CS307 Project3: Multi-threaded Sorting Application & Fork-Join Sorting Application

Junjie Wang 517021910093

May 23, 2019

1 Programming Thoughts

The main logic here is to use the link list to store all the tasks first and then implement different algorithms according to their own conditions. A point that worths attention here is when we construct the link list from the input file(schedule.txt), the task with larger id will be closer to the head. So usually we need to locate to the tail of the link list first.
FCFS: Quite easy. Simply move to the tail of the link list and run the task at that place.
SJF: Each time we traverse the link list, we find the task with the minumum burst time, then running that task.
PRIORITY: Each time we traverse the link list, we find the task with the largest priority number(higher priority according to the rule) and execute that task.
ROUND ROBIN(RR): First we reverse the chain to make tasks with smaller id closer to the head. Then each time we traverse the chain, we check whether the remaining burst time of a task is smaller than the time quantum. If yes, we execute that job with all its remaining burst time and remove that job from the new link list.If no, we execute that task for a slice of time quantum.
PRIORITY ROUND ROBIN: This is probably the hardest part of these five algorithms. First we also reverse the link list. Then each time we traverse the new link list, we find the tasks with the maximum priority number and record how many tasks have this maximum priority number. If more than one, we use round robin strategy among them. If exactly one, we execute that one task and remove it from the link list.

2 Execution Results And Snapshots

The execution results of these five algorithms are listed as follows:

```
dreamboy@Elon_Mask:~/OSProjects/project4$ ./fcfs schedule.txt
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.
dreamboy@Elon_Mask:~/OSProjects/project4$
```

Figure 1: FCFS

```
dreamboy@Elon_Mask:~/OSProjects/project4$ ./sjf schedule.txt
Running task = [T6] [1] [10] for 10 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 = [T8] [10] [25] for 25 units.
Running task = [T3] [3] [25] for 25 units.
Running task = [T2] [3] [25] for 25 units.
Running task = [T7] [3] [30] for 30 units.
dreamboy@Elon_Mask:~/OSProjects/project4$
```

Figure 2: SJF

```
dreamboy@Elon_Mask:~/OSProjects/project4$ ./priority schedule.txt
Running task = [T8] [10] [25] for 25 units.
Running task = [T5] [5] [20] for 20 units.
Running task = [T4] [5] [15] for 15 units.
Running task = [T1] [4] [20] for 20 units.
Running task = [T7] [3] [30] for 30 units.
Running task = [T3] [3] [25] for 25 units.
Running task = [T2] [3] [25] for 25 units.
Running task = [T6] [1] [10] for 10 units.
dreamboy@Elon_Mask:~/OSProjects/project4$
```

