**Process Management with Round-Robin Scheduling**

**for**

**COSC 519**

**Matthew Bourque  
Ravi Chandra Saluru Venkata**

**11/11/2014**

Contents

[**Roles and Responsibilities** 3](#_Toc404604161)

[**Project Description** 3](#_Toc404604162)

[**Design** 4](#_Toc404604163)

[**Development** 6](#_Toc404604164)

[**Analysis** 6](#_Toc404604165)

[**Evaluation** 9](#_Toc404604166)

[**Results** 9](#_Toc404604167)

[**User Guide** 10](#_Toc404604168)

[**References** 10](#_Toc404604169)

[**Code** 10](#_Toc404604170)

# **Roles and Responsibilities**

# 

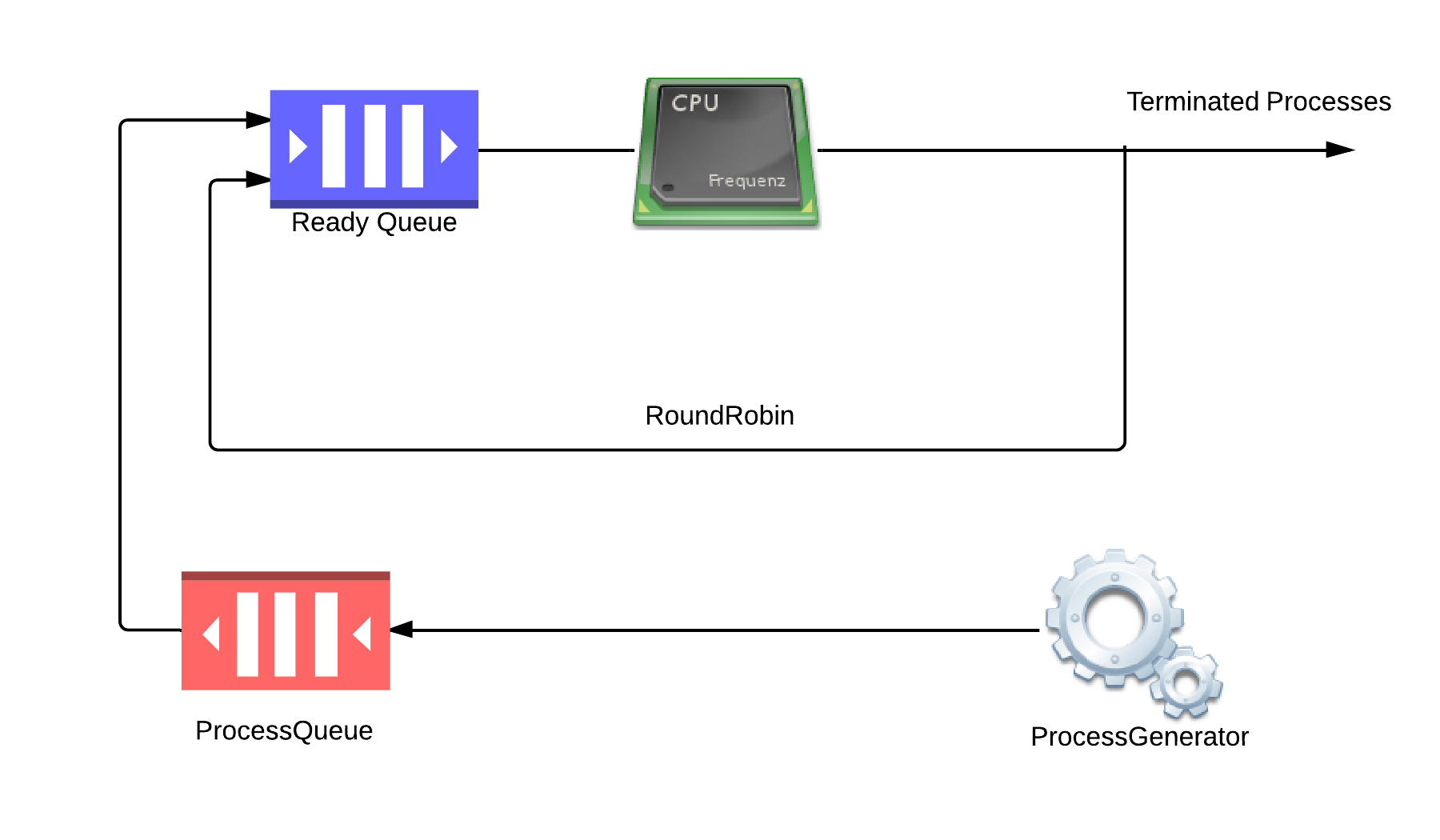
# **Project Description**

A process is a program in execution. Process is a key OS abstraction that users see. Process management is an integral part of any operating system. The operating system provides processes an abstract, which is the illusion that each process is the only process on the computer.

