CSE344 – Systems Programming

Homework 2 Report

Oğuzhan Agkuş - 161044003

Firstly, I am sorry for late submission. I had a big misunderstanding about homework. It took time to correct this misunderstanding. Finally, I solve it. I want to say thank you for one day extension. A penalty of 25% is better than missing submission.

My homework consist of 4 files:

- helper.h
- helper.c
- program.c
- makefile

Helper files contains following statistical functions and methods:

- mean
- mean deviation
- standard deviation
- mean absolute error
- mean squared error
- root mean squared error
- least squares method

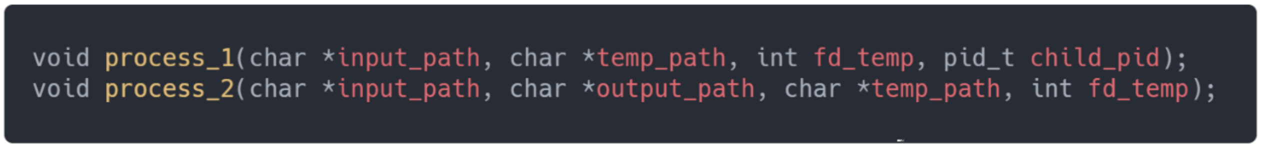
I had implemented these functions according to Wikipedia definitions.

Also he helper header includes 2 structures that represents point and line. They make easier to understand reading and writing operations.

My program gets 2 inputs: input and out paths. In case of given invalid or missing arguments it exits with -1. I used the getopt() function to parsing arguments.

If there is no problem with arguments it creates a temporary file with mkstemp system call. This file is temporary. It will be removed after all jobs are done.

The program has two functions: process_1 and process_2. The first one runs as a parent and the second one runs as child. The se process is forked from process_1. These functions get some parameters they will need.



```
void process_1(char *input_path, char *temp_path, int fd_temp, pid_t child_pid);
void process_2(char *input_path, char *output_path, char *temp_path, int fd_temp);
```

process_1 reads from input file and write to common temp file. The reading operation is not complex. In every loop it reads 20 bytes, then convert each byte to a numeric value and store it as a coordinate. After reading 20 coordinates (bytes) it calculates with help of least squares method (function). It returns a line equation like y=ax+b.

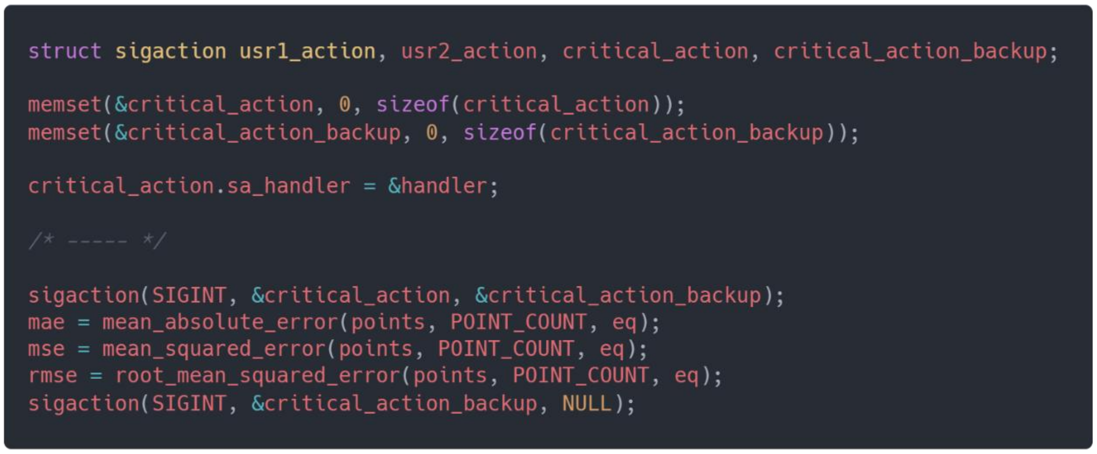
```c
line_t least_squares_method(point_t *points, size_t count) {
  int i;
  line_t eq;
  double m, temp;
  double x_sum = 0, y_sum = 0, x_sq_sum = 0, xy_sum = 0;

  for(i = 0; i < count; i++) {
    x_sum += points[i].x;
    y_sum += points[i].y;
    x_sq_sum += (points[i].x * points[i].x);
    xy_sum += (points[i].x * points[i].y);
  }

  m = (count * xy_sum - (x_sum * y_sum)) / (count * x_sq_sum - (x_sum * x_sum));
  temp = (y_sum - (m * x_sum)) / count;

  eq.a = m;
  eq.b = temp;

  return eq;
}
```

