**SO EJERCICIOS TEMA 2**

**1. Suponga un sistema con un único procesador. Describa detalladamente qué ocurre en el sistema cuando se está sirviendo por ocurre una interrupción de reloj en ese momento.**

El reloj genera una interrupción, y se accede al vector de interrupción correspondiente donde está la rutina de servicio de interrupción. Se transfiere el control a dicha rutina (habiendo guardado previamente los registros usados por la llamada al sistema). Tras la interrupción se contabiliza el tick de Reloj actual. Si no se ha agotado el quantum, se restaura el proceso anterior y se le da el control; si se ha agotado, se planifica.

**2. Supongamos un sistema que está ejecutando 2 procesos: un proceso P1 está en estado Bloqueado esperando que se pulse un único carácter desde teclado con la llamada al sistema correspondiente (read()); y un proceso P2 que se está Ejecutando. Sea un sistema operativo que utiliza una política de planificación apropiativa. Indique detalladamente qué pasos se siguen cuando el usuario pulsa una tecla, qué funciones del sistema operativo se encargan de realizar estas acciones y que cambios de estados se producen en los procesos.**

En un SO apropiativo, el proceso que se está ejecutando actualmente puede ser interrumpido y pasado al estado de Preparado por parte del sistema operativo. La decisión de apropiarse de la CPU puede llevarse a cabo cuando llega un nuevo proceso, cuando se produce una interrupción que lleva a un proceso Bloqueado al estado Preparado o periódicamente, en función de una interrupción del reloj.

Al pulsar la tecla, el dispositivo de E/S manda una interrupción. El proceso que estaba ejecutándose, P2, pasa a Preparado. Se accede al vector de interrupción correspondiente, donde está la dirección de la RSI (rutina de servicio de interrupción) de teclado, y se transfiere el control a dicha rutina. Si no hay ningún error, se le pasa la información a P1 y se marca como Preparado, insertándolo en la cola de preparados. Ahora se debe decidir si se reanuda la ejecución del proceso P2 o si se expulsa dicho proceso en favor de un proceso Preparado de mayor prioridad. Para ello se planifica (schedule()) y el proceso que tenga más prioridad (según la política de planificación que se use) pasará a ejecutarse.

**3. Si invocamos la función clone() con los argumentos que se muestran a continuación, indicar qué tipo de hebra/proceso estamos creando. Justificarlo mostrando cómo quedarían los descriptores de las tareas. clone(funcion,...,CLONE\_VM|CLONE\_FILES|CLONE\_FS|CLONE\_THREAD, …)**

funcion() es un puntero a una función que es ejecutada por el proceso hijo. Cuando funcion() regresa, el proceso hijo termina. El entero devuelto es el código de salida del proceso hijo.

Se crea una hebra/proceso con las siguientes características:

* CLONE\_FILES: Hilo padre e hijo comparten los mismos archivos abiertos.
* CLONE\_FS: Padre e hijo comparten la información del sistema de archivos.
* CLONE\_VM: Padre e hijo comparten el espacio de direcciones de usuario.
* CLONE\_SIGHAND: Comparten los manejadores de señales y señales bloqueadas.
* CLONE\_THREAD: Ambos procesos/hilos pertenecen al mismo grupo (mismo GID).

