Scheduling Algorithm

Overview

This project involves implementing several different process scheduling algorithms. The scheduler will be assigned a predefined set of tasks and will schedule the tasks based on the selected scheduling algorithm. Each task is assigned a priority and CPU burst. The following scheduling algorithms will be implemented:

- **First-come,first-served(FCFS)**,which schedules tasks in the order in which they request the CPU.
- **Shortest-job-first (SJF)**, which schedules tasks in order of the length of the tasks' next CPU burst.
- **Priority scheduling,** which schedules tasks based on priority.
- **Round-robin (RR) scheduling**, where each task is run for a time quantum (or for the remainder of its CPU burst).
- **Priority with round-robin**, which schedules tasks in order of priority and uses round-robin scheduling for tasks with equal priority. Priorities range from 1 to 10, where a higher numeric value indicates a higher relative priority. For round-robin scheduling, the length of a time quantum is 10 milliseconds.

Program Structure

The file `driver.c` reads in the schedule of tasks, inserts each task into a linked list,and invokes the process scheduler by calling the `schedule()` function.The schedule() function executes each task according to the specified scheduling algorithm.Tasks selected for execution on the CPU are executed by invoking the run() function defined in the `CPU.c` file. A `Makefile` is used to determine the specific scheduling algorithm that will be invoked by driver.

Moreover, I write a `vector.c` to implement some basic operations on linked list. Detail can be seen in the source code.

```
//vector.h
/*
    Simple vector to implement the link list.
*/

#ifndef VECTOR_H
#define VECTOR_H


#include "task.h"
struct node* head;
struct node* tail;

#define bool int
#define true 1
#define false 0


struct node {
    Task *task;
```

```
    struct node *next;
};
void addTohead(struct task* newTask);
void insert(struct node* p, struct task* newTask);
void addTotail(struct task* newTask);
bool isEmpty();
void deleteHead();
void moveTotail();
void delete(struct node*p);

#endif
```

Algorithm implements

FCFS

```
void add(char *name, int priority, int burst)
{
    struct task* newTask = malloc(sizeof(struct task));
    newTask->name = name;
    newTask->priority = priority;
    newTask->burst = burst;
    addTotail(newTask);
}


void schedule()
{
    while(!isEmpty())
    {
        run(head->next->task, head->next->task->burst);
        deleteHead();
    }
}
```

SJF

When adding the task to the scheduler, place it in the proper place to satisfy the rule, shortest time first.

```
void add(char *name, int priority, int burst)
{
    struct task* newTask = malloc(sizeof(struct task));
    newTask->name = name;
    newTask->priority = priority;
    newTask->burst = burst;


    struct node *p = head->next;
```

```
    if(!p || p->task->burst >= newTask->burst)
    {
        addTohead(newTask);
    }
    else
    {
        struct node* tmp;
        while(p)
        {
            //printf("make");
            tmp = p->next;
            if(!tmp)    addTotail(newTask);
            else if(tmp->task->burst >= newTask->burst)
            {
                insert(p,newTask);
                break;
            }
            p = tmp;
        }

    }


}

void schedule()
{
    while(!isEmpty())
    {
        run(head->next->task, head->next->task->burst);
        deleteHead();
    }
}
```

Priority

When adding the task to the scheduler, place it in the proper place to satisfy the rule, priority first.

```
void add(char *name, int priority, int burst)
{
    struct task* newTask = malloc(sizeof(struct task));
    newTask->name = name;
    newTask->priority = priority;
    newTask->burst = burst;


    struct node *p = head->next;

    if(!p || p->task->priority < newTask->priority)
    {
        addTohead(newTask);
    }
    else
```

```
    {
        struct node* tmp;
        while(p)
        {
            //printf("make");
            tmp = p->next;
            if(!tmp)    addTotail(newTask);
            else if(tmp->task->priority < newTask->priority)
            {
                insert(p,newTask);
                break;
            }
            p = tmp;
        }

    }


}


void schedule()
{
    while(!isEmpty())
    {
        run(head->next->task, head->next->task->burst);
        deleteHead();
    }
}
```

