CISC 361 Final Project

Jingqing Liu & Zhiwen Zhu & Yuchen Zhang

May 13, 2022

Introduction:

In this project, we have designed and implemented a program that simulated the job scheduling, CPU scheduling, and deadlock avoidance of an operating system.

We used C++ language to make this project work as we expected. We implemented Shortest Job First and First In First Out methods in job scheduling(especially in Queue 1 and Queue 2) and the Round Robin method in process scheduling(with a quantum). Moreover, we also implemented the Banker's Algorithm to avoid deadlock.

Hints and assumptions:

- If there is a completion of a job, check Wait Queue then the two Hold Queues.
- When a job completes, it releases main memory and implicitly releases the devices.
- The only constraints to move from one of the Hold Queues to the Ready Queue are main memory and devices.
- Priority is for the job scheduling, not process scheduling.
- if more resources are needed for a job than in the system then do not consider that job.
- If jobs have same run-time and same priority, use FIFO scheduling (FIFO within SJF).
- Handle all internal events before external.
- Internal Event: The events that are related to the CPU quantum (i.e., execution, completion, and quantum interruption) are considered as internal events. When a device requests or releases devices that is an internal event.
- External Event: Arrival of new jobs (reading from the file) and display events are considered as external events.
- There will never be two external events at the same time.
- The program must not read the entire input file in the beginning of the program to preprocess or use any advanced information. Execution is line by line and your program's operations should not, for example, use future information about device requests.

- When an external release occurs and jobs are still in Ready Queue, check the Wait Queue
- to determine if something can start (i.e., get into the Ready Queue).
- Devices are only requested by jobs while running on the CPU, and if this happen, the job's time slice is interrupted.
- On an arrival, the "S=" denotes the maximum number of device(s) the job will ever use.
- Pretend to allocate devices before using Banker's.
- A job that requests a device in the middle of a time quantum, whether it gets the device or not, does NOT finish its time quantum. It blocks immediately.
- Do not use an array of size 100 for jobs.

Design Approach:

1. Input specification

First, we create a command line-based system configuration. In the input text file, we should see a command like "C 9 M=45 S=12 Q=1". Where C represents the start time for the simulator, M represents the total main memory, S represents the total serial devices, and Q represents the time slice.

Job arrival: A 10 J=1 M=5 S=4 R=3 P=1

The example above states that job number 1 with priority 1 arrives at time 10, requires 5 units of main memory, holds no more than 4 devices at any point during execution, and runs for 3 units of time.

A request for devices: Q 10 J=3 D=4

The example above states that at time 10, job number 3 requests for 4 devices. A job only requests devices when it is running on the CPU. The Quantum is interrupted to process the request. If request is granted, the process goes to the end of the Ready Queue, else it goes to the Wait Queue.

A release of devices:

L 10 J=5 D=1

The example above states that at time 10, job number 5 releases one device. A job only releases devices when it is running on the CPU. Quantum is interrupted. One or more jobs may be taken off the Wait Queue due to this.

display current system status:

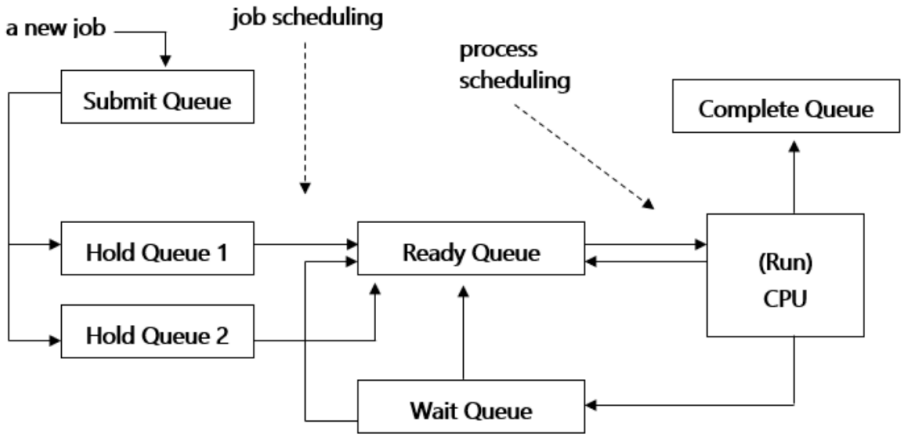
D 11

The example above states that at time 11 an external event is generated, and all the contents of each queue and should be printed. It will also print out which job is running in CPU and remaining service time for unfinished jobs and other time.

