GEBZE TECHNICAL UNIVERSITY
COMPUTER ENGINEERING

SYSTEM PROGRAMMING
CSE 344 - 2020 SPRING

HW2 REPORT

SILA BENGİSU YÜKSEKLİ
1801042877

❖ *Problem Definition*

The purpose of this problem is to enable two different processes to communicate with each other and to process the same files as quickly as possible. p1 process should process all the contents of the input file and write processed data to temporary file it has previously opened. p2 process should process all the contents of the temporary file and write processed data to output file. However, this situation must be completed as soon as possible, means that processes doesn't have to wait for each other. For this purpose, I developed a program that consists of three main parts.

❖ *Solution of the Problem*

➔ *calculations.c*

Only mathematical operations are included in this part of the program. There are three functions to calculate error metrics which are mean absolute error, mean squared error and root mean squared error. There is another function to calculate line equation. This part of the program is kept separate in order to make the code understandable. Also, coordinates are represented as struct to make code easier.
I use some resources in below to write these calculations:

- https://medium.com/@ewuramaminka/mean-absolute-error-mae-sample-calculation-6eed6743838a
- https://veribilimcisi.com/2017/07/14/mse-rmse-mae-mape-metrikleri-nedir/
- https://www.youtube.com/watch?v=BnYQAJIhTf0

➔ *program.c*

This part of the program represents p1 process. First, it gets command lines and parses them. To do this, I use getopt library method. If arguments are not invalid, missing or more than it should be, process is forked, creates a child process and executes it.

After this line program has two processes that runs concurrently, but we can't know what the operating system will run first and when the operating system does context switch. For this purpose, signals and lock operation are used to prevent crashing.

p1 process reads 20 bytes and calculates a line equation. While it is calculating line equation, SIGINT signal is blocked. There are no ways to ignore or catch SIGSTOP signal, so only SIGINT signal is blocked. To do this, sigprocmask is used.
By this way, program creates a critical region, so it can not be interrupted by this signal.

It writes processed data and the line equation to a temporary file, but before that, it locks the file in case some other process tries to write to this file.

There is a global variable to understand if the p2 process is started. If it starts, it sends a SIGUSR1 signal to p1 process. When p1 process catches it, the handler runs and set global variable , isStarted , to one.

After it writes once to the file and if p2 process is started, p1 process sends a SIGUSR2 signal to p2 process to notify that a line is ready and this line can be processes. But p2 process has not started yet, p1 process does not send any signal to it. After that, it continues to read twenty bytes, if any. When it reaches to end of the file, it prints on the screen informations about how many bytes are processed and line equaitons are estimated. If any SIGINT signal has arrive, while it was in the critical section, it prints on the screen.

If p2 has not started yet, I use busy waiting to force context swtiching at the end of the p1 process. Then p1 sends two different signals which are SIGUSR1 and SIGUSR2. SIGUSR1 indicates that p1 has finished and done all the work. SIGUSR2 is for sigsuspend

➔ *program2.c*

When p2 process has started, it first blocks the SIGUSR2 signal to prevent this situation : if this signal has arrive before the line that includes sigsuspend, p2 process can be pended forever. Then, it sends a signal to p1 process to notify that it has started. There are two different handlers for SIGUSR1 and SIGUSR2 signal.

There is a global variable, isFinished, when signal arrives, handler runs and sets this variable to one. This variable is needed for understanding whether p1 is finished or not. Because if it is finished, that means p2 can process all the contents of the temporary file.

sigsuspend was used in two places to ensure synchronization.

First one:

```
sigsuspend(&newMask);

fd = open(temp_name,O_RDWR);
if (fd == -1)
{
        ...
}

while(ch != '\n'){

        size = read(fd,&ch,1);
        ++count;
        if (size == -1)
        {
                ...
        }

...
```

Before it enters a loop, first it calculates how many bytes first row has, but to do this, p1 has to processed a line. If p2 is started before p1, it should not enter this loop. Because there is no content at this time. It has to wait. That's way sigsusped is used in this line.

Second one:

```
....

while(ch != '\n'){
                size = read(fd,&ch,1);
                if (size == -1)
                {
                        ...
                }
                if (ch == 'x')
                        totalLine+=1;

                if (size == 0)
                {
                        ch = '\n';
                        if (isFinished != 1)
                        {
                                sigsuspend(&newMask);
                                ch = ' ';
                                --count;
                        }
                }

                ++count;
        }
....
```

After p2 processed one line, it calculates the number of the bytes of the next row, but it should not do this calculation if p1 does not processed next line yet. If size equals to zero, that means there are no more inputs in the next row. There can be two reasons of that : it is the end of the file or p1 does not finish its work yet. So, sigsuspend is used in this line. If p1 has not finished, p1 sends a SIGUSR2 when it has started and p2 will be able to continue.

❖ *How to run*

To run this program, you can use makefile. After compile with make file, you should write “ ./program ” to terminal. By this way, there are two executable files will be created. Program2 is also run after fork function by using execve by this way.