An Operating System with the help of Process Management, allocates resource for a process, enables sharing and protection of data, and puts it in a job queue using a job scheduler. Once a process comes in the job queue, it needs be assigned to a CPU for it to be able to complete its tasks. A CPU scheduler will fetch the process from the queue and feed to the CPU. Which process gets selected by CPU scheduler depends on the scheduling algorithm implemented.

The goal of this project is to simulate the Process Management component of the Operating System in a single CPU environment. Since this project is about the simulation of Process Management in a single-CPU environment, many other aspects of the Process Management will be not implemented. We will focus solely on how allocate a process to CPU for a fixed time quantum and swapped out of the CPU. The process selection will be based on “**Round Robin**” principle.

# **Design**



The simulator has following components:

- Process

- ProcessGenerator

- Scheduler

- Queue

Multiple processes will generate using the ProcessGenerator, with random burst time. The process generated by ProcessGenerator will be queued into waiting queue, which eventually de-queued to ready queue based on the availability in read queue. The scheduler will schedule a process from ready queue and feed it to CPU. When a process is given the CPU, a timer is set for whatever value has been set for a time quantum. If the process finishes its burst before the time quantum timer expires, then it is swapped out of the CPU. If the timer goes off first, then the process is swapped out of the CPU and queued back to the ready queue. The ready queue is maintained as a circular queue, so when all processes have had a turn, then the scheduler gives the first process another turn, and process goes on.

**ProcessGenerator**:

* The Process generator creates new process with random burst time.

**Scheduler:**

* Scheduler invokes Process-Generator to create new process and add it to queue.
* Scheduler checks the availability of ready queue and put the process into ready queue. Scheduler will make sure ready queue is not empty.
* The Scheduler will schedule a process from ready queue and feed it to CPU. When a process is given the CPU, a timer is set for whatever value has been set for a time quantum.

**Queue (Ready Queue):**

* Incoming jobs are added to the tail of the queue
* Jobs leave the ready queue from the top
* The size of the queue is customizable.

**Class Design**

# **Development**

# **Analysis**

This analysis presents a broad picture that portrays the comparative study of various scheduling algorithms for a single CPU and shows why we chose Round Robin. Using this representation, it becomes much easier to understand what is going on inside the system and why a different set of processes is a candidate for the allocation of the CPU at different time. The objective of the analysis is to study the high efficient CPU scheduler on design of the high quality scheduling algorithms, which suits the scheduling goals.

**First-come First-served (FCFS)**

* + Jobs are scheduled in order of arrival
  + Non-preemptive
* **Problem:**
  + waiting time depends on arrival order
  + Troublesome for time-sharing systems
  + Also called head of the line blocking
  + Convoy effect short process behind long process
* **Advantage:** really simple!

**Shortest-Job-First (SJF) Scheduling**

Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time

* + Non-preemptive – once CPU given to the process it cannot be preempted until completes its CPU burst
  + Pre-emptive – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is know as the   
    Shortest-Remaining-Time-First (SRTF)

SJF is optimal – gives minimum average waiting time for a given set of processes

* **Advantages:** Minimizes average wait time. Provably optimal
* **Disadvantages:** 
  + Not practical: difficult to predict burst time
  + Possible: past predicts future
  + May starve long jobs

