CMPSC 473 project report

1. How to use:

   To run this file, if you are using it in online GDB or Windows IDE (like VScode), simply compile and run the `main.c` should work; If you try to run it in a Linux shell, please use the `cd` command to move to the folder contains `main.c`, and then type `gcc main.c > a.out` for compilation and output file creation; Then in the same shell just type `./a.out`, then you should get the right output from the shell window.

   Be sure to put `main.c`, `input.txt` at the same folder! Otherwise, the program will not work.

   This program reads the data from the `input.txt` in the same folder. To use this program, you have to put the **pid**, **arrival time**, and **burst time** that is separated by space into the `input.txt` as in the test case shown in the below section.

The main process flow is like below:

1. `read_to_arr` will read from `input.txt` to array, this array will be passed to the `init_process_list(double arr[], struct process process_list[], int size)`.

   If you want to use another input file you could modify the underlined part of the function in the below image:

   ```
   double *read_to_arr() {  // This function reads a
       // this function has passed the test, it corr
       FILE *fp = fopen("./input.txt", "r");
       static double result[MAX_LINE_LENGTH] = {0};
       int temp[MAX_LINE_LENGTH] = {0};
   ```

   Simply replace `input.txt` with `desired_name.txt`, and put `desired_name.txt` in the same level of `main.c` should solve all the problems.

2. `init_process_list(double arr[], struct process process_list[], int size)` will initialize the `struct` array: `process_list` by the return array of `read_to_arr`.

3. pass the `process_list` to the `SJF (struct process process_list[], double alpha, int method, int size)`, when `method=1` means use average method when `method=2` means use aging method. The parameter alpha should be in this range: [0,1]. The size

parameter will be calculated automatically in the `main()` function, just put `size` in this parameter.

Parameters of the `SJF` method could be modified in this part of the code:

```
int main() {
    double *input = read_to_arr();
    int size = input_arr_len / 3;  // calculate the size of process_list
    //for (int i = 0; i<size; i++){printf("arr_element %d: %d\n", i, input[i]);}
    printf("length of input array: %d\n", size);
    //printf("size of new struct list: %d\n", size);
    struct process process_list[size];
    init_process_list(input, process_list, size);
    /*for (int i=0; i< size; i++) {
        printf("Struct %d: pid: %d, arrival time: %d, burst time: %d \n", i, process_list[i].pid, process_list[i].
            process_list[i].burst_time);
    }*/
    double* result1 = SJF(process_list, 0.6, 1, size);
    printf("The average waiting time and trunaround time of method 1 is: %f and %f.\n", result1[0], result1[1]);
    double* result2 = SJF(process_list, 0.6, 2, size);
    printf("The average waiting time and trunaround time of method 2 is: %f and %f.\n", result2[0], result2[1]);
}
```

The result will show you the burst time and turnaround time of the method you have chosen. The result will appear in the console like this:

```
read to array, finished
length of input array: 16
Choose to use method 1: average
The average waiting time and trunaround time of method 1 is: 34.375000 and 40.160639.
Choose to use method 2: aging
The average waiting time and trunaround time of method 2 is: 26.625000 and 31.653379.


...Program finished with exit code 0
Press ENTER to exit console.
```

2. testing case:

Input1:

```
0 0 9
1 1 8
2 2 2
3 5 2
3 30 5
1 31 2
2 32 6
0 38 8
```

```
2 60 7
0 62 2
1 65 3
3 66 8
1 90 10
0 95 10
2 98 9
3 99 8
```

alpha = 0.8

```
read to array, finished
length of input array: 16
Choose to use method 1: average
The average waiting time and trunaround time of method 1 is: 34.375000 and 40.160639.
Choose to use method 2: aging
The average waiting time and trunaround time of method 2 is: 28.312500 and 33.518489.


...Program finished with exit code 0
Press ENTER to exit console.
```

alpha = 0.6

```
read to array, finished
length of input array: 16
Choose to use method 1: average
The average waiting time and trunaround time of method 1 is: 34.375000 and 40.160639.
Choose to use method 2: aging
The average waiting time and trunaround time of method 2 is: 26.625000 and 31.653379.


...Program finished with exit code 0
Press ENTER to exit console.
```

