Bu forklarin biri **s**hifted inverse matrisi hesaplarken diğeri convolution matris değerini hesaplar .

Sonuc olarak ürettikleri değerleri pipe yolu ile mainClienta bildirirler.

```c
MatrixArrayType temp;

double convolutionTime;
double shiftTime;
    while(wait(NULL)!=-1 || errno==EINTR);
close(pipearr[1]);
while (read(pipearr[0],&temp,sizeof(MatrixArrayType))>0 )
{
    if(temp.type=='C')
    {
        for ( i = 0; i < MATRIXMAXSIZE; ++i){
            for ( j = 0; j < MATRIXMAXSIZE; ++j)
                convuliton[i][j]=temp.arr[i][j];
            convolutionTime=temp.elapsedtime;
        }
    }
    else if(temp.type=='S'){
        for ( i = 0; i < MATRIXMAXSIZE; ++i){
            for ( j = 0; j < MATRIXMAXSIZE; ++j)
                shiftedMatrix[i][j]=temp.arr[i][j];

            shiftTime=temp.elapsedtime;
        }
    }
}
close(pipearr[0]);
```

**MainClient bu matris için log aldındaki seewhatlog klasörü içine bir log dosyası oluşturur. Bu dosyanın adı sırası client pid si ve kaçıncı matris oluğunu içeren bilgidir.** İçerisine Orijinal,shifted ve Conv.matrisin içeriklerini yazıp dosyayı kapatır.En son MainClient  Subprocesslerden gelen matrislerle.elde edilen result1 ,result2 ve bunların oluşma zamanlarını bir struct yapısa ekler. Eğer varsa showresult ın kendi fifo dosyasına bu struct değişkenini yazar ve matrisin işlemleri bitmiş olur.

```
MatrixResultsType result;
result.shiftElapsed=shiftTime;
result.convElapsed=convolutionTime;
result.result1=result1;
result.result2=result2;
result.cliendPid=getpid();
result.requestNum=logcount;
int writeShowResult=open("showresult.fifo",O_WRONLY);
if(writeShowResult!=-1){
    write(writeShowResult,&result,sizeof(MatrixResultsType));
    close(writeShowResult);
}
char cliendLogname[255];
strcpy(cliendLogname,"");
sprintf(cliendLogname,"log/seewhatLogs/%d",getpid());
char strCountname[255];
sprintf(strCountname,"-%d",logcount);
++logcount;

strcat(cliendLogname,strCountname);
FILE *writeClientLog=fopen(cliendLogname,"w");
    /* write oorginal matrix*/
char tempstr[255];
char temp2[2];
strcpy(tempstr,"");
sprintf(tempstr,"Orginal Matrix =[\n");
fwrite(tempstr,sizeof(char),strlen(tempstr),writeClientLog);
for ( i = 0; i < matrixSize; ++i)
{
    for ( j = 0; j < matrixSize; ++j)
    {
        char temp2[10];
        strcpy(temp2,"");
        sprintf(temp2,"%.2f,",arr[i][j]);
        fwrite(temp2,sizeof(char),strlen(temp2),writeClientLog);

    }
    char temp2=';';
    fwrite(&temp2,sizeof(char),1,writeClientLog);
}
```

```
    sprintf(tempstr,"Shifted Matrix =[\n");
    fwrite(tempstr,sizeof(char),strlen(tempstr),writeClientLog);
    for ( i = 0; i < matrixSize; ++i)
    {
        for ( j = 0; j < matrixSize; ++j)
        {
            char temp2[10];
            strcpy(temp2,"");
            sprintf(temp2,"%.2f,",shiftedMatrix[i][j]);
            fwrite(temp2,sizeof(char),strlen(temp2),writeClientLog);

        }
        char temp2=';';
        fwrite(&temp2,sizeof(char),1,writeClientLog);
    }


    strcpy(temp2,"]\n");
    fwrite(temp2,sizeof(char),2,writeClientLog);

    strcpy(tempstr,"");
    sprintf(tempstr,"convuliton Matrix =[\n");
    fwrite(tempstr,sizeof(char),strlen(tempstr),writeClientLog);
    for ( i = 0; i < matrixSize; ++i)
    {
        for ( j = 0; j < matrixSize; ++j)
        {
            char temp2[10];
            strcpy(temp2,"");
            sprintf(temp2,"%.2f,",convuliton[i][j]);
            fwrite(temp2,sizeof(char),strlen(temp2),writeClientLog);
        }
        char temp2=';';
        fwrite(&temp2,sizeof(char),1,writeClientLog);
    }

    strcpy(temp2,"]\n");
    fwrite(temp2,sizeof(char),2,writeClientLog);

    fclose(writeClientLog);
}
free(matrix);
matrix=NULL;
```

.İşlemler bittiğinde subprocesse SIGUSR1 sinyalı yollar. Ve SubTimeserver yeni bir matrix yollar.

```
            fwrite(temp2,sizeof(char),2,write

            fclose(writeClientLog);
        }

        kill(serverpid,SIGUSR1);
    }
```

**SeeWhat CTRL+C:**

```
void killhander(int signo){
    if(signo==SIGINT && parentClintpid==getpid()){
        close(readFifo);
        close(writeServerDes);
        kill(0,SIGINT);
        while(wait(NULL)!=-1);
        unlink(myfifoname);
        if(matrix!=NULL)
            free(matrix);
        kill(parentTimeServerpid,SIGINT);
        fprintf(stderr, "Ctrl+C signal was taken\n");
        exit(0);
    }

    else
        exit(0);
}
```

MainClient malloc işlemi sırasında gelen sinyali blocklar block kalkınca sinyal fonksiyonu tüm sub clientlara kill(0) komutu gönderir. Eğer client sub client ise açık dosyaları kapatıp çıkar. Daha

sonra mainclient kendisi için oluşturduğu fifo dosyasını silme işlemi gerçekleştirir. Ve timeServerin parent pidsine kill sıgint komutu gönderir.

