CS356 Project 4

Name : HaotianXue

Student ID : 518021910506

Email : xavihart@sjtu.edu.cn

Introduction

This is the report for project4 for CS356, also for COS book chapter5. In this project we implement five CPU schedule algorithms: FCFS, SJF, RR, Priority, and RR-Priority, we simulate the schedule process of the CPU and calculate the average waiting time, average turnaround time and average response time respectively.

General Ideas

- We use linked list to save all the process to be served
- For each process we have some properties, including burst, priority, starting time, last end time, end time, start or not
- Once a process is done, we delete it from the list and we use it to check the termination of the program
- Last end time is crucial for implementation of RR-related strategies, since a process can be done in more than one pieces

FCFS

This is the easiest to implement, we just traverse the list from the head and run the process in sequence:

```
Perf* schedule(){
    Perf* results = (Perf*)malloc(sizeof(Perf));
    //traverse(head);
    results -> AveResponseTime = 0;
    results -> AveTurnAroundTime = 0;
    results -> AveWaitingTime = 0;
    int time = 0;
    struct node*now = head;

    while(now){
       traverse(head);
        if(now -> task -> start == 0){
            results -> AveResponseTime += time;
            results -> AveWaitingTime += (time - now ->
task -> ltime);
            time += now -> task -> burst;
            results -> AveTurnAroundTime += time;
            now -> task -> start = 1;
            now -> task -> etime = time;
        }else{
            results -> AveResponseTime += time;
            results -> AveWaitingTime += (time - now ->
task -> ltime);
            time += now -> task -> burst;
            results -> AveTurnAroundTime += time;
            now -> task -> etime = time;
        }
        run(now -> task, now -> task -> burst);
        now = now -> next;
    }

  return results;
}
```

SJF

Each time we traverse the list to find the process with the shortest burst and run it until the list become empty:

```
Perf* schedule(){
    Perf* results = (Perf*)malloc(sizeof(Perf));
    //traverse(head);
    results -> AveResponseTime = 0;
    results -> AveTurnAroundTime = 0;
    results -> AveWaitingTime = 0;
    int time = 0;
    struct node*now = head;

    while(head){
        now = head;
        int shortest_burst = 1e9+7;
        //traverse(head);
        while(now){
            if(now -> task -> burst < shortest_burst)
               shortest_burst = now -> task -> burst;
            now = now -> next;
        }
        now = head;
        while(now){
            if(now -> task -> burst ==  shortest_burst){
                printf("\\");
                results -> AveResponseTime += time;
                results -> AveWaitingTime += (time - now -
> task -> ltime);
                now -> task -> ltime = time;
                time += now -> task -> burst;
                results -> AveTurnAroundTime += time;
                now -> task -> start = 1;
                now -> task -> etime = time;
                run(now -> task, now -> task -> burst);
                delete(&head, now -> task);
                break;
            }else{
                now = now -> next;
                continue;
            }
        }

    }


    return results;
}
```

Priority

It is almost the same as SJF, since we just need to change the comparison into the priority of the process, where in each cycle we choose the remaining process with the highest priority:

```

Perf* schedule(){
    Perf* results = (Perf*)malloc(sizeof(Perf));
    //traverse(head);
    results -> AveResponseTime = 0;
    results -> AveTurnAroundTime = 0;
    results -> AveWaitingTime = 0;
    int time = 0;
    struct node*now = head;

    while(head){
        now = head;
        int prio= -1;
        //traverse(head);
        while(now){
            if(now -> task -> priority > prio)
               prio = now -> task -> priority;
            now = now -> next;
        }
        now = head;
        while(now){
            if(now -> task -> priority ==  prio){
                //printf("\\");
                results -> AveResponseTime += time;
                results -> AveWaitingTime += (time - now -
> task -> ltime);
                now -> task -> ltime = time;
                time += now -> task -> burst;
                results -> AveTurnAroundTime += time;
                now -> task -> start = 1;
                now -> task -> etime = time;
                run(now -> task, now -> task -> burst);
                delete(&head, now -> task);
                break;
            }else{
                now = now -> next;
                continue;
            }
        }

    }


    return results;
}
```

RR

The quantum is set to be 10 in this case. We traverse the list from head and run a piece of the process in sequence, it the remaining burst time is no more than 0, we kick the process out of the list.

