Grenze International Journal of Engineering and Technology, June Issue

Implementation and Visualization of Context Switching

Pavithra H[1], A Sania Kulsum[2], BH Abhisha[3] and Bhumika K[4]
[1-4]R V College of Engineering/Computer Science, Bengaluru, India
Email: {pavithrah, asaniakulsum.cs22, bhabhisha.cs22, bhumikak.cs22}@rvce.edu.in

***Abstract*—This paper presents a novel approach utilizing the GTK3 toolkit to simulate and visualize context switching in operating systems, focusing particularly on Round Robin scheduling. Context switching, crucial for multitasking efficiency, involves saving and restoring process states. Through dynamic transitions between processes, the framework elucidates Round Robin's CPU resource allocation, aiding in understanding its efficiency and fairness. The visualization serves as a valuable tool for researchers, educators, and practitioners, offering insights into CPU scheduling intricacies. Additionally, it provides a platform for experimentation, enabling the exploration of scheduling performance under various parameters. Overall, this work advances comprehension of CPU scheduling algorithms and offers a practical tool for real-world scenario analysis.**

***Index Terms*— Context switching, Round Robin, Ready queue, Blocked queue.**

I. INTRODUCTION

We've integrated context switching and round-robin (RR) scheduling into our operating system architecture. Whenever a process switch occurs, we ensure seamless transition by preserving process states, crucial for uninterrupted execution. This includes retaining execution context when time slices complete, allowing the processor to smoothly transition to the next task while preserving the previous one's state. Our implementation lies on the Process Control Block (PCB) to store essential process information like Process Identifier (PID), state descriptors, Program Counter (PC), and Stack Pointer (SP), facilitating precise execution management and memory allocation. In line with the operating system's five-state model, we've utilized queues to manage process states effectively, including Blocked, Running, and Ready states. Context switching is seamlessly executed through these queues, with the Ready Queue employing a circular structure to maintain processes in the ready state until termination or completion of execution. Our scheduling strategy, centred on RR, ensures fair treatment to all processes, minimizing starvation risk with a quantum set to 2. By encapsulating critical process details, the PCB enables efficient context switching, allowing multitasking within our operating system environment.

II. RELATED WORK

[1] explores memory system design, particularly caching levels, emphasizing the empirical nature of their design. It introduces the single-pass method as an effective approach for evaluating various cache designs in one pass over a trace. However, it acknowledges that multiprogramming can degrade memory performance due to context switching. To address this, the paper proposes a method extending the recurrence/conflict model to account for context switching effects. This method accurately assesses program susceptibility to context switching across different cache dimensions and intensities, demonstrating good accuracy compared to other test methods and