This calculation is in critical section. It means it cannot be stopped with any signals. But some signal cannot be ignored. For example: SIGSTOP. But SIGINT can be ignored. I can ignore this by using sigaction(). I defined a new handler for SIGINT. I set it just before the doing calculations. So when an SIGINT comes during calculation, it does not stop. It continues and then just after calculations finished, I restore the handler. So, SIGINT will be available until next calculation.

```c
struct sigaction critical_action, critical_action_backup;

memset(&critical_action, 0, sizeof(critical_action));
memset(&critical_action_backup, 0, sizeof(critical_action_backup));

critical_action.sa_handler = &handler;

/* ----- */

sigaction(SIGINT, &critical_action, &critical_action_backup);
eq = least_squares_method(temp, POINT_COUNT);
sigaction(SIGINT, &critical_action_backup, NULL);
```

Then it writes the points and the equation to temp file as a line. Also, it sends a signal to process_2 for saying temp file has data to read. I used SIGUSR2 signal to provide this communication. When process_1 done with input file, it closes it and prints the total bytes read and estimated equation count into terminal. Also, it sends a signal to process_2 for saying it is done with input file.

process_2 reads the temp file concurrently with process_1. I need to lock this file to prevent leaks. process_2 reads a line which contains of 10 points and a line equation. It does some error calculations such as MAE, MSE, RMSE. These calculations is also doing in the critical section just like in the process_1.



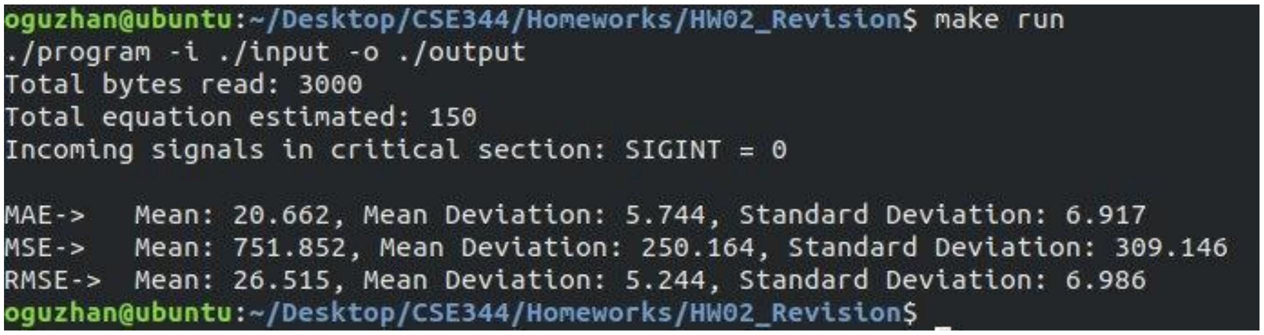
```
struct sigaction usr1_action, usr2_action, critical_action, critical_action_backup;

memset(&critical_action, 0, sizeof(critical_action));
memset(&critical_action_backup, 0, sizeof(critical_action_backup));

critical_action.sa_handler = &handler;

/* ----- */

sigaction(SIGINT, &critical_action, &critical_action_backup);
mae = mean_absolute_error(points, POINT_COUNT, eq);
mse = mean_squared_error(points, POINT_COUNT, eq);
rmse = root_mean_squared_error(points, POINT_COUNT, eq);
sigaction(SIGINT, &critical_action_backup, NULL);
```

When it read a line, it deletes the line from temp file. The deleting is overwriting operation with decreasing file size.

Until process_1 says to process_2 that there will not be input, it continues to read. If the process_2 runs more time, it consumes the temp file. In this situation, it suspends and sends a signal (SIGUSR1) to process_1 to say that process_2 is waiting for process_1 to produce some data to read. If the process_1 produces some data, it sends a signal (SIGUSR2) to process_2. They are communicating each other about temp file with SIGUSR1 and SIGUSR2 signals. After process_1 says to process_2 that there no data, process_2 calculates mean, mean deviation and standard derivation of each error metrics. Then prints them into terminal.

Finally, all processes close the all open files and process_2 removes temp file and input file. There is only left output file.

I did my tests with random lorem ipsum inputs. It works well. Also, I generate a reference output for checking test results. I write a Python script to check differences between reference and current output.



```
oguzhan@ubuntu:~/Desktop/CSE344/Homeworks/HW02_Revision$ make run
./program -i ./input -o ./output
Total bytes read: 3000
Total equation estimated: 150
Incoming signals in critical section: SIGINT = 0

MAE->   Mean: 20.662, Mean Deviation: 5.744, Standard Deviation: 6.917
MSE->   Mean: 751.852, Mean Deviation: 250.164, Standard Deviation: 309.146
RMSE->  Mean: 26.515, Mean Deviation: 5.244, Standard Deviation: 6.986
oguzhan@ubuntu:~/Desktop/CSE344/Homeworks/HW02_Revision$
```