It should be noted that we need to use the `last_end_time` to calculate the waiting time for each process:

```
Perf* schedule(){
    Perf* results = (Perf*)malloc(sizeof(Perf));
    //traverse(head);
    results -> AveResponseTime = 0;
    results -> AveTurnAroundTime = 0;
    results -> AveWaitingTime = 0;
    int time = 0;
    struct node*now = head;

    while(head){
        now = head;
        while(now){
            if(now -> task -> start == 0){
                now -> task -> start = 1;
```

```
                    int bst = now -> task -> burst;
                    int time_csmd = (bst > quantum) ? quantum
: bst;
                    results -> AveResponseTime += (time - 0);
                    results -> AveWaitingTime += (time - 0);
                    now -> task -> burst -= time_csmd;
                    time += time_csmd;
                    now -> task -> ltime = time;
                    run(now -> task, time_csmd);
                }else{
                    int bst = now -> task -> burst;
                    int time_csmd = (bst > quantum) ? quantum
: bst;
                    results -> AveWaitingTime += (time - now -
> task -> ltime);
                    now -> task -> burst -= time_csmd;
                    time += time_csmd;
                    now -> task -> ltime = time;
                    run(now -> task, time_csmd);
                }

                if(now -> task -> burst == 0){
                    results ->AveTurnAroundTime += time;
                    delete(&head, now -> task);
                }

                now = now -> next;
            }
        }

    return results;
}
```

RR-Priority

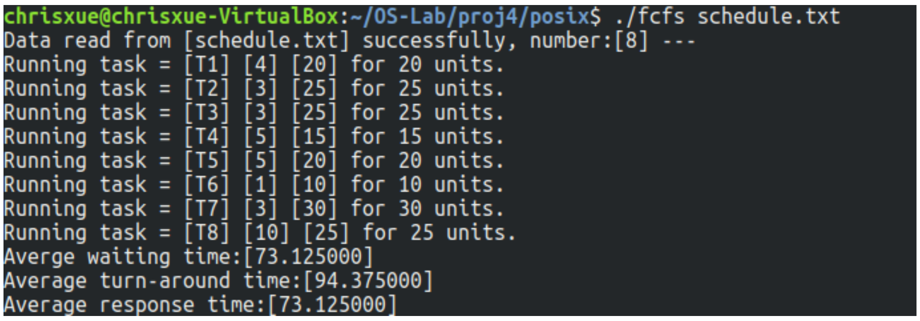
It is a little complex than other schedule strategies, and we should use three levels of while loop to do this: in the first level we decide if the list is empty and choose the highest priority ; in the second level we run in cycle until no process with the highest priority exist; in the third level we traverse the list repeatedly to run processes with the highest priority.

```
Perf* schedule(){
    Perf* results = (Perf*)malloc(sizeof(Perf));
    //traverse(head);
    results -> AveResponseTime = 0;
    results -> AveTurnAroundTime = 0;
    results -> AveWaitingTime = 0;
    int time = 0;
    struct node*now = head;

    while(head){
        now = head;
        int prio=-1, f;
        while(now){
            if(now -> task -> priority > prio)
               prio = now -> task -> priority;
            now = now -> next;
        }

        while(1){
            f = 0;
            now = head;
            while(now){
               if(now -> task -> priority == prio){
                   f = 1;
                   if(now -> task -> start == 0){
                        now -> task -> start = 1;
                        int bst = now -> task -> burst;
                        int time_csmd = (bst > quantum) ?
quantum : bst;
                        results -> AveResponseTime +=
(time - 0);
                        results -> AveWaitingTime += (time
- 0);
                        now -> task -> burst -= time_csmd;
                        time += time_csmd;
                        now -> task -> ltime = time;
                        run(now -> task, time_csmd);
                   }else{
                        int bst = now -> task -> burst;
                        int time_csmd = (bst > quantum) ?
quantum : bst;
                        results -> AveWaitingTime += (time
- now -> task -> ltime);
                        now -> task -> burst -= time_csmd;
                        time += time_csmd;
                        now -> task -> ltime = time;
                        run(now -> task, time_csmd);
                   }

                   if(now -> task -> burst == 0){
                        results ->AveTurnAroundTime +=
time;
                        delete(&head, now -> task);
                   }
               }

                now = now -> next;
            }

            if(!f) break;
        }
```

