CL2006 - Operating Systems Spring 2024
LAB # 7 MANUAL (Common)

Please note that all labs' topics including pre-lab, in-lab and post-lab exercises are part of the theory and labsyllabus. These topics will be part of your Midterms and Final Exams of lab and theory.

Objectives:

1. Understand working of scheduling algorithms through test scenarios. Calculation of average response, waiting and turnaround time for each case.

Lab Tasks:

1. Compile the main.cpp file containing C code for various scheduling algorithms.
2. Understanding the test-case generation format
3. Run at all test cases for each algorithm and compute by hand average response, waiting and turnaround times based on each test case outputs.
4. Make two test cases on your own for trace and stats

Delivery of Lab contents:

Strictly following the following content delivery strategy. Ask students to take notes during the lab.

1st Hour

- Test cases 1 – 10

2nd Hour

- Test cases 11 – 12, Make new test cases.

3rd Hour (Not possible in Ramadan Schedule, Lab before Ramadan should utilize this hour)

- Use 3rd hour to make new cases.
- Optionally study the code.

Materials: GitHub Repo of Mr. Yousef Kotp **Created by:** Nadeem Kafi (053/03/2024)
DEPARTMENT OF COMPUTER SCEICEN, FAST-NU, KARACHI

EXPERIMENT 7
CPU Scheduling Algorithms

Algorithms (Self reading only: No coverage in lab during)

First Come First Serve (FCFS)

- First Come First Served (FCFS) is a scheduling algorithm in which the process that arrives first is executed first. It is a simple and easy-to-understand algorithm, but it can lead to poor performance if there are processes with long burst times. This algorithm does not have any mechanism for prioritizing processes, so it is considered a non-preemptive algorithm. In FCFS scheduling, the process that arrives first is executed first, regardless of its burst time or priority. This can lead to poor performance, as longer running processes will block shorter ones from being executed. It is commonly used in batch systems where the order of the processes is important.

Round Robin with varying time quantum (RR)

- Round Robin (RR) with variable quantum is a scheduling algorithm that uses a time-sharing approach to divide CPU time among processes. In this version of RR, the quantum (time slice) is not fixed and can be adjusted depending on the requirements of the processes. This allows processes with shorter burst times to be given smaller quanta and vice versa.
- The algorithm works by maintaining a queue of processes, where each process is given a quantum of time to execute on the CPU. When a process's quantum expires, it is moved to the back of the queue, and the next process in the queue is given a quantum of time to execute.
- The variable quantum allows the algorithm to be more efficient as it allows the CPU to spend more time on shorter processes and less time on longer ones. This can help to minimize the average waiting time for processes. Additionally, it also helps to avoid the issue of starvation, which occurs when a process with a long burst time prevents other processes from executing.

Shortest Process Next (SPN) – Textbook: Shortest Job First (non-preemptive)

- Shortest Process Next (SPN) is a scheduling algorithm that prioritizes the execution of processes based on their burst time, or the amount of time they need to complete their task. It is a non-preemptive algorithm which means that once a process starts executing, it runs until completion or until it enters a waiting state.
- The algorithm maintains a queue of processes, where each process is given a burst time when it arrives. The process with the shortest burst time is executed first, and as new processes arrive, they are added to the queue and sorted based on their burst time. The process with the shortest burst time will always be at the front of the queue, and thus will always be executed next.
- This algorithm can be beneficial in situations where the objective is to minimize the average waiting time for processes, since shorter processes will be executed first, and thus will spend less time waiting in the queue. However, it can lead to longer running processes being blocked by shorter ones, which can negatively impact the overall performance of the system.
- In summary, SPN is a scheduling algorithm that prioritizes the execution of processes based on their burst time, it's non-preemptive and it's commonly used in situations where the objective is to minimize the average waiting time for processes.

Shortest Remaining Time (SRT) - Textbook: Shortest Remaining Time First (non-preemptive)

- Shortest Remaining Time Next (SRT) is a scheduling algorithm that is similar to the Shortest Process Next (SPN) algorithm, but it is a preemptive algorithm. This means that once a process starts executing, it can be interrupted by a new process with a shorter remaining time.
- The algorithm maintains a queue of processes, where each process is given a burst time when it arrives. The process with the shortest remaining time is executed first, and as new processes arrive, they are added to the queue and sorted based on their remaining time. The process with the shortest remaining time will always be at the front of the queue, and thus will always be executed next.
- This algorithm can be beneficial in situations where the objective is to minimize the average waiting time for processes, since shorter processes will be executed first, and thus will spend less time waiting in the queue. Additionally, it can be useful in situations where the burst time of processes is not known in advance, since the algorithm can adapt to changes in the remaining time as the process is executing.
- In summary, SRT is a scheduling algorithm that prioritizes the execution of processes based on their remaining time, it's a preemptive algorithm, which means that it can interrupt a process that's already executing if a new process with a shorter remaining time arrives and it's commonly used in situations where the objective is to minimize the average waiting time for processes and burst time is not known in advance.

Feedback (FB) - Textbook: Multi-Level Queue

- Feedback is a scheduling algorithm that allocates CPU time to different processes based on their priority level. It is a multi-level priority algorithm that uses multiple priority queues, each with a different priority level.
- Processes with higher priority levels are executed first, and as new processes arrive, they are added to the appropriate priority queue based on their priority level. When a process completes execution, it is moved to the next lower priority queue.
- This algorithm can be beneficial in situations where the system needs to handle a mix of short and long-running processes, as well as processes with varying priority levels. By having multiple priority queues, it ensures that processes with higher priority levels are executed first, while also allowing lower-priority processes to eventually be executed.
- In summary, Feedback is a scheduling algorithm that allocates CPU time based on priority levels, it uses multiple priority queues with different levels of priority, processes with higher priority levels are executed first and when process completes execution, it is moved to the next lower priority queue, it's commonly used in situations where system needs to handle a mix of short and long-running processes, as well as processes with varying priority levels.

