CPU SCHEDULING SIMULATION

PROJECT
IN PARTIAL FULFILLMENT
OF REQUIREMENTS OF
COSC 519
DR. R. KARNE
SPRING, 2016

BY

GROUP ONE:
FRANCESCO BUTTS
ALEXANDER SCHECK
LAUREL VALENTI
TABLE OF CONTENTS

COVER PAGE
TABLE OF CONTENTS
INTRODUCTION
CPU SCHEDULING DESCRIPTION
  GOALS OF CPU SCHEDULING
DESIGN
  PROCESS FLOW DIAGRAM OF CPU SCHEDULING
  ALGORITHMS COMPARED IN THIS CPU SCHEDULING PROGRAM
  PLANNED EVALUATION METHODS FOR ALGORITHMS
  PROCESS FLOW DIAGRAM OF IMPLEMENTATION
DEVELOPMENT
  OVERALL DESIGN NOTES
  DESCRIPTIONS OF CLASSES USED
  DESCRIPTIONS OF METHODS WITHIN CLASSES
ANALYSIS
  COMPARISON OF APPROACHES
  DELAYS AND LIMITS
EVALUATION AND RESULTS
  DISCUSSION OF RESULTS
  STATISTICAL RESULTS TABLE
REFERENCES
INTRODUCTION

The objective for this project is to implement a CPU scheduling simulation and compare the results of standard algorithms used. The group members are students in Dr. R. Karne's COSC 519 Operating Systems class: Francesco Butts, Alexander Scheck, and Laurel Valenti. All group members participated in project proposal development, design, development, testing, evaluation of results and final reporting. Alexander Scheck is the development manager and head developer. Francisco Butts is senior developer. Laurel Valenti is junior developer and group report writer.

CPU SCHEDULING DESCRIPTION

As originally conceived and developed, computers executed programs sequentially. One job entered the CPU and completed before another job began. That led to periods of time when the CPU was idle, waiting for I/O or other interrupts. It was a waste of the vital computing resource of the CPU.

Multiprogramming began as a response to the CPU idle time. Computers execute jobs in parallel in multiprogramming, multiple jobs running at the same time. If there is just one CPU, however, at a given moment the CPU can only execute one program. The concept of multiprogramming was expanded to multiprocessing, where a process in a CPU is defined as a program being executed. Processes can move in and out of the CPU when waiting for I/O, memory or other resources. When a running process is allowed to finish before the next one loads, it is called non-preemptive scheduling. The alternative occurs when the CPU scheduler stops a running process to load another, waiting process, it is called preemptive scheduling.

Multiprocessing requires scheduling processes to move in and out of a ready queue for the CPU. This idea is expanded below in the design section, because it is integral to the simulation design that was developed. There are different algorithms that may be employed to schedule process movement, depending on the purpose of the scheduling. The original purpose of multiprocessing was to maximize CPU utilization, but there are other possible advantages to be gained, leading to different algorithms to employ to schedule processes.

Other advantages sought through multiprocessor scheduling are to maximize the time that all resources are employed, not just the CPU. This leads to scheduling algorithms designed to manage the use of I/O, memory, and other system resources as well as the CPU. The criteria for evaluating this type of algorithm would be maximization of time utilized for all of these resources.

Another algorithm to schedule processes for the CPU might maximize throughput, the number of processes that may be completed in one unit of time. Another could be set to minimize turnaround time, all of the time that a process is in the system, from the time it is submitted to the time it is finished. Turnaround time includes I/O + wait time for memory+ wait time for the ready queue+ CPU execution time. If throughput measures number of processes in a given time, turnaround time is the inverse of throughput. Still another algorithm might have minimal response time as a goal, which is the time interval between the submission of a process to the CPU and the start of a response from the CPU.

CPU execution time, the amount of time that a process uses the CPU, is called burst time. Just as one program may have many processes, one program may have many CPU bursts. The CPU executes a series of instructions for process 1, executes another waiting process 2 while it waits for I/O for process 1, and so forth. A histogram chart of frequency versus burst duration reveals different characteristics for different types of programs. If there are many long duration bursts, the CPU is taking a long time to execute a series of instructions, and the program may be assumed to be complicated code. This process is called a CPU bound process. Short duration bursts are more characteristic of a great deal of interactive I/O; this is called an I/O bound process. The burst time is part of the total turnaround time and is included in the evaluation of scheduling algorithms.