## ShowResult:

```
while(read(readshowresult,&result,sizeof(MatrixResultsType))>0){

    sigprocmask(SIG_BLOCK, &sinyal.sa_mask, NULL);
    if(flag==0){
        FILE *readpid=fopen("serverpid.temp","r");
        fscanf(readpid, "%d", &mainServerpid);
        fclose(readpid);
        flag=1;
    }
    writelog=fopen("log/showresult.log","a+");
    fprintf(stderr, "pid=%d-%d\t, Result1=%lf\t, Result2=%lf\n",result.cliendPid,result.requestNum,
        result.result1,     result.result2);
    char tempstr[255];
    sprintf(tempstr,"cliend Pid=%d-%d\nResult1 Time=%lf\nResult2 Time=%lf\n",result.cliendPid,result.requestNum,result.shiftElapsed,result.convElap
    fwrite(tempstr,sizeof(char),strlen(tempstr),writelog);
    fclose(writelog);
    writelog=NULL;
    sigprocmask(SIG_UNBLOCK, &sinyal.sa_mask, NULL);
}
```

Program ilk açıldığında ilk olarak kendi özel fifo dosyasını oluşturur. Sonra kendi pidsni temp bir dosyaya yazar. Bu dosyanın adı sabit bir bilgidir. Bu işlemler bittiğinde kendisi okuma moduna alarak oluşturduğu fifoyu sürekli olarak logları okuma işlemi yapar. Ve okurken bir adet oluşturduğu showresult.log dosyasına sürekli olarak gerekli log bilgiler yazar.Aynı zamanda ekranada gerekli olan log bilgilerini yazma işlemi gerçekleştirir.İşlemler sırasında önemli yerlerde blocklama işlemi yapmaktadır.

```
if(mkfifo("showresult.fifo",0666)==-1);
readshowresult=open("showresult.fifo",O_CREAT|O_RDONLY);
struct sigaction sinyal;
sinyal.sa_handler = killhander;
sigemptyset(&sinyal.sa_mask);
sigaddset(&sinyal.sa_mask, SIGINT);
if (sigaction(SIGINT, &sinyal, NULL) == -1) {
    fprintf(stderr, "Cannot set sigaction...\n");
    exit(1);
}
sigprocmask(SIG_BLOCK, &sinyal.sa_mask, NULL);

FILE *writePid=fopen("showresultpid.temp","w");
fprintf(writePid, "%d\n",getpid() );
fclose(writePid);

sigprocmask(SIG_UNBLOCK, &sinyal.sa_mask, NULL);
```

ShowResult CTRL+C:

Sürekli olarak dosyaya yazdığı için işlem sırasında sinyal blocklanır. Blocklama bittikten sonra temp ve fifo dosyalarını silme işlemi gerçekleştirir.Ve Timeserverin temp dosyası varsa eğer pidsini okuyup kill sinyalı gönderir. Ve kendisini kapatır.

```
void killhander(int signo){
    if(readshowresult!=-1)
        close(readshowresult);
    char tempstr[255];
    strcpy(tempstr,"");
    if(writelog!=NULL){
        sprintf(tempstr,"Kill signal was taken");
        fwrite(tempstr,sizeof(char),strlen(tempstr),writelog);
        fclose(writelog);
    }
    kill(mainServerpid,SIGINT);
    unlink("serverpid.temp");
    unlink("showresultpid.temp");
    unlink("showresult.fifo");
    exit(0);
}
```

Ek olarak

- TimerServer çalışmadan diğer programların çalışması durumunda program hata mesajı verip çıkış yapar.
- Herhangi bir durumda matrisin tersi alınamıyosa tersi alınamıyan matrisin tersi bir sayisına eşitlenir.
- **Program Abdullah Akay hocamızın moodle a koymuş olduğu virtual machine ile derlenmiştir.**
- **Test klasörü içerisinde önceden derlenmiş program bulunmaktadır Herhangi. bir sorun karşısında derlenmiş programla test edilebilir durumdadır.**
- **Test klasörü içerisinde önceden denenmiş log dosyaları bulunmaktadır. İnceleme yapılabilir.**

Yararlanılan Kaynaklar:


- Shifted inverse matris hesaplanması için ; http://www.sanfoundry.com/c-program-find-inverse-matrix/ adresinden yararlanılmıştır.

- 2d Convolution Matris Hesaplanması için ; http://www.songho.ca/dsp/convolution/convolution.html adresinden yararlanılmıştır.