UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE SISTEMAS INFORMÁTICOS

Máster en Software de Sistemas Distribuidos y Empotrados

Sistemas de Tiempo Real Distribuidos

Glosario

Alejandro Casanova Martín

N.º de matrícula: bu0383

Madrid, 14 de abril 2024

Sistema de Tiempo Real

Consiste en la combinación de software, hardware, y dispositivos de entrada/salida para lograr el control de determinados aspectos ambientales y reaccionar dinámicamente frente a sus cambios. El software de estos sistemas está generalmente compuesto por tareas independientes y concurrentes, cuyo objetivo no solo será realizar las funciones necesarias, sino también cumplir con los plazos, periodos de ejecución y requisitos de sincronización del sistema. Tanto el dron como el software del sistema de estabilización conforman un sistema de tiempo real, debido a las características físicas y ambientales del sistema. Deberán cumplirse unos estrictos requisitos temporales para asegurar que el dron se mantiene estable independientemente de perturbaciones, y tenga un funcionamiento seguro que no ponga en riesgo la integridad de personas ni del propio sistema.

FreeRTOS

FreeRTOS es un kernel de tiempo real orientado a aplicaciones embebidas. Es simple, ligero, está escrito principalmente en C, y cuenta con los principales recursos necesarios para programar un sistema de tiempo real (tareas concurrentes, recursos compartidos, prioridades, y herramientas de sincronización). Debido a su sencillez y reducido tamaño, FreeRTOS es ideal para aplicaciones embebidas, en las que la potencia del hardware del sistema puede ser reducida, como es el caso del microcontrolador utilizado para implementar el sistema de estabilización de un dron. Las características físicas del sistema y sus requisitos de tiempo real también hicieron de FreeRTOS una opción ideal dados los recursos que ofrece.

Semáforo

Un semáforo es un recurso software de sincronización utilizado habitualmente para controlar el acceso de hilos concurrentes a un medio compartido. Un semáforo puede ser de tipo binario, si únicamente admite los valores 0 y 1, o puede ser de tipo contador, si acepta valores superiores. En caso de ser su valor 0, un semáforo bloqueará a la próxima tarea que trate de adquirirlo, mientras que en caso de ser su valor superior, la tarea decrementará en 1 el valor del semáforo y continuará con su ejecución, siempre que no deba ejecutarse otra tarea más prioritaria. Al ser liberado por una tarea, el semáforo incrementará su valor en 1, y en caso de ser un semáforo contador, podrá ser incrementado su valor un número arbitrario de veces por cualquier tarea. Para permitir el acceso de un número arbitrario de hilos a un recurso, podrá utilizarse un semáforo contador, mientras que para lograr exclusión mutua puede emplearse un semáforo binario. Sin embargo, es habitual utilizar un *mutex* en este segundo caso, debido a que cuenta con recursos adicionales de seguridad para garantizar la protección de regiones críticas.

Un uso habitual del semáforo binario es la sincronización y envío de señales entre procesos. Dado que el semáforo binario se inicializa a cero, una tarea quedará bloqueada la primera vez que intente adquirirlo, y podrá ser despertada como respuesta a un evento, o cuando otra tarea incremente el semáforo. Es típico su uso en escenarios del tipo productor-consumidor, en los que el productor se mantendrá dormido, esperando a que el consumidor le despierte con una petición.

En el sistema de estabilización del dron, se utilizó semáforos binarios para la sincronización de tareas esporádicas. Tanto la tarea de activación/desactivación del sistema, como la tarea de control de motores (en la implementación distribuida) debían bloquearse esperando a la llegada de una interrupción hardware. Utilizando un semáforo, se logró señalizar la llegada de dichas interrupciones,

liberando el semáforo desde la rutina de gestión de interrupción, y despertando a la tarea esporádica que había quedado bloqueada en dicho semáforo.

Mutex

Un *mutex* (también llamado candado, o cerrojo) es un recurso software de sincronización, utilizado habitualmente para evitar que varios hilos de ejecución accedan a una región crítica de código al mismo tiempo. De esta manera se evitan posibles condiciones de carrera, dado que sólo podrá acceder simultáneamente al código un único hilo (aquel que esté en posesión del *mutex*). A diferencia del semáforo binario, que tiene un funcionamiento similar, este se inicializa con un valor de 1, para permitir el acceso a la región crítica a la primera tarea que lo solicite. Adicionalmente, un *mutex* guarda constancia de qué tarea lo ha bloqueado, y debe ser liberado únicamente por esa misma tarea. Esta restricción permite contrarrestar ciertos errores e incorporar protocolos como el de herencia de prioridad, o techo de prioridad inmediato, para manejar el problema de inversión de prioridad. El *mutex* ha sido una herramienta esencial en el desarrollo del proyecto, a la hora de proteger los recursos compartidos utilizados por las diferentes tareas, y sincronizar su acceso para evitar condiciones de carrera o la corrupción de los datos por lecturas/escrituras simultáneas.

Planificación con Prioridades Estáticas y "Expulsión" (Fixed-Priority Pre-emptive Scheduling)

Es una política de planificación muy común en los sistemas de tiempo real, que garantiza que en cualquier momento el procesador estará ejecutando siempre la tarea de mayor prioridad, de todas las que se encuentren disponibles para ser ejecutadas.

