Sistemas Operativos II

Profesor

Carlos Mendez Rodriguez

Estudiante:

Javier Díaz Mora / 20200120139

Planificación en tiempo real

Usos robótica, militar, control de sistemas

Características

- **Determinación:** determina duración de reconocimiento de interrupciones.
- **Respuesta:** determina duración después de reconocer la interrupción.
- **Control de usuario:** usualmente no es en tiempo real y no tiene control sobre el funcionamiento de la planeación, solo guía.
- **Fiable:** si el programa llega a fallar puede ser catastrófico al ser en tiempo real.
- **Sistema contra fallas:** el sistema debe ser capaz de soportar fallos sin perder datos. determina prioridad de tareas

planificación en tiempo real

Introducción a la programación en tiempo real:

La programación en tiempo real es un área de investigación activa en informática, abarcando varias metodologías para gestionar el tiempo de ejecución de las tareas. Se analizan dos clases populares de algoritmos de programación.

Tipos de algoritmos de programación en tiempo real:

Según [RAMA94], los enfoques de programación dependen de si el sistema realiza un análisis de programabilidad, y si es así, si se hace estáticamente o dinámicamente, y si el análisis genera un plan para la ejecución de tareas. Basado en esto, se identifican cuatro clases de algoritmos:

- **Enfoques estáticos basados en tablas:** Realizan un análisis estático y producen un horario fijo para la ejecución de tareas.
- **Enfoques estáticos basados en prioridades:** Realizan un análisis estático y asignan prioridades a las tareas sin generar un horario fijo.

- **Enfoques dinámicos basados en planificación:** Determinan la factibilidad en tiempo real y generan un plan de ejecución.
- **Enfoques dinámicos de mejor esfuerzo:** No realizan análisis de factibilidad y tratan de cumplir con todas las tareas, abortando las que no cumplen con sus plazos.

Detalles de cada tipo de programación:

Programación estática basada en tablas: Adecuada para tareas periódicas, genera un horario predecible pero inflexible, necesitando rehacerse si cambian los requisitos de las tareas.

Programación estática basada en prioridades: Utiliza un mecanismo de prioridades común en sistemas multiprogramación no en tiempo real, asignando prioridades según los plazos de las tareas.

Programación dinámica basada en planificación: Revisa la factibilidad de agregar nuevas tareas en tiempo real, ajustando el horario para acomodarlas si es posible.

Programación dinámica de mejor esfuerzo: Asigna prioridades basadas en las características de las tareas sin análisis previo, usando técnicas como el horario del plazo más cercano, con la desventaja de no poder garantizar el cumplimiento de plazos hasta el final.

Objetivo de los sistemas operativos en tiempo real:

Los sistemas operativos en tiempo real actuales están diseñados para iniciar tareas en tiempo real lo más rápido posible, pero esto no es un buen indicador de rendimiento. Las aplicaciones en tiempo real se centran en completar o iniciar tareas en el momento más valioso, no demasiado pronto ni demasiado tarde, a pesar de las demandas y conflictos dinámicos de recursos, sobrecargas de procesamiento y fallos de hardware o software. Las prioridades son una herramienta básica que no capturan completamente esta necesidad.

Propuestas para una programación más efectiva:

Se han propuesto enfoques más adecuados para la programación de tareas en tiempo real, basados en información adicional sobre cada tarea. La información que puede ser utilizada incluye:

- **Tiempo de preparación:** Momento en que la tarea está lista para ejecutarse.

- **Fecha límite de inicio:** Momento en que una tarea debe comenzar.
- **Fecha límite de finalización:** Momento en que una tarea debe completarse.
- **Tiempo de procesamiento:** Tiempo necesario para ejecutar la tarea hasta su finalización.
- **Requisitos de recursos:** Conjunto de recursos necesarios para la ejecución de la tarea.
- **Prioridad:** Mide la importancia relativa de la tarea.
- **Estructura de subtareas:** La tarea puede dividirse en una subtarea obligatoria y una opcional.

