CSCI340, Fall 2014
Assignment 5: Writing a Scheduler

California State University - Chico
By Bryan Dixon

Due Date: Sunday, November 23rd, 2014 11:59pm

Introduction

The purpose of this assignment is for you gain insight into how schedulers work on the system.

Logistics

The only "hand-in" will be electronic. Any clarifications and revisions to the assignment will be posted on my web page and emailed out to the class.

Hand Out Instructions

I recommend you use an Ubuntu Linux virtual machine to complete this assignment. Alternatively, you can use the jaguar machines or your native Linux install.

Download the file `scheduler-handout.tar` from the course "assignment" page.

Start by copying the file `scheduler-handout.tar` to the protected directory (the *lab directory*) in which you plan to do your work. Then do the following:

- Type the command `tar xvf scheduler-handout.tar` to expand the tarfile.
- Type the command `make` to compile and link some test routines.
- Enter your name in the header comment at the top of `schedule.cc`.

Looking at the `schedule.cc` file, you will see that it contains a rudimentary scheduler API to add a process, remove a process, and get the next process from your scheduler. Your job is to implement the code in schedule.cc that implements a variant of a Multi Level Priority Scheduler using Round Robin schedulers.

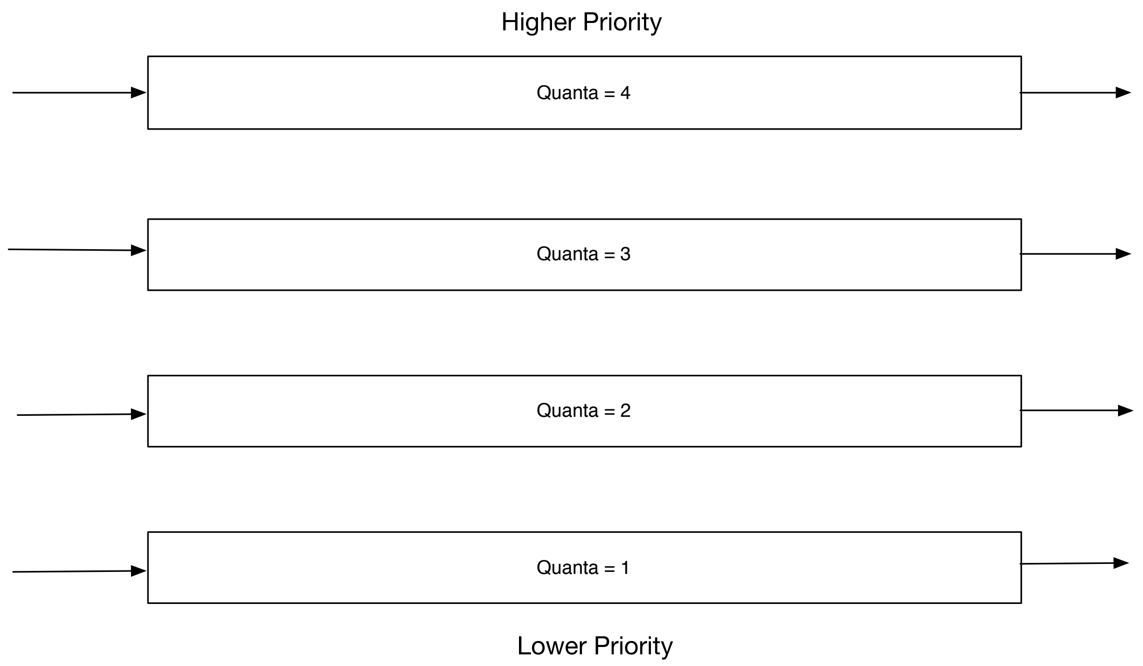

Figure 1: Multi Level Round Robin Priority Scheduler

General Overview of Schedulers

Taken from Wikipedia[1]

In computer science, scheduling is the method by which threads, processes or data flows are given access to system resources (e.g. processor time, communications bandwidth). This is usually done to load balance and share system resources effectively or achieve a target quality of service. The need for a scheduling algorithm arises from the requirement for most modern systems to perform multitasking (executing more than one process at a time) and multiplexing (transmit multiple data streams simultaneously across a single physical channel).

The scheduler is concerned mainly with:

- **Throughput** - The total number of processes that complete their execution per time unit.
- **Latency**, specifically:
  - **Turnaround time** - total time between submission of a process and its completion.
  - **Response time** - amount of time it takes from when a request was submitted until the first response is produced.
- **Fairness** - Equal CPU time to each process (or more generally appropriate times according to each process' priority and workload).

- **Waiting Time** - The time the process remains in the ready queue.

In practice, these goals often conflict (e.g. throughput versus latency), thus a scheduler will implement a suitable compromise. Preference is given to any one of the above mentioned concerns depending upon the user's needs and objectives.

In real-time environments, such as embedded systems for automatic control in industry (for example robotics), the scheduler also must ensure that processes can meet deadlines; this is crucial for keeping the system stable. Scheduled tasks can also be distributed to remote devices across a network and managed through an administrative back end.

Your Task

The task for this assignment is to implement the scheduler API provided to you in the `schedule.cc` file with variant of a multi level priority scheduler using Round Robin schedulers. This scheduler iterates through the different levels starting with the highest priority (priority 1) and choses the process next in the queue for that priority level to schedule for a specified quanta or time. The higher the priority the more quanta is given. Your implementation should mirror the quanta and number of priorities shown in Figure 1.

You aren't allowed to import any libraries not already provided in the `schedule.cc` file.

Data Structures in C

As you can't include any libraries for data structures you'll likely want to implement your own data structure to implement the FCFS scheduler. As such you'll need to do this in C code. As an example here is a simple implementation of a linked list in C using structs:

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

struct node {
    int value;
    struct node *next;
};

int main(){
    int i;
    /* This will be the unchanging first node */
    struct node *root;
    /* cur node for manipulating linked list */
    struct node *cur;

    /*
    * Now root points to a node struct
    * Dynamic allocation of memory the size of
    * a node in C (similar to new node in C++)
    */
    root = (struct node *) malloc( sizeof(struct node));

    cur = root;
    for(i=0; i<10;i++){
        cur->value = i;
        cur->next = (struct node *) malloc( sizeof(struct node));
        cur = cur->next;
    }
    //Create final values
    cur->value = 10;
    cur->next = NULL;

    /* Now let's print it out */
    cur=root;
    while(cur){
        printf("%d ", cur->value);
        cur=cur->next;
    }
    printf("\n");

    /* Let's free up memory */
    while(root){
        cur = root;
        root = root->next;
        free(cur); //like delete
    }
}
```

Checking Your Work

I have provided some tools to help you check your work.

Reference solution. I've included a reference output file with the expected solution. Your program should produce identical output to that of the `ref.out` file. There is also a `simref` executable that will generate the reference output as well.

Hints

- Read **Section 6.3** of the Operating System Concepts Essentials 2nd edition book.

Evaluation

Your solution will be tested agains the reference output. You will get full credit if your assignment reproduces the reference output.

Hand In Instructions

You only have to change `schedule.cc`. You need to upload `schedule.cc` to the `http://turnin.ecst.csuchico.edu/` page to mark your completion time.

Extra Credit - 10pts

I'm in the process of working out a potential EC scheduler for you to implement in addition to this assignment. If one gets created details will be updated here.

References

[1] Wikipedia. "Scheduling (computing)". Wikipedia, The Free Encyclopedia. 2012. `http://en.wikipedia.org/wiki/Process_scheduler`. Online; accessed 16-February-2014.