Feedback with varying time quantum (FBV) - Textbook: Multi-Level Feedback queue

- Same as Feedback but with varying time quantum.
- Feedback with varying time quantum also uses multiple priority queues and assigns a different time quantum for each priority level, it allows the algorithm to be more efficient by spending more time on higher-priority processes and less time on lower-priority processes.

Cloning a GitHub repository in bash shell

1. On GitHub.com, navigate to the main page of the repository.
2. Above the list of files, click Code.
3. Copy the URL for the repository.
   - ✓ To clone the repository using HTTPS, under "HTTPS", click
4. Open Bash Shell
5. Change the current working directory to the location where you want the cloned directory.
6. Type git clone, and then paste the URL you copied earlier.

7. Press Enter to create your local clone.

```
nakafi@ubuntu:~$
nakafi@ubuntu:~$ git clone https://github.com/yousefkotp/CPU-Scheduling-Algorithms.git
Cloning into 'CPU-Scheduling-Algorithms'...
remote: Enumerating objects: 252, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 252 (delta 2), reused 1 (delta 1), pack-reused 249
Receiving objects: 100% (252/252), 396.13 KiB | 516.00 KiB/s, done.
Resolving deltas: 100% (145/145), done.
nakafi@ubuntu:~$
```

Now follow the above procedure to close our GitHub repository in an appropriate folder: https://github.com/yousefkotp/CPU-Scheduling-Algorithms.git

Installation, Compilation, Execution

- Clone the repository
- Install g++ compiler and make (Installed)
- Compile the code using make command
- Run the executable file
    - ✓ Ensure that the output file is lab4

```
nakafi@ubuntu:~/CPU-Scheduling-Algorithms$ pwd
/home/nakafi/CPU-Scheduling-Algorithms
nakafi@ubuntu:~/CPU-Scheduling-Algorithms$
nakafi@ubuntu:~/CPU-Scheduling-Algorithms$
nakafi@ubuntu:~/CPU-Scheduling-Algorithms$ make
g++ -c main.cpp
g++ -o lab4 main.o
nakafi@ubuntu:~/CPU-Scheduling-Algorithms$ cat makefile
target:
        g++ -c main.cpp
        g++ -o lab4 main.o

nakafi@ubuntu:~/CPU-Scheduling-Algorithms$
```

- $cat 01a-input.txt

```
nakafi@ubuntu:~/CPU-Scheduling-Algorithms/testcases$ cat 01a-input.txt
trace
1
20
5
A,0,3
B,2,6
C,4,4
D,6,5
E,8,2
```

- $./lab4

```
nakafi@ubuntu:~/CPU-Scheduling-Algorithms$ ./lab4
trace
1
20
5
A,0,3
B,2,6
C,4,4
D,6,5
E,8,2
FCFS  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
------------------------------------------------
A     |*|*|*| | | | | | | | | | | | | | | | |
B     | | |.|*|*|*|*|*|*| | | | | | | | | | |
C     | | | | |.|.|.|.|.|*|*|*|*| | | | | | |
D     | | | | | | |.|.|.|.|.|.|.|*|*|*|*|*| | |
E     | | | | | | | | |.|.|.|.|.|.|.|.|.|.|*|*|
------------------------------------------------
```

Input Format of test cases

Line 1: "trace" or "stats"
Line 2: a comma-separated list telling which CPU scheduling policies to be analyzed/visualized along with their parameters, if applicable. Each algorithm is represented by a number as listed in the introduction section and as shown in the attached testcases. Round Robin and Aging have a parameter specifying the quantum q to be used. Therefore, a policy entered as 2-4 means Round Robin with q=4. Also, policy 8-1 means Aging with q=1.

FCFS (First Come First Serve)
RR (Round Robin)
SPN (Shortest Process Next)
SRT (Shortest Remaining Time)
FB-1, (Feedback where all queues have q=1)
FB-2i, (Feedback where q= 2i)

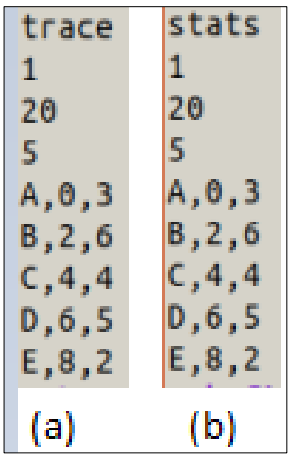

Figure 1

Line 3: An integer specifying the last instant to be used in your simulation and to be shown on the timeline.

Line 4: An integer specifying the number of processes to be simulated.

Line 5: Start of description of processes. Each process is to be described on a separate line. For algorithms 1 through 7, each process is described using a comma-separated list specifying:

1- String specifying a process name
2- Arrival Time
3- Service Time

Processes are assumed to be sorted based on the arrival time. If two processes have the same arrival time, then the one with the lower priority is assumed to arrive first.

In-Lab

1. **Complete the installation.**
2. **Compile and execute the file using the testcase shown above.**
3. **Read "Input Format of test cases" and verify it with file 01a-input.txt and 01b-input.txt shown in the Figure 1 part (a) and (b).**
4. **Now run all test cases and calculate the values by hand.**
5. **Submit snapshots of your hand-written work on GCR and hardcopy to your instructor.**