**Round Robin**

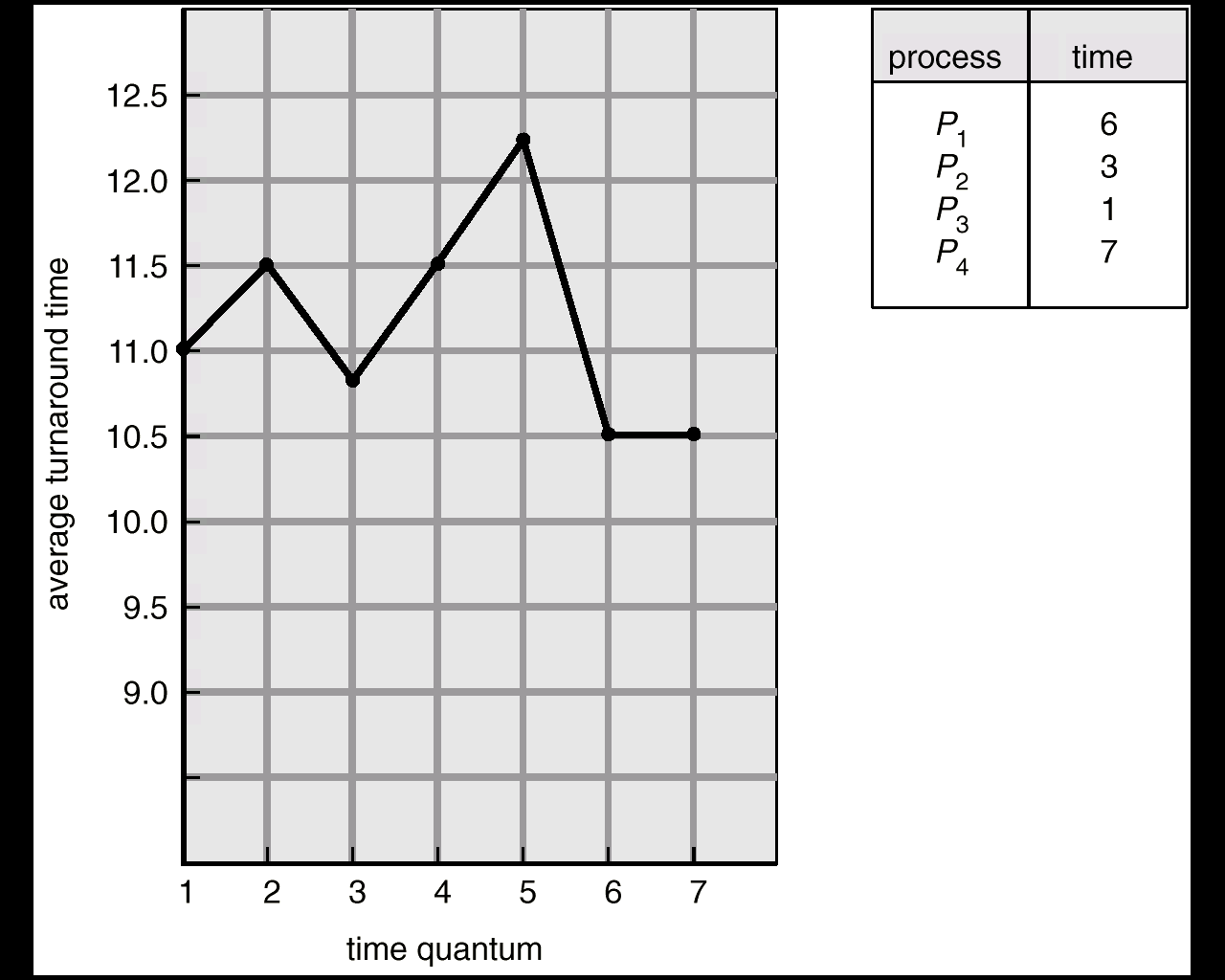
* Each process gets a small unit of CPU time (*time quantum*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
* If there are ***n*** processes in the ready queue and the time quantum is ***q***, then each process gets **1/*n*** of the CPU time in chunks of at most ***q*** time units at once. No process waits more than **(*n*-1)*q***time units.
* **Advantages**
  + Low response time, good interactivity
  + Fair allocation of CPU across processes
  + Low average waiting time When job lengths vary widely
* **Disadvantages**
  + Poor average waiting time when jobs have similar lengths
  + Performance depends on length of time slice
* Typically, higher average turnaround than SJF, but better response.
* Performance

q large ⇒ FCFS

q small ⇒ q must be large with respect to context switch, otherwise overhead is too high.

**Turnaround Time Varies With The Time Quantum**

TAT can be improved if most process finish their next CPU burst in a single time quantum.



**Comparison of SJF with FIFO and Round Robin**

If all jobs are the same length, SJF runs same as FIFO. In other words, in that instance, FIFO is as good as you can do. Under SJF if jobs have varying length, short jobs do not get stuck behind long jobs.

The following is an example that illustrates the benefit of SJF. Suppose we have three jobs: A, B, and C. A and B are both CPU bound, and each runs for a week. Job C consists of an I/O bound loop, which consumes 1 msec of CPU and 10 msec of disk I/O (assuming complete overlapping of I/O and CPU processing). By itself, C uses 90% of the disk. By itself, A or B could use 100% of the CPU. What happens if we try to share the system among A, B, and C?

Under FCFS, A and B will take two weeks before getting to job C.

Under round robin with 100 msec of time slice, we can only get 5% of disk utilization, since job C uses disk for 10 msec of every 201 msec.

A (100 msec)

B

C

A

B

CPU

I/O

However, if we reduce the time slice down to 1 msec, we can get 90% disk utilization again—almost as good as job C running alone. Also, we have not slowed down A or B by all that much; they will get 90% of the CPU.

# **Evaluation**

The purpose of this project is to simulate and understand the implementation of Round Robin process management which incorporates creating, managing them (Circular Queue theoretically) and their execution of processes as per the round robin scheduler. We have developed a simulation, which will create a certain number of processes and add them to a ready queue. Then these process would processed by the scheduler.

In simulation, user needs to specify the value of the time quantum in a plan text file. We noticed the too short or too long time quantum will lead to performance decrease due to increasing context switching overhead, and degradation of the RR method to FCFS algorithm.

Simulation successfully maintains the queue so that all processes have had a turn, then the scheduler places the process back at the end of the queue if it has not been completed, and so on. It also pushes the process from ready queue to terminated queue once the process has been completed. The scheduler makes sure that the ready queue is never empty.

We had tested the program with various inputs and it appears that the design workes as expected.

Validations on user input fields are:

- Quantum Time value cannot be zero or empty. If user try to start this simulation without any value or with zero, system shall show an error message.

# **Results**

# **User Guide**

**System Requirements:**

* Windows 7 or Windows 8
* Java JDK 1.7+

**Steps to run the application:**

# **References**

# **Code**