REAL-TIME SCHEDULING

Background

En la computación en tiempo real, las acciones deben realizarse en momentos específicos para que el sistema funcione correctamente (es como un baile sincronizado donde cada paso debe tomarse en el momento adecuado para que todo funcione sin problemas.)

Por ejemplo, Un sistema de control de tráfico aéreo. Si un avión necesita cambiar de rumbo, el sistema debe hacerlo en el momento exacto para evitar colisiones. Si el sistema se retrasa, podría haber graves consecuencias.

en un sistema operativo en tiempo real, la prioridad es garantizar una respuesta rápida y predecible a eventos críticos para mantener la seguridad y la eficiencia del sistema.

Characteristics of Real-Time Operating Systems

Los sistemas operativos en tiempo real tienen cinco características principales:

1. **Determinismo**: Realizan operaciones en momentos predeterminados. Es decir, actúan de manera predecible y precisa según un plan establecido.
2. **Capacidad de respuesta**: Responden rápidamente a eventos externos, como interrupciones, sin demoras significativas.
3. **Control del usuario**: Los usuarios tienen más influencia sobre cómo se asignan los recursos y se priorizan las tareas en comparación con los sistemas operativos regulares.
4. **Fiabilidad**: Son altamente confiables y pueden continuar operando incluso después de experimentar fallos, minimizando el impacto en el sistema.
5. **Operación de falla suave**: En caso de fallo, el sistema intenta mantener su funcionamiento y conservar la mayor cantidad posible de datos y capacidad.

Real-Time Scheduling

Hay diferentes enfoques para programar en tiempo real, pero se pueden agrupar en cuatro clases principales:

1. **Enfoques estáticos basados en tablas**: Realizan un análisis estático de los horarios factibles de ejecución. Se crea un horario que determina cuándo debe comenzar la ejecución de cada tarea.(Los enfoques estáticos basados en tablas son adecuados para tareas periódicas y proporcionan una programación predecible pero inflexible.)
2. **Enfoques estáticos basados en prioridades preemptivas**: Se realiza un análisis estático para asignar prioridades a las tareas, pero no se elabora un horario específico. Se utiliza un programador preemptivo basado en prioridades.(Los enfoques estáticos basados en prioridades preemptivas asignan prioridades estáticas a las tareas según la longitud de sus períodos.)
3. **Enfoques dinámicos basados en planificación**: La viabilidad se determina dinámicamente en tiempo de ejecución. Una tarea nueva solo se acepta si es factible cumplir con sus restricciones de tiempo. Se crea un plan o horario para decidir

cuándo se despacha cada tarea.(Los enfoques dinámicos basados en planificación intentan crear un horario que cumpla con los plazos de todas las tareas)

4. **Enfoques dinámicos de mejor esfuerzo**: No se realiza un análisis de viabilidad. El sistema intenta cumplir con todos los plazos y aborta cualquier proceso iniciado que no cumpla con su plazo.(los enfoques de mejor esfuerzo intentan cumplir con los plazos sin un análisis previo.

Los diferentes enfoques de programación en tiempo real tienen ventajas y desventajas, desde la previsibilidad hasta la facilidad de implementación, y se eligen según las necesidades específicas del sistema.

Deadline Scheduling

El enfoque de programación de plazos (deadline scheduling) es crucial para las aplicaciones en tiempo real, donde completar (o iniciar) tareas en los momentos más valiosos es fundamental. Este método considera varios datos sobre cada tarea, como su tiempo de llegada, plazo de inicio, plazo de finalización, tiempo de ejecución, requisitos de recursos y prioridad.

Existen diferentes estrategias para planificar tareas según los plazos. Por ejemplo, el algoritmo de planificación con la fecha límite más temprana minimiza la fracción de tareas que incumplen sus plazos, ya sea usando plazos de inicio o de finalización. Cuando se especifican plazos de inicio, un enfoque no preemptivo puede ser apropiado, mientras que para plazos de finalización, una estrategia preemptiva es más adecuada.

Un ejemplo práctico es el control de datos de sensores A y B, donde cada sensor tiene plazos específicos para recolectar datos. Siempre se debe dar prioridad a la tarea con el plazo más cercano, garantizando así que se cumplan todos los requisitos del sistema.

En resumen, el enfoque de programación de plazos se adapta a las demandas dinámicas de las aplicaciones en tiempo real, asegurando que las tareas se completen o inicien en los momentos más valiosos, evitando tanto la anticipación como la tardanza excesiva.

Rate Monotonic Scheduling

La planificación de tareas con el método de tasa monótona (RMS) asigna prioridades a las tareas basadas en sus períodos. En este enfoque, la tarea con el período más corto tiene la prioridad más alta, y así sucesivamente. Cuando hay varias tareas listas para la ejecución, se atiende primero la que tiene el período más corto. Esta estrategia garantiza que todas las tareas se completen dentro de sus plazos de manera eficiente.

En RMS, el período de una tarea (T) es el tiempo entre la llegada de una instancia de la tarea y la llegada de la próxima. La tasa de una tarea es simplemente el inverso de su período, expresada en Hertz. La ejecución de la tarea (C) es el tiempo de procesamiento necesario para cada instancia de la misma.

Aunque la planificación con plazos más tempranos puede lograr una mayor utilización del procesador, RMS sigue siendo ampliamente utilizado debido a su estabilidad y al pequeño impacto de rendimiento en la práctica. Además, RMS facilita la estabilidad en situaciones de sobrecarga o errores transitorios, lo que lo hace preferible en muchos sistemas en tiempo real.

Priority Inversión

La inversión de prioridad es cuando una tarea importante tiene que esperar por una tarea menos importante debido a cómo está organizado el sistema. Por ejemplo, si una tarea importante necesita usar un recurso que está siendo usado por una tarea menos importante, la tarea importante tiene que esperar hasta que el recurso esté libre.

Este problema se vuelve más grave cuando la espera puede ser impredecible y durar mucho tiempo. En el caso de la misión Mars Pathfinder, esta demora causó reinicios del sistema, lo que es muy problemático en misiones espaciales.

Para evitar este problema, hay dos soluciones comunes. Una es la "herencia de prioridad", donde una tarea importante toma temporalmente la prioridad de una tarea menos importante mientras necesita usar un recurso compartido. La otra es el "techo de prioridad", donde cada recurso tiene una prioridad asociada, y cuando una tarea lo usa, su prioridad se eleva temporalmente para asegurar que no sea interrumpida por tareas menos importantes.