RELACIONES SISTEMAS OPERATIVOS

Daniel Monjas Miguélez

2 DGIIM Universidad de Granada

17 de noviembre de 2019

Índice

1. Relacion 1

1.1. Cuestiones generales relacionadas con un SO:

¿Qué es el núcleo (kernel) de un SO?

El núcleo o kernel es la parte central de un sistema operativo y es el encargado de realizar toda la comunicación segura entre el software y el hardware del ordenador. Este es la parte más importante del sistema operativo Unix y sus derivados.

¿Qué es un modelo de memoria para un programa?. Explique los diferentes modelos de memoria para la arquitectura IA-32.

Es el mecanismo que utiliza un programa para acceder de forma indirecta a la memoria física del computador. Los programas referencian a memoria con direcciones lógicas, estas se transforman en direcciones lineales al espacio lineal de memoria y finalmente el procesador transforma la dirección lineal en direcciones físicas de memoria.

En IA-32 tenemos los siguiente modelos de memoria:

- **Flat memory model.** Es un espacio linea con direcciones consecutivas en el ragno [0 , 2^{32-1}](linear address space). Permite direccionar con granularidad de un byte. Se direcciona de la forma <selector, offset>, donde en el flat memory model el selector es siempre 0. (Realmente el selector solo es el identificador de una entrada en la tabla de direcciones la cual, por cada entrada tiene asociada una dirección base y un limite de forma que la dirección base + el offset no se puede salir de dicho limite o daría un error, esto es aplicable para los selector de los dos modelos que aparecen más abajo).

- **Segmented memory model.** Los programas ven el espacio de memoria como un grupo de espacios independientes llamados segmentos. Para acceder a un byte dentro de un segmento el programa utiliza una dirección bidimiensional (dirección lógica), con la forma <selector de segmento, offset>, donde el selector de segmento contiene la dirección lineal del inicio del segmento. IA-32 permite identificar 16383 segmentos de diferentes tipos y tamaños con un tamaño máximo por segmento de 2^{32} bytes.

- **Real-address mode memory model.** Proporcionado por compatibiliadad hacia atrás con el procesador 8086. Implementación de segmentación con limitaciones en el tamaño de los segmentos a 64 KB y en el espacio de memoria final accesible, 2^{20} bytes. Las limitaciones del tamaño de los segmentos se debe a que la direccion logica tiene una forma <selector, offset>donde el selector tiene unicamente 16 bits.

¿Cómo funciona el mecanismo de tratamiento de interrupciones mediante interrupciones vectorizadas? Explique que parte es realizada apor el hardware y que parte por el software.

En las interrupciones vectorizadas, el periférico que ha interrumpido envía un código o número de vecto a la CPU a partir del cual se puede calcular la dirección de comienzo de la rutina de tratamiento de interrupciones de ese periférico particular.

Cuando el periférico recibe una señal de confirmación o reconocimiento de interrupción INTA(interruption act) envía el numerod e vector a través del bus de datos.

A partir del numerod e vector se calcula una dirección de memoria donde está almacenada el comienzo de la RTI.

1. El periférico activa la señal de interrupción (INTR=0)
2. La CPU activa una señal de confirmacion (INTA=1) que se conecta a los dispositivos de forma encadenada (daisy-chain).
3. Un periférico que no ha interrumpido, cuando recibe la señal INTA, la propaga al siguiente.
4. Cuando el periférico que interrumpió recibe la señal INTA vuelca su número de vector sobre el bus de datos y desactiva la señal de petición de interrupción. Este periférico no propaga INTA.
5. La CPU salva el contexto en pila (CPU y registro de estado) y salta a la RTI.
6. Se guardan los registros accesibles por programa, se ejcuta la operación de E/S y se retoma la interrupción al programa principal restaurando previamente todo el contexto.

El hardware realiza las siguientes tareas:

- El controlador de dispositivo u otro sistema hardware genera una interrupción.
- El procesador termina la ejecución de la instrucción actual.
- El procesador indica el reconocimiento de la interrupción.
- El procesador apila PSW y el PC en la pila de control
- El procesador carga un nuevo valor en el PC basado en la interrupción.

