Simulación del rendimiento de procesadores interconectados

Primavera 2023

Contenidos:

1. Introducción

Queremos simular el funcionamiento de una arquitectura multiprocesador, donde cada procesador trabaja exclusivamente con su propia memoria y puede ejecutar más de un proceso simultáneamente.

Tenemos un *clúster* con un número de procesadores variable, distribuidos en forma de árbol binario, con identificadores únicos. El primer procesador puede ser cualquiera de ellos. Cada procesador puede tener dos, uno o cero procesadores sucesores, también llamados auxiliares. Todo clúster ha de tener como mínimo un procesador.

Un procesador dispone de una memoria en la que guardará y ejecutará los procesos. Dicha memoria está compuesta de una cierta cantidad de posiciones, indexadas desde 0. Cada uno de los procesadores puede tener un número diferente de posiciones de memoria.

Independiente del clúster, se dispone de un área de procesos "pendientes", es decir, en espera de ser ejecutados. En cada momento puede haber procesos pendientes y procesos ejecutándose en el clúster (activos). El sistema permite enviar procesos directamente al clúster, indicando en qué procesador se han de ejecutar, sin pasar primero por el área de espera. También permite enviar al clúster procesos pendientes e introducir procesos pendientes nuevos. Por último, el sistema permite finalizar procesos activos en cualquier procesador en cualquier momento.

Si un proceso entra en el área de espera, se ha de indicar su prioridad. Existen varias prioridades diferentes designadas por strings. Las prioridades están ordenadas según su nombre: a menor nombre, mayor es la prioridad. Por ejemplo `"11"` es más prioritario que `"12"` y menos que `"10C"`.

La información relativa a un proceso consiste en su identificador, la cantidad de memoria estimada que va a necesitar para ejecutarse y el tiempo estimado de ejecución. No puede haber identificadores de procesos repetidos entre todos los procesos pendientes de una misma prioridad, ni en un mismo procesador.

Cada procesador ha de conocer qué procesos está ejecutando y qué posiciones de su memoria ocupan. Un proceso se puede colocar en un procesador si dicho procesador no contiene ya un proceso con el mismo identificador y si dispone de 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 necesarias 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.

El sistema incluye un mecanismo para gestionar el paso del tiempo, de forma que se sepa en todo momento cuanto tiempo le queda a cada proceso para acabar de ejecutarse. A medida que transcurra el tiempo, se irán eliminando del clúster los procesos que hayan acabado de ejecutarse, quedando libre la memoria que ocupaban.

2. Funcionalidades

2.1. Decisiones sobre los datos

Todos los identificadores de prioridades y de procesadores tienen el mismo formato: son strings que constan solo de letras y dígitos. Los identificadores de procesos son enteros no negativos.

La memoria de cualquier procesador será un número entero (de posiciones) mayor que cero. La cantidad de memoria reservada para un proceso y su tiempo previsto también serán enteros mayores que cero, así como el número de prioridades posibles de los procesos pendientes.

En todo momento, se leerán solo datos sintácticamente correctos y no será necesario comprobar dicha corrección. Por último, algunas funcionalidades tendrán restringidos los datos que pueden recibir, con el objetivo de simplificar su ejecución. Dichas restricciones se introducen con la expresión "Se garantiza que ..." y no se han de comprobar.

2.2. Programa principal: estructura y comandos

El programa principal comenzará inicializando un primer clúster sin procesos. A continuación inicializará un área de espera vacía, leyendo para ello un número inicial de prioridades $N > 0$ y los identificadores de N prioridades.

Terminadas las inicializaciones, se procesará una serie de comandos, incluido el comando `fin` que terminará la ejecución. La estructura general del programa principal será la siguiente:

```
inicializa un cluster
inicializa un area de procesos pendientes
lee comando;
while (comando != "fin") {
  procesa comando;
  lee comando;
}
```

Los comandos aceptados se describen a continuación. Todo ellos se presentan en dos versiones, una con el nombre completo y otra con el nombre abreviado. La sintaxis exacta de la entrada y la salida de cada comando se podrá derivar del juego de pruebas público del ejercicio creado en el Jutge para cada entrega.

1. `configurar_cluster`: lee los procesadores del clúster, sus conexiones y la memoria de cada uno de ellos. Si ya existía un clúster anterior, este deja de existir. Se garantiza que los identificadores de los procesadores no están repetidos. El comando admite la forma abreviada `cc`.

   Esta funcionalidad se ha de aprovechar para la inicialización del primer clúster.

2. `modificar_cluster`: lee un identificador p de un procesador del clúster y un nuevo clúster. Si p no existe, o tiene procesos en ejecución, o tiene procesadores

auxiliares, se imprime un mensaje de error. En caso contrario, el nuevo cluster se coloca en el lugar de p, haciendo crecer así el cluster original (si p era auxiliar de otro procesador, la raíz del nuevo cluster pasa a ser el correspondiente auxiliar de dicho procesador). Se garantiza que no habrá repetición de identificadores en el cluster modificado resultante. El comando admite la forma abreviada `mc`.

3. `alta_prioridad`: lee un identificador de prioridad. Si ya existe en el área de espera una prioridad con el mismo identificador, se imprime un mensaje de error. En caso contrario se añade esta prioridad al área de espera, sin procesos pendientes. El comando admite la forma abreviada `ap`.

