AHMEDABAD UNIVERSITY
School of Engineering and Applied Science

CPU Scheduling Algorithm

Subject: CSC341- Operating Systems Lab

Guided by: Dr. Sanjay Choudhary
Prof. M. T. Savaliya

Presented by: **Group 02**

Charmi Chokshi (201501021)
Hena Ghonia (201501032)

Outline

- Problem Definition
- Project Modules and Technical Details
- Flow Chart
- Operating Systems Concepts used
- Why we Modified existing RR Algorithm?
- Live Demo
- Linux Scheduling
- Conclusion
- Future Work
- References
- Q&A

Problem Definition

- **Process Scheduling:** An **act** of determining which process in the ready state should be moved to the running state
- **CPU scheduling:** A **process** which allows one process to use the CPU while the execution of another process is on hold/ in waiting state
- The **Aim** of CPU scheduling is to make the system
  - **Efficient**
  - **Fast**
  - **Fair**
- For achieving this, the scheduler must apply appropriate rules/ algorithms for swapping processes IN and OUT of CPU.
- There are many different criteria to check when considering the "**best**" scheduling algorithm:
  - CPU utilization
  - Throughput
  - Waiting time
  - Response Time
  - Turnaround time
  - Starvation
  - Context Switching
  - Possibility of Deadlock

Project Modules and Technical Details

Project Modules

This project is aimed at **Implementation** and **Analysis** of the following CPU scheduling Algorithms:

- **Uniprocessor Scheduling Algorithms:**
  - First-Come-First-Served (FCFS)
  - Round Robin (RR)
  - Shortest-Job-First (SJF)
  - Priority Scheduling (PS)
- **Real Time Scheduling Algorithms:**
  - Earliest Deadline First (EDF)
  - Rate Monotonic Scheduling (RMS)
- The project also include implementation of **Modified Round Robin Algorithm**

There is a **Simulator** which runs different CPU scheduling algorithms and gives different Utilization matrices:

- **Waiting time** of each process and average waiting time
- **Turn-around time** of each process and average turnaround time
- Number of **context switches** in case of RR and Modified RR algorithms
- **Throughput** for given time span
- **Gantt Chart**