Other algorithms may be designed to minimize the wait time of a process in the ready queue, or minimize the time the system takes to respond to a user on interactive console inputs, or to create an environment which does not starve some processes of CPU time in favor of other processes. Some algorithms may be developed according to a priority rankings, although this type of scheduling risks starving low priority processes. Schedulers which
involve only process time spent ready to load into the CPU (ready queue) and in execution in the CPU itself are short term schedulers. Schedulers which add time spent in a job queue and time spent loading a job into memory are called long term schedulers.

In this project, we included elements of long term and short term scheduling in job queue time, wait time, burst time and total turnaround time. We simulated three of the most popular short term scheduling algorithms, discussed in detail in the design section. The three algorithms we studied are FIFO, first come, first served; SJF, shortest job first; and RR, round robin scheduling. The results were recorded in a series of statistical tables, to evaluate performance of the three algorithms.

DESIGN

PROCESS FLOW DIAGRAMS OF CPU SCHEDULING SIMULATION

From our textbook, a visual representation of CPU scheduling is shown below. This flow diagram is used as the framework for our group’s CPU scheduling. The simulation includes a ready queue, partially executed processes which are swapped in and out, and I/O waiting queues. Three short term scheduling algorithms are used in the simulation, First In First Out (FIFO), Round Robin (RR), and Shortest Job First (SJF). Each scheduler runs in sequence. At the beginning of the run, the user is asked to enter the number of processes desired, between 10 and 50. The selected number of processes with varying burst times are randomly generated and placed in the ready queue for each scheduler. The processes are released to the CPU by the scheduler according to the particular algorithm being tested.

Source: Silberschatz et.al. p. 107

A new process admitted to the ready queue progresses through a series of states, from ready to running to terminated in a straightforward case. This series of states may be seen in our simulation as it runs. However, while processes are running, a CPU may be subject to random interrupts, causing a different change of state. The process that was running may return directly to the ready queue, or be placed in a wait state after which it returns to the ready queue. State progressions are shown in the diagram below. Our simulation includes random interrupts and system calls which return processes to waiting or back to the ready queue for further processing.

Source: Silberschatz et.al. p. 101
ALGORITHMS COMPARED IN CPU SCHEDULING SIMULATION

FIFO – First In, First Out

The first come, first served (FIFO) scheduling algorithm causes processes to be released to the CPU in order of their arrival in the ready queue. The first process to ask for CPU time is the one that receives it.

RR - Round Robin

The round robin (RR) scheduling algorithm causes the ready queue to be viewed as a circular queue. It is generally a first come, first served scheduler with an additional stipulation. A time quantum is defined as a finite time interval, and is then used by the scheduler to set maximum times for each incoming process. If a process in execution consumes the entire time quantum, it is interrupted and returned to the ready queue.

SJF - Shortest Job First

The shortest job first (SJF) scheduling algorithm looks at the CPU burst times for the processes in the ready queue. It releases the one with the shortest burst time to the CPU for execution. The SJF algorithm may be programmed to be either non-preemptive or preemptive. In the non-preemptive case, a new process arriving at the ready queue with a shorter burst time than the process currently executing waits for the currently executing process to finished before it goes to the CPU for execution. In the pre-emptive case, the newly arriving shortest process displaces the currently executing process in the CPU. Our simulation was designed for the non-preemptive case.

PLANNED EVALUATION METHODS FOR ALGORITHMS

In our simulation, FIFO, RR and CPU schedulers executing 10 - 50 random CPU or I/O bound processes, including random interrupt handling, will be observed on the output console. Statistics will be collected for the number of processes scheduled and executed under each algorithm. The number of processes will be varied in order to see if there is a difference between schedulers under different size process batches. The statistics to be used for all are: average wait time; minimum wait time; maximum wait time; average turnaround time; minimum turnaround time; maximum turnaround time; average response time; maximum response time; and minimum response time.