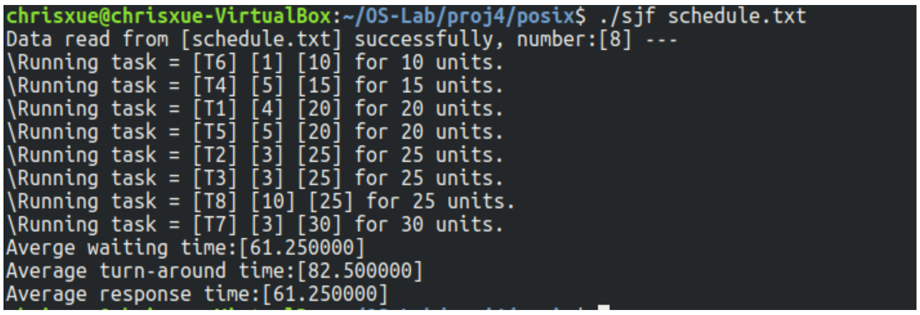
```
    }


    return results;
}
```

Test for correctness

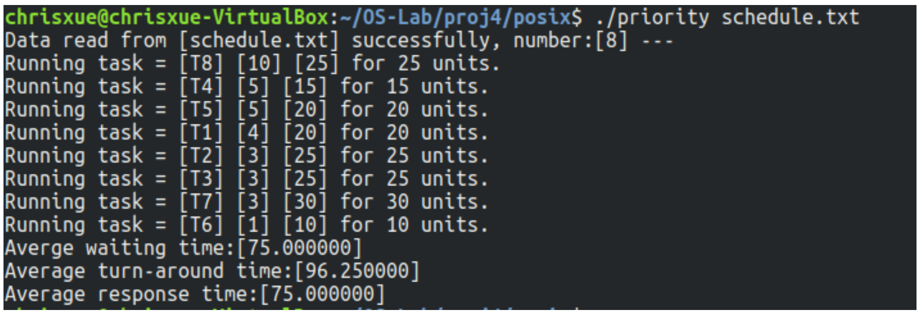
We use process from the source code of the book to test:



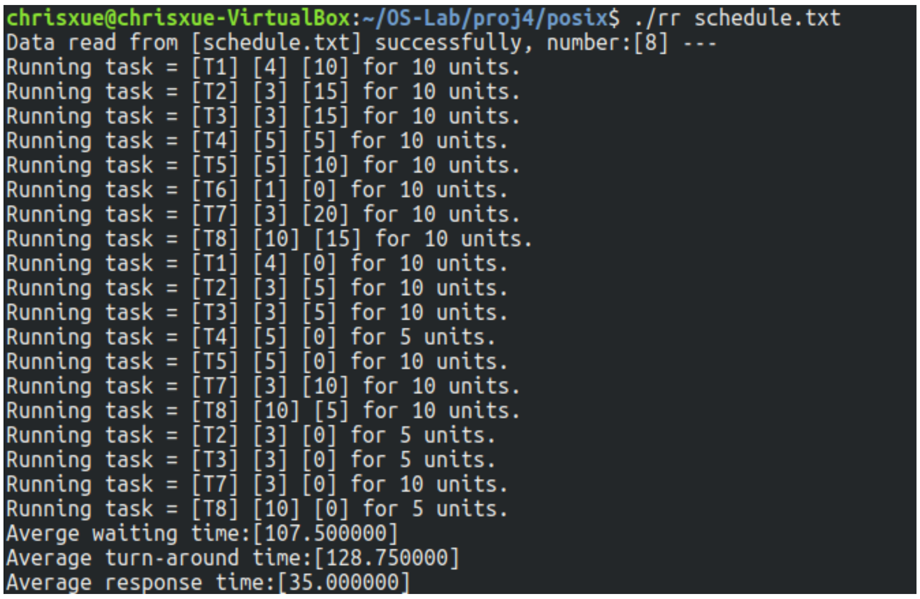
```
chrisxue@chrisxue-VirtualBox:~/OS-Lab/proj4/posix$ ./fcfs schedule.txt
Data read from [schedule.txt] successfully, number:[8] ---
Running task = [T1] [4] [20] for 20 units.
Running task = [T2] [3] [25] for 25 units.
Running task = [T3] [3] [25] for 25 units.
Running task = [T4] [5] [15] for 15 units.
Running task = [T5] [5] [20] for 20 units.
Running task = [T6] [1] [10] for 10 units.
Running task = [T7] [3] [30] for 30 units.
Running task = [T8] [10] [25] for 25 units.
Averge waiting time:[73.125000]
Average turn-around time:[94.375000]
Average response time:[73.125000]
```



