Práctica PRO2

Otoño 2018

31 de octubre de 2018

1. Introducción

Queremos simular el sistema de gestión de tareas de una arquitectura mutiprocesador y multiusuario tipo NUMA, donde cada procesador trabaja exclusivamente con su propia memoria y puede ejecutar más de un proceso simultáneamente.

Tenemos un *cluster* con n procesadores distribuidos en forma de árbol, con identificadores $1, \ldots, n$. Hay un primer procesador con id cualquiera. Cada procesador tiene un máximo de dos procesadores inmediatamente siguientes, reconocibles como "siguiente por la izquierda" y "siguiente por la derecha" (aunque solo tenga uno).

En cada momento, hay un cierto número de usuarios conectados al sistema que quieren ejecutar procesos en el *cluster*. Un proceso a la espera de ser ejecutado se denomina *pendiente*. Posteriormente se explicará como los usuarios pueden enviar procesos al *cluster*. Un proceso en el *cluster* se denomina *activo*.

Cada usuario tiene un identificador formado por minúsculas y números, que comienza por una letra. Se pueden añadir y eliminar usuarios al sistema siguiendo unas ciertas normas.

La información de un proceso consiste en:

- un identificador de proceso (numérico)
- un identificador de usuario (anteriormente descrito)
- la cantidad de memoria reservada para el proceso, en posiciones de memoria
- el tiempo estimado que va a necesitar, en unidades enteras de tiempo.

Además el sistema permite enviar procesos directamente al *cluster*, indicando en qué procesador se han de ejecutar, sin pasar primero por el estado *pendiente*. El sistema también permite finalizar procesos activos en cualquier procesador en cualquier momento.

No hay id de procesos repetidos entre todos los procesos activos y todos los pendientes, aunque un mismo identificador se puede usar varias veces una vez que el proceso correspondiente ha acabado o se ha hecho terminar.

Un usuario tiene que saber en todo momento qué procesos pendientes tiene para poder enviarlos al *cluster* cuando tenga oportunidad. El usuario ha de poder saber en todo momento cual es la antigüedad relativa de sus procesos pendientes.

Un procesador tiene una cantidad de memoria en la que podrá guardar y ejecutar los procesos. Dicha memoria está compuesta de una cierta cantidad de posiciones, indexadas desde 0. Cada uno de los n procesadores puede tener un número diferente de unidades de memoria. Cada procesador ha de conocer qué procesos está ejecutando y qué posiciones de la memoria ocupan.

Un proceso se puede colocar en un procesador del *cluster* si dicho procesador contiene un hueco (posiciones libres consecutivas) mayor o igual que la memoria que necesita el proceso. Si existen varios huecos posibles, se selecciona el más ajustado con posición inicial más pequeña, y el proceso

ocupa las posiciones necesaria a partir de dicha posición. No hay que tocar la memoria de los procesos que ya estén en el procesador. La ocupación es siempre en posiciones consecutivas, nunca se recurre a fragmentar la memoria de un proceso.

Es posible que se intente enviar un proceso al *cluster* y que no sea aceptado en ese momento porque no haya memoria libre consecutiva suficiente en ningún procesador. Un proceso que se haya intentado colocar en el *cluster* sin éxito se devuelve a su usuario y pasa a ser el más reciente de los pendientes.

A medida que transcurre el tiempo, se irán eliminando de los procesadores los procesos que hayan acabado, quedando libre la memoria que ocupaban.

2. Operaciones

Los identificadores de procesador siempre estarán entre 1 y n. Los identificadores de proceso siempre serán enteros mayores que cero. La cantidad de memoria reservada para un proceso y su tiempo previsto también será mayor que cero. La memoria de cualquier procesador será un número mayor que cero de posiciones.

- `configurar_cluster`. Los datos siempre comenzarán por una configuración de *cluster*. Se leerá el número de procesadores, sus conexiones y la memoria de cada uno de ellos en orden creciente de `id`. Esta operación se puede realizar varias veces. Si se vuelve a configurar el *cluster* más adelante, los procesos activos se eliminan, pero los procesos pendientes de los usuarios se conservan para el siguiente *cluster* en el mismo estado que antes.

- `poner_usuario`. Se da un id de usuario. Este nuevo usuario no tiene procesos pendientes. Si ya existe no se hace nada.

- `quitar_usuario`. Se da un id de usuario y se elimina. Si no existe el usuario o le quedaban procesos pendientes no se hace nada.

- `poner_proceso_en_procesador`. Se leen el id de un procesador y los datos de un proceso. Si el usuario no existe no se hace nada. Si hay un hueco de memoria libre consecutiva en el que quepa el proceso, pasa a ejecutarse en dicho procesador y la memoria que use pasa a estar ocupada. Si el proceso no cabe, pasa a ser el proceso pendiente más reciente del usuario.

- `quitar_proceso_de_procesador`. Se lee el identificador del proceso y del procesador y si existe dicho proceso en dicho procesador la memoria que usaba pasa a quedar libre y el proceso desaparece. Si no existe el proceso en dicho procesador no se hace nada.

- `enviar_proceso_a_usuario`. Se leen los datos de un proceso y pasa a ser un proceso pendiente del usuario, el más reciente de todos.

- `consultar_usuario`. Se lee el identificador de un usuario. Se escribe cuantos procesos pendientes tiene y el identificador del más antiguo. Si no tiene procesos pendientes solo se escribe 0.

- `avanzar_tiempo`. Se lee un entero positivo que indica las unidades de tiempo que han transcurrido desde el momento inicial (al configurar el *cluster*) o desde la última vez que se avanzó el tiempo. Se eliminan todos los procesos activos que hayan acabado en ese intervalo de tiempo.

- `consultar_procesador`. Se lee el identificador del procesador. Se escribe por orden de inicio de memoria dicha posición y el resto de datos de cada proceso. El tiempo que se escribe es el tiempo que falta para que el proceso acabe, no el tiempo inicial de ejecución.

- `enviar_procesos_a_cluster`. Se lee un entero positivo n_p que indica cuántos procesos pendientes hay que enviar al *cluster*. Dichos procesos se envían siguiendo un sistema de turnos. Los turnos se organizan así: el primer turno le corresponde al proceso pendiente más antiguo del usuario con menor identificador que tenga más procesos pendientes; en cada turno posterior se aplica el mismo criterio, pero sin tener en cuenta los procesos que hayan sido devueltos en los turnos anteriores. Los turnos avanzan hasta que se hayan colocado n_p procesos o cuando no se puedan colocar más procesos, bien porque no queden procesos pendientes o bien todos los que quedan no se pudieron colocar cuando les tocó el turno. Evidentemente si un proceso no cabe cuando le toca el turno, con más razón no cabrá después puesto que ningún procesador libera memoria en está operación.

  Para poner un proceso en el *cluster* se escoge el procesador, de entre los que pueden aceptar el proceso, que tenga más memoria libre en ese momento. En caso de empate, se escoge el más cercano a la raíz, y si persiste el empate el de más a la izquierda.

- `acabar`. Sin datos. Cesa la ejecución del programa.

Para ver los detalles concretos conviene estudiar el juego de pruebas público, que aparecerá en breve.