TEMA2: PROCESOS Y HEBRAS

Relación de problemas

1.Cuestiones generales sobre procesos y asignación de CPU:

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

Teoría pura y dura.

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

No, (una llamada al sistema es una petición explícita), por ejemplo, las excepciones no recuperables pasan por el sys_exit(), que es el liberador de recursos. Pero se puede llegar a la sys_exit mediante otras llamadas o excepciones, no sólo el exit().

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

```
context_switch(){
  PID=planifCPU() //o scheduler
  PID.ESTADO='BLOQUEADO_HD'
  ENCOLAR(PID, 'BLOQUEADOS_HD');
  DISPATCH(PIDcpu, PID);
}
```

d) ¿Qué debería hacer cualquier planificador a corto plazo cuando es invocado pero no hayningún proceso en la cola de ejecutables?

Siempre hay una tarea "doomie", tarea tonta, que cuando no hay trabajos, entra y vuelve en bucle. Es una tarea que siempre está ahi, meramente estadística.

e)¿Qué algoritmos de planificación quedan descartados para ser utilizados en sistemas detiempo compartido?

Los que no valen en un timesharing son:
los no apropiativos y el first Come First Served (LOS DOS PRIMEROS DE TEORÍA), junto con el de prioridades

2. Cuestiones sobre el modelo de procesos extendido:

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

TEORÍA: Pedir PCB, inicializarlo, etc.

b) ¿Qué pasos debe llevar a cabo un SO para poder pasar un proceso ejecutándose en CPUa estado “FINALIZADO”?

Tenemos que hacer un sys_exit(), como terminación de procesos.
Se requiere:

- Liberar recursos
  (Después de eso nos queda el PCB)
- PCB.estado_finalizacion= EXIT_SUCCESS (valor concreto)
  -Enviar señal de finalización al padre (SIGCHILD), pues un proceso puede estar esperando a la finalización de un proceso hijo.
- El proceso hijo puede tener procesos hijos (Descendientes), que se colocan como hijos de INIT.

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

El context_switch se ha usado en rutinas de E/S, en el timeout y el `sys_exit` .
Hay que tener cuidado con adaptar dónde se encola el proceso anterior, porque varía. Si está en TIMEOUT, se encola en "LISTOS", sys_exit() e, "FINALIZADOS", etc.

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

a) ¿Qué elementos de información es imprescindible que contenga una estructura de datosque permita gestionar hebras en un kernel de SO? Describa las estructuras task_t y lathread_t.

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

c)

Si se bloquea una hebra me da igual, el planificador sigue tal cual.

6. ¿Puede el procesador manejar una interrupción mientras está

ejecutando un proceso sinhacer context_switch() si la política de planificación que utilizamos es no apropiativa?¿Y si es apropiativa?

El procesador tiene que manejar la interrupcion, con una política no apropiativa, significa que solo se llama a context_switch desde timeout o, tipo A y tipo B. Las apropiativas son las tipo B.
Si es de las antiguas, no apropiativa, timeout, manejo la interrupcion RSI, ¿se puede servir context_switch? (ver teoría sobre RSI). No se puede ejecutar el context_switch.

Si es apropiativa, tampoco, pues la RSI y el vector de interrupción es lo más bajo. Nada que se haga por encima puede variar su comportamiento.

7. Suponga que es responsable de diseñar e implementar un SO que va a utilizar una política de planificación apropiativa (preemptive). 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).

Las partes que trabajan sobre la planificación B que va de nuevo a listo se llaman las `program load` : pedir PID, PCB, cargar en memoria, ...
El código de sistema operativo que debería incluirse es (sólo hay 3 con dispatch):

```
if (Planif_CPU)
then
  DISPATCH
  //encolar en listo el estado del PID listo
```

Sólo sabemos RSI, al final de la RSI, antes del pop y ret, se hace la verificacion `if (Planif_CPU)``