Por su parte el software realiza:

- Salva el resto de la información de estado del proceso.
- Procesa la interrupción
- Restaura la información de estado del proceso.
- Restaura antiguas PSW y PC.

Describa detalladamente los pasos que lleva a cabo el SO cuando un programa solicita una llamada al sistema

Si durante la ejecución de un proceso en modo usuario se encuentra una llamada al sistema, entonces se busca en la biblioteca el código asociado a la llamda del sistema, después el procesador cambia a modo kernel e invoca al manejador de llamadas del sistema, el cual salvará a la pila de sistema los valores de los registros del procesador para luego reanudar la ejecución del programa. A continuación el manejador de llamadas al sistema busca la llamada utilizando el código pasado por el procesador y una vez la encuentra carga en el PC el primer valor de la rutina de servicio de llamada. Esta se ejecuta hasta que finaliza o se bloquea. Se puede dar el caso de que la llamada sea bloqueante, lo que conllevará que una vez se ejecute la llamada al sistema el proceso se bloqueará a la espera de la ocurrencia de un evento. Si la llamada no es bloqueante, una vez terminada la ejecución de la llamada al sistema se restaurán de la pila del sistema los valores llamados y entonces se continúa con la ejecución del programa por donde se había dejado.

1.2. Explique tres responsabilidades asignadas al gestor de memoria de un SO y tres asignadas al gestor de procesos

Gestor de memoria:

- Se encarga de la protección de la región de memoria principal ocupada por el kernel y protección de las regiones de memoria principal ocupadas por los programas.
- Se encarga de la compartición de parte de las regiones ocupadas por los programas para permitir comunicación entre estos.
- Se encarga de la gestión automática de la asignación/liberación de la memoria disponible para un programa en cualquier nivel de la jerarquía de memoria y de forma transparente al programador.

Gestor de procesos:

- Se encarga de la creación del PCB asociado aun programa que va a ejecutarse. Además, se encarga también de la eliminación del miso una vez finalizada la ejecución.
- Se encarga del bloqueo(sleep) y desbloqueo(wakeup) de los procesos dependiendo de los eventos por los que debe esperar un programa para poder continuar su ejecución.
- Se encarga de proporcionar mecanismos que posibiliten la comunicación y la sincronización entre procesos.

1.3. Contraste las ventajas e inconvenienes de una arquitectura de SO monolítica frente a una arquitectura microkernel

Las arquitecturas de SO monolítica tienen como gran inconveniente que en ellas no se tenía en cuenta los problemas causados por la dependiencia mutua e interracción. En estos sistemas operativos practicamente cualquier porceso podía llamar a cualquier otor. Esta falta de estructura se hizo insostenible a medida que los SO crecieron. Además entre las desventajas de estos sistemas se encuentra que disponían de un gran núcleo monolítico, el cual incluía gran parte de las funcinalidades consideradas propias del SO, lo que conlleva que si los cambios se hacen sobre cualquier porción del sistema operativo, todos los módulos y rutinas deben volverse a enlazar y reinstalar, y el sistema se debe reiniciar para que tengan efecto los cambios, y por consiguiente cualquier modificación es dificil. La ventaja es que al estar gran parte de las funcionalidades como parte del núcleo se puede acceder a ellas de forma rápida pues residen constantemente en memoria.

Por su parte las ventajas de un micronúcleo son:

- Una interfaz uniforme que permite que no haya que diferenciar los procesos entre nivel núcleo y nivel usuario, como se hace en la arquitectura monolítica, pues todos se proporcionan a través de paso mensajes.
- Facilita la extensibilidad, permitiendo agregar nuevos servicios, así como la realización de múltiples servicios en la misma área funcional. En la arquitectura micronúcleo al añadir o modificar servicios solo se actualiza el servidor que este relacionado, a diferencia del sistema operativo monolitica donde una modificación en una porción del SO implica reenlazar y reinstalar todos los módulos y rutinas, lo que conlleva una gran dificultad para cualquier modificación minimo.
- La portabilidad del SO micronúcleo se basa en que gran parte o todo el código específico del procesador está en el micronúcleo. Por tanto, los cambios necesarios para transferir el sistea a un nuevo procesador son menores y tienden a estar unidos en grupos lógicos. Como ya hemos visto anteriormente cualquier modificación en una porción cualquier del SO monlítico es tremendamente compleja y afecta a gran parte o todo el sistema.
- Fiabilidad. El pequeños tamaño del micronúcleo permite que se puede verificar de forma rigurosa y por tanto sea más fiable. En contraste con el núcleo monolítico que es de gran tamaño y es más dificil asegurar su fiabilidad. Es más en un núcleo monolítico como todos los componentes fucnionales del núcelo tienen acceso a todas sus estructuras de datos internas y a sus rutinas, un error en una rutina se podría propagar a todo el sistema.
- El micronúcleo permite el soporte de sistemas operativos distribuidos. Cuando se envía un mensaje desde un cliente a un servidor, el mensa-

je debe incluir un identificador de servicio. Si se configura un sistema distribuido donde los procesos y servicios tengan identificadores únicos, habrá una sola imagen del sistema a nivel micronúcleo.

- Una arquitectura micronúcleo funciona bien en el contexto de un sistema operativo orientado a objetos.

Desventajas del micronúcleo:

La principal desventaja de la arquitectura micronúcleo es que el paso constante de mensajes entre procesos a través del núcleo da lugar a un pobre rendimiento en comparación con la arquitectura monolítica que al esta muchas funcionalidades en el núcleo tiene un alto rendimiento. Si bien esto es una desventaja una posible solución sería reducir incluso más el tamaño del micronúcleo.

Tambiéne se consideran desventajas la complejidad en la sincronización de todos los módulos que componen el micronúcleo y su acceso a memoria, la complejidad del código y las limitaciones de diversas funciones.

2. Relacion 2

2.1. Cuestiones generales sobre procesos y asignación de CPU:

¿Cuáles son los motivos que pueden llevar a la creación de un proceso

- **Nuevo proceso por lotes.** El SO dispone de un flujo de control de lotes de trabajos. Cuando el SO está listo para procesar un nuevo trabajo, leerá la siguiente secuencia de mandatos de control de trabajos.
- **Sesión interactiva.** Un usuario desde un terminal entra en el sistema.
- **Creado por el SO para proporcionar un servicio.** El SO puede crear un proceso para realizar una función en representación de un programa de usuario, sin que el usuario tenga que esperar.
- **Creado por un proceso existente.** Por motivos de modularidad o para explotar el paralelismo, un programa de usuario puede ordenar la creación de un numero de procesos.

Es necesario qeu lo último que haga todo proceso antes de finalizar sea una llamda al sistema para finalizar de forma explícita, por ejemplo exit()?

No es estrictamente necesario, pues se podría dar el caso de que un programa produzca una excepción o error irrecuperable que lleve al SO a ejecutar una rutina que termine el proceso de forma no explícita (es decir, sin que se indique en el propio proceso).

Cuando un proceso pasa a "BLOQUEADO",¿Quién se encarga de cambiar el valor de su estado en el PCB?

El dispatcher() es el encargado de cambiar el estado de los procesos. Luego si un proceso va a pasar a estado bloqueado se llamará al dispatcher() para que cambie el estado del programa de ejecutando a bloqueado y lo almacenará en el PCB del proceso.

Qué debería hacer cualquier planificador a corto plazo cuando es invocado pero no hay ningún proceso en la cola de ejecutables

Debería llamar al planificador de medio plazo para que si hay algún elemento en la cola de listos/supendido se transfiera dicho proceso a memoria principal. En caso de que esta cola también esté vacía llamaría al planificador a largo plazo para que introduzca nuevos procesos en la cola de ejecutables.

Qué algoritmos de planificación quedan descartados para ser utilizados en sistemas de tiempo compartido

Todos los tipos de planificación que no sean expulsivos.

