**POSIBLES PREGUNTAS EXAMEN FINAL**

1. Explicar paso por paso la herencia de prioridades con mutex.

La herencia de prioridades con mutex es un mecanismo utilizado en sistemas operativos para evitar la inversión de prioridades.

Supongamos que tenemos dos hilos, donde la prioridad es ‘A’ > ‘B’, y donde tenemos un recurso compartido. El hilo ‘A’, de mayor prioridad, necesita acceder al recurso protegido e intenta adquirir el mutex asociado a ese recurso. Si el mutex está libre, el hilo ‘A’ adquiere el mutex y puede acceder al recurso de forma segura.

Pero si el hilo ‘B’ está utilizando el mismo recurso, el sistema operativo temporalmente aumenta la prioridad del hilo ‘A’ a la del hilo ‘B’ para evitar la espera indefinida, y que el hilo ‘A’ pueda adquirir el mutex, acceder al recurso y liberarlo rápidamente. Después, el hilo ‘B’ puede adquirir el mutex y acceder al recurso compartido.

Si hay más hilos esperando por el mutex, el sistema operativo utiliza su política de planificación para determinar el próximo hilo que obtendrá acceso al recurso.

1. Preguntas cortas del parcial.
   1. Que es RTOS y significado de las siglas.
      1. Siglas: Sistemas Operativos en Tiempo Real.
   2. Que es una RTA y características.
      1. Siglas: Aplicaciones en Tiempo Real.
      2. Características:
         1. No son aplicaciones que se ejecuten rápidamente.
         2. Son aplicaciones determinadas en el tiempo, el tiempo de respuesta es prácticamente constante.
         3. Puede haber pequeñas desviaciones en el tiempo de respuesta. (Hablamos de milisegundos o de segundos y los llamamos “Soft RTAs”).
         4. Las “Hard RTAs” deben cumplir todas sus funciones en un tiempo muy estricto, de no cumplirlo, puede ocurrir un fallo absoluto. Existen también los “Real RTAs” que son aún más restrictivos.
   3. Concepto de que es tiempo real.
      1. Sistema donde la exactitud del cómputo realizado no solo depende de la exactitud lógica del cómputo sino también del instante de tiempo producido.
   4. Que es el Tick Interrupt.
      1. Tick Interrupt es un mecanismo en los sistemas operativos en tiempo real que permite generar interrupciones regulares en intervalos de tiempo fijos. El uso de estas interrupciones permite al RTOS realizar una variedad de tareas importantes, como la planificación de tareas, la medición del tiempo y la actualización de temporizadores.
      2. Se mide en unidades de tiempo de ticks del sistema.
   5. Que es un context switch y cuando se produce.
      1. Es una operación realizada en los sistemas operativos multitarea que permite a múltiples procesos o tareas ejecutarse en paralelo. Se produce cuando el sistema operativo interrumpe la ejecución de un proceso o tarea para permitir la ejecución de otro proceso o tarea.
      2. Durante el cambio de contexto, la información como el valor de los registros de la CPU, el puntero de la pila y otros datos relevantes, se guardan el estado actual del proceso que se está ejecutando, se carga el estado del siguiente proceso y se cambia el contexto de ejecución.
   6. Diferencia entre RTOS cooperativo y preemptive.
      1. Un RTOS preemptive, el sistema operativo interrumpe las tareas en ejecución para dar prioridad a las tareas de mayor prioridad, mientras que en un RTOS cooperative, las tareas solo ceden el control del procesador voluntariamente después de completar su ejecución. La elección de uno u otro depende de la aplicación específica y de las necesidades de tiempo real y prioridad de esta.
   7. Que es un tcb de una tarea.
      1. Task Control Block almacena información de cada tarea en la RAM. Kernel lo usa para tener decisiones sobre que tarea se ejecuta.
   8. Que es la inversión de prioridades.
      1. La inversión de prioridad se produce cuando una tarea de menor prioridad (LP) accede previamente a un recurso compartido (con una tarea de alta prioridad HP). En este caso, la tarea de alta prioridad se mantiene bloqueada a la espera de que la tarea de menor prioridad libere el recurso compartido. El efecto producido es el siguiente: La tarea de baja prioridad se comporta como una tarea de alta prioridad. Y viceversa, la tarea de alta prioridad se comporta como una de baja prioridad.
   9. Que es lo de la variable os\_error y porque se utiliza
      1. Puntero a una variable que almacena cualquier error durante creación de tareas. Al lanzar un mensaje de salida al usuario para evitar la ejecución de la tarea se hay error.
   10. Que es el time cuanta y cuando se utiliza
       1. Es el tiempo de ejecución permitido a una tarea donde por turnos comparten y ejecutan la CPU.
   11. Cuando es una aplicación determinista
       1. Sistema predecible para realizar tareas en un tiempo determinado siendo consistente y confiable y permite continuar realizando tareas mientras espera.
2. Explicar los componentes y elementos de la función createThread().

Este código en C está creando una tarea en un sistema operativo en tiempo real basado en el kernel de Micrium OS (uC/OS-III). El objetivo de este código es crear una tarea llamada "Tarea de inicialización" que se ejecutará en el sistema operativo.

* `(OS\_TCB \*)&init\_task\_TCB`: Es un puntero al bloque de control de tarea (Task Control Block, TCB) de la tarea de inicialización.
* `(CPU\_CHAR \*)"Tarea de inicialización"`: Es el nombre de la tarea.
* (OS\_TASK\_PTR)InitTask`: Es un puntero a la función que será ejecutada por la tarea de inicialización. En este caso, la función se llama `InitTask`.
* `(void \*)0`: Es un puntero a un argumento que se pasará a la tarea de inicialización. En este caso, se pasa un puntero nulo (0).
* `(OS\_PRIO)3`: Es la prioridad asignada a la tarea de inicialización. En este caso, se le asigna una prioridad de 3.
* (CPU\_STK \*)&init\_task\_STK[0]`: Es un puntero al inicio de la pila (stack) de la tarea de inicialización.
* `(CPU\_STK\_SIZE)0u`: Es el tamaño mínimo requerido para la pila de la tarea de inicialización. En este caso, no se especifica un tamaño mínimo.
* `(CPU\_STK\_SIZE)1024u`: Es el tamaño de la pila asignado a la tarea de inicialización. En este caso, se le asigna un tamaño de 1024 bytes.
* `(OS\_MSG\_QTY)0u`: Es la cantidad máxima de mensajes en la cola de mensajes asociada a la tarea de inicialización. En este caso, no se utiliza una cola de mensajes.
* `(OS\_TICK)0`: Es el tiempo de espera antes de que la tarea de inicialización se ejecute por primera vez. En este caso, se establece en 0, lo que significa que la tarea se ejecutará inmediatamente después de su creación.
* `(void \*)0`: Es un puntero a una variable que se pasará a la tarea de inicialización cuando se ejecute. En este caso, se pasa un puntero nulo (0).
* `(OS\_OPT)(OS\_OPT\_TASK\_STK\_CHK|OS\_OPT\_TASK\_STK\_CLR)`: Son opciones adicionales para la tarea de inicialización. En este caso, se habilita la verificación de la pila de la tarea (`OS\_OPT\_TASK\_STK\_CHK`) y se limpia la pila antes de ejecutar la tarea (`OS\_OPT\_TASK\_STK\_CLR`).
* `(OS\_ERR \*)&os\_err`: Es un puntero a una variable de error donde se almacenará el resultado de la llamada a la función `OSTaskCreate()`.