Round Robin

Time slice is 5

```
void add(char *name, int priority, int burst)
{
    struct task* newTask = malloc(sizeof(struct task));
    newTask->name = name;
    newTask->priority = priority;
    newTask->burst = burst;
    //printf(name);
    addTotail(newTask);
    //addTohead(newTask);
}



void schedule()
{
    int time = 0;
    while(!isEmpty())
    {
        time = min(5,head->next->task->burst);
```

```
        run(head->next->task, time);
        head->next->task->burst -= time;
        if(head->next->task->burst == 0)
            deleteHead();
        else
            moveTotail();
    }
}
```

Priority with RR

When adding the task to the scheduler, place it in the proper place to satisfy the rule, priority first. Then apply RR to the tasks with same priority until the list is empty.

```
void add(char *name, int priority, int burst)
{
    struct task* newTask = malloc(sizeof(struct task));
    newTask->name = name;
    newTask->priority = priority;
    newTask->burst = burst;


    struct node *p = head->next;

    if(!p || p->task->priority < newTask->priority)
    {
        addTohead(newTask);
    }
    else
    {
        struct node* tmp;
        while(p)
        {
            //printf("make");
            tmp = p->next;
            if(!tmp)    addTotail(newTask);
            else if(tmp->task->priority < newTask->priority)
            {
                insert(p,newTask);
                break;
            }
            p = tmp;
        }

    }


}


void schedule()
{
    while(!isEmpty())
```

```
    {
        struct node *end;
        end = head->next;
        int priority = end->task->priority;
        // find tasks that have same priority
        while(end->next)
        {
            if(end->next->task->priority == priority)
                end =end->next;
            else
                break;
        }

        struct node* tmp1 = tail;
        struct node* tmp2 = end->next;
        tail = end;
        tail->next = NULL;

        int time = 0;
        while(!isEmpty())
        {
            time = min(10,head->next->task->burst);
            run(head->next->task, time);
            head->next->task->burst -= time;
            if(head->next->task->burst == 0)
                deleteHead();
            else
                moveTotail();
        }

        if(!tmp2)   break;
        else
        {
            head->next = tmp2;
            tail = tmp1;
        }
    }
}
```