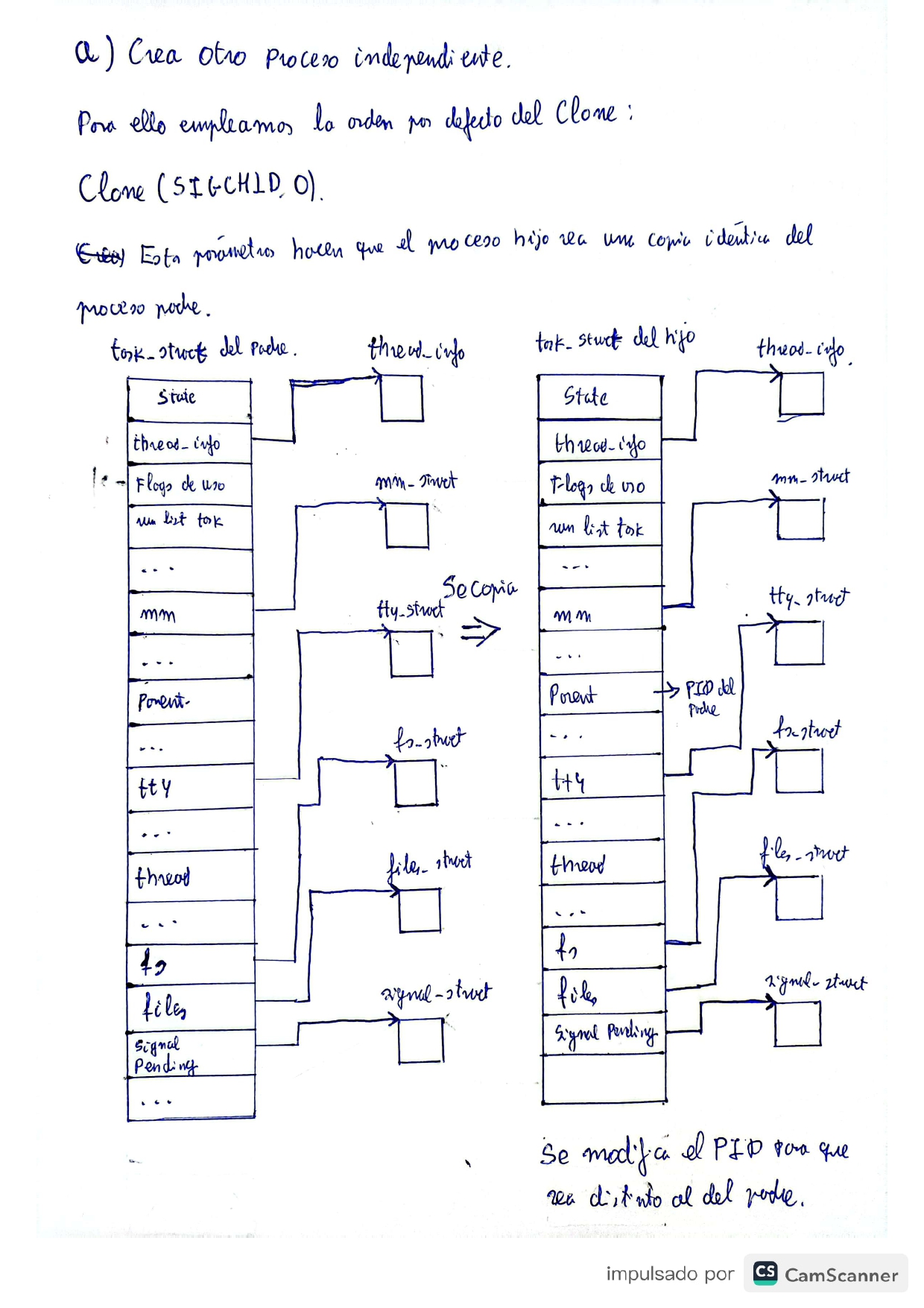
Sin embargo, clone() debe tener algunas precauciones:

* Algunos indicadores no tienen sentido juntos, por ejemplo, CLONE\_NEWNS y CLONE\_FS.
* Otros deben aparecer a la vez: CLONE\_THREAD con CLONE\_SIGHAND, o CLONE\_SIGHAND con CLONE\_VM.

