EXAMEN PARCIAL 2017-2018 SISTEMAS OPERATIVOS TEMAS 1 Y 2:

Autor: 2º DGIIM 19/20

1. [1.25] Con respecto al apoyo hardware al SO:

a. Explica los modos de direccionamiento de memoria de la arquitectura IA32, modelos de memoria flat, segmented y real-address mode

- Un modelo de memoria describe las interacciones de los hilos a través de la memoria y su uso compartido de los datos.
- Para la ia 32 son el flat memory model, segmented memory model y real-adress mode memory model:
    - **Flat memory model.** Es un espacio lineal con direcciones consecutivas en el rango [0,2^32-1] (linear address space). Permite direccionar con granularidad de un byte.
    - **Segmented memory model.** Los programas ven el espacio de memoria como un grupo de espacios independientes llamados segmentos.
    - **Real-address mode memory model.**Proporcionado por compatibilidad hacia atrás con el procesador 8086. Implementación de segmentación con limitaciones en el tamaño de los segmentos, 64KB, y en el espacio de memoria final accesible, 2^{20} bytes.

b. Describe los pasos, hardware y software, que se llevan a cabo para la resolución de una llamada al sistema.

PARTE HARDWARE

- 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 llamada 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 registros del procesador para luego reanudar la ejecución del programa.
- A continuación el manejador de llamadas al sistema busca la llamada utilizando el código pasando por el procesador y una vez la encuentra carga en el PC el primer valor de la rutina de servicio de llamada. Ésta se ejecuta hasta que finaliza o se bloquea

PARTE SOFTWARE

- Se puede dar el caso de que la llamada sea bloqueante, lo que conllevará que una vez se ejecute la llamda 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 llmada al sistema se restaurarán de la pila del sisetam los valores llamados y entonces se continúa con la ejecución del programa por donde se había dejado.

2. [1.25] Con respecto a virtualización:

a. ¿Cuál es la diferencia entre los dos enfoques de virtualización explicados en clase hipervisor tipo1 e hipervisor tipo2?

- **Tipo 1, native o bare-metal.** Se ejecutan directamente en el host HW para proporcionar VM a los SO invitados. Ej. Xen y Vmware ESX/ESXi.
- **Tipo 2, hosted**: Se ejecutan sobre un SO convencional como el resto de programas y el SO invitado se ejecuta sobre la abstracción proporcionada por el hipervisor. Ej. VMware Workstation y VirtualBox .

b. Describe en qué consiste la técnica de virtualización denominada virtualización asistida por hardware. Básese para la explicación en la arquitectura x86 y los rings del procesador

- La **Virtualización Asistida por Hardware**, son extensiones introducidas en la arquitectura del procesador x86 para facilitar las tareas de **virtualización** al software corriendo sobre el sistema.
- Si cuatro son los niveles de privilegio o *anillos* de ejecución en esta arquitectura, desde el 0 o de mayor privilegio, que se destina a las operaciones del kernel de SO, al 3, con privilegios menores que es el utilizado por los procesos de usuario, en esta nueva arquitectura se introduce un anillo interior o ring -1 que será el que un hypervisor o Virtual Machine Monitor usará para aislar todas las capas superiores de software de las operaciones de virtualización.

3. [2.5] Responda a las siguientes cuestiones sobre el concpeto de hebra:

a. ¿Qué ventajas proporciona el modelo de hebras frente al modelo de proceso tradicional?

- Se reduce el tiempo de cambio de contexto entre hebras, así como el tiempo de creación y terminación de hebras.
- Mientras una hebra de una tarea está bloqueada, otra hebra de la misma tarea puede ejecutarse, siempre que el kernel sea *multithreading*. Por ejemplo, cualquier GUI.
- Usan los sistemas multiprocesador de manera eficiente y transparente, a mayor número de procesadores, mayor rendimiento.
- La comunicación entre hebras de una misma tarea se realiza a través del espacio de direcciones asociado a la tarea por lo que no necesitan utilizar los mecanismos del núcleo.

b. ¿Cuál es el inconveniente en la implementación de hebras de usuario a la hora de que se realice una llamada al sistema bloqueante por parte del programa?

- La mayoría de las llamadas al sistema son bloqueantes, por lo que si una hebra realiza una llamada, el resto de hebras se bloqueará.

c. Justifique el grado de paralelismo real alcanzado por una aplicación con varias hebras, teniendo en cuenta que los programas utilizan una biblioteca de hebras a nivel usuario y el núcleo no planifica hebras sino procesos.

- El grado de paralelismo es *NULO*:
- (INTERPRETACIÓN 1: SI LA APLICACIÓN ESTÁ FORMADA POR UN ÚNICO PROCESO). Cuando se utilizan hebras a nivel usuario, el kernel sólo asigna procesos a procesadores, por lo que no se puede asignar más de un procesador a más de una hebra de la misma tarea. En conclusión, las hebras se ejecutan secuencialmente.
- (INTERPRETACIÓN 2: SI LA APLICACIÓN ESTÁ FORMADA POR MÁS DE UN PROCESO). No obstante, si la aplicación consta de varios procesos, lograremos paralelismo a nivel global de la aplicación, pero no desde el punto de vista de los hilos de sus procesos.

4. [1.25] ¿Cómo implementa Linux el concepto de hebra? Explíquelo utilizado el PCB de Linux (struct task_struct) y la llamada al sistema `clone()`.

