

## SISTEMAS OPERATIVOS

### 2º Curso

#### Ejercicios del Tema 2

1. Suponga un sistema con un único procesador. Describa detalladamente qué ocurre en el sistema cuando se está sirviendo por el núcleo monolítico una llamada al sistema para abrir un archivo (`open()`) y ocurre una interrupción de reloj en ese momento.
2. Supongamos un sistema que está ejecutando 2 procesos: un proceso  $P_1$  está en estado *Bloqueado* esperando que se pulse un único carácter desde teclado con la llamada al sistema correspondiente (`read()`); y un proceso  $P_2$  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 qué cambios de estados se producen en los procesos.
3. Si invocamos la función `clone()` con los argumentos que se muestran a continuación, indicar qué tipo de hebra/proceso estamos creando. Justificarlo mostrando como quedarían los descriptores de las tareas.

```
clone(funcion, ..., CLONE_VM|CLONE_FILES|CLONE_FS|CLONE_THREAD, ...)
```

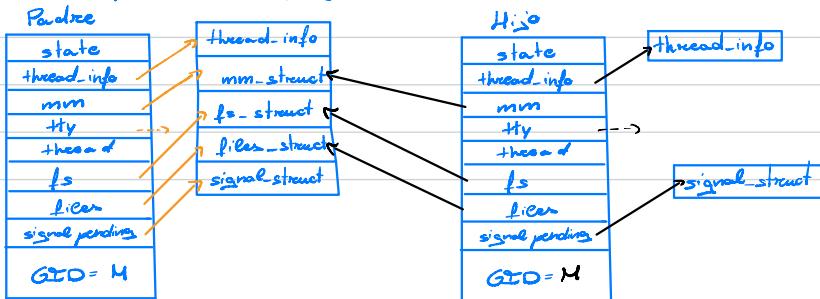
4. La llamada al sistema `clone()` en Linux se utiliza tanto para crear procesos como hilos (hebras). Indicar cómo la utilizaríamos, y dibujar los descriptores de procesos, cuando:
  - a) Un proceso crea otro proceso independiente.
  - b) Un proceso crea un hilo de la misma tarea.
5. El algoritmo de planificación *CFS* (*Completely Fair Scheduler*) de Linux reparte imparcialmente la CPU entre los procesos de la clase correspondiente. Justificar como 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 computo (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.
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 *SCED\_RR*, y el otro a la clase *SCED\_FIFO*) y 2 de tiempo compartido con prioridad 120 de la clase *SCED\_NORMAL*. Indique la secuencia de ejecución de dichos procesos que se realizará en el sistema.
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.
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?
  - 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.
  - c) Si en la clase de tiempo compartido tenemos tres procesos:  $P_1$  y  $P_2$  con prioridad 120 y  $P_3$  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?

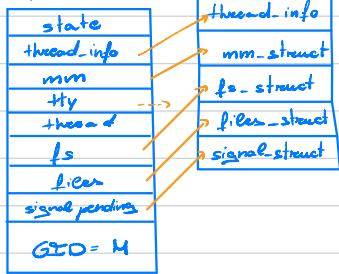
9. Entre los diferentes pasos que sigue el planificador de Linux, función `schedule()`, está el de seleccionar la siguiente tarea a ejecutar (función `pick_next_task()`). A la vista de lo estudiado, esbozar los pasos que debe seguir esta función para seleccionar el proceso teniendo en cuenta que existen tres clases de planificación (`SCHED_FIFO`, `SCHED_RR` y `SCHED_NORMAL`) y de las características/propiedades de los procesos dentro de cada clase.
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.
11. ¿Qué son los *Gobernadores (Governors)* en un sistema Linux? Indicar cuales son los dos gobernadores de usuario y su función.
12. El planificador a medio plazo 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?
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.
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.
15. ¿Qué mecanismos utiliza el kernel de Linux para gestionar el consumo de energía de los procesadores?

- 1.- Como el sistema está tratando la llamada del open(), tecminará esta, bloqueará el proceso guardando el procesador el PC y el PSW de este para ir a la rutina de tratamiento de interrupción, cuya dirección encontrará en la tabla de vectores de interrupción en base a su id. Una vez se realice esta, se produce el retorno, restaurando así la ejecución del proceso
- 2.- Como el proceso estaba bloqueado por una operación de E/S que finaliza, el planificador marcará que es necesario replanificar, activando el bit USED-KSCHED? cambiando task\_struct.state a TASK\_RUNNING. Una vez llegado a un punto de apropiación, con los contenidos de la pila Kernel seguros y sin posibilidad de causar una condición de carrera, se replanificará, comprobando si tiene mayor prioridad el proceso 1 para provocar un cambio de contexto y que este pase a ejecutarse, pasando a la cola de preparados el proceso 2.

- 3.- Estaríamos creando una hebra del mismo proceso que su padre, ya que compartiría el mismo GID por la flag CLONE\_THREAD.



Padre



Hijo

