CPU SCHEDULER

CECS 326 - OPERATING SYSTEM

PROJECT 4 REPORT

PREPARED BY:

- THANH NGUYEN
- FIONA LE

PREPARED FOR: PROF. HAILU XU

DATE: 05/09/2021

Name: Thanh Nguyen, Fiona Le
CECS 326
Professor Hailu Xu
Project 4 Report
Due date: 05/09/2021

Table Of Contents

Name: Thanh Nguyen, Fiona Le
CECS 326
Professor Hailu Xu
Project 4 Report
Due date: 05/09/2021

I. Task List

```
/**
 * list data structure containing the tasks in the system
 */

#include "task.h"

struct node {
    Task *task;
    struct node *next;
};

// insert and delete operations.
void insert(struct node **head, Task *task);
void delete(struct node **head, Task *task);
void traverse(struct node *head);
```

- It is the list of tasks which need to be scheduled.
- We use **linked-list** for this type of list where each struct node is defined as one task in the system. Each **node** will contains **2 values**: one is ***Task* task*** including the information about the task like name, ip, priority and burst time, and another is ***struct node* next*** is the pointer points to the node or the task next to it.
- This linked-list has 3 basic functions which are insert, delete and traverse.
    - ***Insert function:*** It uses **head** as the beginning node of the list. Whenever one new task is added into the list, the head will point to the address of this new node, then the next field of head will point to the address of the old head node. It means the new task will be placed in front of the previous task.
    - ***Delete function:*** It also uses head as the first node of the list. Then it will go over each node inside the list using the next field, then check the name of each task. If the name is matched then delete the task from the list. It is important to separately check the head, if the head is deleted then move the pointer to the next node.
    - ***Traverse function:*** Go over the list and print its task information.
- For 3 algorithms, we usually use insert and delete functions to do CPU scheduling, so it is necessary to know about these functions' roles and structures.

Name: Thanh Nguyen, Fiona Le
CECS 326
Professor Hailu Xu
Project 4 Report
Due date: 05/09/2021

II. Code

A. First-come first-served (FCFS)

1. Code

```
/**
 * FCFS scheduling
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include "task.h"
#include "list.h"
#include "CPU.h"
#include "schedulers.h"

static struct node* taskList = NULL;

 // add a task to the list
void add(char* name, int priority, int burst)
{
    // create a new task based on the parameters
    Task* newTask = malloc(sizeof(Task));
    newTask->priority = priority;
    newTask->burst = burst;
    // copy the name of task to store in the list
    newTask->name = malloc(sizeof(char) * (strlen(name) + 1));
    strcpy(newTask->name, name);

    // add the new task to list, the new task will be placed before the
head
    insert(&taskList, newTask);
}

// pick the next Task based on FCFS algorithm
Task* pickNextTask()
{
    // get the latest task from the list
    struct node* tail = taskList;

    while (tail->next)
    {
        tail = tail->next;
    }

    return tail->task;
}
```

Name: Thanh Nguyen, Fiona Le
CECS 326
Professor Hailu Xu
Project 4 Report
Due date: 05/09/2021

```
// invoke the scheduler
void schedule()
{
    while (taskList)
    {
        Task* nextTask = pickNextTask();
        run(nextTask, nextTask->burst);
        delete(&taskList, nextTask);
    }
}
```

2. Explanation

- We use 1 local object to do this type of scheduler, which is a **taskList** - *the list of tasks in the system*
- **add function:**
  - We add the tasks to the list **taskList** as the same order like the Schedule.txt provides. The new task will be placed before the old task.
  - All the information of the new task will be copied into the list.
- **pickNextTask function:** We will find the next task which needs to be scheduled next.
  - The algorithm is FCFS, the function pickNextTask() picks the latest task (the past node) in the list and assigns it to the next task.
- **scheduler function:**
  - At each time, we use pickNextTask() function to get the next task which needs to be scheduled next.
  - After the task is executed, it will be deleted from the taskList.
  - Continue these steps until the list is empty, which means all the tasks are scheduled.