alpha = 0.4

```
read to array, finished
length of input array: 16
Choose to use method 1: average
The average waiting time and trunaround time of method 1 is: 34.375000 and 40.160639.
Choose to use method 2: aging
The average waiting time and trunaround time of method 2 is: 23.750000 and 28.332530.


...Program finished with exit code 0
Press ENTER to exit console.
```

alpha = 0.2

```
read to array, finished
length of input array: 16
Choose to use method 1: average
The average waiting time and trunaround time of method 1 is: 34.375000 and 40.160639.
Choose to use method 2: aging
The average waiting time and trunaround time of method 2 is: 17.187500 and 21.049859.


...Program finished with exit code 0
Press ENTER to exit console.
```

Input2:

```
1 0 3
2 1 4
3 2 2
4 5 3
```

alpha = 0.8:

```
read to array, finished
length of input array: 4
Choose to use method 1: average
The average waiting time and trunaround time of method 1 is: 2.250000 and 4.750000.
Choose to use method 2: aging
The average waiting time and trunaround time of method 2 is: 0.500000 and 2.020000.


...Program finished with exit code 0
Press ENTER to exit console.
```

alpha = 0.6:

```
read to array, finished
length of input array: 4
Choose to use method 1: average
The average waiting time and trunaround time of method 1 is: 2.250000 and 4.750000.
Choose to use method 2: aging
The average waiting time and trunaround time of method 2 is: 0.250000 and 1.605000.


...Program finished with exit code 0
Press ENTER to exit console.
```

alpha = 0.4:

```
read to array, finished
length of input array: 4
Choose to use method 1: average
The average waiting time and trunaround time of method 1 is: 2.250000 and 4.750000.
Choose to use method 2: aging
The average waiting time and trunaround time of method 2 is: 0.250000 and 1.130000.


...Program finished with exit code 0
Press ENTER to exit console.
```

alpha = 0.2:

```
read to array, finished
length of input array: 4
Choose to use method 1: average
The average waiting time and trunaround time of method 1 is: 2.250000 and 4.750000.
Choose to use method 2: aging
The average waiting time and trunaround time of method 2 is: 0.000000 and 0.595000.


...Program finished with exit code 0
Press ENTER to exit console.
```

3. Analysis

Below are the formulas used for burst time prediction:

Method 1 (Average): $T_{n+1} = \frac{\sum_1^n t_i}{n}$, where t_n is the actual burst time for the n^{th} process

Method 2 (Aging): $T_{n+1} = \alpha t_n + (1-\alpha)T_n$, α is the smooth factor

From the above test cases, We could tell:

1. Aging methods usually have a better performance than the Average method.
2. Aging methods with a smaller alpha factor have a better performance than a larger alpha factor.
3. At least, the average method is stable in prediction and do not affect by the change of smooth factor.

The value of the alpha factor means the weight of recent operations (in this case, the actual burst time). It seems a small weight on past actual burst time works better than a large weight.

I didn't try this program with an extra-long test input which may provide evidence of the Average method has a better performance than the Aging method; And, I am also not sure that the way I implemented the SJF with the prediction algorithm in the most appropriate way. So, the above result could still be somehow inaccurate.

However, from test cases with different smooth factors, I could tell that the Aging method provides totally better performance than the Average method. Thus, it is reasonable most SJF schedulers choose the Aging method as their default method.

4. Contribution

Name	**Code**	**MoM**	**Debug**	**test cases**	**report and analysi**
Pengwen Zhu	100%	75%	100%	100%	100%

5. Reference

1. https://www.geeksforgeeks.org/shortest-job-first-cpu-scheduling-with-predicted-burst-time/ (the formulas and another test case come from here)
2. https://github.com/doughgle/scheduling-algorithms/blob/master/simulator.py (one test case come from here)

3. https://onlinegdb.com/dkuF4qtB3 (The online GDB I have used, this link guides you to the online source code.)