**Aclaración:** cuando aparece el indicador CLONE\_xxx , la estructura xxx\_struct se comparte (no se copia).

**4. La llamada al sistema clone() en Linux se utiliza tanto para crear procesos como hilos (hebras). Indicar cómo la utilizaríamos, y dibujar los descriptores de procesos, cuando:**

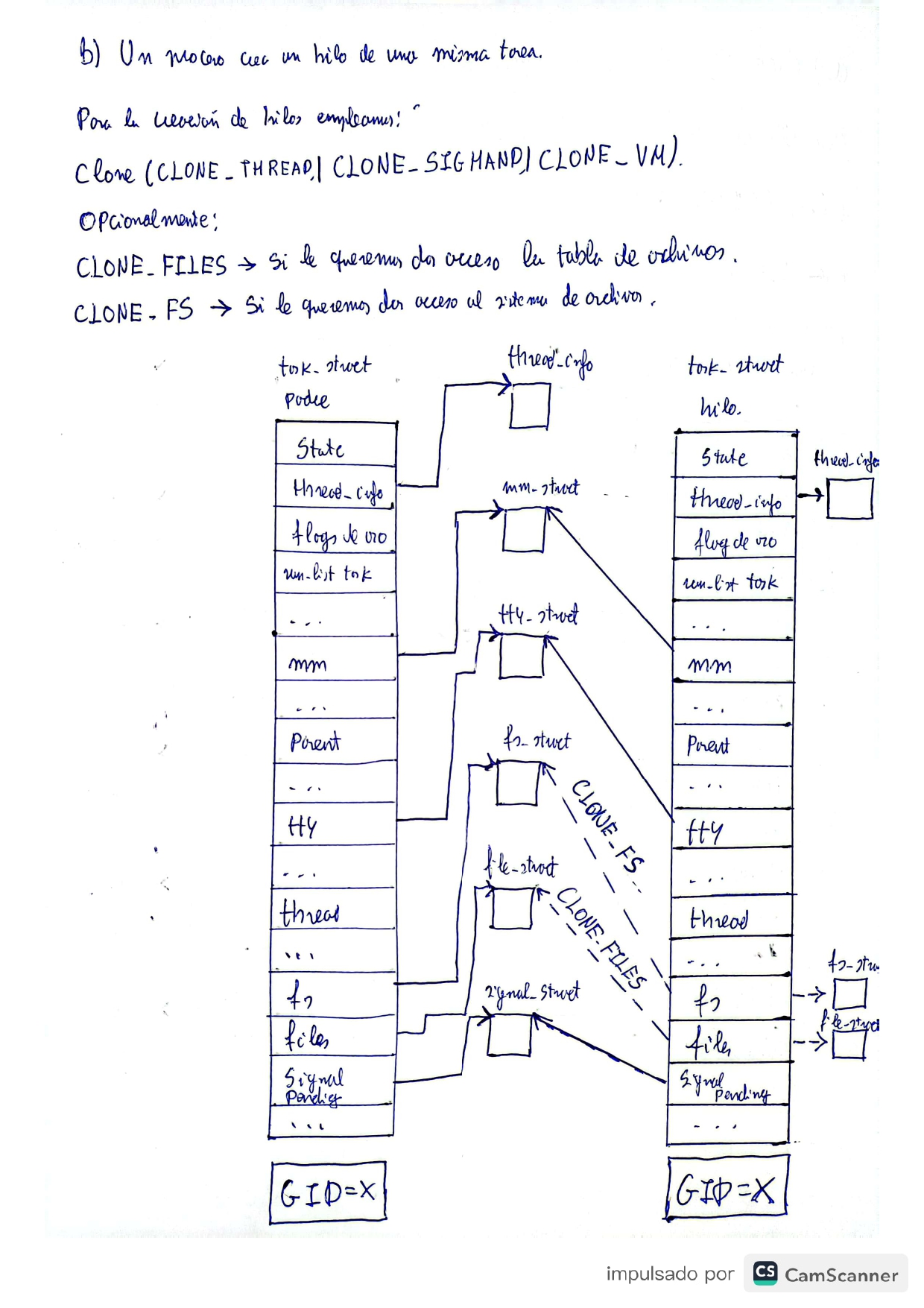
**a) Un proceso crea otro proceso independiente.**

****

Cuando se crea un hijo se copia todo menos el signal\_struct, el del hijo apunta al del padre.