2.2. Cuestiones sobre el modelo de procesos extendido

¿Qué pasos debe llevar a cabo un SO para poder pasar un proceso de reciente creación de estado 'NUEVO' a estado 'LISTO'?

Se actualiza el estado del proceso y se cambia el cambio del PCB de nuevo a listo, entonces si no está cargado el proceso en memoria principal se llamará al planificador de medio plazo para que cargue en memoria principal el programa, lo que conllevará que se muevan a memoria principal el PCB, el programa, la pila de usuario y de núcleo, etc.

¿Qué pasos debe llevar a cabo un SO para poder pasar un proceso ejecutándose en CPU a estado finalizado?

Una vez se haya terminado la ejecución del proceso en la CPU se llama al contex_switch() para salvaguardar el contexto del programa que va a dejar la CPU y cargar en CPU el nuevo programa que se haya planificado. Entonces se pueden mantener los datos del programa saliente para tomar estadísticas e información de auditoría y finalmente se eliminará el proceso por completo.

Hemos explicado en clase que la función context_switch() realiza siempre dos funcionalidades y que además es necesario que el kernel la llame simepre cuando el proceso en ejecución pasa a estdo "FINALIZADO.º "BLOQUEADO". ¿Qué funcionalidades debe realizar y en qué funciones del SO se llama a esta función?

Se llama al dispatcher() para actualizar el estado del proceso cuyo estado va a cambiar y entonces se llama al planif_CPU(), el cual selecciona el siguiente proceso que se va a ejecutar, entonces se llama al dispatcher() el cual actualiza el estado del proceso que se va a meter en la CPU a ejecutar. Finalmente se carga el PC del proceso y comienza la ejecución.

Se llama a context_switch() siempre que se realiza una llamada a sistema bloqueante, interrupciones de E/S, una preemption, algún tipo de error irrecuperable (excepciones) ....

Indique el motivo de la aparición de los estados "SUSPENDIDO-BLOQUEADOz "SUSPENDIDO-LLISTO.en el modelo de procesos extendido.

El proceso SUSPENDIDO/BLOQUEADO surge debido a que como la diferencia entre la velocidad de la CPU y la de los dispositivos/operaciones E/S es muy dispar se podría llegar a dar el caso de que todos los procesos en memoria principal estuvieran bloqueados a la espera de un evento. Esto implica que el procesador estaría ocioso. Suponiendo que toda la memoria estuviese completamente llena y todos los procesos en memoria estuvieran bloqueados se perdería mucho tiempo de procesamiento. Entonces surge el nuevo estado por el cual se puede sacar un proceso o más de las lista de bloqueados y llevarlos a una lista de SUSPENDIDOS/BLOQUEADO que se encuentra en disco dejando memoria libre para así traer nuevos procesos a memoria principal. Como consecuencia surge la cola SUSPENDIDO/LISTO ya que podría suceder el evento por el cual espera el proceso mientras este está en disco, entonces pasaría de la cola de SUSPENDIDO/BLOQUEADO a la de LISTO/BLOQUEADO.

2.3. ¿Tiene sentido mantener odenada por prioridades la cola de procesos bloqueados?,¿en qué casos sería útil hacerlo?

No tiene sentido, pues realmente en la cola de bloqueados lo que importa es el evento por el que espera un proceso. Si tendría sentido, que en la cola de peticiones de E/S este tuviese prioridades, de forma que si un proceso que requiere un operación de E/S urgente su petición se trate preferentemente en la cola por el planificador de E/S, sin embargo no importa la posición que ocupe el proceso en la cola de bloqueados, sino la posición que ocupe la petición en la cola de peticiones de E/S.

2.4. Explique las diferentes formas que tiene el kernel de ejecutarse en relación al contexto de un proceso y al modo de ejecución del procesador

En un gran número de sistemas la única forma que tiene el kernel de ejecutarse es a través del modo núcleo del procesador, el cual permite que se accedan a zonas de memoria protegida e instrucciones privilegiadas (como son las del kernel), en cualuqier otro caso no se permite la ejecución del kernel. Este se ejecuta en las llamadas al sistema e interrupciones de E/S.

