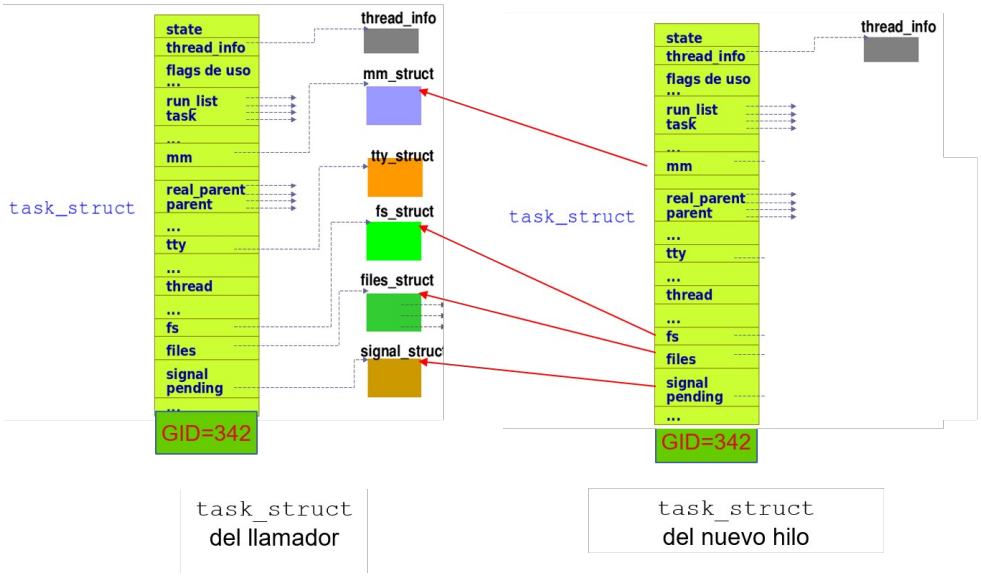
SISTEMAS OPERATIVOS
2º Curso
Ejercicios del Tema 2

Tema 2:

1. Si invocamos la función `clone()` con los argumentos que se muestran a continuación, indicar que tipo de hebra/proceso estamos creando. Justificarlo mostrando como quedarían los descriptores de las tareas.

```
clone(funcion,...,CLONE_VM|CLONE_FILES|CLONE_FS|CLONE_TRHEAD, …)
```



2. 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.
      Llamada al sistema directa: clone(SIGCHLD, 0)
   b) Un proceso crea un hilo de la misma tarea.

      Usando la función de glibc: clone(fn, pila, CLONE_VM|…| CLONE_SIGHAND)

3. 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.
   Pc = proceso computo y Pes= proceso E/S
   Si Pc y Pes tiene la misma prioridad base, recibiran el mismo tiempo de CPU. Cada vez que se inserte Pes en la cola de preparados tras esperar por un E/S, se invocará al planificador y se le dará el control a Pes pues dado que al no haberse ejecutado su

vruntime será menor que el de Pc. Es decir, Pes no se penaliza pues en cuanto esta lista para ejecutarse tomará el control de la CPU.

4. Explicar cómo se lleva a cabo la finalización de un proceso/hebra 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 destrucción de un proceso que ha finalizado se realiza en dos pasos:
   1º) La función exit() realiza la llamada correspondiente del SO que destruye las esturcturas de datos mm_struct, fs_struct y files_strcut, notifica al padr de su finalización y pone al proceso en estado ZOMBIE.
   2º) Cuando el padre (real o init/systemd) hace un wait() sobre recoge el código de finalización del task_struct del hijo y ya se puede destruir el descriptor del proceso.

5. En un sistema Linux pueden existir procesos que pertenecen a diferentes clases de planificación. Indicar:
   a) ¿Qué clases son estas y que algoritmo implementan?
      Teoría: minimas SCHED_CFS, SCHED_FIFO y SCHED_RR
   b) Si en un momento dado, hay en el sistema al menos un proceso que pertenece a cada una de las clases, indicar en que orden se planificarán.
      Teoría: 1) Tiempo-real y 2) tiempo compartido
   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 ~~110~~ 9548?
      P1 y P2: 100*1024 / (1024+1024+9548) ~ 9 %
      P3: 100*9548 / (1024+1024+9548) ~ 82 %

6. 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.

```
Estructura clases: clase_tiempo-real → clase_cfs → clase_idle
for_each_class (class) {
            p = class →pic_next_task(rq, prev)
return p
}
```

   donde `class→pick_next_task` aplica el algoritmo de cada clase (ver teoría)

7. 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 esta 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 esta usando ahora el proceso de tiempo compartido.
   El proceso de tiempo-real (Prt) se desbloquea solicita el recurso compartido. El cerrojo que esta en uso esta cerrado al esta en uso el recurso que protege por proceso cfs (Pcfs). El sistema detecta el evento y el cerrojo contiene el PID del proceso que lo esta bloqueando. El mecanismo de herencia de prioridad usado por Linux hace que que prioridad herreda de Pcfs sea igual a la prioridad de Prt, para evitar que se produzca inversión de prioridad. Este proceso continua ejecutandose hasta que termine (si no

hay proceso más prioritarios que Prt) y al terminar libera el recurso, que ya podrá ser usado por Prt.

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

9. 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?
      El planificador no se invoca inmediatamente cuando un proceso esta listo para ejecutarse sino que se espera a invocarlo cuando se alcanza un punto de apropiación (if TF_NEED_RESCHED == 1). Cuando se debe activar el planificador, se activa la bandera de replanificación en su lugar (TF_NEED_RESCHED = 1)
   b) ¿Qué ventajas e inconvenientes presenta respecto a una política totalmente apropiativa?
      Ventaja: menos esfuerzo de programación al no tener que proteger todas y cada una de las ED del kernel con un cerrojo.
      Incoveniente: menos responsibidad dado que la distancia entre dos puntos de apropiación puede ser grande.

10. 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.
    Teoría

11. 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.
    La distancia entre puntos de aproación puede no tener una cota de ejecución adecuada para estas aplicaciones

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

13. Sobre los grupos de control (*cgroups*) en Linux:
    a) ¿Qué utilidades tiene esta construcción?
       Asignar/limitar/contabilizar/control el uso de ciertos recursos por parte de los procesos/entidades
    b) ¿Cómo dan soporte a la virtualización?
       La MVs obtiene aislamiento/asignacion/control de los recursos de cgroups
    c) Indicar al menos tres subsistemas de grupos de control.
       Directo de teoría