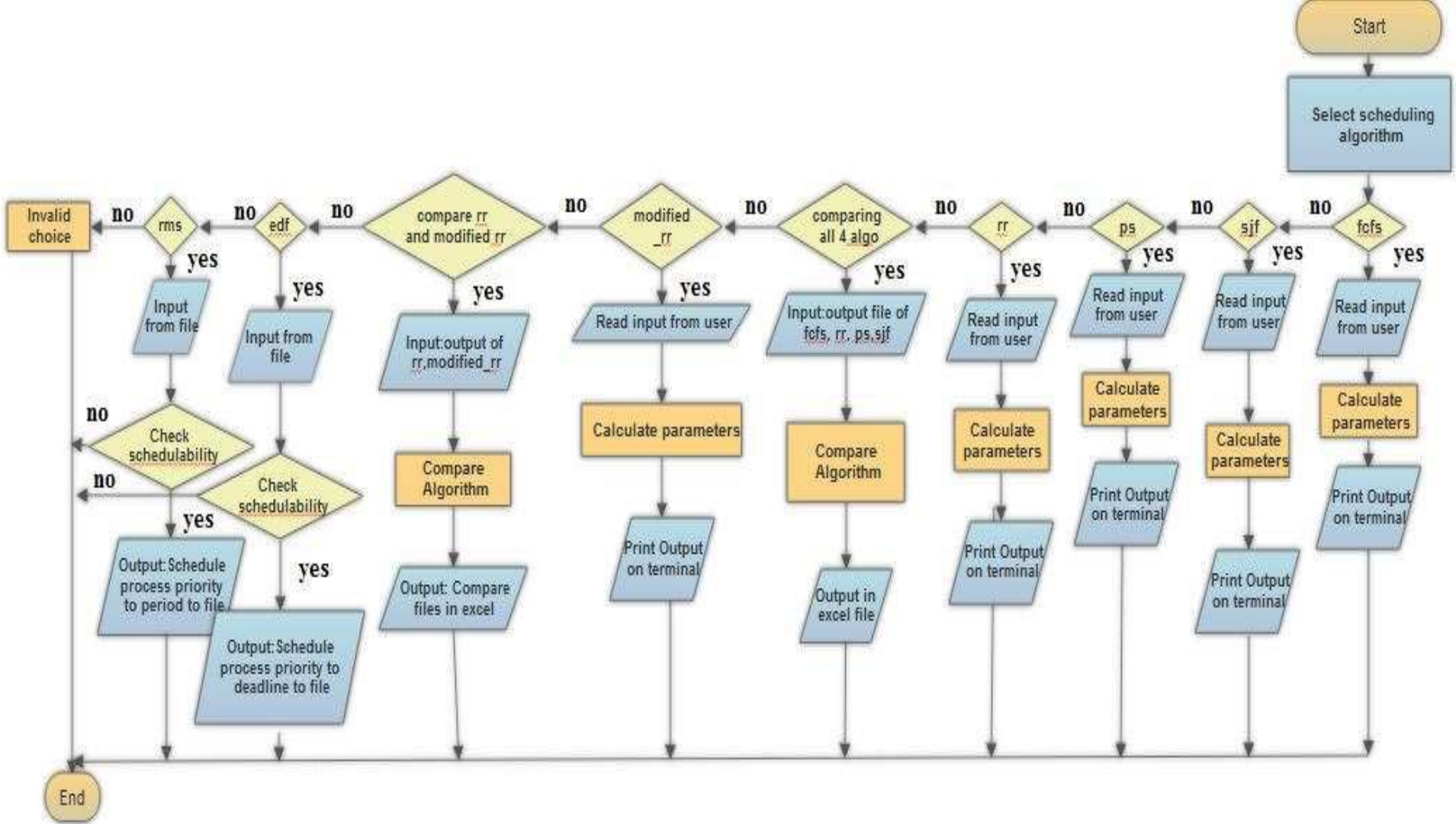
Assumptions

- Arrival time for each processes in FCFS, PS, RR, SJF and Modified RR Algorithm is 0 ms.
- Deadline and Time Periods in EDF and RMS Algorithm are of type integer.
- Number of processes to compare all 4 algorithms is set to 4.
- Number of processes to compare RR and Modified RR is set to 5.
- The value of Time Quantum in Modified RR is doubled after first iteration.

Note

- The project is made using **C Programming language.**
- Comparison of different scheduling algorithms by generating Bar Graphs is done using **Libxlsxwriter** which is a C library for creating Excel XLSX files.
- It contains 1 C file and 11 header files for implementation of different tasks.
- Some inputs are taken from user at run time and some are taken from file.

Flow Chart


Start
Select scheduling algorithm
fcfs
no
yes
Read input from user
Calculate parameters
Print Output on terminal
sjf
no
yes
Read input from user
Calculate parameters
Print Output on terminal
ps
no
yes
Read input from user
Calculate parameters
Print Output on terminal
rr
no
yes
Read input from user
Calculate parameters
Print Output on terminal
comparing all 4 algo
no
yes
Input:output file of fcfs, rr, ps,sjf
Compare Algorithm
Output in excel file
modified _rr
no
yes
Read input from user
Calculate parameters
Print Output on terminal
compare rr and modified rr
no
yes
Input:output of rr,modified_rr
Compare Algorithm
Output: Compare files in excel
edf
no
yes
Input from file
Check schedulability
no
yes
Output: Schedule process priority to deadline to file
rms
no
yes
Input from file
Check schedulability
no
yes
Output: Schedule process priority to period to file
Invalid choice
End

