GEBZE TECHNICAL UNIVERSITY
COMPUTER ENGINEERING

SYSTEM PROGRAMMING
CSE 344 - 2020 SPRING

HW3 REPORT

SILA BENGİSU YÜKSEKLİ
1801042877

➢ *Explanation of the way of work*

User can compile by typing make on terminal. User should give command line arguments like “-i input.txt - j input2.txt -n 1” , then it can be run by writing “./program” on terminal.

Program takes two input files and a positive integer number. Number of 2^n represents the row and column of the matrix. Program read 2^n x 2^n characters from input files and interpretes them as elements of the matrix. Then it procures product matrix by multiplying with each other. It calculates singular values of the product matrix, prints it on the screen, then exits.

It makes the calculation in a distributed fashion. The main process has four child process and it uses them to calculate every quarter piece, but while it is doing this, it should communicate with child processes somehow so that parent process don’t try to combine pieces before child processes calculate. For this purpose pipe structure is used in this program.

➢ *Communication between parent process and its child processes*

There are four bidirectional pipes between parent and its child processes. To make one bidirectional pipe, two pipe should be used. So, eight pipes are created. Pipe has own synchronization. Namely, it waits for an input when program reaches the reading operation from pipe.

After four children are created, program closes the read-end of the first pipes ,they are unused, then it starts writing required parts of matrix A and B to the pipe. If an interrupt occurs and one of the children starts running, pipe waits for an input to read. That’s way there will be no problem. After parent process is done with writing, child processes start running and read informations from pipe, but before that they close the write end of the first pipes. Then, they close the read-end of the first and second pipe. Read-end of the second pipe is unused. They calculate the parts and save the results in an array. They send them to parent process by writing to pipe. When they’re done with writing, they close the write-end of the second pipe so that parent process can understand whether it can start reading from pipe. Parent process reads results from four pipes, combines them and saves it in matrix C. After that, it calculates singular values of product matrix. This calculation is taken elsewhere.

➢ *Handling of Zombie Processes*

When a child process dies, parent process receives a signal which is SIGCHLD. To prevent processes from being zombie, there are several methods. In this program waitpid method is used in a SIGCHLD handler. Handler catches SIGCHLD signal when a child process dies as seen in the code snippet below.

```
int stat;
  while(waitpid(-1,&stat,WNOHANG) > 0){
        --aliveChildren;
  }
```

This part of code checks if any zombie-children exits. If any, it collects the status of the dead processes, thus it avoids this situation.

The global variable aliveChildren is used, because parent process must not die before its children. Value of this variable is initially set at four. Whenever a status is collected of a dead process, program decreases its value. As seen in the code snippet below, parent process is suspended until all of its children become dead. Thus, children don't become orphans.

```
while(aliveChildren > 0){
        sigsuspend(&newMask);
}
```

➢ *Handling of SIGINT signal*

There is a handler for each separate process. Initially, parent process creates a handler for this signal and whenever a process is forked, this structure is passed on to children as inheritance, because I don't use execve. All processes must exit gracefully, if program receives this signal. However every process has own allocated memory. So free operation is different from each other. That's way I kept separate their handlers.

I keep global variables which has type of void**. Whenever I allocate memory, I save its address in this array so that if an SIGINT signal is come, handlers can free resources.

This signal can also come between two lines that contains free. So I keep global variables as count. If a resources is freed, I decrease counter so that probability of double free error is prevented.