(Es un error en el dibujo).

**b) Un proceso crea un hilo de la misma tarea.**

****

La línea discontinua que va a files\_struct y fs\_struct se debe a que si se incluyen los flags mencionados CLONE\_FILES,CLONE\_FS el puntero apuntará a los del padre, si no creará una copia de los del padre y serán completamente independientes.

El introducir o no estos flags, depende de las tareas que queremos que realice nuestra hebra.

**5. El algoritmo de planificación CFS (Completely Fair Scheduler) de Linux reparte imparcialmente la CPU entre los procesos de la clase correspondiente. Justificar cómo es posible que un proceso que realiza muchas entradas-salidas (por tanto, tiene ráfagas de CPU muy cortas) obtenga un trato imparcial respecto de un proceso acotado por cómputo (pocas entradas-salidas y que consume las ráfagas asignadas completamente). Para el razonamiento, podemos suponer que solo hay un proceso de cada tipo y que la prioridad base de ambos es 120.**

CFS asigna una proporción del procesador dependiendo de la carga del sistema (número de procesos). Cada proceso se ejecuta durante un tiempo proporcional a su peso dividido por la suma total de pesos. Se establece un tiempo mínimo. Para seleccionar un proceso se usa como criterio el tiempo virtual de ejecución (vruntime). CFS intenta equilibrar los tiempos virtuales eligiendo primero al proceso con menor vruntime. De esta manera, los procesos que consumen todo el quantum, los atraso; y los que consumen poco les doy prioridad para que se ejecuten.

La fracción de CPU es la misma para los dos procesos. Si consume toda o casi toda la CPU, mayor es el vruntime=>se atrasa su ejecución.

**6. Respecto a la planificación que usa Linux, suponga que actualmente existen 4 procesos: 2 de Tiempo-real ambos con valor de prioridad 20 (uno pertenece a la clase de planificación SCHED\_RR, y el otro a la clase SCHED\_FIFO) y 2 de tiempo compartido con prioridad 120 de la clase SCHED\_NORMAL. Indique la secuencia de ejecución de dichos procesos que se realizará en el sistema.**

El planificador de Linux consta de al menos tres clases de planificación: 2 de tiempo-real y 1 de tiempo-compartido. El planificador de Linux da prioridad a los procesos de tiempo-real frente a los de tiempo compartido, luego se ejecutarán primero los 2 de tiempo-real. Ahora veremos cual de los dos de tiempo-real se ejecutará antes. Como pertenecen a clases de planificación distintas, no podemos decir cuál de los dos se ejecutará antes, solo que lo harán antes de sus tiempos límite (deadline). Si la cola de RR, está antes que el FIFO, …. y si es al revés, se ejecutará hasta que finalice.

Por otro lado, a continuación de los de tiempo-real, los dos procesos de tiempo-compartido. El CFS determinará en función del vruntime cual se ejecutará antes, ya que tienen la misma prioridad.

**7. Explicar cómo se lleva a cabo la finalización de un proceso/hilo por parte del kernel de Linux tras una invocación explícita de la llamada al sistema exit() hasta la liberación de todos los recursos usados por el mismo.**

La llamada al sistema exit() libera los recursos utilizados y devuelve un valor de estado de salida que recibirá el proceso padre con wait().

El hijo realiza un exit(), entonces se cambia de modo usuario a modo kernel y se realiza:

• Cambio estado para decir al resto de procesos y planificador ‘esto no se toca’.

• Activa PF\_EXITING

• Decrementa los contadores de uso de mm\_struct, fs\_struct, files\_struct. Si estos contadores alcanzan el valor 0, libera los recursos.

