

Práctica D

Modificación del algoritmo de planificación de MINIX.

- Edite y lea las siguientes funciones asociadas al paso de mensajes y la planificación:
 - `src/kernel/proc.c:pick_proc(), ready(), unready(), sched()`.
- Edite y lea las siguientes funciones asociadas a la gestión del reloj (tarea de reloj y rutina de servicio de interrupción):
 - `src/kernel/clock.c:init_clock()`
 - `src/kernel/clock.c:clock_handler(), clock_task(), do_clocktick()`
 - `src/kernel/proc.c:interrupt()`
- **Visualización del efecto convoy cuando cambiamos el algoritmo de planificación de RR a FIFO:**
 - **Espacio de usuario:** prepare una carga de trabajo formada por 1 proceso que necesite entre 10 y 20 segundos de servicio y 3 procesos cuya ejecución se realice en alrededor de 1 segundo.
 - Una buena opción es realizar el cálculo de números primos. Uno de los procesos calculará los necesarios para tardar alrededor de 10 segundos y el resto los necesarios para tardar 1 segundo. Para conseguirlo utilice el conocido método del ensayo y error ya que el tiempo necesario dependerá del hardware subyacente, como a estas alturas ya estará imaginando.
 - Cada uno de los cuatro procesos únicamente imprimirá un mensaje al comenzar y otro justo antes de terminar. No imprima nada entre medias ya que lo que necesitamos es una larga ráfaga de cpu (sin e/s, por definición).
 - Los cuatro procesos deben ser lanzados, vía **fork()**, desde un único fichero *a.out*
 - Tenga la precaución de **lanzar el proceso largo en último lugar**. ¿Por qué? Excelente pregunta, no deje de buscar la respuesta. La respuesta está en el código.
 - **Espacio de sistema:** implemente una llamada al sistema que modifique el cuanto de tiempo de CPU.
 - La función **system.c:do_asops()** que ha realizado en la práctica anterior debe ahora ser modificada para convertirse en un **repartidor**. Esto es, una función que se encargue de llamar a otras funciones dependiendo del primer argumento recibido (*m_ptr->m1_i1*), enviando a dichas funciones el segundo y tercer argumento (*m_ptr->m1_i2*, *m_ptr->m1_i3*).
 - Para esta práctica invoque a una función llamada, por ejemplo, **cambiarQ(m_ptr->m1_i2)**

- Escriba **clock.c:cambiarQ(nq)**:
 - Puede que necesite declarar una nueva variable externa, por ejemplo, *nuevoQ*; hágalo sin problemas en el fichero clock.c. No olvide inicializar esta variable con la constante SCHED_RATE
 - Asigne **nq** a la nueva variable declarada la cuál a su vez será asignada a **sched_ticks**.
 - Es recomendable modificar aquí también la variable sched_ticks porque si el valor del cuanto elegido es muy grande puede ocurrir que nunca llegue a cero y por lo tanto no se vuelva a asignar un nuevo cuanto; por ello cuando se reduzca el valor del cuanto no se verá traducido a efectos prácticos.
- Una vez hechos los cambios en el espacio de sistema y, por supuesto, después de su compilación, deberá reiniciar Minix para que su nuevo núcleo pase a estar en uso.

○ **Experimentación:**

- Sin cambiar el valor del cuanto: ejecute su proceso de usuario (ya sabe, los 3 cortos más el largo, todos ellos iniciados desde un único a.out)
- Incremente el valor del cuanto para pasar de RR a FIFO no apropiativo. Para conseguirlo el cuanto debe ser mayor que el mayor tiempo de ráfaga de los procesos que está ejecutando. En nuestro caso el cuanto deberá ser mayor que el tiempo de ráfaga del proceso largo. Ejecute de nuevo su proceso de usuario. Deberá ver el efecto convoy.