Grenze ID: 01.GIJET.10.2.671

aligning with previous research on multiprogramming effects. [2] addresses the challenges associated with context switching and proposes strategies to mitigate its impact, aiming to optimize process execution for improved efficiency. However, it acknowledges that previous research and conclusions in this area may not offer a definitive solution or a positive outlook. Despite quoting previous studies, the paper suggests that the current understanding of context switching optimization may still be limited or inconclusive.[3] highlights the discrepancy in utilizing flash memory's performance in generic operating systems like Linux, optimized for hard disks. It addresses the issue of excessive context switching when using flash memory, which burdens the system despite its fast access time. By optimizing the context switching scheme, the paper achieves a 5% overall performance increase across various workloads.[4] emphasizes the critical role of cache performance in sustaining the speed of fast processors, which relies on the locality of reference. Context switching in operating systems can disrupt this locality, resulting in cache performance degradation beyond kernel operation costs. Through simulation using cache traces, the paper estimates that context switching imposes a significant cost, ranging from thousands of cycles to tens or hundreds of microseconds, depending on cache parameters. [5] investigates the impact of multitasking and interruptions on software developers' productivity across multiple projects. Through self-reported work logs, it quantitatively evaluates cross-project interruptions, finding that developers spend an average of 17% of their effort on such interruptions. The study reveals discrepancies between estimated and actual interruption effort, weak correlation between project number and interruption effort, and a strong correlation between project number and reported interruptions. [6] investigates the performance and overhead of context switches in modern AI workloads, revealing their significant impact on cloud servers and edge/IoT devices. Through empirical study, it highlights the complexities and inefficiencies introduced by context switching, leading to performance degradation and system inefficiency. The findings underscore the need for optimizing the system stack of modern operating systems to better support AI workloads on both cloud and edge systems.[7] delves into context awareness within business process modeling (BPM), introducing the Context Model for BPM (CM4BPM) and a Role-Based BPM (RBPM) model. It proposes an approach for enacting processes based on contextual knowledge, aiming to enhance assignment adequacy during process enactment. By incorporating contextual knowledge, the approach seeks to improve the design and implementation of business processes, enabling support for context-aware process enactment within existing process modeling languages.[8] addresses the static nature of context in contemporary business process management software. It proposes enabling runtime context switching within the object-aware process management paradigm, allowing for flexibility in adapting to changing physical or logical surroundings. The paper introduces algorithms for implementing context switches at different granularity levels, from shifting entire process instances to migrating sub-processes between parent processes. By showcasing these advanced concepts, the paper demonstrates the maturity of data-centric BPM. [9] addresses the need for high process variability and flexibility in response to customer expectations, product diversity, and regulatory requirements. It introduces the concept of context-aware process injection, allowing for sophisticated modeling and dynamic execution of process variants based on contextual factors. The approach enables the adjustment of running processes to accommodate contextual changes and emerging demands, even facilitating dynamic data flow. A case study demonstrates the feasibility and benefits of this approach, showcasing its potential to provide high process flexibility at runtime.

III. Proposed Method

There are initialized processes, each represented by a different text file with execution instructions in the same directory with certain burst value times. The contents of these files are including data, execution instructions unique to each process, and computer code. The process lifecycle in an operating system involves key states. Beginning as "New," processes are prepared for creation but not yet instantiated. Upon creation, they transition to "Ready," awaiting CPU allocation for execution. Once granted CPU time, they enter the "Run" state, actively executing instructions. If processes require external resources, they move to "Blocked" or "Wait," pausing execution until resources are available. Upon completion, they transition to "Terminated," releasing allocated resources. Processes can also enter "Suspend Ready" or "Suspend Wait," indicating temporary storage in external memory due to constraints, with potential return upon memory availability or task completion. These states delineate the dynamic lifecycle of processes, influenced by resource availability and task demands.

A circular queue data structure is used to create a ready queue. The set of processes that are prepared for CPU execution is represented by this queue. Round Robin scheduling allocates CPU time to processes in fixed time slices, cycling through them in a circular order. Each process gets a turn to execute for a predetermined time quantum before being pre-empted. This algorithm ensures fairness among processes, particularly useful in time-

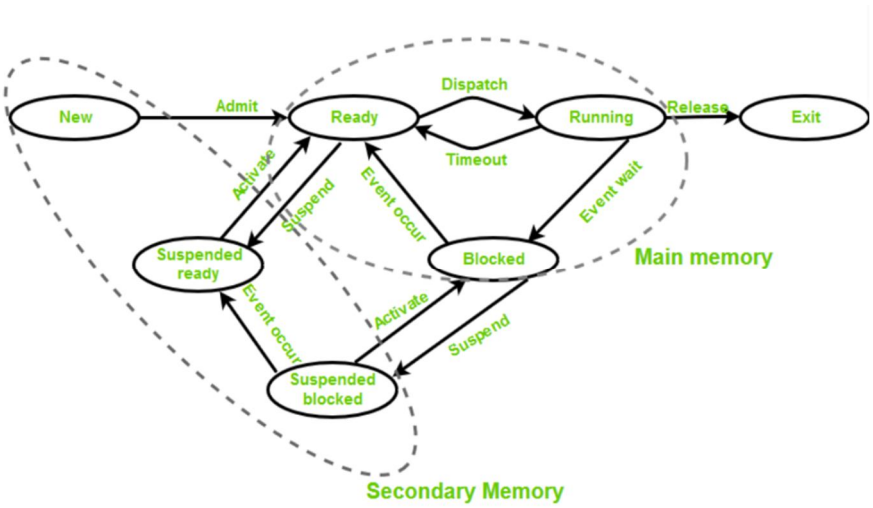

Fig .1 Process States

sharing systems. However, frequent context switches can lead to overhead, impacting overall system performance. Round-robin scheduling is the scheduling algorithm that is used. The amount of time that each process spends using the CPU is set 2s.