• Ajusta el exit\_code del descriptor, que será devuelto al padre, con el valor pasado al invocar a exit().

• Envía al padre la señal de finalización; si tiene algún hijo le busca un padre en el grupo o el init, y pone el estado a TASK\_ZOMBIE.

• Invoca a schedule() para ejecutar otro proceso.

• Después se procede a llamar wait(). (Trabajan juntos exit() y wait()). El padre recibirá así el código de salida y se elimina el task\_struct.

**8. En un sistema Linux pueden existir procesos que pertenecen a diferentes clases de planificación. Indicar:**

**a) ¿Qué clases son estas y que algoritmo implementan?**

Planificación de tiempo real, planificación neutra o limpia (CFS) y planificación de la tarea ‘idle’. Implementan el algoritmo por prioridad apropiativo.

**b) Si en un momento dado, hay en el sistema al menos un proceso que pertenece a cada una de las clases, indicar en qué orden se planificarán.**

Deadline,Tiempo real, CFS y ‘idle’.

**c) Si en la clase de tiempo compartido tenemos tres procesos: P1 y P2 con prioridad 120 y P3 con prioridad 110 ¿qué porcentaje de CPU la asignará en planificador a cada uno sabiendo que la prioridad 120 tiene un peso de 1024 y a la prioridad 110 le corresponde un peso de 9548?**

Dividir el peso de cada uno entre la suma de todos los pesos:

p1 y p2 = 8,83

p3 = 82,34

**9. Entre los diferentes pasos que sigue el planificador de Linux, función schedule(), está el de seleccionar la siguiente tarea a ejecutar (función pick\_next\_task()). A la vista de lo estudiado, esbozar los pasos que debe seguir esta función para seleccionar el proceso teniendo en cuenta que existen tres clases de planificación (SCHED\_FIFO, SCHED\_RR y SCHED\_NORMAL) y de las características/propiedades de los procesos dentro de cada clase.**

Función pick\_next\_task(): Primero tareas de tiempo real y se planifica según sean de FIFO o RR y luego pasa a las de tiempo compartido. Siempre selecciona para ejecución a un proceso que pertenezca al mayor nivel dentro de la jerarquía de clases. Si pertenecen a la misma clase

1. Con SCHED\_FIFO: Esta política manejada por el planificador de tiempo real consiste en que en el momento en que un proceso pasa a listo, se une a la cola de preparados. Cuando el proceso actualmente en ejecución deja de ejecutarse, se selecciona el que más tiempo lleva en la cola de listos, luego la función se encarga de llamar al siguiente proceso en la cola de preparados(rq) cuando el anterior termine su ejecución o se bloquee
2. Con SCHED\_RR: Esta política también manejada por el planificador de tiempo real consiste en que se genera una interrupción de reloj cada cierto intervalo de tiempo, cuando sucede, el proceso actual en ejecución se sitúa en la cola de preparados y se selecciona el siguiente trabajo según FCFS(FIFO). Luego la función realiza la misma tarea que con SCHED\_FIFO(solo cambia el cuándo la realiza).
3. Con SCHED\_NORMAL: Se aplica a los procesos normales de tiempo compartido, la función debe escoger según el nivel de prioridad (tiene dos subclases, SCHED\_BATCH→Tareas menos importantes y SCHED\_IDLE→Prioridad mínima), y en caso de que haya dos procesos con la misma prioridad, ejecutará al que lleva más tiempo esperando.