2. Job Scheduling

The flowing graph indicated the general design of the whole program.

A graphic view of the simulator



According to the above diagram, there will be three queues containing all the upcoming jobs, and in hold Queue 1, we will implement the SJF method to schedule all the jobs in Q1.

In hold Queue 2, we will implement the FIFO method to plan all jobs in Q2. There is also a priority ordering between Q1 and Q2 (Q1 is prioritized over Q2).

The Ready Queue contains all the jobs that are ready to be executed, and if there is space in the main memory for the job to be executed, it will be put in the Ready Queue.

Once a new job arrives, it should come with new arrivals time, job id, required main memory, max demand of resources, runtime, and priority.

The following is what can happen when a job arrives:

1. if there is not enough main memory or total number of devices, the job is rejected and will not be placed in any waiting queue.

2. If there is not enough available main memory, the job is placed in Q1 or Q2 depending on priority and waits for enough main memory to be available.

3. If there is enough main memory available, a process is created for the job, the memory needed for the job is allocated in main memory, and the job is placed in the ready queue.

3. Process Scheduling

For process scheduling, we use the round-robin comparison algorithm. Ready Queue, Complete Queue, and Wait Queue contain many jobs.

When there is a job in the Ready Queue, it is put into the CPU to run. If the job finishes during a one-time slice, the job will be pushed into the Complete Queue, and then the allocated memory and devices are released.

In this case, we use Banker's algorithm to check if there is a suitable job that can be pushed into the Ready Queue. At the same time, the new job will be pushed into the CPU for execution.

4. Deadlock

When a running job requests devices, the Banker’s algorithm is used to determine if the request can be satisfied. If the request cannot be satisfied, the process is immediately switched from the CPU to the Wait Queue.

Whenever a running job releases devices and the Wait Queue is not empty, the Banker’s algorithm is executed in FIFO order on each job in the Wait Queue to determine if any jobs in the Wait Queue can be allocated its last request of devices. If necessary, the entire Wait Queue is checked to restart as many jobs as possible.

A similar remark applies when a job that holds devices terminates, because a job implicitly releases devices upon termination.

The Wait Queue is checked FIRST (before the Hold Queues) upon a completion.

Project execution instructions:

In order to compile the program and generate the a.out file, we should do the following command in the terminal!

In terminal:

1. cd <folder name>
2. g++ -std=c++11 Main.cpp
3. ./a.out

The result will display in the terminal window.

If you choose to execute the program and get the result via Eclipse, simply build it and run it.

Input:

```
C 1 M=200 S=12 Q=4
A 3 J=1 M=20 S=5 R=10 P=1
A 4 J=2 M=30 S=2 R=12 P=2
A 9 J=3 M=10 S=8 R=4 P=1
Q 10 J=1 D=5
A 13 J=4 M=20 S=4 R=11 P=2
Q 14 J=3 D=2
A 24 J=5 M=20 S=10 R=9 P=1
A 25 J=6 M=20 S=4 R=12 P=2
Q 30 J=4 D=4
Q 31 J=5 D=7
L 32 J=3 D=2
D 9999
```

Output:

```
At time:  9999
Current Available Main Memory = 80
Current Devices = 12
Completed Jobs:
------------------------------------------------------------
Job ID    Arrival Time    Finish Time    Turnaround Time
============================================================
3           9               19              10
1           3               29              26
2           4               33              29
4           13               56              43
5           24               57              33
6           25               61              36

Hold Queue 1:
-------------------------
Job ID    Run Time
=========================

Hold Queue 2:
-------------------------
Job ID    Run Time
=========================

Ready Queue:
----------------------------------
Job ID    Run Time    Time Accrued
==================================

Process running on CPU:
-----------------------------------
Job ID    Time Accrued    Time Left
===================================

Wait Queue:
----------------------------------
Job ID    Run Time    Time Accrued
==================================


************************************************************
************************************************************

System Turnaround Time: 16.5
```