- Linux implementa el concepto de hebra como un proceso sin más, que simplemente comparte recursos con otros procesos.
- Cada hebra tiene su propia `task_struct`, que es una estructura que engloba el concepto de proceso e hilo. La llamada al sistema `clone()` crea un nuevo proceso o hebra dependiendo del parámetro *flags*. A diferencia de `fork()`, esta llamada permite al proceso hijo compartir partes de su contexto de ejecución con el proceso invocador, tales como el espacio de memoria, la tabla de descriptores de fichero y la tabla de manejadores de señal.
- A veces, es útil que el kernel realice operaciones en segundo plano, para lo cual se crean hebras kernel. Las hebras kernel no tienen un espacio de direcciones y por tanto su puntero mm es NULL. Estas hebras son ejecutadas únicamente en el espacio del kernel. Son planificadas y pueden ser expropiadas. Terminan cuando realizan una operación `do_exit` o cuando otra parte del kernel provoca su finalización.

5. [1.25] En un SO con una política de planificación apropiativa, enumere las distintas partes del SO que deben comprobar la posibilidad de desplazar al proceso que actualmente se está ejecutando y proponga un pseudocódigo que describa cómo se realizaría dicha comprobación en cada parte.

- Hay un total de 4 situaciones:

1. El proceso finaliza su ejecución. En este caso el planificador `schedule()` es llamado dentro de la rutina de núcleo que finaliza un proceso `sys_exit()` o `do_exit()`.
2. El proceso realiza una llamada bloqueante. En este caso, después de una llamda a E/S o una llamada `wait()`, se llama a `schedule()`.

3. El proceso agota su rodaja de tiempo (es interrumpido por la señal de reloj). La llamada a `schedule()` ocurre durante la interrupción de `RSI_reloj()`.
4. Se añade un proceso nuevo a la cola de Listos. Una vez que el SO ha creado el proceso nuevo o un proceso ha recibido la señal que lo desbloquea o se haya movido principal (en resumen, se pasa a la cola de Listos), se llama a `schedule()`, esto suele hacerlo el planificador a largo o medio plazo, excepto en el caso de que se desbloquee un proceso.

6. [1.25] Con respecto a la planificación de procesos responda las siguientes cuestiones:

a. ¿Qué algoritmo de planificación provoca una mayor penalización a los procesos limitados por E/S frente a los procesos limitados por CPU? ¿Por qué?

- El **FCFS**, ya que con procesos largos que no incluyan E/S, el proceso se ejecuta de una sola vez. Sin embargo, en los procesos cortos que tengan E/S, el proceso se bloqueará por cada una de las E/S que tenga, aumentando con creces el tiempo de espera y la pnealización con él.

b. Describa los factores a considerar a la hora de diseñar un algoritmo de planificación basado en colas múltiples con realimentación. En particular, justifique cómo asociaría los conceptos de quantum y prioridades a su diseño.

- Requiere definir los siguientes parámetros:
  - *Número de colas junto con su prioridad*.
  - *Quantum de tiempo asociado a cada cola.*
  - *Algoritmo de planificación para cada cola.*
  - *Método utilizado para determinar cuando trasladar a un proceso a otra cola.*
  - *Método utilizado para determinar en qué cola se introducirá un proceso cuando necesite un servicio.*
  - *Algoritmo de planificación entre colas.*
- Para intentar evitar el problema de que un proceso largo se alargue aún más o incluso pueda sufrir inanición, se suelen variar los tiempos de cada cola, de modo que un proceso que se encuentre en la cola de prioridad i, Cl_i, se le permite ejecutar 2^i unidades de tiempo antes de ser expulsado.

7. [1.25] Con respecto al núcleo de Linux visto en clase:

a. Describe los pasos que ejecuta el núcleo de Linux en la función `do_exit()`.

- Activa el flag `PF_EXITING` de `task_struct`.
- Para cada recurso que esté utilizando se decrementa el contador. Si este es 0 se realiza la operación de destrucción oportuna sobre el recurso.
- El campo *exit_code* de `task_struct` se pasará como argumento a `exit()` , y es la información de terminación para que el padre pueda hacer un `wait()` o `waitpid()` y recogerla.
- Se manda señal al padre indicando la finalización de su hijo. Si el proceso aún tiene hijos, se establece su padre al proceso init (podría darse el caso de poner como padre a otro proceso del mismo grupo). Se establece el campo *exit_state* de `task_struct` a `EXIT_ZOMBIE`.
- Se llama a `schedule()` para que el planificador elija un nuevo proceso a ejecutar.
- `do_exit()` nunca retorna.

b. Describa cómo se comporta el planificador de Linux, `schedule()`, para los procesos planificados mediante la clase de planificación CFS (Complete Fair Scheduling).

1. Determina la runqueue actual y establece el puntero prev a la `task_struct` del proceso actual.
2. Actualiza estadísticas y limpia el flag *TIF_NEED_RESCHED*
3. Si el proceso actual tiene el estado *TASK_INTERRUPTIBLE* y ha recibido la señal que esperaba, se cambia su estado a *TASK_RUNNING*.
4. Se llama a *pick_next_task* de la clase de planificación CFS, la cual selecciona como siguiente proceso a ejecutar el que tenga el menor valor de **vruntime** (tiempo virtual que ha consumido el proceso).
5. Si se neesita un cambio en la asignación de CPU, se hace un cambio de contexto mediante `context_switch()` ¡NUESTRO DISPATCHER()!!!!!.