**10. Suponga un sistema Linux con dos procesos: uno de tiempo-real de la clase SCHED\_RR que está actualmente bloqueado; otro de la clase CFS que está en ejecución usando un recurso compartido del sistema. Cuando el proceso de tiempo-real se desbloquea, indicar la secuencia de eventos que se producen hasta que pasa a ejecutarse sabiendo que debe usar el recurso que está usando ahora el proceso de tiempo compartido.**

El proceso de tiempo compartido al ser de Linux será de la clase SCHED\_NORMAL. El proceso bloqueado de tiempo real, en cambio, pertenece a la clase SCHED\_RR. En el caso de que el proceso de T-R tenga una prioridad mayor que la del de tiempo compartido, se bloqueará a este último para liberar el recurso que necesita el otro proceso. Si, en cambio, son de igual prioridad, o el de tiempo compartido tiene mayor prioridad que el de T-R, este se empezará a ejecutar una vez el de T-R haya cumplido su plazo (*deadline*) o haya acabado su ejecución.

CORRECCIÓN:

P1=el que está preparado, P2=el que está ejecutándose. Esperar a que p2 haya acabado, pasa a listo y el planificador tiene que decidir y prioriza a p1 si es que no hay otro en la cola. Cuando le da la cpu, p2 puede que haya tardado mucho en liberar el recurso y cuando llegue a p1 no necesita llevar ya a cabo el proceso. P2 “hereda” la prioridad de P1 (le presta su prioridad), con la intención de provocar que termine el proceso P2 cuanto antes y reducir la latencia de despacho.

**11. ¿Qué son los Gobernadores (Governors) en un sistema Linux? Indicar cuales son los dos gobernadores de usuario y su función.**

El gobernador es un módulo que define las características de energía de la CPU del sistema que a su vez afectan el rendimiento de la CPU. Cada gobernador tiene su propia conducta, propósito e idoneidad en términos de carga de trabajo.

Los gobernadores de usuario son el gobernador PowerSaved, que se utiliza junto con el daemon cpuspeed. ****

**12. El planificador de Linux implementa una política apropiativa denominada “apropiatividad mediante puntos de apropiación”. a) ¿En qué consiste y/o cómo se implementa? b) ¿Qué ventajas e inconvenientes presenta respecto a una política totalmente apropiativa?**

1. Los kernel actuales de Linux son apropiativos y permiten ajustar el grado de apropiatividad según el uso que le vayamos a dar.

Se implementa a través de puntos de apropiación: puntos del flujo de ejecución kernel donde es posible apropiar al proceso actual sin incurrir en una condición de carrera.

Dado que la invocación asíncrona del planificador podría generar condiciones de carrera, esta se difiere hasta alcanzar un punto de apropiación.

1. Ventajas: se evitan condiciones de carrera, desde el punto de vista de programación es más fácil diseñar el código.

Desventaja: no es totalmente apropiativo, entre dos puntos no hay una distancia fija, luego no controlo cuan instanciado están y por tanto no tengo una cota de tiempo para saber cuando voy a planificar.

No planifico, pongo un punto de apropiación, entre dos puntos no hay una distancia fija, lo que quiere decir que podría tener una tarea cuya deadline sea mayor, por lo que es tarea ya no cumpliria el plazo.

**13. En el algoritmo de planificación de la clase CFS siempre se elige como siguiente proceso para ejecución al proceso cuyo vruntime es menor. Indicar que representa este parámetro y como se calcula.**

La clase de Planificación de CFS (*Completely Fair Scheduling*) reparte el tiempo de CPU de forma imparcial, garantizando que todos los procesos se ejecutarán y, dependiendo del número de procesos, asignarles más o menos tiempo de uso de CPU. El kernel calcula un peso para cada proceso. Cuanto mayor sea el valor de la prioridad estática de un proceso, menor será el peso que tenga.

vruntime (virtual runtime) de una entidad es el tiempo virtual que un proceso ha consumido y se calcula a partir del tiempo real que el proceso ha hecho uso de la CPU, su prioridad estática y su peso. El tiempo partido por la carga (peso de todos los procesos del sistema).

El valor vruntime del proceso actual se actualiza:

* periódicamente (el planificador periódico ajusta los valores de tiempo de CPU consumido)
* cuando llega un nuevo proceso ejecutable
* cuando el proceso actual se bloquea

Cuando se decide qué proceso ejecutar a continuación, se elige el que tenga un valor menor de vruntime. Para realizar esto CFS utiliza un rbtree (red black tree).