DEVELOPMENT

OVERALL DESIGN NOTES

The CPU scheduling code was developed in the Java programming language, hosted on Rioux subversion website using the Eclipse IDE. At the outset of program execution, the user is presented with the screen below, asking for the number of processes desired.

![Input Screen](image)

After the user selects the total number of processes, the user is asked to select the maximum number of processes which will be in the ready queue at any one time. The user is presented with the following screen to enter this parameter.
Execution begins in the class Environment which constructs the schedulers in turn, and calls each to create and run according to the user chosen parameters. Simulation of the processes running in the three schedulers is displayed on the console. Statistics for minimum, maximum and average turnaround times, wait times, and response times are collected, to be displayed at the conclusion of the runs.

The random processes generated are identified as CPU or IO bound, causing associated variations in the times. As noted, CPU bound processes are CPU intensive, with many calculations and operations in the CPU. CPU bound processes may cause peripheral system resources such as I/O to be under used. I/O bound processes are the opposite; I/O use is intensive, CPU is not, and the CPU may experience idle time. Both are included in the random processes used by the three schedulers. Also during execution, random interrupts are generated in the Processor class to simulate a realistic CPU operation. Here is a list of the Java classes.

### JAVA CLASSES WITH MEMBER DESCRIPTIONS

<table>
<thead>
<tr>
<th>Class/Enum</th>
<th>Member Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bound (Enum)</td>
<td>Sets processes as CPU or IO bound</td>
</tr>
<tr>
<td>BurstComparator</td>
<td>Compare(Process,Process) burst times</td>
</tr>
<tr>
<td>EnvironmentUI</td>
<td>Main, executes program and collects statistics</td>
</tr>
<tr>
<td>ExecutionStats</td>
<td>Get wait/runtime stats per scheduler</td>
</tr>
<tr>
<td>FIFO Scheduler</td>
<td>First in, first out scheduler</td>
</tr>
<tr>
<td>Process</td>
<td>Get/set state, bound, times arrival, burst, burst remain, start, wait, execution</td>
</tr>
<tr>
<td>Processor</td>
<td>Random interrupt processing, quantum for RR</td>
</tr>
<tr>
<td>RR Scheduler</td>
<td>Round robin scheduler</td>
</tr>
<tr>
<td>Scheduler</td>
<td>Create processes load job, ready, terminated, io queues; reset each sched</td>
</tr>
<tr>
<td>SchedulerType</td>
<td>FIFO, RR, SJF calls in turn</td>
</tr>
<tr>
<td>SJF Scheduler</td>
<td>Shortest job first scheduler</td>
</tr>
<tr>
<td>State (Enum)</td>
<td>ReadyQ: New, Running, Waiting, Ready, Terminated</td>
</tr>
<tr>
<td>StatsCalculator</td>
<td>Perform calculations for stats</td>
</tr>
</tbody>
</table>
**TESTING**

The arrival, start and wait times were tested to ensure they were providing accurate and logical results to be used for the analysis. The process objects were tested by tracking the object through the simulation. A narrative console was used to display the track of the object so we could see precisely the travel of the object. The narrative console was also used to ensure the proper object was chosen at the appropriate times. For example, when sjf is running, we tested to see if the shortest job was actually chosen to be sent to the processor.

**ANALYSIS**

**COMPARISON OF APPROACHES**

**FIFO Scheduler**

Here is a simple example of what can be expected to happen to three processes arriving at the same time at a CPU ready queue, assuming no interrupts occur, where the FIFO scheduling algorithm is used for the CPU.

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Burst Time</th>
<th>Order of Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>P2</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>P3</td>
<td>0</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

A time schedule, also called a Gantt chart, for the FIFO scheduler execution is:

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Burst Time</th>
<th>Order of Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td>23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td>27</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

P1 executes immediately. P2 will wait 20 units until P(1) finishes. P3 will wait 20+3 = 23 units. The average waiting time for all three processes is (0+20+23)/3 = 14 units of time.

