Sitemas Operativos 2020-1
Tarea 1

Edgar Quiroz

10 de marzo de 2020

Todas las respuestas deben estar justificadas.

1. Al terminar un proceso de E/S, el dispositivo pertinente debe interrumpir al CPU para indicar que los datos están listos, y el kernel debe recuperar estos datos. Esto es muy lento. Mencione la técnica utilizada para optimizar este proceso.

   Se suelen dividir a los tipos de interrupciones en dos. Los que son sensibles a tiempo y los que pueden esperar.

   Los primeros (como lectura de disco duro) son atentidos por un manejador de interrupcione de primer nivel. Este usa tablas de rutinas que se ejecutan directamente. Esto las hace más rápidas, aunque más complicadas y poco escalable. Sólo las rutines que realmente necesitan la velocidad son atendidas por este.

   Los demás son atendidos que un manejador de segundo nivel. Este es más parecido a un proceso normal. Tiene un hilo de ejecución por cada programa de interrupción pertinente. Esto lo hace mas robusto y escalable, aunque lento.

2. Sean t_1 y t_2 procesos. t_1 ejecuta `thread_yield`. El calenderizador escoge a t_2 para ejecutarse. ¿Qué proceso ejecutará la línea justo despuésde la llamada a `thread_yield`?

   t_2 ejecutará la línea, puesto que las rutinas de bloqueo son idénticas para todos los procesos, por lo que no son afectados por el cambio de contexto.

   Al terminar la rutina, t_2 regresaría a donde se bloqueó, puesto que esta información se guarda en su contexto, que es restaurado al cambiar el proceso en ejeución.

3. Cuando se ejecuta una rutina de interrupción, ¿siempre hay cambio de contexto?

   Cuando se detecta una señal en el canal de interrupciones, el procesador guarda el estado actual y salta al manejador de interrupciones en la dirección de memoria adecuada.

   El manejador determin la causa, ejecuta una acción pertienete, restaura el estado anterior del procesador y vuelve al punto donde ocurrió la interrupción.

   Entonces siempre es necesario tener un cambio de contexto, pues es la única manera de entrar y salir de la subrutina de la interrupción.

4. ¿Quién duerme a los procesos?

   En caso de esperas para interrupciones, el proceso sabe que tiene que dormir, así que se marca como no ejecutable, y pasa el control a otro proceso.

   Posteriormente, algún otro proceso revisa que otro procesos están listos para despertar y los despierta. En cuanto haya procesador libre, volverán a ser ejecutados.

5. Enumere los casos de transición de un proceso del estado `running` al estado `ready`.

   Puede ser cuando se empieza a ejecutar un proceso hijo.

   O cuando se transfiere de forma no voluntiara el control del procesador.

   O cuando se tiene que atender una interrupción (como E/S).

6. Lo más sencillo para lidiar con una interrupción sería pasar el control a un proceso especial para que determine la rutina a llamar. Utilice el hecho de que los tipos de interrupción están predefinidos para dar una optimización a este método.

   Se utiliza una tabla de códigos de interrupción, pues son finitos, y de apuntadores hacia el código que se necesita ejecutar.

   Esto elimina el proceso intermedio para determinar el tipo de interrupción, haciendo todo más eficiente.

7. ¿Cuál es la máxima cantidad de llamadas a `wait` que se pueden completar sobre un semáforo de valor inicial n?

   Cada llamada de `wait` a lo menos disminuye en 1 el valor del semáforo. Después de este punto, el lugar de disminuir el semáforo, se bloquean los procesos.

   Entonces, a lo más se puede llamar n veces un semáforo con valor inicial de n si únicamente hay llamadas `wait`.

   En el caso de que haya desbloqueos `signal`, entonces habría $n + numSignalCalls$ número de posibles llamadas a `wait`.

8. Hay tres maneras para pasar parámetros a una llamada al sistema. Menciónelos. ¿Cuál es el más eficiente respecto a espacio y a tiempo?

   - Guardarlos en registros.
   - Almacenarlos en un bloque de la memoria y pasar la dirección
   - Insertarlos en el stack

   En espacio, son mejores los registros. En tiempo, es mejor en bloque o en el stack, pues hay menos limitaciones sobre el acceso a ellos y a la cantidad límite que se puede guardar.

9. ¿En cuantas colas puede estar formado un proceso?

   Un proceso entra a una cola cuando no puede continuar su ejecución por un bloqueo de recursos.

   Para entrar a otra cola, debe tener otra solicitud bloqueada de recursos. Y para que esto pase debe primero tener tiempo otra vez en el procesador.Y esto sólo pasa después de que salga de la primera cola.

   Entonces, a lo más puede estar en una cola a la vez.

10. Compara la comunicación por memoria compartida y por paso de mensajes.

    En el paso de mensajes, no se tienen problemas de bloqueo por acceso simultaneo. Es más sencillo conceptualmente y fácil de implementar. El paso de mensajes requiere llamadas a sistema. El acceso a recursos es más lento y sólo es factible con pocos datos.

    En memoria compartida, el acceso es rápdipo y se pueden mantener grandes cantidades de datos. La consulta a memoria no requiere uso del kernel. Pero pueden surgir problemas al varios procesos modificar la memoria simultaneamente. Así que una implementación adecuada puede ser muy complicada.

11. Da dos ejemplos de instrucciones protegidas y las consecuencias en el caso que no lo fueran.

    `set_intr` permite desactivar interrupciones. En caso de no ser protegida, procesos de usuario podrían apropiarse del procesador sin que hubiera un mecanismo para removerlos.

    `switch_thread` permite cambiar el hilo actualmente en el procesador. De no ser protegida, todos los procesos podrían mandar a llamara cualquier otro código, sin restricciones ni consideraciones de seguridad.

12. Explica como simular la instrucción `goto` en un modelo de pila.

    Se puede hacer con llamadas de cola.

    Esto es que en cada llamada a `goto`, se guarda todas las instrucciones faltantes de la rutina actual, y se empieza a ejecutar el código de la subrutina. Esto hace que todas las llamadas a subrutinas sean la última acción en cada rutina.

    No es necesario recordar el origen ni los registros.