**14. Justificar por qué el kernel de Linux no es apto para la ejecución de aplicaciones de tiempo real duras, es decir que tienen plazos estrictos de ejecución. Observación, recordad la definición de latencia de apropiación y la implementación que hace de esta el kernel.**

Incluso usando un mecanismo para evitar la inversión de prioridad, eso no garantiza que un proceso pueda cumplir sus plazos. Si tengo un punto de apropiación en la instrucción n y otro en la m, el tiempo que tardo en volver a planificar, es decir, en llegar a la instrucción m, puede ser mayor que el deadline de un proceso de tiempo real y por tanto se produciría un error fatal.

Sabemos que las aplicaciones de tiempo real duro son aquellas que deben realizarse en el tiempo de respuesta límite de otro modo se producirá un daño inaceptable o error fatal del sistema.



LATENCIA DE DESPACHO (DISPATCH LATENCY) – tiempo del despachador de parar un proceso e iniciar otro. La parte más complicada es la fase de conflicto, en la que se producen dos pasos:

1. Apropiación de cualquier proceso que se está ejecutando en modo kernel.
2. Liberar recurso de proceso con baja prioridad necesitado por proceso con alta prioridad.

Para todo esto debemos estar tratando con planificadores kernel APROPIATIVOS.

En el caso de Linux no es apto porque su planificación puede producir la llamada inversión de prioridad, por la que algunos procesos nunca lleguen a ejecutarse (o tarden demasiado) debido a que las circunstancias de un sistema fuercen a una tarea de mayor prioridad a esperar por una de menor prioridad. Esto no es compatible con aplicaciones con fuertes restricciones de tiempo de ejecución, puesto que si se produce es fácil que se sobrepase el límite y se produzcan errores fatales (ejemplo del robot de Marte que comenzó a fallar).

**15. ¿Qué mecanismos utiliza el kernel de Linux para gestionar el consumo de energía de los procesadores?**

Para gestionar la potencia, el subsistema CPUfreq ajusta explícitamente la frecuencia del procesador (a más frecuencia más consumo de energía y más potencia).

Se separan las políticas (gobernadores) de los mecanismos (drivers específicos de CPU).

Las herramientas son las siguientes:

-Cpufrequtils – podemos ver, modificar los ajustes del kernel relativos al

subsistema CPUfreq. Las órdenes cpufreq\* son útiles para modificar los estados-P,

especialmente escalado de frecuencia y gobernadores.

-Cpupower – ver todos los parámetros relativos a potencia de todas las CPUs,

incluidos los estados-turbo. Engloba a la anterior.

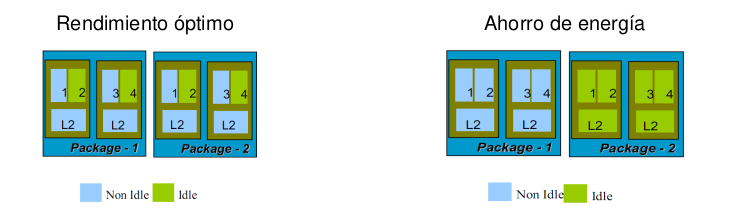
-PowerTOP – ayuda a identificar las razones de un consumo alto innecesario, por ejemplo, procesos que despiertan al procesador del estado ocioso.

- Se pueden crear perfiles en /etc/pm-profiler.

-TLP herramienta para gestionar energía de forma avanzada.

En cuanto a la planificación, también se puede optimizar el consumo de energía.

Para el ahorro de energía se opta por ocupar los mínimos procesadores posibles, y dejar el resto sin hacer nada (se apagan), es decir, se termina de ocupar un procesador antes de encontrar uno vacío. Para un rendimiento óptimo, se reparten equitativamente los procesos por todos los procesadores.

****

