Proyecto SO: Fase 3

César Bonilla Garrido
Sergio Bugallo Enjamio
Marcos Cela López

1. Introducción:

Nuestro proyecto tiene que cumplir 8 requisitos funcionales y 2 requisitos no funcionales:

-RF0: Se usarán 3 colas multinivel.
- ·Ya existen en Linux, no es necesario implementar nada a mayores.

-RF1: Las colas designadas a los procesos de pagos y anulaciones usarán FIFO.
- ·Cuando creamos el proceso, se envía el parámetro SCHED_FIFO.

-RF2: La cola designada para el resto de procesos propios usará RR.
- ·Cuando creamos el proceso, se envía el parámetro SCHED_RR.

-RF3: Se asignarán 4 niveles de prioridad a los distintos procesos.
- ·Para ello modificaremos el struct task struct* en sched.h

-RF4: Se utilizará una ventana de admisión para regular los procesos entrantes.
- ·Implementado en el gestor mediante la función habilitar().

-RF5: En las colas FIFO se usará un temporizador para evitar procesos bloqueados.
- ·Implementado en el gestor mediante la función buscar_bloqueados().

-RF6: En la cola RR se usará un número máximo de ciclos.
- ·Implementado en el gestor mediante la función buscar_bloqueados().

-RF7: Se empleará un algoritmo con apropiación.
- ·Implementado en el gestor mediante la función mata_proceso(..).

-RNF0: La herramienta se desarrollará mediante una herramienta de desarrollo libre.
-RNF1: La herramienta del sistema se desarrollará sobre un sistema linux.

2. Creamos el Gestor:

Para crear el gestor tendremos que crear un archivo .h (contiene las cabeceras de las funciones, así como las constantes que usaremos) y un archivo .c (contiene la definición de las funciones).

En el .h definimos:
Las 4 constantes que usaremos para asignar las distintas prioridades:

-PRI_PAGOS: para procesos de pagos.

-PRI_ANULACIONES: para procesos de anulaciones.

-PRI_OTROS: para procesos de eventos, gradas y pre-reservas.

-PRI_DEFECTO: para los procesos del sistema.(habrá que modificar el exec.c)

Definimos también las cabeceras de las funciones del gestor:

-int crear_procesos(struct task_struct*): crea un proceso nuevo.

-int habilitar(struct task_struct*): devuelve 0 si se puede crear el proceso o -1 en otro caso.

-int detectar_tipo_proceso(struct task_struct*): devuelve el tipo del proceso.

-int buscar_bloqueado(void): busca si hay algún proceso bloqueado en el sistema. Devuelve 0 si se ha matado algun proceso bloqueado, -1 en otro caso.

-int buscar_victima(int): busca un proceso que matar para dejar sitio a otro de mayor prioridad. Devuelve 0 si se ha matado algún proceso, -1 en otro caso.

-int mata_proceso(struct task_struct*): mata el proceso y decrementa en una unidad el valor de la variable que contiene número de procesos de ese tipo en el sistema

-void revisor(): comprueba el estado del sistema y se encarga de su mantenimiento;

-void actualizar_procesos(struct task_struct*): actualiza las variables que contienen el número de procesos de cada tipo.

En el .c definimos las funciones:

-int crear_procesos(struct task_struct*): creamos el proceso, y si no se puede crear, lo eliminamos.

-int habilitar(struct task_struct*):

-int buscar_bloqueado(void):

-int buscar_victima(int):

-void revisor():

-void actualizar_procesos(struct task_struct*):

3. Modificación de exit.c:

Modificamos la función void do_exit(long code) para que cuando termine de ejecutarse un proceso, se actualice el número de procesos propios en el sistema. Para ello simplemente añadimos:

```
actualizar_procesos(tsk);
```

4. Modificación de sched.h:

Dentro de struct task_struct introducimos nuestras variables

```
int tipo;
```

5. Modificación de exec.c:

Modificamos la función set_task_comm(struct task_struct *tsk, char *buf) para asignar los valores por defecto de los nuevos campos. Para ello añadimos:

```
crear_proceso(tsk);
```

6. Modificación de core.c:

Dentro de scheduler_tick, llamamos a la función revisor(), esta se ejecutará periódicamente para eliminar procesos bloqueados en el sistema.

7. Pruebas:

1-Ejecución de 2 procesos con la misma prioridad:Pagos.
Tenemos dos procesos de pagos, p1 y p2, en cola, si p1 fue el primer proceso en entrar en ella, será también el primero en atenderse. Para comprobarlo se mostrará el tiempo de uso de cpu de cada proceso.

2-Ejecución de 2 procesos con la misma prioridad:Gradas y Pre-reservas.
Tenemos dos procesos en cola, g1 (gradas) y pr1 (Pre-reservas), y que las colas con mayor prioridad están vacías, los procesos se atenderán siguiendo el algoritmo RR. Para comprobarlo se mostrará el tiempo de uso de cpu de cada proceso.

3-Ejecución de 2 procesos con diferente prioridad al sistema: Pagos y Pre-reservas.
Con el sistema libre de procesos, exceptuando un proceso en la cola de Pagos, p1, y uno en la de Pre-reservas, pr1, se atenderán según su nivel de prioridad: primero Pagos y después Pre-reservas. Para comprobarlo se mostrará el tiempo de uso de cpu de cada proceso.

4-Proceso de Pagos supera el umbral t.
Tenemos un proceso p1 que lleva en el sistema un tiempo t, el sistema matará al proceso y atenderá al siguiente. Para comprobarlo se imprimirá por pantalla el mensaje "El proceso px lleva x segundos el sistema, por lo tanto, se matará", siendo px el identificador(p1) del proceso.

5-Proceso de Pre-reservas supera x veces el cuanto (igualando el tiempo t).
Tenemos un proceso de pre-reservas pr1 que ha superado x veces el cuanto. El sistema matará al proceso y atenderá al siguiente. Para comprobarlo se imprimirá por pantalla el mensaje "El proceso pr1 lleva x segundos el sistema, por lo tanto, se matará".

6-Proceso de Pagos cuando se supera el umbral k pero no n de procesos en el sistema.
Tenemos un proceso de pagos p1 que quiere entrar al sistema y dentro de este ya hay 3 procesos de pagos. En este caso se admitirá al proceso. Para demostrarlo se imprimirá por pantalla "entra el proceso p1 de pago y el número de procesos en el sistema es x".

7-Proceso de Anulaciones cuando se supera el umbral k pero no n de procesos en el sistema.
Tenemos un proceso de Anulaciones a1 que quiere entrar al sistema y dentro de este ya hay 3 procesos de anulaciones. En este caso se no admitirá al proceso. Para demostrarlo se imprimirá por pantalla "no entra el proceso a1 de anulación y el número de procesos en el sistema es x".

8-Proceso de Pagos llega cuando se ha alcanzado el número máximo de procesos en sistema (n).
Tenemos un proceso de Pagos p1 que quiere entrar al sistema y dentro de este ya hay 5 procesos de pagos. En este caso se no admitirá al proceso. Para demostrarlo se imprimirá por pantalla "no entra el proceso p1 el número de procesos en el sistema es x".

9-Se está ejecutando un proceso de Gradas y llega un proceso de Anulaciones.
Tenemos 3 procesos de Gradas en el sistema, y llega un proceso de Anulaciones p1. Se matará un proceso de Gradas y se atenderá al de Anulaciones. Para ello imprimirá por pantalla "Se mata al proceso x por tener menor prioridad".

César Bonilla Garrido

Sergio Bugallo Enjamio
Marcos Cela López