A process's completion time grows by the quantum duration during each quantum, and the ongoing process's Program Counter (PC) is increased. Each quantum executes two instructions, which suggests that the execution process is likely multi-step. Context switching involves saving and restoring the state of processes to facilitate multitasking, crucial for efficient CPU management. It allows the operating system to switch between different tasks, preserving their progress and data while ensuring fair allocation of CPU resources. However, frequent context switches can incur overhead, impacting overall system performance. To preserve their state across quantum intervals, the process's variable values are pushed into a stack. Any process that gets blocked is added to a block queue, signifying that it is awaiting the fulfilment of a specific requirement or resource.

Information like the process ID, state of execution at that moment, PC value, stack pointer (SP), and maybe other pertinent data are probably contained in the PCB. To guarantee continuous execution, the processor is subsequently turned over to the following process in the ready queue. Each process's Process Control Block (PCB) state is shown both before and after it is executed.

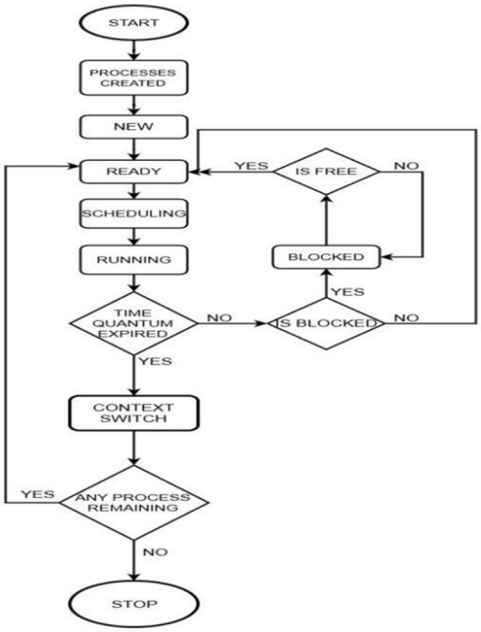

Fig .2 Flowchart of Proposed Algorithm

In our GUI built with GTK3, we've integrated buttons to represent the allocation and release of resources within the operating system. These buttons simulate the actions of assigning resources to processes and freeing them up when they're no longer needed. When a user clicks on the "Allocate" button, it signifies the allocation of resources to a specific task or process, whereas clicking on the "Free" button indicates the release of those resources back to the system for other tasks to use. This interactive approach provides users with a hands-on experience, allowing them to directly engage with the resource allocation process in a visually intuitive manner. Here, when the user wishes to allocate a resource to the processes, we make use of the GUI so that a short-term scheduler can select one of the processes for scheduling, and when the CPU gets allocated, they switch their state to running.

IV. Experimental Results and Discussion

Thus, the output appears as a GUI interface that appears after the process has been run. The PCB contents of the processes, including as the PID, PC, and state, are displayed on the graphical user interface (GUI) once the user has chosen and selected the resource he wants. We can access the processes in the ready queue and blocked queue using the output interface. Additionally, a message stating "process (number of the process) is completed" appears on the screen after the procedure is finished. The corresponding parts, such as the read queue and blocked queue, show the processes as they run and undergo modifications. As the application runs, their PCB contents are also updated and shown.