Por otro lados los programas de usuario se ejecutan en un modo del procesador cuyo privilegio sea menor que el modo núcleo, realizando un cambio a modo núcleo cuando alguno de estos programas realiza una llamada al sistema o a una interrupción de E/S.

Los distintos modos en los que se ejecuta un programa se almacenan en un bit o más de la PSW.

2.5. Responda a las siguientes cuestiones relacionadas con el concepto de hebra:

¿Qué elementos de información es imprescindible que contenga una estructura de datos que permita gestionar hebras en un kernel de SO? Describa las estructuras task_t y thread_t

Para gestionar las hebras al igual que se hacía con los procesos se requiere de un TCB(*thread control block*), uno por cada hilo, y cada uno de estos debe dispones de almenos:

- Estado de ejecución del hilo
- Contexto de hilo que se almacena cuando no está en ejecución.
- Una pila de ejcución
- Una pila de núcleo
- Espacio de almacenamiento para variables locales
- Acceso a la memoria y recursos de su procesos, compartido con todos los hilos del mismo

En un implementación con una biblioteca de usuario en la cual cada hebra de usuario tiene una correspondencia N:1 con una hebra kernel, ¿Qué ocurre si se realiza una llamada al sistema bloqueante, por ejemplo read()?

Al ser la correspondencia N:1 la planificación del procesador se realiza a nivel de proceso y dentro de cada proceso se disponde de un biblioteca de hilos a nivel de usuario la cual gestiona los hilos, crea y destruye dentro del mismo proceso.

Cuando se realiza una llamada a sistema por parte de una hebra el procesador entiende como que todo el proceso se bloquea y no solo la hebra, por tanto llamará a context_switch() y el proceso pasará a estar bloqueado y el procesador lo ocupará un nuevo proceso. Una vez se realice el evento de E/S y se vuelva a ejecutar el proceso la misma hebra que ha llamado al proceso seguirá ejecutando, lo que implica que las hebras dentro del proceso mantienen su estado. Además, como consecuencia se deduce que como por ser la planificación a nivel proceso si se bloquea el proceso se bloquearán todas las hebras, y por consiguiente no se bloquea la hebra en ejecución y se ejecuta otra del proceso como cabría esperar.

¿Qué ocurriría con la llamada al sistema read() con respecto a la tarea de la pregunta anterior si la correspondencia entre hebras usuario y hebras kernel fuese 1:1?

A diferencia del caso anterior, en este caso la planificación se realiza a nivel de hebra, luego el planificador si es consciente de todas las hebras de las que dispone un proceso, luego cuando una hebra en ejecución se bloquea debido a la llamada read, se cambia su estado y se almacena el contexto y se pondrá a ejecutar una nueva hebra que bien puede ser del mismo proceso o de otro, dependerá del orden de la cola de listos y del tipo de planificación.

2.6. ¿Puede el procesador manejar una interrupción mientras está ejecutando un proceso sin hacer context_switch() si la política de planificación que utilizamos es no apropiativa?¿Y si es apropiativa?

En ambos casos la afirmación es false, siempre que se produce una interrupción se requiere de un cambio de contexto, pues en el procesador se pasará a ejecutar un RSI. Es decir, es indiependiente del tipo de planificación pues siempre que se produzca una interrupción habrá un context_switch().

2.7. Suponga que es el responsable de diseñar e implementar un SO que va a utilizar una política de planificación apropiativa (preemtive). Suponiendo que el sistema ya funciona perfectamente con multiprogramación pura y que tenemos implementada la función Planif_CPU(), ¿qué otras partes del SO habría que modificar para implementar tal sistema? Escriba el código que habría que incorporar a dichas partes para implementar apropiación (preemption).

Para empezar había que modificar los planificadores a largo y medio plazo para que cada vez que se añada un proceso a la cola de listos este llame al planif_CPU(), para así comprobar si se tiene que realizar una preemption.