4. `baja_prioridad`: lee un identificador de prioridad. Si no existe en el área de espera una prioridad con el mismo identificador o si tiene procesos pendientes, se imprime un mensaje de error. En caso contrario se elimina esta prioridad del área de espera. El comando admite la forma abreviada `bp`.

5. `alta_proceso_espera`: lee los datos de un proceso y un identificador de prioridad. Si no existe la prioridad en el área de espera o si ya había un proceso con ese identificador y esa prioridad, se imprime un mensaje de error. En caso contrario el proceso pasa al área de espera con dicha prioridad. El comando admite la forma abreviada `ape`.

6. `alta_proceso_procesador`: lee el identificador de un procesador y los datos de un proceso. Si no existe el procesador en el clúster o ya contiene un proceso con el mismo identificador o el proceso no se puede colocar en el procesador, se imprime un mensaje de error. En caso contrario, el proceso pasa a ejecutarse en dicho procesador y la memoria que usa pasa a estar ocupada. El proceso se coloca en la posición más pequeña que deje el hueco más ajustado, tal como se explica más arriba. El comando admite la forma abreviada `app`.

7. `baja_proceso_procesador`: lee el identificador de un procesador y de un proceso. Si no existe el procesador o el proceso no está en el procesador, se imprime un mensaje de error. En caso contrario se elimina el proceso del procesador. El comando admite la forma abreviada `bpp`.

8. `enviar_procesos_cluster`: lee un entero no negativo n y se intentan enviar al clúster n procesos pendientes. El comando admite la forma abreviada `epc`.

   Los procesos se intentan colocar en el clúster por orden de prioridad. Dentro de una misma prioridad, se intentan colocar primero los procesos más antiguos. Los intentos continúan hasta haber colocado n procesos en el clúster o hasta que no queden procesos pendientes o hasta que todos los que queden pendientes se hayan intentado colocar sin éxito.

   Si un proceso no se ha podido colocar, no tiene sentido volver a intentar enviarlo al clúster en la misma operación porque, dado que en ella no se libera memoria, seguirá sin caber en ningún procesador. Entonces se considera que ha sido rechazado por el clúster. Los procesos rechazados por el clúster vuelven al área de espera como si fueran nuevos, con la misma prioridad.

Si un proceso se puede colocar en más de un procesador, se elige el que disponga de un hueco más ajustado respecto a tal proceso. En caso de empate entre varios procesadores, se designa el que quede con más memoria libre. Si persiste el empate, se escoge el más cercano a la raíz del árbol de procesadores y si todavía es necesario desempatar, el de más a la izquierda.

9. `avanzar_tiempo`: lee un entero positivo t que indica las unidades de tiempo que han transcurrido desde la última vez que se avanzó el tiempo (o desde el momento inicial, si es la primera vez que se aplica). Se eliminan todos los procesos activos que hayan acabado en ese intervalo de tiempo. La ejecución de todos los procesos de todos los procesadores han progresado t unidades de tiempo. El comando admite la forma abreviada `at`.

10. `imprimir_prioridad`: lee un identificador de prioridad. Si no existe una prioridad con el mismo identificador en el área de espera se imprime un mensaje de error. En caso contrario, se escriben los procesos pendientes de dicha prioridad por orden decreciente de antigüedad, desde su última alta. Además, escribe el número de procesos de la prioridad colocados en el clúster por la operación `enviar_procesos_a_cluster` y el número de rechazos (un mismo proceso cuenta tantas veces como rechazos haya sufrido). El comando admite la forma abreviada `ipri`.

11. `imprimir_area_espera`: Escribe los procesos pendientes de todas la prioridades del área de espera por orden creciente de prioridad. Para cada prioridad se escribe lo mismo que en la funcionalidad anterior. El comando admite la forma abreviada `iae`.

12. `imprimir_procesador`: lee el identificador de un procesador. Si no existe un procesador con el mismo identificador en el clúster se imprime un mensaje de error. En caso contrario, se escriben los procesos de dicho procesador por orden creciente de primera posición de memoria, incluyendo 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. El comando admite la forma abreviada `ipro`.

13. `imprimir_procesadores_cluster`. Escribe todos los procesadores del clúster por orden creciente de identificador. Para cada procesador se escribe lo mismo que en la funcionalidad anterior. El comando admite la forma abreviada `ipc`.

14. `imprimir_estructura_cluster`. Escribe la estructura de procesadores del clúster. El comando admite la forma abreviada `iec`.

15. `compactar_memoria_procesador`: lee el identificador de un procesador. Si no existe un procesador con el mismo identificador en el clúster se imprime un mensaje de error. En caso contrario, se mueven todos los procesos hacia el principio de la memoria del procesador, sin dejar huecos, sin solaparse y respetando el orden inicial. El comando admite la forma abreviada `cmp`.

16. `compactar_memoria_cluster`: en cada procesador, se mueven todos los procesos hacia el principio de la memoria, sin dejar huecos, sin solaparse y respetando el orden inicial. El comando admite la forma abreviada `cmc`.

17. `fin`:

La única operación que modifica el tiempo es `avanzar_tiempo`. Se considera que cada una de las restantes operaciones se ejecuta en el instante de tiempo definido por la ejecución más reciente de `avanzar_tiempo` (o en el instante 0, si aún no se ha ejecutado ninguna vez).