RELACIÓN SO: TEMA 2

1. Cuestiones generales sobre procesos y asignación de CPU:

- *¿Cuáles son los motivos que pueden llevar a la creación de un proceso?:*
    - En sistemas batch: en respuesta a la recepción y admisión de un trabajo.
    - "Logon" interactivo. Cuando el usuario se autentifica desde un terminal (logs on), el SO crea un proceso que ejecuta el intérprete de órdenes asignado.
    - El SO puede crear un proceso para llevar a cabo un servicio solicitado por un proceso de usuario.
    - Un proceso puede crear otros procesos formando un árbol de procesos (relación padre-hijo).

- *¿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()?:*
    - Sí, para que seguidamente se mande el aviso de finalización al padre, se guarde el estado de finalización y el SO libere los recursos asociados al proceso. Si no se realizara no se mataría el proceso.
    - Sin embargo, si se produce una excepción no recuperable (error en el proceso o terminación inesperada por el propio SO), no se llama a sistema desde el proceso, sino que se ejecuta `SYS_exit`.

- *Cuando un proceso pasa a estado "BLOQUEADO", ¿Quién se encarga de cambiar el valor de su estado en el descriptor de proceso o PCB?:*
    - El SO es el encargado de cambiar el contexto de los procesos, incluyendo su estado. La función concreta que realiza esto es `context_switch()`, que está compuesto por:
    - `context_switch()`:
        - `schedule()`: Es el planificador corto (planificador de CPU).
        - `dispatch()`: Función del SO que da el control de la CPU al proceso seleccionado por el planificador a corto plazo.
    - Concretamente, es la función `dispatch()` la que se encarga de cambiar el contexto de registros desde el punto de vistea de la CPU, incluyendo así el cambio de estado del proceso, escribiendo dicho nuevo estado en el PCB. Después se llamaría a `scheduler()` para mandar esta tarea a la cola de procesos bloqueados.

- *¿Qué debería hacer cualquier planificador a corto plazo cuando es invocado pero no hay ningún proceso en la cola de ejecutables?*:
    - Llama al proceso idle(). Cuando algún procesador se queda sin tareas, llama a idle() y ese tiempo desocupado lo aprovecha ejecutando tareas del núcleo.

- *¿Qué algoritmos de planificación quedan descartados para ser utilizados en sistemas de tiempo compartido?:*
    - Los **planificadores no apriopiativos**: En los SO de tiempo compartido varios usuarios acceden al mismo conjunto de recursos, por lo que si un proceso se queda en CPU no

se puede después expropiar en caso de que venga otro proceso que se requiera ejecutar con más urgencia.
    - Los **planificadores por prioridad**: En sus ambas modalidades (apropiativa y no apropiativa), si hay un proceso en la CPU que dispone de mayor prioridad que todos los que hay en la cola de Listos, éste permanecerá ahí hasta que termine su ejecución (y sucede lo mismo que el caso anterior).

2. Cuestiones sobre el modelo de procesos extendido:

- *¿Qué pasos debe llevar a cabo un SO para poder pasar un proceso de reciente creación de estado "NUEVO" a estado "LISTO"?:*
    - Cuando se crea un proceso aún no es admitido por el SO. En general los procesos que se encuentran en este estado todavía no han sido cargados a memoria. Cuando pasa a estado LISTO, está listo para ser ejecutado y está esperando a que el planidifcador así lo disponga.
- *¿Qué pasos debe llevar a cabo un SO para poder pasar un proceso ejecutándose en CPU a estado "FINALIZADO"?*:
    - Aviso de finalización al padre ↪ se guarda el estado de finalización ↪ se libreran losrecursos asociados al proceso por parte del SO.
- 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?
    - Debe realizar la función `schdeule()` y `dispatch()`.
- Indique el motivo de la aparición de los estados "SUSPENDIDO-BLOQUEADO" y "SUSPENDIDO-LISTO" en el modelo de procesos extendido.
    - PROCESOS SUSPENDIDOS:
        - El procesador es más rápido que la E/S.
        - Suspender consiste en pasar una parte o todo el proceso al disco para liberar la memoria principal. Disminuye así el nivel de multiprogramación.
        - Cuando procesos en memoria principal están bloqueados, el SO puede suspender un proceso poniéndolo en suspendido.
        - El estado bloqueado se convierte en suspendido cuando el proceso está swappeado en disco:
            - **Bloqueado/Suspendido**: El proceso se encuentra en memoria secundaria esperando un proceso.
            - **Suspendido/Listo**: El proceso está en memoria secundaria pero está disponible para ejecutar tan pronto como se cargue en memoria principal.

3. ¿Tiene sentido mantener ordenada por prioridades la cola de procesos bloqueados? Si lo tuviera, ¿en qué casos sería útil hacerlo? Piense en la cola de un planificador de E/S, por ejemplo el de HDD, y en la cola de bloqueados en espera del evento “Fin E/S HDD”.