Dimensiones de la función de programación en tiempo real:

Al considerar fechas límite, la programación debe decidir qué tarea programar y qué tipo de interrupción se permite. Se ha demostrado que la política de programar la tarea con la fecha límite más cercana minimiza la proporción de tareas que no cumplen sus fechas límite, tanto en configuraciones de un solo procesador como multiprocesador.

Estrategias de preempción:

No preempción: Utilizada cuando se especifican fechas límite de inicio, permitiendo que la tarea bloquee otras tareas tras completar su parte crítica.

Preempción: Más adecuada para sistemas con fechas límite de finalización, permitiendo interrumpir una tarea para ejecutar otra que tenga una fecha límite más cercana.

Ejemplo de programación con fechas límite de finalización:

En un sistema que procesa datos de dos sensores con diferentes fechas límite, se muestra que la programación basada en la fecha límite más cercana permite cumplir todos los requisitos del sistema, utilizando un enfoque de programación basado en tablas estáticas.

Tareas aperiódicas con fechas límite de inicio:

Para tareas aperiódicas, un esquema sencillo programa la tarea lista con la fecha límite más cercana, pero puede no ser eficiente si se niega el servicio a tareas urgentes. Una política refinada, llamada "fecha límite más cercana con tiempos de inactividad no forzados", mejora el rendimiento permitiendo que el procesador permanezca inactivo si la tarea elegible aún no está lista, cumpliendo así con todos los requisitos de programación.

Introducción al Rate Monotonic Scheduling (RMS):

RMS es un método para resolver conflictos de programación multitarea en tareas periódicas, asignando prioridades basadas en los períodos de las tareas. La tarea con el período más corto tiene la mayor prioridad, seguida por la segunda más corta, y así sucesivamente. Esta asignación de prioridades crea una función monótonamente creciente.

Parámetros de tareas periódicas:

- **Período (T):** Tiempo entre la llegada de una instancia de la tarea y la siguiente.
- **Tasa (en hertz):** Inverso del período.
- **Tiempo de ejecución (C):** Tiempo de procesamiento requerido para cada instancia de la tarea.

Ejemplo de RMS:

En un sistema uniprocesador, si la suma de las utilizaciones del procesador de todas las tareas no supera 1, se pueden cumplir todos los plazos.

Comparación con otros algoritmos:

RMS es menos eficiente que el scheduling por fecha límite más cercana (EDF), pero la diferencia de rendimiento es pequeña en la práctica. RMS puede lograr una utilización del procesador de hasta el 90% en la práctica.

Ventajas de RMS:

Diferencia de rendimiento pequeña: A pesar de ser conservadora, RMS puede lograr una alta utilización del procesador.

Manejo de componentes en tiempo real suaves: RMS permite la inclusión de tareas no críticas que pueden ejecutarse a niveles de prioridad más bajos.

Estabilidad: RMS facilita garantizar los plazos de tareas esenciales bajo condiciones de sobrecarga o errores transitorios, ya que las prioridades son estáticas.

Definición de Priority Inversion:

Ocurre cuando una tarea de mayor prioridad debe esperar por una tarea de menor prioridad debido a un recurso bloqueado.

Un ejemplo simple es cuando una tarea de baja prioridad bloquea un recurso que luego necesita una tarea de alta prioridad, resultando en que la tarea de alta prioridad quede bloqueada hasta que la de baja prioridad libere el recurso.

Unbounded Priority Inversion:

Es una situación más seria donde la duración de la inversión de prioridad depende no solo del tiempo necesario para manejar un recurso compartido, sino también de las acciones impredecibles de otras tareas no relacionadas.

Caso famoso: Misión Mars Pathfinder, donde el software experimentó reinicios totales del sistema debido a una inversión de prioridad no controlada.

Priority Ceiling Protocol:

Se asigna una prioridad a cada recurso, un nivel más alto que la tarea de mayor prioridad que pueda usar el recurso.

El planificador asigna dinámicamente esta prioridad a cualquier tarea que acceda al recurso.

Una vez que la tarea termina con el recurso, su prioridad vuelve a su nivel normal.