Here is a sample of a portion of our FIFO scheduler execution, operating with a maximum of 15 processes in the ready queue at one time. The processes execute in order from the ready queue, except when an interrupt occurs and is processed.

**START: Starting FIFO scheduler**

- CPU bound Process 1 has entered the Ready Queue with a burst time of: 1030
- CPU bound Process 3 has entered the Ready Queue with a burst time of: 2977
- CPU bound Process 5 has entered the Ready Queue with a burst time of: 3290
- IO bound Process 2 has entered the Ready Queue with a burst time of: 624
- IO bound Process 4 has entered the Ready Queue with a burst time of: 3262
- IO bound Process 6 has entered the Ready Queue with a burst time of: 2416
- IO bound Process 8 has entered the Ready Queue with a burst time of: 2777
- IO bound Process 10 has entered the Ready Queue with a burst time of: 3159
- IO bound Process 11 has entered the Ready Queue with a burst time of: 897
- IO bound Process 12 has entered the Ready Queue with a burst time of: 3253
- IO bound Process 13 has entered the Ready Queue with a burst time of: 2999
- IO bound Process 14 has entered the Ready Queue with a burst time of: 2111
- IO bound Process 15 has entered the Ready Queue with a burst time of: 2560
IO bound Process 17 has entered the Ready Queue with a burst time of: 2052
IO bound Process 18 has entered the Ready Queue with a burst time of: 2223
Process 1 has entered processor and is executing
TERMINATED: CPU bound Process 1
CPU bound Process 7 has entered the Ready Queue with a burst time of: 3388
Process 3 has entered processor and is executing
Interrupt occurred
Process 3 waiting
Process 3 sent to IO Queue to execute IO request
Process 3 IO request executed. Sent back to Ready Queue.
BACK TO READY QUEUE: Process 3
Process 5 has entered processor and is executing
Interrupt occurred
Process 5 waiting

Round Robin Scheduler

Below is a simple example of process execution time using a round robin CPU scheduler, again assuming no interrupts occur. If the time quantum is specified as 1, and processes with burst times greater than 1 arrive, the processes will partially execute up to the level of the quantum and return to the ready queue, allowing another process to partially execute. The process execution table and Gantt expected for a round robin scheduler under these conditions are as follows.

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Burst Time</th>
<th>Order of Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0</td>
<td>20</td>
<td>1 to qtm or fin, then</td>
</tr>
<tr>
<td>P2</td>
<td>0</td>
<td>3</td>
<td>2 to qtm or fin, then</td>
</tr>
<tr>
<td>P3</td>
<td>0</td>
<td>4</td>
<td>3 to qtm or fin, back to 1</td>
</tr>
</tbody>
</table>

Gantt chart:

```
Time 0 1 2 3 4 5 6 7 8 9 10 11 12 to 27
P1 P2 P3 P1 P2 P3 P1 P2 P3 P1 P3 P1 P1
```

P1 will wait a total of 7 units, P2, 5 units, and P3, 7 units. The average waiting time under round robin is \((7+5+7)/3 = 6.33\) units, less than the 14 unit waiting time for FIFO. Here is a sample of a portion of our round robin scheduler execution, with a maximum of five processes in the ready queue. It can be seen that processes move in and out of CPU execution as the quantum level is reached.

START: Starting Round Robin scheduler
CPU bound Process 1 has entered the Ready Queue with a burst time of: 3295
IO bound Process 2 has entered the Ready Queue with a burst time of: 2537
IO bound Process 4 has entered the Ready Queue with a burst time of: 1451
IO bound Process 5 has entered the Ready Queue with a burst time of: 2670
IO bound Process 6 has entered the Ready Queue with a burst time of: 1993
Process 1 has entered processor and is executing
BACK TO READY QUEUE: Process 1
Process 2 has entered processor and is executing
BACK TO READY QUEUE: Process 2
Process 4 has entered processor and is executing
Interrupt occurred
Process 4 waiting
Executing interrupt.
Process 5 has entered processor and is executing
Interrupt occurred
Process 5 waiting
Process 5 sent to IO Queue to execute IO request
Process 5 IO request executed. Sent back to Ready Queue.
BACK TO READY QUEUE:
Process 6 has entered processor and is executing

**Shortest Job First Scheduler**

The shortest job first (SJF) scheduling algorithm looks at the CPU burst times for the processes in the ready queue and releases the one with the shortest burst time to the CPU for execution. Below is a simple example of process execution time using a shortest job first CPU scheduler, assuming no interrupts occur and a non-preemptive scheduler is deployed. The process execution table and Gantt expected for this shortest job first scheduling algorithm are as follows.