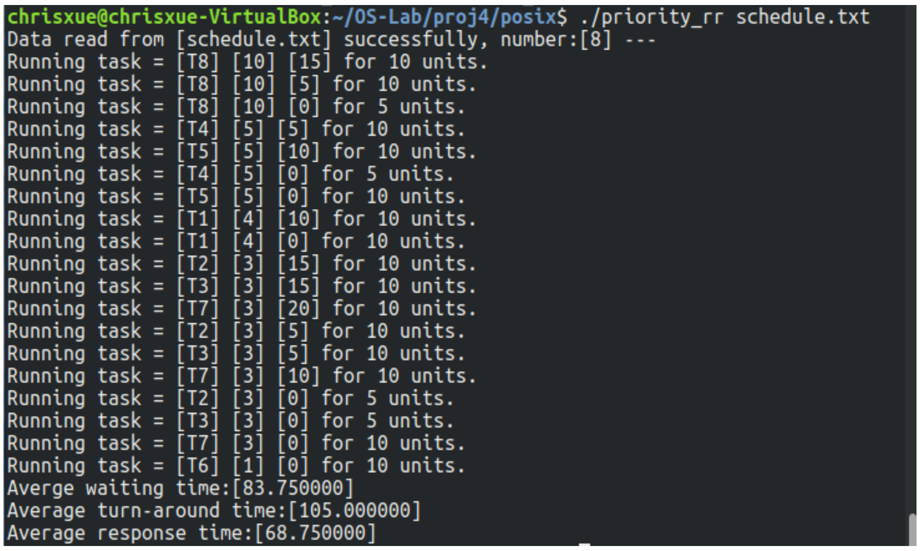
```
chrisxue@chrisxue-VirtualBox:~/OS-Lab/proj4/posix$ ./sjf schedule.txt
Data read from [schedule.txt] successfully, number:[8] ---
\Running task = [T6] [1] [10] for 10 units.
\Running task = [T4] [5] [15] for 15 units.
\Running task = [T1] [4] [20] for 20 units.
\Running task = [T5] [5] [20] for 20 units.
\Running task = [T2] [3] [25] for 25 units.
\Running task = [T3] [3] [25] for 25 units.
\Running task = [T8] [10] [25] for 25 units.
\Running task = [T7] [3] [30] for 30 units.
Averge waiting time:[61.250000]
Average turn-around time:[82.500000]
Average response time:[61.250000]
```



```
chrisxue@chrisxue-VirtualBox:~/OS-Lab/proj4/posix$ ./priority schedule.txt
Data read from [schedule.txt] successfully, number:[8] ---
Running task = [T8] [10] [25] for 25 units.
Running task = [T4] [5] [15] for 15 units.
Running task = [T5] [5] [20] for 20 units.
Running task = [T1] [4] [20] for 20 units.
Running task = [T2] [3] [25] for 25 units.
Running task = [T3] [3] [25] for 25 units.
Running task = [T7] [3] [30] for 30 units.
Running task = [T6] [1] [10] for 10 units.
Averge waiting time:[75.000000]
Average turn-around time:[96.250000]
Average response time:[75.000000]
```



```
chrisxue@chrisxue-VirtualBox:~/OS-Lab/proj4/posix$ ./rr schedule.txt
Data read from [schedule.txt] successfully, number:[8] ---
Running task = [T1] [4] [10] for 10 units.
Running task = [T2] [3] [15] for 10 units.
Running task = [T3] [3] [15] for 10 units.
Running task = [T4] [5] [5] for 10 units.
Running task = [T5] [5] [10] for 10 units.
Running task = [T6] [1] [0] for 10 units.
Running task = [T7] [3] [20] for 10 units.
Running task = [T8] [10] [15] for 10 units.
Running task = [T1] [4] [0] for 10 units.
Running task = [T2] [3] [5] for 10 units.
Running task = [T3] [3] [5] for 10 units.
Running task = [T4] [5] [0] for 5 units.
Running task = [T5] [5] [0] for 10 units.
Running task = [T7] [3] [10] for 10 units.
Running task = [T8] [10] [5] for 10 units.
Running task = [T2] [3] [0] for 5 units.
Running task = [T3] [3] [0] for 5 units.
Running task = [T7] [3] [0] for 10 units.
Running task = [T8] [10] [0] for 5 units.
Averge waiting time:[107.500000]
Average turn-around time:[128.750000]
Average response time:[35.000000]
```



```
chrisxue@chrisxue-VirtualBox:~/OS-Lab/proj4/posix$ ./priority_rr schedule.txt
Data read from [schedule.txt] successfully, number:[8] ---
Running task = [T8] [10] [15] for 10 units.
Running task = [T8] [10] [5] for 10 units.
Running task = [T8] [10] [0] for 5 units.
Running task = [T4] [5] [5] for 10 units.
Running task = [T5] [5] [10] for 10 units.
Running task = [T4] [5] [0] for 5 units.
Running task = [T5] [5] [0] for 10 units.
Running task = [T1] [4] [10] for 10 units.
Running task = [T1] [4] [0] for 10 units.
Running task = [T2] [3] [15] for 10 units.
Running task = [T3] [3] [15] for 10 units.
Running task = [T7] [3] [20] for 10 units.
Running task = [T2] [3] [5] for 10 units.
Running task = [T3] [3] [5] for 10 units.
Running task = [T7] [3] [10] for 10 units.
Running task = [T2] [3] [0] for 5 units.
Running task = [T3] [3] [0] for 5 units.
Running task = [T7] [3] [0] for 10 units.
Running task = [T6] [1] [0] for 10 units.
Averge waiting time:[83.750000]
Average turn-around time:[105.000000]
Average response time:[68.750000]
```

Conclusion

This project helps me review the five CPU schedule strategies, and we implement the five strategies FCFS, SJF, RR, Priority, and RR-Priority . Also it helps me learn more about c program in linux.

Thanks for the instructions and useful help offered by Prof. Wu and all the TAs!