COMPSCI 3SH3 Winter, 2021
Student name: Khizar Siddiqui
Student ID: 400109902
Date: 01-April-2021

Lab 5 Report

Shortest-job-first Scheduler

```
/**
 * Implementation of various scheduling algorithms.
 *
 * SJF scheduling
 */
 #include <stdlib.h>
#include <stdio.h>
#include <stddef.h>

#include "task.h"
#include "list.h"
#include "cpu.h"

struct node *head = NULL;

Task *pickNextTask();

// add a new task to the list of tasks
void add(char *name, int priority, int burst) {
    // first create the new task
    Task *newTask = (Task *) malloc(sizeof(Task));

    newTask->name = name;
    newTask->priority = priority;
    newTask->burst = burst;

    // insert the new task into the list of tasks
    insert(&head, newTask);
}
```

```
/**
 * Run the priority scheduler
 */
void schedule()
{
    Task *current;

    while (head != NULL) {
        current = pickNextTask();

        run(current,current->burst);

        delete(&head, current);
    }
}

/**
 * Returns the next task selected to run.
 */
Task *pickNextTask()
{
struct node *temp;
Task *hp = head->task;
temp = head->next;

    while (temp != NULL) {
        if (temp->task->burst < hp->burst)
            hp = temp->task;

        temp = temp->next;
    }

    return hp;
}
```

This implementation was quite similar to priority schedling. The only difference was in ‘pickNextTask()’ where we had to compare burst size and find the smallest one to schedule first.

Priority with round-robin scheduler

```
/**
 * Implementation of various scheduling algorithms.
 *
 * Round-robin priority scheduling
 */

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

#include "task.h"
#include "list.h"
#include "cpu.h"

struct node *head = NULL;

// pointer to the struct containing the next task
struct node *tmp;

int ctr;

Task *pickNextTask();

void insert_tail(struct node **head, Task *task);

// add a new task to the list of tasks
void add(char *name, int priority, int burst) {
    // first create the new task
    Task *newTask = (Task *) malloc(sizeof(Task));

    newTask->name = name;
    newTask->priority = priority;
    newTask->burst = burst;

    // insert the new task into the list of tasks
    insert(&head, newTask);
}

/**
 * Run the priority scheduler
 */
void schedule()
{
    Task *current;
    tmp = head;

    while (head != NULL) {
        current = pickNextTask();

        if (ctr > 1) {
            if (current->burst > QUANTUM) {
                run(current, QUANTUM);

                current->burst -= QUANTUM;

                delete(&head, current);
                insert_tail(&head, current);
            }
            else {
                run(current, current->burst);

                current->burst = 0;

                printf("Task %s finished.\n",current->name);
                delete(&head, current);
            }
        } else {
            run(current, current->burst);
            current->burst = 0;
            printf("Task %s finished.\n",current->name);
            delete(&head, current);
        }

    }
}

/**
 * Returns the next task selected to run.
 */
Task *pickNextTask() {
    struct node *temp;
    Task *hp = head->task;
    temp = head->next;

    while (temp != NULL) {
        if (temp->task->priority > hp->priority) {
            hp = temp->task;
            ctr = 1;
        } else if (temp->task->priority == hp->priority) {
            ctr++;
        }
        temp = temp->next;
    }

    return hp;
}

void insert_tail(struct node **head, Task *task){
    struct node* newNode = malloc(sizeof(struct node));
    newNode->task = task;
    newNode->next = NULL;

    if ((*head) == NULL){
        *head = newNode;
    }
    else{
        struct node *tmp = *head;
        while (tmp->next != NULL)
            tmp = tmp->next;

        tmp->next = newNode;
    }
}
```

The implementation for this combined priority with round-robin by using ‘pickNextTask()’ of priority and ‘schedule()’ of round-robin. It was slightly modified to account for instances of duplicate priorities in which case the round-robin would occur, this was done with the help of a function that inserted the current task at the end of the schedule to be rescheduled again.