B. Priority

1. Code

```
/*
 * Priority scheduling
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include "task.h"
#include "list.h"
#include "CPU.h"
#include "schedulers.h"
```

Name: Thanh Nguyen, Fiona Le
CECS 326
Professor Hailu Xu
Project 4 Report
Due date: 05/09/2021

```c
static struct node* taskList = NULL;

// add a task to the list
void add(char* name, int priority, int burst)
{
    // create a new task based on the parameters
    Task* newTask = malloc(sizeof(Task));
    newTask->priority = priority;
    newTask->burst = burst;
    // copy the name of task to store in the list
    newTask->name = malloc(sizeof(char) * (strlen(name) + 1));
    strcpy(newTask->name, name);

    // add the new task to list, the new task will be placed before the
head
    insert(&taskList, newTask);
}

// pick the next Task based on Priority algorithm
Task* pickNextTask()
{
    // get the task has the highest priority (smallest number)

    // assign the head is the task has highest priority
    struct node* currentTask = taskList;
    Task* highestPriorityTask = currentTask->task;

    // check the other tasks
    currentTask = currentTask->next;

    while (currentTask)
    {
        if (currentTask->task->priority < highestPriorityTask->priority)
            highestPriorityTask = currentTask->task;
        currentTask = currentTask->next;
    }

    return highestPriorityTask;
}

// invoke the scheduler
void schedule()
{
    while (taskList)
    {
        Task* nextTask = pickNextTask();
        run(nextTask, nextTask->burst);
        delete(&taskList, nextTask);
    }
}
```

Name: Thanh Nguyen, Fiona Le
CECS 326
Professor Hailu Xu
Project 4 Report
Due date: 05/09/2021

2. Explanation

- We use 1 local object to do this type of scheduler, which is **taskList** - *the list of tasks in the system*
- **add function:**
  - We add the tasks to the list **taskList** as the same order like the Schedule.txt provides. The new task will be placed before the old task.
  - All the information of the new task will be copied into the list.
- **pickNextTask function:** We will find the next task which needs to be scheduled next.
  - Because the scheduler prioritizes the task which has the highest priority, hence the pickNextTask function will pick the task to execute next based on its priority number.
  - The smallest priority number, for example the task with priority number 1 will be scheduled first.
  - We will go over the list and choose the task with the smallest priority number, then return this task as the next task needs to be scheduled.
- **scheduler function:**
  - At each time, we use pickNextTask() function to get the next task which needs to be scheduled next. This task will has the smallest priority number.
  - After the task is executed, it will be deleted from the taskList.
  - Continue these steps until the list is empty, which means all the tasks are scheduled.

C. Round-Robin (RR)

1. Code

```
/**
 * Round-robin scheduling
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include "task.h"
#include "list.h"
#include "CPU.h"
#include "schedulers.h"

static struct node* taskList = NULL;
static struct node* currentNode = NULL;
```

Name: Thanh Nguyen, Fiona Le
CECS 326
Professor Hailu Xu
Project 4 Report
Due date: 05/09/2021

```
// add a task to the list
void add(char* name, int priority, int burst)
{
    // create a new task based on the parameters
    Task* newTask = malloc(sizeof(Task));
    newTask->priority = priority;
    newTask->burst = burst;
    // copy the name of task to store in the list
    newTask->name = malloc(sizeof(char) * (strlen(name) + 1));
    strcpy(newTask->name, name);

    // add the new task to list, the new task will be placed before the
head
    insert(&taskList, newTask);
}

struct node* getLastNode()
{
    struct node* tail = taskList;

    while(tail->next)
        tail = tail->next;

    return tail;
}

struct node* findNextNode(Task* currentTask)
{
    // if the list only has one node
    if (!taskList->next)
        return taskList;
    else
    {
        struct node* temp = taskList;
    struct node* prev;
        // interior or last element in the list
        prev = temp;
        temp = temp->next;
        while (strcmp(currentTask->name,temp->task->name) != 0)
        {
            prev = temp;
            temp = temp->next;
        }

        return prev;
    }
}

// pick the next Task based on RR algorithm
```

Name: Thanh Nguyen, Fiona Le
CECS 326
Professor Hailu Xu
Project 4 Report
Due date: 05/09/2021

```
Task* pickNextTask()
{
    Task* nextTask = currentNode->task;
    // if the node is not the first node in the list => currentNode goes to
the next one
    if (strcmp(currentNode->task->name,taskList->task->name) != 0)
        currentNode = findNextNode(currentNode->task);
    // if the node is the first node in the list => currentNode begins at
the tail
    else
        currentNode = getLastNode();

    return nextTask;
}

// invoke the scheduler
void schedule()
{
    int burstTime;
    currentNode = getLastNode();
    while (taskList)
    {
        Task* nextTask = pickNextTask();
        if (QUANTUM <= nextTask->burst)
            burstTime = QUANTUM;
        else
            burstTime = nextTask->burst;
        run(nextTask, burstTime);
        nextTask->burst -= burstTime;
        if (!nextTask->burst)
            delete(&taskList, nextTask);
    }
}
```

2. Explanation (**QUANTUM = 10**)

- We use 2 local objects to do this type of scheduler, which are **taskList** - *the list of tasks in the system* and the pointer **currentNode** - *that points to the current scheduled task.*
- **add function:**
    - We add the tasks to the list **taskList** as the same order like the Schedule.txt provides. The new task will be placed before the old task.
    - All the information of the new task will be copied into the list.
- **pickNextTask function:** We will find the next task which needs to be scheduled next.
    - Because the order of the list is opposite with the Schedule.txt (new task is placed before the old task), we have to find the next task using **findNextNode** function based on the task's name.