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Burst Time</th>
<th>Order of Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0</td>
<td>20</td>
<td>3</td>
</tr>
<tr>
<td>P2</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

Gantt chart:

<table>
<thead>
<tr>
<th>P2</th>
<th>P3</th>
<th>P1</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
<td>27</td>
</tr>
</tbody>
</table>

In SJF, P2 executes immediately because it is the shortest process. P3 will wait 3 units until P1 finishes. P1 will wait $3 + 4 = 7$ units. The average waiting time for all three processes is $(0 + 3 + 7)/3 = 3.33$ units of time. Compared to FIFO (14 units) and RR (6.33), non-preemptive SJF scheduling with no interrupts is expected to produce the least amount of wait time. Here is an excerpt from the SJF scheduler execution in our simulation, with a maximum of 5 processes in the ready queue. The shortest job, process 4 in this case, executes first.

START: Starting SJF scheduler
CPU bound Process 1 has entered the Ready Queue with a burst time of: 3295
IO bound Process 3 has entered the Ready Queue with a burst time of: 3161
IO bound Process 4 has entered the Ready Queue with a burst time of: 1451
IO bound Process 5 has entered the Ready Queue with a burst time of: 2670
IO bound Process 6 has entered the Ready Queue with a burst time of: 1993

Starting SJF
Process 4 has entered processor and is executing
Interrupt occurred
Process 4 waiting
Process 4 sent to IO Queue to execute IO request
Process 4 IO request executed. Sent back to Ready Queue.
BACK TO READY QUEUE: Process 4 with 840 burst remaining
Process 4 has entered processor and is executing
TERMINATED: IO bound Process 4
IO bound Process 7 has entered the Ready Queue with a burst time of: 1835

Process 7 has entered processor and is executing
TERMINATED: IO bound Process 7
IO bound Process 9 has entered the Ready Queue with a burst time of: 2629
Process 6 has entered processor and is executing
Interrupt occurred
Process 6 waiting
Executing interrupt.
BACK TO READY QUEUE: Process 6 with 134 burst remaining
Process 6 has entered processor and is executing

All three of these scheduling algorithms execute for the user selected number of processes and ready queue maximum during one run of the program. At the end of a run, statistics are displayed on the console for each scheduler. Below is an example of the statistics collected at the end of a sample run.

FIFO___________________
avg Turnaround time: 31093
max Turnaround time: 59403
min Turnaround time: 1002
avg wait time: 20966
max wait time: 45057
min wait time: 0
avg Response time: 18130
max Response time: 26201
min Response time: 0

RR___________________
avg Turnaround time: 39007
max Turnaround time: 61370
min Turnaround time: 1001
avg wait time: 25092
max wait time: 51066
min wait time: 0
avg Response time: 10724
max Response time: 16160
min Response time: 0

SJF___________________
avg Turnaround time: 23970
max Turnaround time: 59659
min Turnaround time: 783
avg wait time: 17325
max wait time: 55323
min wait time: 0
avg Response time: 17325
max Response time: 55323
min Response time: 0

DELAYS AND LIMITS

The three algorithms considered in this CPU scheduler simulation are widely implemented. They all have pros and cons. FIFO is simple to understand and code, but processes scheduled under FIFO may experience long delays if a process that arrived earlier has a long burst time. The processes are executed strictly in a serial fashion, one after another. If an early arrival is I/O bound, the CPU may sit idle for a period of time. This does not use the CPU system resource efficiently, and does not take advantage of the multiprocessing abilities of a CPU.