Figure 3: Priority

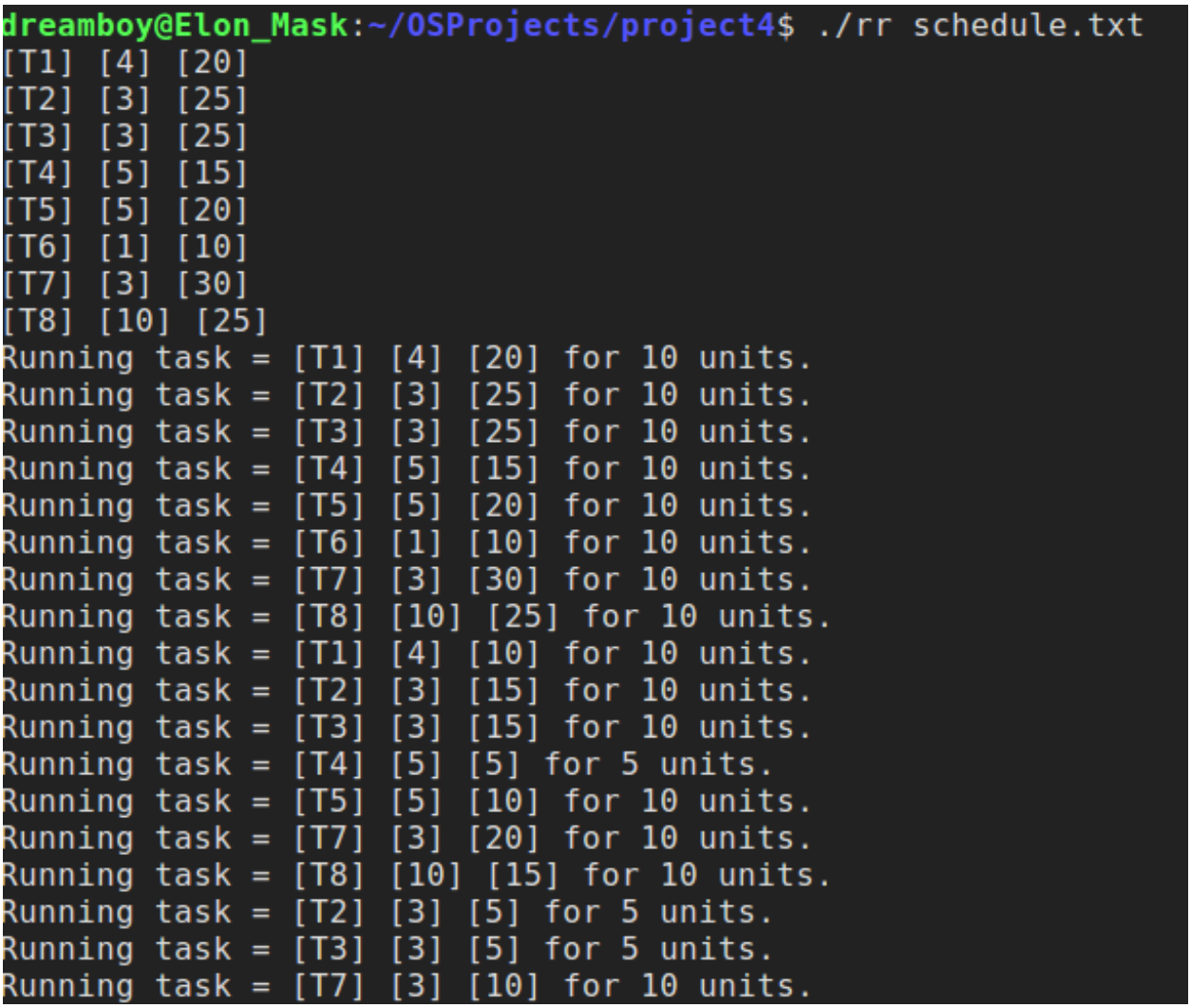

Figure 4: Round Robin

```
dreamboy@Elon_Mask:~/OSProjects/project4$ ./priority_rr schedule.txt
Running task = [T4] [5] [15] for 10 units.
Running task = [T5] [5] [20] for 10 units.
Running task = [T4] [5] [5] for 5 units.
Running task = [T5] [5] [10] for 10 units.
Running task = [T1] [4] [20] for 20 units.
Running task = [T2] [3] [25] for 10 units.
Running task = [T3] [3] [25] for 10 units.
Running task = [T7] [3] [30] 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 5 units.
Running task = [T3] [3] [5] for 5 units.
Running task = [T7] [3] [10] for 10 units.
Running task = [T6] [1] [10] for 10 units.
dreamboy@Elon_Mask:~/OSProjects/project4$
```

Figure 5: Priority Round Robin

3 Code Explanation

For brevity here I only show the implementation of the priority round robin. First we reverse the link list and use the headNew pointer we get to implement the core logic.

```
void schedule(){
/* implement the priority round robin schedule algorithm
* first reverse the list */
struct node **headNew = malloc(sizeof(struct node *));
/* must maually set it to NULL */
*headNew = NULL;

Task *task = malloc(sizeof(Task));
struct node *newNode;
struct node **p = malloc(sizeof(struct node *));
while((*head)->next!=NULL){
task = (*head) -> task;
newNode = malloc(sizeof(struct node));
newNode -> task = task;
newNode -> next = *headNew;
*headNew = newNode;
*head = (*head)-> next;
}
task = (*head) -> task;
newNode = malloc(sizeof(struct node));
newNode -> task = task;
newNode -> next = *headNew;
*headNew = newNode;
```

```
/* implement the priority RR based on the headNew
* first use an array to store the priority counts */
*p = *headNew;
int counts[MAX_PRIORITY+1], i;
for(i=0;i<=MAX_PRIORITY;i++) counts[i] = 0;
while((*p)->next != NULL){
task = (*p) -> task;
counts[task->priority]++;
*p = (*p) -> next;
}

i = MAX_PRIORITY;
while(1){
for(;counts[i] == 0 && i>0; i--) ;
if(i == 0) break;

if(counts[i] == 1){
/* with only one task, we just execute it */
*p = *headNew;
while((*p)->next != NULL){
if((*p) -> task -> priority == i){
run((*p)->task, (*p)->task->burst);
delete(headNew, (*p)->task);
break;
}
*p = (*p) -> next;
}
counts[i] = 0;
}else{
/* with more than one task, we use the round robin */
*p = *headNew;
while(counts[i]>0){
if((*p)->task->priority == i){
task = (*p) -> task;
if(task->burst <= RR_TIME){
/* with time less than the limit */
run(task, task->burst);
/* move on */
delete(headNew, task);
counts[i]--;
}else{
/* with time more than the Q */
run(task, RR_TIME);
task->burst -= RR_TIME;
}
}
```

```
if((*p)->next == NULL) *p = *headNew;
else *p = (*p) -> next;
}
}
}
free(newNode);
free(headNew);
free(p);
free(task);
}
```