The first scheduling algorithm that caught my eye was the “Round-Robin” scheduling algorithm. The Round-Robin algorithm has two major components, a queue and a “Time Quantum” (also known as a “Time Slice” or “Time Slot”). The time quantum is a unit of time typically in the size of milliseconds, and it’s the allotted time that the process can be acted upon before it gets sent back to the end of the process queue. This scheduling algorithm gets its name from the cyclical nature in which it manages processes. If the process can’t be completed during the time quantum, then it must wait until the other processes have had a turn to be operated upon.

This scheduling algorithm is well liked because it’s fair. Every process gets an equivalent amount of CPU attention before it’s preempted (interrupted). This fairness also typically results in a quick response time for a process to be acted upon since long running processes are limited by the time quantum. A short quantum would result in tasks moving quickly through the queue, which typically means an even faster response time.

This algorithm isn’t perfect though. It requires a lot of overhead for the processor to context switch since it needs to save the state of the current process and load the state of the next one. A short quantum will exacerbate the overhead problem since processes would have a higher probability of being preempted. This algorithm can also result in “starvation” of critical processes, which means that critical processes could repeatedly be preempted in favor of low priority processes for the sake of fairness. There are variations of the round-robin algorithm that attempt to resolve the starvation issue by assigning priority to tasks. This priority assignment gives high priority tasks either a more favorable position in the queue after the quantum is completed, an increased time quantum, or both.

The second scheduling algorithm that stood out to me is the multilevel queue scheduling algorithm. This algorithm is much more advanced than the round-robin algorithm. Each process is assigned a priority level, typically determined by process type (interactive user process or background batch process), execution time, resource utilization, completion deadline constraints, and many other variables. This scheduling algorithm consists of many queues, one for each assigned priority level. After the priority of a process is determined, it’s added to its respective priority queue. Each priority queue has its own scheduling algorithm that’s configured based upon what priority of processes it’s managing. For example, the high priority queue might have a round-robin scheduling algorithm with a short quantum so that those processes complete rapidly, and a low priority queue might have a round-robin scheduling algorithm with a large quantum to keep processing overhead low. The high priority queue can also preempt the low or medium priority queue so that critical tasks get processed timely.

The biggest advantage of this algorithm is how efficiently it manages processes with different priorities and properties. Critical tasks are always processed with high authority, and low priority tasks are put behind the high priority tasks. This is also a highly responsive scheduler. If you consider that interactive user processes are typically high priority tasks, the responsiveness of the system from the user’s perspective would be very high. The preemption of low priority processes in favor of high-priority processes is also a contributing factor to the responsiveness of the system.

A disadvantage of this algorithm is the possible starvation of low-priority tasks. Most multilevel queue algorithms resolve this problem by reassigning priority to processes after quantum completion with aging being factored into the priority calculation. So, if a low priority task has sat for an extended period in the low priority queue, it can elevate itself in priority based upon much it has aged in the system. Another disadvantage would be the complexity of configuration. There is a lot to manage when

considering assigning priority to processes (especially when considering aging policies), configuring sub-queue scheduling algorithms, and preemption policies. There is also a lot of context switching in this scheduling algorithm between the different priority queues and preemption of low priority tasks, which can lead to a lot of overhead as well.

After submitting my discussion board, I realized that my response was incomplete. I wrote my original post completely off the rubric and didn't answer the real discussion questions, so it looks like I got some work completed toward my IP2 😅 Here is my response to the questions asked in the assignment details:

The suitability of the Round-Robin scheduling algorithm depends on the specific environment it operates in. In non-virtual settings where dedicated hardware resources are readily available, Round-Robin offers advantages in terms of fairness and predictable response times. It ensures that each process receives an equal share of CPU attention, making it an excellent choice for stable and controlled systems. However, there are drawbacks to consider, including the potential for high context-switching overhead, especially when using a short time quantum. Additionally, the algorithm might pose challenges in environments where efficiency and resource utilization are important due to the risk of starving critical processes.