The delays and limits of the round robin approach are due to the high degree of context switching required to move processes in and out of the CPU as the quantum level is reached. Also, there are no priorities assigned in the RR scheduler; all jobs are considered equal.

The shortest job first approach has three major issues. First, it assumes that the scheduler knows all of the burst times for all possible programs that will arrive in the ready queue. SJF also assumes that all of these programs arrive at the same time, so it is possible to know in what order they should execute. Third, if a long program arrives
and is scheduled at the end of the line according to the SJF algorithm, and more jobs with shorter burst times arrive after it, the short jobs will continue to push the original job to the end of the queue. It may effectively starve.

EVALUATION AND RESULTS

DISCUSSION OF RESULTS

To evaluate the three scheduler simulations, FIFO, RRR and SJF. 5 execution runs were conducted for 10, 30 and 50 processes set at 5,10 or 15 processes in the ready queue at any given time. The results were compared against the other schedulers and the different number of processes simulated. The summary results are displayed in the table below.

In turnaround times, the schedulers behaved as expected. The average turnaround times for SJF for 10 processes were lowest; RR averaged the highest turnaround times. With a relatively small number of processes, SJF would be expected to produce the lowest average turnaround time. Similar results were observed for 30 processes and 50 processes. The average turnaround time for SJF was lowest; average turnaround time for RR was highest. This pattern is most apparent in the minimum turnaround times, which are lowest for SJF, then FIFO, then RR. However, the maximum turnaround times for 10, 30, and 50 processes were highest for SJF and lowest for FIFO. This may be attributed to the starvation effect of SJF scheduling discussed above. The starvation effect of SJF scheduling was observed in all three cases. The higher the number of processes, the more pronounced the effect.

The average response time for Round Robin was lowest in all cases in our simulation. FIFO average response times and SJF average response times were approximately equal, higher than RR. This may be attributed to the limit that RR places on process time in the CPU, up to the quantum. Long burst time processes will wait in the ready queue under SJF scheduling. Under FIFO, longer average response times could also be observed depending on the order in which jobs are received. Minimum response time was zero in all cases. Maximum response time was lowest for RR and highest for SJF, attributable to the SJF starvation effect with long processes.

Average wait time was found to be shortest for SJF and longest for RR. Maximum wait time was longest by far under the SJF algorithm, and shortest for FIFO. Minimum wait time was shortest for SJF (zero in all cases), intermediate for FIFO and longest for RR. The RR scheduler, requiring use of a quantum limit, evens out the wait times.

