Lab 5 Report

Name: Insiyah Ujjainwala
Email: ujjainwi@mcmaster.ca
Student ID: 400357483
Date: 3/12/2023

Shortest Job First Scheduler

- This C code is for a Shortest Job First (SJF) scheduler, part of a CPU task scheduling simulation.
- It includes the core function `schedule()` which manages the execution of tasks based on their burst time, and a helper function `pickNextTask()` which selects the next task to run by finding the one with the shortest burst time.
- The `add()` function creates and adds a new task, with a unique task identifier (TID), to the global linked list `head`.
- Each task has a name, priority, and burst duration.
- The `schedule()` function iteratively selects a task, runs it to completion, and then removes it from the list.
- During each iteration, the `traverse()` function checks the state of the list.
- The tasks are managed as nodes in a linked list, and memory allocation for new tasks is handled via `malloc`.
- The code embodies a simple and efficient approach to CPU scheduling.

```c
#include <stdlib.h>
#include <stdio.h>
#include "task.h"
#include "list.h"
#include "cpu.h"

// reference to the head of the list

struct node *head = NULL;

// sequence counter of next available threat identifier
int nextTID = 0;

Task *pickNextTask();

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

newTask->name = name;
newTask->tid = nextTID++;
newTask->priority = priority;
newTask->burst = burst;

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

```
/**
* Run the SJF scheduler
*/

void schedule(){
Task *current;

// sanity checker
traverse(head);

while (head!=NULL){
current = pickNextTask();
run(current, current->burst);
delete(&head, current);
}
}

/**
* Returns the next task selected to run.
* Picks the task with the shortest burst time.
*/

Task *pickNextTask(){
struct node *temp;
Task *shortestJob = head -> task;
temp = head->next;

while (temp != NULL){
if (temp -> task -> burst < shortestJob -> burst){
shortestJob = temp -> task;
}
temp = temp->next;
}
return shortestJob;
}
```

Output:

```
insiyah@insiyah:~/Desktop/Lab5Project$ make sjf
make: 'sjf' is up to date.
insiyah@insiyah:~/Desktop/Lab5Project$ ./sjf schedule.txt
[T8] [10] [25]
[T7] [3] [30]
[T6] [1] [10]
[T5] [5] [20]
[T4] [5] [15]
[T3] [3] [25]
[T2] [3] [25]
[T1] [4] [20]
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.
```

Priority with round-robin Scheduler

- This C code is a priority-based round-robin scheduler.
- It adds tasks to a linked list, assigns them priority and burst times, and uses `pickNextTask()` to determine execution order.
- Tasks with the highest priority run first; if multiple tasks share the highest priority, they are executed in round-robin fashion, defined by a quantum time slice.
- Tasks exceeding the quantum are re-queued; others complete and are removed.
- The `highestPriority()` function identifies the highest priority at any time, ensuring the scheduler adheres to priority ordering.

```c
#include <stdlib.h>
#include <stdio.h>
#include "task.h"
#include "list.h"
#include "cpu.h"

struct node *head = NULL;

// pointer for round-robin within the same priority level
struct node *nextNode = NULL;

Task *pickNextTask();



// add a new task to the list of tasks
```

```
void add(char *name, int priority, int burst){
// first create a 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);
}

/**
* finds the highest priority among the tasks
*/

int highestPriority(){
struct node *temp;
Task *highestPriority = head->task;
temp = head->next;

while (temp != NULL){
if (temp->task->priority > highestPriority -> priority)
highestPriority = temp -> task;
temp = temp -> next;
}

return highestPriority -> priority;
}
/**
* Run the priority with round-robin scheduler
*/

void schedule(){
Task *current;


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

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

//restart the task at the end of its priority group
```

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

/**
* returns the next task selected to run
* if multiple tasks have the same highest priority, round-robin is applied
*/

Task *pickNextTask(){
if (nextNode == NULL || nextNode -> task -> priority != highestPriority()){
nextNode = head;
}

Task *nextTask = nextNode->task;
nextNode = (nextNode->next == NULL)?head : nextNode->next;

return nextTask;
}
```

Output:

```
insiyah@insiyah:~/Desktop/Lab5Project$ make priority_rr
gcc -Wall -c driver.c
gcc -Wall -c list.c
gcc -Wall -c CPU.c
gcc -Wall   -c -o schedule_priority_rr.o schedule_priority_rr.c
gcc -Wall -o priority_rr driver.o schedule_priority_rr.o list.o CPU.o
insiyah@insiyah:~/Desktop/Lab5Project$ ./priority_rr schedule.txt
Running task = [T8] [10] [25] for 10 units.
Running task = [T8] [10] [15] for 10 units.
Running task = [T8] [10] [5] for 5 units.
Running task = [T7] [3] [30] for 10 units.
Running task = [T7] [3] [20] for 10 units.
Running task = [T7] [3] [10] for 10 units.
Running task = [T6] [1] [10] for 10 units.
Running task = [T5] [5] [20] for 10 units.
Running task = [T4] [5] [15] for 10 units.
Running task = [T4] [5] [5] for 5 units.
Running task = [T5] [5] [10] for 10 units.
Running task = [T3] [3] [25] for 10 units.
Running task = [T3] [3] [15] for 10 units.
Running task = [T3] [3] [5] for 5 units.
Running task = [T2] [3] [25] for 10 units.
Running task = [T1] [4] [20] for 10 units.
Running task = [T1] [4] [10] for 10 units.
Running task = [T2] [3] [15] for 10 units.
Running task = [T2] [3] [5] for 5 units.
```