On the other hand, in virtual environments where resource sharing and adaptability are crucial, Round-Robin's fairness and quick response times can be beneficial for ensuring equal resource allocation among virtual entities. However, the context-switching overhead issue can be magnified in virtual environments with resource contention. The adaptability and dynamic resource allocation requirements in virtual environments make Round-Robin a better fit, emphasizing flexibility and shared resources over rigid fairness. Therefore, Round-Robin is better suited for virtual environments.

In non-virtual environments, the multilevel queue scheduling algorithm offers the advantage of efficiently managing processes with diverse priorities, ensuring that high-priority tasks receive immediate attention, and enhancing overall system responsiveness. However, it also can potentially lead to low-priority task starvation, which necessitates priority reassignment and aging policies for resolution. Managing sub-queue scheduling, context switching overhead, and preemption challenges is generally more manageable in non-virtual environments with dedicated hardware resources.

In virtual environments, the multilevel queue algorithm excels in adapting to dynamic resource allocation and ensuring equitable resource distribution among virtual entities. Similar to non-virtual environments, it faces the challenge of addressing low-priority task starvation and requires the implementation of priority reassignment and aging strategies. The complexity of configuring the algorithm, managing context switching overhead, and addressing preemption complexities can become more pronounced in virtual settings.

Overall, the multilevel queue algorithm appears better suited for virtual environments, where adaptability and dynamic resource management are critical. While both environments encounter challenges related to low-priority task starvation, the adaptability and flexibility

demanded in virtual environments make it the more suitable choice for efficiently allocating resources among diverse virtual entities.

The multilevel queue scheduling algorithm is the more effective choice over the Round-Robin algorithm for both virtual and non-virtual environments, including real-time scenarios. It excels in managing processes with varying priorities, ensuring that high-priority tasks are quickly addressed while still accommodating lower-priority ones. Its adaptability, particularly important in dynamic virtual environments, makes it the better choice for efficient resource allocation to a range of virtual entities. In non-virtual settings, it offers stability, predictable resource allocation, and fairness. Also, when configured appropriately, it can effectively address real-time constraints by adjusting priority. This versatility makes the multilevel queue algorithm stand out as the more robust and suitable option over Round-Robin for a range of computing environments.

Hey Brian, I'm picking on you again this week! I also looked at Round-Robin scheduling, I think because Dr. Goforth mentioned that it's one of the better liked scheduling algorithms in our live share. I thought it was interesting that you outlined that the algorithm isn't good for tasks with unequal resource needs. I said something similar about prioritization, but I didn't consider that maybe different processes might require different hardware resource utilization that Round-Robin wouldn't recognize. Maybe I read a little too much into what you said though. I really liked the analogies in your segment on First Come First Serve, especially the one about TSA not letting you push to the front of the line if you're late, I thought that was pretty funny and accurate! I agree with you that Round-Robin is a more desirable algorithm overall, I think it definitely depends on the use case of the system. If your system is a super computer targeted at long running training of complex machine learning algorithms or batch processes, maybe FCFS would be a better alternative than Round-Robin. Thanks for sharing your insights!

Aidan

Hi Justin! I liked that you described the disadvantage of priority scheduling in that it can lead to starvation. I think it was briefly mentioned in chapter 9 of our textbook, but some prioritization algorithms evaluate process "age" as a variable in how they prioritize processes. So, if a process is initially assigned to a low priority but it's sat in the queue for an extended period without completion, it can be re-assigned to a higher priority level so that it gets completed. From what I understand this can add a lot of complexity to the scheduling algorithm, but it's a way that some people have answered the starvation problem. I liked how you contrasted the use cases of Round-Robin and Priority scheduling, I think it's really interesting to see how they work together in some of the hybrid scheduling algorithms like multilevel queue scheduling. Thanks for sharing!

Aidan