Result show

```
gavin@ubuntu:~/Documents/Project4$ ./fcfs schedule.txt
[Task = T1] [Priority = 4] [Executing time = 20]
[Task = T2] [Priority = 3] [Executing time = 25]
[Task = T3] [Priority = 3] [Executing time = 25]
[Task = T4] [Priority = 5] [Executing time = 15]
[Task = T5] [Priority = 5] [Executing time = 20]
[Task = T6] [Priority = 1] [Executing time = 10]
[Task = T7] [Priority = 3] [Executing time = 30]
[Task = T8] [Priority = 10] [Executing time = 25]
gavin@ubuntu:~/Documents/Project4$ ./sjf schedule.txt
[Task = T6] [Priority = 1] [Executing time = 10]
[Task = T4] [Priority = 5] [Executing time = 15]
[Task = T5] [Priority = 5] [Executing time = 20]
[Task = T1] [Priority = 4] [Executing time = 20]
[Task = T8] [Priority = 10] [Executing time = 25]
[Task = T3] [Priority = 3] [Executing time = 25]
[Task = T2] [Priority = 3] [Executing time = 25]
[Task = T7] [Priority = 3] [Executing time = 30]
gavin@ubuntu:~/Documents/Project4$ ./priority schedule.txt
[Task = T8] [Priority = 10] [Executing time = 25]
[Task = T4] [Priority = 5] [Executing time = 15]
[Task = T5] [Priority = 5] [Executing time = 20]
[Task = T1] [Priority = 4] [Executing time = 20]
[Task = T2] [Priority = 3] [Executing time = 25]
[Task = T3] [Priority = 3] [Executing time = 25]
[Task = T7] [Priority = 3] [Executing time = 30]
[Task = T6] [Priority = 1] [Executing time = 10]
gavin@ubuntu:~/Documents/Project4$ ./rr schedule.txt
[Task = T1] [Priority = 4] [Executing time = 5]
[Task = T2] [Priority = 3] [Executing time = 5]
[Task = T3] [Priority = 3] [Executing time = 5]
[Task = T4] [Priority = 5] [Executing time = 5]
[Task = T5] [Priority = 5] [Executing time = 5]
[Task = T6] [Priority = 1] [Executing time = 5]
[Task = T7] [Priority = 3] [Executing time = 5]
[Task = T8] [Priority = 10] [Executing time = 5]
[Task = T1] [Priority = 4] [Executing time = 5]
[Task = T2] [Priority = 3] [Executing time = 5]
[Task = T3] [Priority = 3] [Executing time = 5]
[Task = T4] [Priority = 5] [Executing time = 5]
[Task = T5] [Priority = 5] [Executing time = 5]
[Task = T6] [Priority = 1] [Executing time = 5]
[Task = T7] [Priority = 3] [Executing time = 5]
[Task = T8] [Priority = 10] [Executing time = 5]
[Task = T1] [Priority = 4] [Executing time = 5]
[Task = T2] [Priority = 3] [Executing time = 5]
[Task = T3] [Priority = 3] [Executing time = 5]
[Task = T4] [Priority = 5] [Executing time = 5]
[Task = T5] [Priority = 5] [Executing time = 5]
[Task = T7] [Priority = 3] [Executing time = 5]
[Task = T8] [Priority = 10] [Executing time = 5]
[Task = T1] [Priority = 4] [Executing time = 5]
[Task = T2] [Priority = 3] [Executing time = 5]
[Task = T3] [Priority = 3] [Executing time = 5]
```

```
[Task = T5] [Priority = 5] [Executing time = 5]
[Task = T7] [Priority = 3] [Executing time = 5]
[Task = T8] [Priority = 10] [Executing time = 5]
[Task = T2] [Priority = 3] [Executing time = 5]
[Task = T3] [Priority = 3] [Executing time = 5]
[Task = T7] [Priority = 3] [Executing time = 5]
[Task = T8] [Priority = 10] [Executing time = 5]
[Task = T7] [Priority = 3] [Executing time = 5]
```

```
gavin@ubuntu:~/Documents/Project4$ ./rr_priority schedule.txt
[Task = T8] [Priority = 10] [Executing time = 10]
[Task = T8] [Priority = 10] [Executing time = 10]
[Task = T8] [Priority = 10] [Executing time = 5]
[Task = T4] [Priority = 5] [Executing time = 10]
[Task = T5] [Priority = 5] [Executing time = 10]
[Task = T4] [Priority = 5] [Executing time = 5]
[Task = T5] [Priority = 5] [Executing time = 10]
[Task = T1] [Priority = 4] [Executing time = 10]
[Task = T1] [Priority = 4] [Executing time = 10]
[Task = T2] [Priority = 3] [Executing time = 10]
[Task = T3] [Priority = 3] [Executing time = 10]
[Task = T7] [Priority = 3] [Executing time = 10]
[Task = T2] [Priority = 3] [Executing time = 10]
[Task = T3] [Priority = 3] [Executing time = 10]
[Task = T7] [Priority = 3] [Executing time = 10]
[Task = T2] [Priority = 3] [Executing time = 5]
[Task = T3] [Priority = 3] [Executing time = 5]
[Task = T7] [Priority = 3] [Executing time = 10]
[Task = T6] [Priority = 1] [Executing time = 10]
```

How to run the program

To compile files, enter `make scheduler_name`

To run it, enter `./scheduler_name schedule.txt`

Scheduler names are `fcfs/sjf/priority/robin_round/rr_priority`