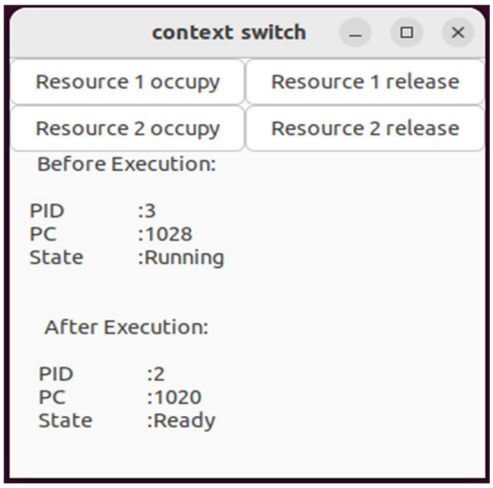

Fig .3 GUI Interface

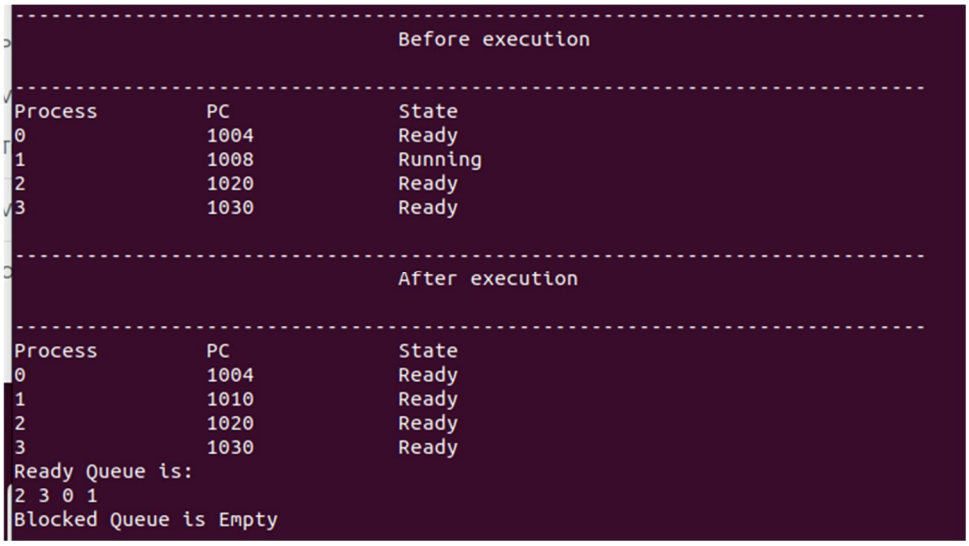

Fig .4 Output Depiction on the terminal

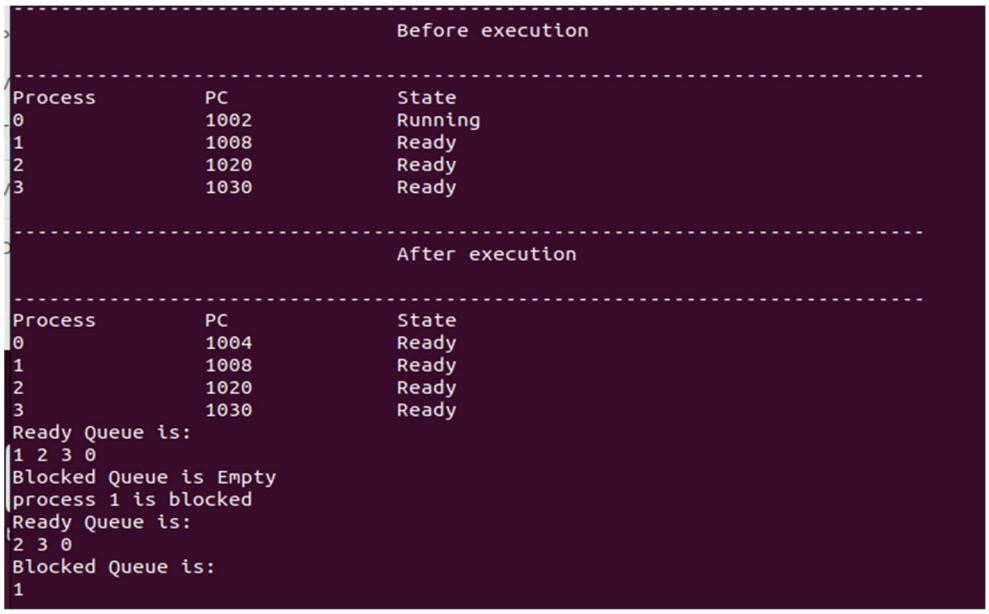

Fig .5 Output Depiction on the terminal

V. CONCLUSION

Operating systems rely on the core idea of context switching, which enables the CPU to effectively manage several processes at once. The CPU's ability to multitask would be severely constrained without context switching, allowing it to only run one process at a time. A variety of scheduling strategies, each intended to maximize resource consumption and system performance, can be implemented with context switching. The most straightforward scheduling algorithm that may be used without context switching is First Come First Serve (FCFS). Processes in FCFS are carried out in the order that they are received, without regard to their priority or execution time. Despite being simple, this method might not be the best for increasing throughput or reducing reaction times. More sophisticated scheduling algorithms, such as Round Robin (RR), Shortest Remaining Time (SRT), Shortest Process Next (SPN), and Highest Response Ratio Next (HRRN), rely heavily on context switching to function effectively. It is difficult to estimate a process's service time (execution time) properly in real-world circumstances, which makes algorithms like SRT, SPN, and HRRN less useful. Nonetheless, these algorithms are useful in academic and research settings and provide as theoretical models for comprehending scheduling ideas. RR with a tiny time quantum is selected to effectively depict context switching. Small quantum numbers cause frequent interruptions in operations, which facilitate quick switching between them by the scheduler. This demonstrates how context switching is dynamic and important in multitasking settings. Furthermore, responsiveness and fairness are balanced by RR with a tiny quantum, which makes it a useful option for illustrating context switching in practice.

