ECE 3058

Thread Scheduling Lab

Name: Mythri Muralikannan

GT Username: mmuralikannan3

Problem 1B

Execution Times:

- 1 CPU: 67.6s

- 2 CPUs: 35.9s

- 4 CPUs: 33.4s

Upon analysis, it is evident that the correlation between the number of CPUs and total execution time is not linear but there is a decrease. The significant decrease in time from 1 to 2 CPUs contrasts with the relatively minor drop in time from 2 to 4 CPUs. Here are several reasons as to why this might be the case.

- Workload Distribution: When transitioning from 1 to 2 CPUs, there is a split in the workload that reduces the execution time considerably as compared to having a single processor do all the work. However, the effectiveness diminishes as more CPUs are added. This is because workload distribution among CPUs is not always evenly balanced. For instance, a program with 200 lines of code may not neatly divide into 50 lines per CPU. Instead, the workload might be unevenly distributed, such as 150, 10, 20, 20, causing less efficiency gains. While more CPUs theoretically offer parallel execution, real-world efficiency depends on how well tasks can be divided and allocated. Uneven task distribution or dependencies between tasks can lead to suboptimal utilization of additional CPUs.
- Overheads: As the number of CPUs increases, so does the overhead associated with managing them. Context switching, synchronization, and coordinating multiple processors introduce overhead that can offset the benefits of parallelism.

In conclusion, the non-linear relationship between the number of CPUs and execution time stems from the a combination of workload distribution, real- work inefficiency of CPU utilization, and increasing overheads. While adding CPUs can initially lead to significant reductions in execution time, the diminishing returns become evident with additional CPUs.

Problem 2B

---

800 ms:

Context Switches: 209

Execution Time: 67.5s

Time Spent in Ready State: 312.6s

600 ms:

Context Switches: 223

Execution Time: 67.5s

Time Spent in Ready State: 302.1s

400 ms:

Context Switches: 266

Execution Time: 67.4s

Time Spent in Ready State: 292.7s

200 ms:

Context Switches: 426

Execution Time: 67.7 s

Time Spent in Ready State: 299.2 s

Analysis

- Total Waiting Time: As seen in the results, the total waiting time decreases as the time slice decreases. This is expected because shorter time slices allow the scheduler to switch between processes more frequently, reducing the time processes spend waiting in the ready state.
- Context Switches: The number of context switches increases as the time slice decreases. This is because shorter time slices lead to more frequent preemptions and context switches.
- Execution Time: Execution time more or less remains constant throughout.

However, in a real operating system, the shortest possible time slice is usually not the best choice for several reasons:

- Overhead: Shorter time slices lead to more frequent context switches, which incur overhead. Each context switch requires saving and restoring process states, which can become significant with very short time slices.
- Scheduling Efficiency: Extremely short time slices can lead to inefficient scheduling, especially in systems with many processes. The scheduler may spend more time switching between processes than actually executing them. This is known as the "overhead of overheads."
- Response Time: Very short time slices might cause delays in responsiveness for interactive tasks. For example, a user interacting with a program might experience lag if the time slice is too short.

While shorter time slices can reduce total waiting time and improve responsiveness for some tasks, they come with trade-offs in terms of overhead and scheduling efficiency. An extremely small time slice will cause a lot of context switches leading to a decrease in processing efficiency. In real operating systems, time slices are typically chosen based on a balance of these factors to achieve optimal system performance.

Problem 3B

None given