- Tiene sentido siempre y cuando solucionemos el problema de inversión de prioridad,ya que un proceso con poca prioridad puede quedarse bloqueado indefinidamente si siguen llegando procesos con mayor prioridad. Esto se soluciona con el uso de “herencia de prioridad” y “techo de prioridad”.

4. Explique las diferentes formas que tiene el kernel de ejecutarse en relación al contexto de un proceso y al modo de ejecución del procesador.

- Contexto de Proceso ↪ Llamadas al sistema y excepciones.
- Contexto de CPU ↪ Tratamiento de interrupciones y tareas del sistema.

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

- *¿Qué elementos de información es imprescindible que contenga una estructura de datos que permita gestionar hebras en un kernel de SO? Describa las estructuras* `task_t` *y la* `thread_t`*.*
  - *INFORMACIÓN IMPRESCINDIBLE*: Contador de programa/Conjunto de registros/Espacio de pila/Estado PCB.
  - `task_t`: PCB {PID, ESTADO, RECURSOS, MEMORIA, LISTA DE HEBRAS}
  - `thread_t`: TCB {TID, ESTADO, CONTEXTO REGISTRADOS, PARAMETROS PLANIFICADORES, PILA USER, PILA KERNEL}
- *En una implementación de hebras con una biblioteca de usuario en la cual cada hebra de usuario tiene una correspondencia N:1 con una hebra kernel, ¿Qué ocurre con la tarea si se realiza una llamada al sistema bloqueante, por ejemplo* `read()` *?*
  - Todas las hebras de usuario de un proceso se vinculan a una única hebra del núcleo. La tarea se bloquea.
- *¿Qué ocurriría con la llamada al sistema read() con respecto a la tarea de la pregunta anterior si la correspondencia entre hebras usuario y hebras kernel fuese 1:1?*
  - Se bloquea solo la hebra correspondiente.

6. ¿Puede el procesador manejar una interrupción mientras está ejecutando un proceso sin hacer context_switch() si la política de planificación que utilizamos es no apropiativa? ¿Y si es apropiativa?

- Si la política es NO APROPIATIVA no puede hacerlo ya que el procesador no se le puede retirar a dicho proceso hasta que éste voluntariamente lo deje (finalice o se bloquee).
- Si la política es APROPIATIVA si puede hacerlo ya que el SO puede apropiarse del procesador cuando lo decida.

7. Suponga que es responsable de diseñar e implementar un SO que va a utilizar una política deplanificación 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).

- Habría que modificar el planificador a corto plazo, que ahora se tiene que encargar de ordenar los procesos cuando llegan y de comprobar que si en la cola de listos hay algún proceso con mayor prioridad que el que se está ejecutando, debe expulsar a este para que entre el de mayor prioridad.

8. Para cada una de las siguientes llamadas al sistema explique si su procesamiento por parte del SO requiere la invocación del planificador a corto plazo (Planif_CPU()):

- *Crear un proceso, fork()*
    - Puede ser necesario cuando nada más llegar el SO decide que debe ejecutarse inmediatamente.
- *Abortar un proceso, es decir, terminarlo forzosamente, abort() .*
    - Sí, porque se libera la CPU.
- *Bloquear (suspender) un proceso, read() o wait() .*
    - Sí, ya que estaba en ejecución.
- *Desbloquear (reanudar) un proceso, RSI o exit() (complementarias a las del caso anterior).*
    - Depende del algoritmo de planificación:
        - Si es apropiativo, puede ser que al desbloquearlo se quiera ejecutar inmediatamente, en este caso si debe intervenir el planificador a corto plazo.
- *Modificar la prioridad de un proceso*
    - Sí, si al modificar la prioridad del proceso entra o sale de la ejecución, en caso contrario no.

9. En el algoritmo de planificación FCFS, el índice de penalización, (M+r)/r, ¿es creciente, decreciente o constante respecto a r (ráfaga de CPU: tiempo de servicio de CPU requerido por un proceso)?Justifique su respuesta.

- Índice de penalización: (M+r)/r
- La penalización es decreciente, es decir, cuanto más corto sea un proceso, mayor penalización. Esto es así porque los procesos cortos requieren poca CPU pero tienen que esperar mucho. (Sobretodo si delante de ellos tienen procesos largos.)

10. Sea un sistema multiprogramado que utiliza el algoritmo Por Turnos (Round-Robin, RR). Sea S el tiempo que tarda el despachador en cada cambio de contexto. ¿Cuál debe ser el

valor de quantum Q para que el porcentaje de uso de la CPU por los procesos de usuario sea del 80%?

- S ↪ Tiempo en cambio de contexto
- Q ↪ Quantum de tiempo
- % Uso de CPU: Q / (Q+S)
    - Q ↪ Tiempo que trabajo
    - Q+S ↪ Tiempo total de ejecución

$$\frac{Q}{Q+S} = 0.8 \iff Q = 0.8 \cdot Q + 0.8 \cdot S \iff 0.2 \cdot Q = 0.8 \cdot S \iff Q = 4 \cdot S$$