REFERENCES

[1] Wen-Mei W. Hwu Thomas M. Conte, "The Susceptibility of Programs to Context Switching " March 21, 1991

[2] Muhammad Q Shahzad “A Survey Paper on Context Switching” Syracuse University

[3] Sehwan Lee, Seungwan Hyun, Kern Koh, “Selective Context Switching on Flash Memory” Published in: 2009 International Conference on Computational Science and Its Applications

[4] Jeffrey C. Mogul and Anita Borg “The Effect of Context Switches on Cache Performance” ASPLOS-IV Proceedings - Forth International Conference on Architectural Support for Programming Languages and Operating Systems, Santa Clara, California, USA, April 8-11, 1991.

[5] Alexey Tregubov, Barry Boehm, Jo Ann Lane, Natalia Rodchenko “Impact of task switching and work interruptions on software development processes” University of Southern California , July 2017 DOI:10.1145/3084100.3084116 Conference: the 2017 International Conference

[6] Patrick Otoo Bobbie, Chih-Cheng Hung,Yong Shi,Kun Suo “Quantifying context switch overhead of artificial intelligence workloads on the cloud and edges” Kennesaw State University, SAC '21: Proceedings of the 36th Annual ACM Symposium on Applied Computing March 2021

[7] Oumaima Saidani,Selmin Nurcan “Context-awareness for adequate business process modelling” 2009 Third International Conference on Research Challenges in Information Science DOI: 10.1109/RCIS.2009.5089281
[8] Kevin Andrews, Sebastian Steinau, and Manfred Reichert “Dynamically Switching Execution Context in Data-Centric BPM Approaches” Institute of Databases and Information Systems, Ulm University, Ulm, Germany, Enterprise, Business-Process and Information Systems Modeling. 2020; 387: 3–19. Published online 2020 May 5. Doi: 10.1007/978-3-030-49418-6_1
[9] Nicolas Mundbrod, Gregor Grambow, Jens Kolb, Manfred Reichert “Context-Aware Process Injection OTM Confederated International Conferences " On the Move to Meaningful Internet Systems.