Name: Thanh Nguyen, Fiona Le
CECS 326
Professor Hailu Xu
Project 4 Report
Due date: 05/09/2021

- When the pointer points to the head then the next task should be the last node of the list, we use **getLastNode** to find the final node.

- **scheduler function:**
  - We begin at the last node of the list which is the first task in the Schedule.txt file.
  - Then we go over all the tasks in the system.
  - If the burst time of this task is bigger than **QUANTUM** (burst time > **QUANTUM**), then we just run the task in **QUANTUM** period time. In contrast, we will complete the task.
  - After each running, we have to update the burst time of the current task using subtract.
  - When the task is completed (burst time = 0), we delete the task from the list taskList.
  - Continue these steps until the list is empty or all the tasks are scheduled.
-

III. Result

A. Input File

```
T1, 4, 20
T2, 2, 25
T3, 3, 25
T4, 3, 15
T5, 10, 10
T6, 1, 30
T7, 5, 6
T8, 8, 24
T9, 7, 35
T10, 6, 20
```

B. Result

1. FCFS

```
thanhnguyen@thanhnguyen:~$ ./fcfs schedule.txt
Running task = [T1] [4] [20] for 20 units.
Running task = [T2] [2] [25] for 25 units.
Running task = [T3] [3] [25] for 25 units.
Running task = [T4] [3] [15] for 15 units.
Running task = [T5] [10] [10] for 10 units.
Running task = [T6] [1] [30] for 30 units.
Running task = [T7] [5] [6] for 6 units.
Running task = [T8] [8] [24] for 24 units.
Running task = [T9] [7] [35] for 35 units.
Running task = [T10] [6] [20] for 20 units.
```

Name: Thanh Nguyen, Fiona Le
CECS 326
Professor Hailu Xu
Project 4 Report
Due date: 05/09/2021

2. Priority

```
thanhnguyen@thanhnguyen:~$ ./priority schedule.txt
Running task = [T6] [1] [30] for 30 units.
Running task = [T2] [2] [25] for 25 units.
Running task = [T4] [3] [15] for 15 units.
Running task = [T3] [3] [25] for 25 units.
Running task = [T1] [4] [20] for 20 units.
Running task = [T7] [5] [6] for 6 units.
Running task = [T10] [6] [20] for 20 units.
Running task = [T9] [7] [35] for 35 units.
Running task = [T8] [8] [24] for 24 units.
Running task = [T5] [10] [10] for 10 units.
```

3. Round-Robin (RR)

```
thanhnguyen@thanhnguyen:~$ ./rr schedule.txt
Running task = [T1] [4] [20] for 10 units.
Running task = [T2] [2] [25] for 10 units.
Running task = [T3] [3] [25] for 10 units.
Running task = [T4] [3] [15] for 10 units.
Running task = [T5] [10] [10] for 10 units.
Running task = [T6] [1] [30] for 10 units.
Running task = [T7] [5] [6] for 6 units.
Running task = [T8] [8] [24] for 10 units.
Running task = [T9] [7] [35] for 10 units.
Running task = [T10] [6] [20] for 10 units.
Running task = [T1] [4] [10] for 10 units.
Running task = [T2] [2] [15] for 10 units.
Running task = [T3] [3] [15] for 10 units.
Running task = [T4] [3] [5] for 5 units.
Running task = [T6] [1] [20] for 10 units.
Running task = [T8] [8] [14] for 10 units.
Running task = [T9] [7] [25] for 10 units.
Running task = [T10] [6] [10] for 10 units.
Running task = [T2] [2] [5] for 5 units.
Running task = [T3] [3] [5] for 5 units.
Running task = [T6] [1] [10] for 10 units.
Running task = [T8] [8] [4] for 4 units.
Running task = [T9] [7] [15] for 10 units.
Running task = [T9] [7] [5] for 5 units.
```

Name: Thanh Nguyen, Fiona Le
CECS 326
Professor Hailu Xu
Project 4 Report
Due date: 05/09/2021

IV. Contribution

Task	Contribution
add Function (for 3 algorithms)	Fiona Le
FCFS algorithm	Fiona Le
Priority algorithm	Thanh Nguyen
RR algorithm	Thanh Nguyen, Fiona Le
Input File (schedule.txt)	Thanh Nguyen
README file	Fiona Le
MAKEFILE	Thanh Nguyen
Recording	Thanh Nguyen
Report	Thanh Nguyen , Fiona Le