Se dice que las prioridades son estáticas, porque estas son definidas de antemano, y no serán modificadas permanentemente por el planificador en tiempo de ejecución. Sin embargo, sí podrían verse modificadas temporalmente debido a algoritmos como el de herencia de prioridad o techo de prioridad inmediato.

Se dice que los kernels o sistemas operativos que tienen este tipo de política de planificación son "apropiativos" o tienen "expulsión", porque priorizarán siempre la ejecución de la tarea de máxima prioridad, independientemente de cuándo esta pase a estar disponible para ejecución. Por ejemplo, en caso de recibirse una interrupción hardware, el planificador abandonará inmediatamente la ejecución de una tarea de menor prioridad para atender la rutina de servicio de interrupción. Por este motivo, esta política de planificación es susceptible de provocar la inanición de las tareas de menor prioridad, si las tareas de mayor prioridad se ejecutan de forma continuada y no dejan tiempo de procesador para las demás. Para evitar esto, es necesario utilizar mecanismos de comunicación y sincronización, que permitan mantener a las tareas en estado de bloqueo o suspensión mientras no deban trabajar, y permitiendo su activación mediante el envío de una señal.

El planificador de FreeRTOS utiliza esta política de prioridades estáticas y expulsión. Además, cuando varias tareas tienen el mismo nivel de prioridad y están listas para ejecutarse, el planificador las atenderá por turnos, saltando de una a otra por cada ciclo de reloj. Sacando provecho a FreeRTOS y su política de planificación, se han podido implementar los diversos procesos encargados de garantizar la estabilidad del dron, simultáneamente realizando tanto lecturas de varios sensores, como correcciones sobre los motores.

Rate Monotonic Scheduling (RMS)

RMS es una política de asignación de prioridades utilizada en kernels y sistemas operativos de tiempo real con prioridades estáticas y típicamente con expulsión. La asignación de prioridades se realiza en función del periodo de las tareas, siendo las tareas de periodos más cortos las que reciben una mayor prioridad. Esta política de asignación de prioridades será óptima cuando los plazos a cumplir por las tareas sean idénticas a los periodos.

Dado que en el sistema de estabilización del dron los periodos y los plazos de las tareas no eran idénticos, se optó por otra política de asignación de prioridades, que se define a continuación.

Deadline Monotonic Scheduling (DMS)

DMS es una política de asignación de prioridades para planificación con prioridades estáticas y expulsión. Consiste en asignar las prioridades de acuerdo con los plazos de las tareas, de modo que las tareas con los plazos más cortos reciben la mayor prioridad. Esta política de asignación de prioridades es especialmente eficiente cuando todos los plazos son iguales o menores que los periodos de sus respectivas tareas, y mayores que los tiempos de ejecución más pesimistas (WCET).

Para el sistema de estabilización del dron se ha empleado asignación de prioridades mediante DMS, dado que se cumplían las dos condiciones recién mencionadas.

Inversión de Prioridad

La inversión de prioridad es un escenario de planificación en el que una tarea de alta prioridad se ve suspendida indirectamente por otra tarea de menor prioridad, lo que supone una violación de los principios de la planificación con prioridades. La inversión de prioridad típicamente ocurre cuando una tarea que ha accedido a un recurso compartido (bloqueando su acceso a otras tareas) es expulsada por una tarea de prioridad intermedia. A continuación, una tarea de prioridad mayor podría entrar en ejecución y quedar bloqueada en el recurso compartido que fue accedido por la tarea de prioridad inferior. En este caso, para que la tarea de máxima prioridad pudiera terminar su ejecución, tendría que terminar ejecutarse primero la tarea intermedia, y luego la tarea inferior hasta quedar liberado el recurso compartido que bloqueaba a la superior.

En muchos casos, la inversión de prioridad puede no tener repercusiones significativas y pasar desapercibida. Sin embargo, en algunos casos podría generar problemas de mayor gravedad; si la tarea de mayor prioridad sufriera inanición, podría provocar fallos en el sistema o incluso desencadenar medidas correctivas, como podría ser el caso de un temporizador *watchdog* que reinicie el sistema por completo (como fue el caso de la nave *Mars Pathfinder* en 1997).

Algunas medidas para evitar por completo la inversión de prioridad son deshabilitar las interrupciones, para evitar la expulsión de tareas que se encuentren en regiones críticas, o evitar los bloqueos en recursos compartidos. Los protocolos de herencia de prioridad y techo inmediato de prioridad permiten mitigar significativamente los efectos de la inversión de prioridad, aunque no eliminarla por completo.

Herencia de Prioridad

El protocolo de herencia de prioridad consiste en que si una tarea de prioridad alta se bloquea al intentar obtener un *mutex* que está en ese momento en posesión de una tarea de prioridad menor,

entonces la prioridad de la tarea en posesión del *mutex* se verá temporalmente incrementada al nivel de la tarea bloqueada. Al liberar el *mutex*, dicha tarea recuperará su prioridad original. De esta manera, se consigue que la tarea de menor prioridad libere el mutex cuanto antes, y que por lo tanto la tarea de mayor prioridad pase el menor tiempo posible bloqueada, minimizando los efectos negativos del fenómeno de inversión de prioridad.

En FreeRTOS, los *mutex* implementan el protocolo de herencia de prioridad. Para el sistema de estabilización del dron, esto nos permitió prevenir y mitigar potenciales problemas de rendimiento e inanición.