OS Concepts used

- **Interrupt Handling:** It is used using signals such as SIGINT and SIGTSTP to avoid unwanted behavior of program at the time of interrupt generation such as ctrl + C or ctrl + Z
- **POSIX Thread:** To compare all 4 uniprocessor scheduling algorithms, 4 threads are being created at runtime to achieve parallelism
- **Inter Process Communication (Using Signalling):** To avoid Floating Point Errors such as, division by 0, SIGFPE is used
- **File Management:** To take inputs such as number of processes, burst time, throughput etc. and after execution, to store the output in text files and excel files
- **Deadlock:** In EDF and RMS Algorithms, when more than 1 processes arrive at the same time, there is a possibility of deadlock
- **Mutual Exclusion:** This is avoided using Hold and Wait which means put one process in wait queue while the other is executing. The selection of process to be executed first is based on priority such as earliest deadline or shortest period.

OS Concepts used cont...

- **Starvation:** The process which has to wait for longer period of time to get the CPU access is a process which starves the most
- **Context Switching:** A process of storing and restoring the state of a process or thread so that execution can be resumed from the same point at a later time, it is used in n RR and Modified RR
- **Burst Time**
- **CPU Utilization**
- **Throughput**
- **Turnaround Time**
- **Waiting Time**
- **Period**
- **Deadline**
- **Uniprocessor Scheduling**
- **Real Time Scheduling**

Why we opt for modifying RR Algorithm?

- Round Robin reduces the penalty that short jobs suffer with FCFS by preempting running jobs periodically, and also saves starving of longer jobs and scheduling effort in case of SJF.
- The main advantage of Round Robin Scheduling is that every process gets the CPU and thus there is no starvation.
- But it has scope of improvement on following parameters:
    - High wait time
    - High turnaround time
    - Low throughput
    - More context switches

Modified Round Robin

Proposed Modified Round Robin Algorithm is an amalgamation of:

- Traditional Round Robin Algorithm
- Priority Scheduling Algorithm
- It also uses concept of change in quantum time

Algorithm of Modified Round Robin

Step 1: Allocate every process to CPU, a **single time** by applying **RR scheduling** with a initial time quantum (say k units)

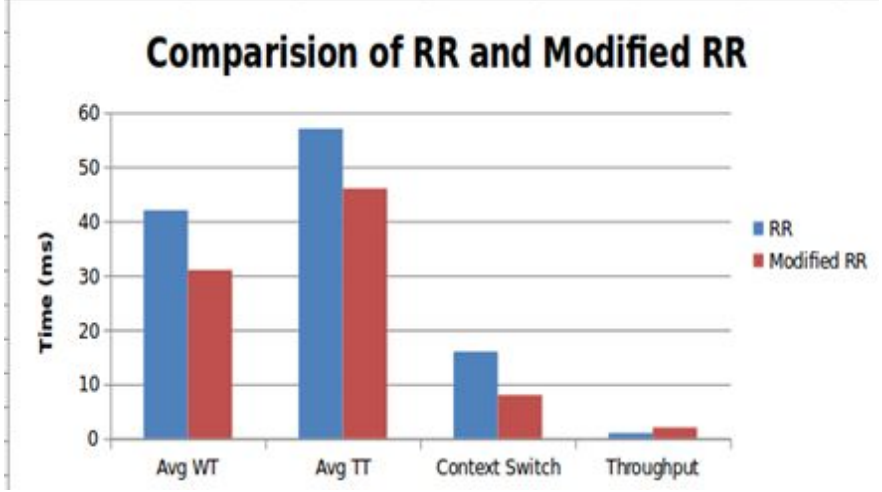
Step 2: After completing first cycle perform the following steps:

- **Double the initial time quantum** (2k units)
- **Select the shortest process** from the waiting queue and assign to CPU
- After that we have to select the next shortest process for execution by **excluding the already executed ones**

Step 3: For the complete execution of all the processes we have to repeat steps 1 and 2 in cycle

