Khaula Molapo

20240001

Operating System – Week 11

RTOS vs GPOS Comparison Table

Feature	GPOS (Example: Linux/Windows)	RTOS (Example: FreeRTOS/VxWorks)
Primary Goal	Focuses on user experience, multitasking, and resource sharing	Focuses on predictable and fast response times
Scheduling Policy	Time-sharing (fair scheduling for all processes)	Priority-based preemptive scheduling
Kernel Size	Large and complex	Small and lightweight
Interrupt Latency	High latency because of many background services	Very low latency for fast real-time response
System Overhead	High due to multitasking and background processes	Low overhead for efficient control
Typical Applications	PCs, servers, smartphones	Embedded systems, robots, medical and automotive systems

Summary

A General-Purpose Operating System (GPOS) like Linux or Windows is designed to handle many tasks at once and keep the system user-friendly. It focuses on multitasking, security, and supporting many programs at the same time. However, this design makes it unpredictable in timing. The system can delay tasks because of interrupts, background services, or heavy CPU use.

A Real-Time Operating System (RTOS) like FreeRTOS or VxWorks, on the other hand, is made for tasks that must happen exactly on time. In something like an airbag controller, even a few milliseconds of delay can mean failure or danger. RTOS ensures that high-priority tasks always run immediately when needed.

You cannot just "tune" Linux to act like an RTOS because the Linux kernel and structure are too large, with too many unpredictable processes. Even optimized, Linux cannot guarantee the precise, consistent timing required in critical systems like medical devices or airbags. That's why true hard real-time systems need a real RTOS.

2. FreeRTOS Simulation Report
Code Example:

```
void Task1(void *pvParameters) {
  while(1) {
    printf("Task 1 Running\n");
    vTaskDelay(1000 / portTICK_PERIOD_MS);  // Delay for 1 second
  }
}

void Task2(void *pvParameters) {
  while(1) {
    printf("Task 2 Running\n");
    vTaskDelay(500 / portTICK_PERIOD_MS);   // Delay for 0.5 seconds
  }
}

int main(void) {
  xTaskCreate(Task1, "Task1", 1000, NULL, 1, NULL);
  xTaskCreate(Task2, "Task2", 1000, NULL, 2, NULL);
  vTaskStartScheduler();
  while(1);
}
```

Observations:

- Task 2 has higher priority, so it runs more often.
- Using vTaskDelay() makes tasks wait without blocking others.
- Semaphores can control access when two tasks share the same resource.
- The experiment shows that task priority directly affects execution timing — higher priority tasks always run first, making the system predictable.