When we hold the total number of processes constant at 10, 30 or 50 and vary the numb in the ready queue, we see that the results for turnaround, response and wait times noted above are still true. The levels of these metrics increase as the number of processes in the ready queues increase, but the relative results are still there.
<table>
<thead>
<tr>
<th></th>
<th>10 process</th>
<th>30 process</th>
<th>50 process</th>
<th>10 process</th>
<th>30 process</th>
<th>50 process</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FIFO</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>avg turnaround</td>
<td>11668</td>
<td>35146</td>
<td>51197</td>
<td>12850</td>
<td>34309</td>
<td>53261</td>
</tr>
<tr>
<td>time:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>max turnaround</td>
<td>20617</td>
<td>64020</td>
<td>100602</td>
<td>21915</td>
<td>64358</td>
<td>99811</td>
</tr>
<tr>
<td>time:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>min turnaround</td>
<td>1997</td>
<td>2957</td>
<td>1727</td>
<td>2576</td>
<td>1889</td>
<td>2177</td>
</tr>
<tr>
<td>time:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>avg wait time:</td>
<td>6487</td>
<td>8041</td>
<td>7606</td>
<td>10659</td>
<td>23291</td>
<td>24568</td>
</tr>
<tr>
<td>max wait time:</td>
<td>16019</td>
<td>18754</td>
<td>19339</td>
<td>19235</td>
<td>51642</td>
<td>56234</td>
</tr>
<tr>
<td>min wait time:</td>
<td>316</td>
<td>564</td>
<td>537</td>
<td>598</td>
<td>499</td>
<td>510</td>
</tr>
<tr>
<td>avg response</td>
<td>5541</td>
<td>6350</td>
<td>6649</td>
<td>8792</td>
<td>18857</td>
<td>21474</td>
</tr>
<tr>
<td>time:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>max response</td>
<td>9340</td>
<td>10533</td>
<td>10777</td>
<td>16989</td>
<td>28128</td>
<td>30806</td>
</tr>
<tr>
<td>time:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>min response</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>time:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>RR</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>avg turnaround</td>
<td>14426</td>
<td>37308</td>
<td>56336</td>
<td>17368</td>
<td>41794</td>
<td>60144</td>
</tr>
<tr>
<td>time:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>max turnaround</td>
<td>22032</td>
<td>65783</td>
<td>107075</td>
<td>22823</td>
<td>66254</td>
<td>102914</td>
</tr>
<tr>
<td>time:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>min turnaround</td>
<td>3390</td>
<td>3716</td>
<td>3116</td>
<td>5423</td>
<td>4465</td>
<td>3424</td>
</tr>
<tr>
<td>time:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>avg wait time:</td>
<td>7687</td>
<td>8297</td>
<td>8319</td>
<td>15085</td>
<td>26818</td>
<td>26089</td>
</tr>
<tr>
<td>max wait time:</td>
<td>13975</td>
<td>19358</td>
<td>17913</td>
<td>19635</td>
<td>54764</td>
<td>58820</td>
</tr>
<tr>
<td>min wait time:</td>
<td>1870</td>
<td>1465</td>
<td>1344</td>
<td>4374</td>
<td>3392</td>
<td>2562</td>
</tr>
<tr>
<td>avg response</td>
<td>3189</td>
<td>3551</td>
<td>3713</td>
<td>5278</td>
<td>10777</td>
<td>11442</td>
</tr>
<tr>
<td>time:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>max response</td>
<td>5327</td>
<td>5293</td>
<td>5604</td>
<td>10812</td>
<td>16324</td>
<td>16633</td>
</tr>
<tr>
<td>time:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>min response</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>10 process</td>
<td>30 process</td>
<td>50 process</td>
<td>10 process</td>
<td>30 process</td>
<td>50 process</td>
</tr>
<tr>
<td>----------------</td>
<td>------------</td>
<td>------------</td>
<td>------------</td>
<td>------------</td>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td>SJF</td>
<td>5 Rdy Q</td>
<td>5 Rdy Q</td>
<td>5 Rdy Q</td>
<td>10 Rdy Q</td>
<td>15 Rdy Q</td>
<td>15 Rdy Q</td>
</tr>
<tr>
<td>avg turnaround time:</td>
<td>9036</td>
<td>28723</td>
<td>46193</td>
<td>9799</td>
<td>26563</td>
<td>42825</td>
</tr>
<tr>
<td>max turnaround time:</td>
<td>20989</td>
<td>62302</td>
<td>101696</td>
<td>21588</td>
<td>63829</td>
<td>102444</td>
</tr>
<tr>
<td>min turnaround time:</td>
<td>969</td>
<td>1046</td>
<td>903</td>
<td>994</td>
<td>872</td>
<td>984</td>
</tr>
<tr>
<td>avg wait time:</td>
<td>5141</td>
<td>7184</td>
<td>7445</td>
<td>7640</td>
<td>19209</td>
<td>21883</td>
</tr>
<tr>
<td>max wait time:</td>
<td>16560</td>
<td>55271</td>
<td>76938</td>
<td>18220</td>
<td>57563</td>
<td>94068</td>
</tr>
<tr>
<td>min wait time:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>avg response time:</td>
<td>5141</td>
<td>7184</td>
<td>7282</td>
<td>7640</td>
<td>19209</td>
<td>21883</td>
</tr>
<tr>
<td>max response time:</td>
<td>16561</td>
<td>55271</td>
<td>74482</td>
<td>18220</td>
<td>57563</td>
<td>94068</td>
</tr>
<tr>
<td>min response time:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
REFERENCES

Class Notes - Operating Systems, COSC 519, Towson University, Dr. R. Karne.