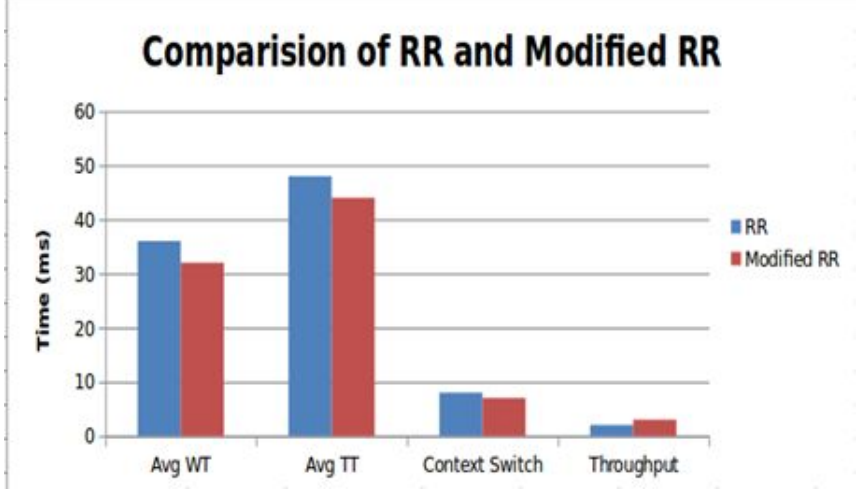
Input: 1
Number of processes: 5
Burst time of processes: 22, 18, 21, 10, 5
Time Quantum: 5
Throughput to test: 30 ms

Process	RR	Modified RR
Avg WT	42	31
Avg TT	57	46
Context Switch	16	8
Throughput	1	2



Input: 2
Number of processes: 5
Burst time of processes: 22, 9, 11, 17, 5
Time Quantum: 10
Throughput to test: 45 ms

Process	RR	Modified RR
Avg WT	36	32
Avg TT	48	44
Context Switch	8	7
Throughput	2	3



Analysis

Performance matrix of proposed RR algorithm as compared to traditional RR Algorithm:

- Approximately **20% less average Waiting Time**
- Approximately **15% less average Turnaround Time**
- Approximately **30-40% less Context Switches**
- Approximately **50-70% higher Throughput**

Thus, proposed algorithm has approximately 40% better performance as compared to simple RR algorithm.

Live Demo

Linux scheduling

- **Linux 2.4**
  - Uses FIFO and RR real time threads and other non real time threads.
  - No preemption, uses single run queue in SMP.
  - Good for load balancing but bad for memory cache.
- **Linux 2.6 (O(1) scheduler)**
  - Separate queue for each priority level (active queue and expire queue).
  - Tasks are scheduled in RR.
  - Task with more sleeping time is given higher priority.

Conclusion

- In this project, we **simulated, visualized** and **compared basic CPU scheduling algorithms** which are used in batch systems, interactive systems and real time systems.
- We come up with result that, for Real Time Systems, this type of algorithms will not work. So, implemented some **Real Time Scheduling Algorithms** using different OS concepts.
- We also proposed one algorithm named **Modified Round Robin Algorithm**, which gives approximately **40% better performance than traditional Round Robin**.

Future Work

- Algorithms we implemented was based on uniprocessor, but this could be done in **multiprocessor system**.
- Also complexities could be modified and multiprogramming can be considered. Even more variables and criteria could be used to provide a **broader range for comparison**.
- Extension of this project could be **Re-Compilation of Linux Kernel** using both real time and non real time scheduling algorithms.

References

- **Research Paper:**
- https://www.researchgate.net/publication/50194216_An_Optimized_Round_Robin_Scheduling_Algorithm_for_CPU_Scheduling
- http://ipasj.org/IIJCS/Volume2Issue11/IIJCS-2014-11-17-28.pdf
- https://www.coursera.org/learn/real-time-systems/.../earliest-deadline-first-example
- **Websites:**
- http://www.geeksforgeeks.org/gate-notes-operating-system-process-scheduling/
- https://en.wikipedia.org/wiki/Scheduling_(computing)
- https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/5_CPU_Scheduling.html
- **Reference Book:**
- Operating Systems-Internals and Design principles by William Stallings.

Any Questions...??

Thank you!