Planificador distribuido

Juan Francisco Cardona Mc'Cormick

15 de abril de 2016

1. Objetivo

Esta práctica tiene como objetivo, el manejo de hilos, procesos y concurrencia.

2. Enunciado

Se desea modelar un sistema distribuido, que implemente dos tipos de planificadores: un planificador de largo plazo (`PLP`) y un planificador de corto plazo (`PLP`). Para simular dicho sistema, se utilizará el concepto de *Anillo de procesos* (Ver [2, pág. 139-166], ver [1, pág. 889-920]). Este es un conjunto de procesos que forma una topología de anillo, en la cual la salida estándar del proceso i, se conecta a la entrada estándar del proceso i + 1 y la salida estándar del proceso n - 1[1] está conectada a la entrada estándar del proceso 0, las conexiones se harán a través de tuberías.

En el anillo viajará un mensajes a través de todos los procesos. Este mensaje será recibido y modificado, sí así lo requiere, por cada proceso y una vez procesado será enviado al siguiente proceso. Este mensaje servirá para tres propósitos: en primer lugar, llevará la información sobre los procesos a ejecutar; segundo, servirá para determinar un mecanismo de exclusión mutua que controla el acceso de un recurso compartido, que será el error estándar en todos los procesos (estará rediridigido a la terminal, la consola, o un archivo) y tercero, como un mecanismo para recompilar la información sobre las tareas que han terminado.

[1] donde n es el número de máximo de procesos y estos se numeran desde 0

El planificador de largo plazo (PLP), será el proceso 0, y se encargará de generar un número limitado de tareas, que serán ejecutados por los planificadores de corto plazo[2] (PCP) y este se encarga de generar el mensaje que viajara por el anillo. El mensaje consta de dos partes, de las tareas que faltan por ejecutar y de las tareas ejecutadas. El PLP iniciará el envío del primer mensaje que contiene una lista de tareas (el número de lista de tareas tiene una cuota mínima y máxima, que se genera de forma aleatoria), dentro del mensaje se indica que tareas y la información de asignación de asignación de cada una[3]. Cuando el mensaje llega al PCP, cada uno de ellos tiene un conjunto de hilos que se realizará una de las tareas, el PCP mira que hilos tiene disponibles y que tareas hay disponibles en el mensaje, si hay tareas e hilos disponibles lanzará para cada hilo libre una tarea libre, marcará dicha tarea dentro del mensaje como asignado (el PCP debe tratar de ocupar todos sus hilos libres). Si una tarea ya ha sido terminada por un hilo dentro Una vez haya revisado el mensaje, lo pasa al siguiente proceso, hasta que le mensaje alcanza de nuevo el PCP. Una vez allí el PCP, eliminará las entradas asignadas, recolocará las tareas no asignadas en el mensaje y añadirá un conjunto nuevo de tareas que irán en el nuevo mensaje.

Cada hilo creará un nuevo proceso que ejecutará la tarea asignada, esperará que la tarea termine, una vez esta finalice, tratará de mostrar el estado de finalización de la tarea en el error estándar. Debido a que este es un recurso compartido, el proceso imprimirá la salida, únicamente cuando el mensaje se encuentra dentro del proceso que lo contiene.

3. Detalles

3.1. Formato del mensaje

El mensaje que se pasará entre los procesos, lleva el siguiente formato:

```
1 const int MAX_TAREAS = 256;
  const int MAX_TEXT_TAREA = 64;
3
  struct Tarea {
5    bool asignado;
     char tareaAEjecutar[MAX_TEXT_TAREA];
7    unsigned short procesoId;
```

[2] los procesos restantes

[3] Cuando el mensaje sale del PLP, ninguna tarea habrá sido asignada

```
    unsigned short hiloId;
};

struct Estadistica {
    char tareaAEjecutar[MAX_TEXT_TAREA];
    unsigned short procesoId;
    unsigned short hiloId;
};

struct Mensaje {
    unsigned nTareas;
    unsigned nEstadisticas;
    Tarea tareas[nTareas];
    Estadistica estadisticas[nEstadisticas];
};
```

En la estructura `Tarea` el campo asignado indica si una tarea ha sido asignada (`False`-indica si no ha sido asignada y True si lo ha sido). `tareAEjecutar`, es la tarea que será lanzada por el hilo dentro del `PCP`, si la tarea y sus parámetros ocupa menos de MAX_TEXT_TAREA, los restantes carácteres deben valores nulos. procesoId es el número del proceso dentro del anillo de procesos (0 a 19). hiloId es el número de hilo dentro del proceso (0 a 19). nTareas es el total de tareas que hay dentro del mensaje. tareas es un vector de estructuras tarea de tamaño nTareas.

En la estructura

3.2. Programas

La practica debe desarrollar inicialmente tres programas base: `PCP`, `PLP` y `planificador`. Cada programa tiene una funcionalidad descrita anteriormente, pero cada programa tiene una forma de ejecución particular. Las tareas son en

3.2.1. `PCP`

El planificador de proceso de corto plazo tiene las siguiente lista de comandos de ejecución.

```
pcp -i nProceso [-t nHilos]
```

Donde, `-i` y `nProceso` indica el identificador del proceso dentro del anillos de procesos. `-t` y `nHilos` indica el número total de hilos para tareas dentro del `PLP`, tiene valor por omisión 3 y puede tener un valor máximo de 10.

3.2.2. `PLP`

El comando de ejecución `PLP` es el siguiente:

```
$ plp
```

El `PLP` se encarga de generar tareas para ejecutar el número total de tareas que se genera en cada mensaje es un valor aleatorio entre 3 y 255 tareas. Cuando se genera el total de tarea, estas se ponen en el mensaje y al recibir de nuevo el mensaje, este verifica cuantas hay independientes es menor crear el nuevo conjunto de tareas que sea igual al total a generar.

3.2.3. `planificador`

Se encarga de construir el anillo de procesos, conectar el `PLP` con los `PCP`, a través de las tuberías y crear cada uno de los `PCP` que el usuario indique y crear el `PLP` también, también de conectar el error estándar de todos los procesos a una misma salida.

El comando de ejecución del planificador es el siguiente:

```
planificador [-l <plpname>] -n N [-t npcp totalHilos ...]
```

Donde `-l` indica el nombre del planificador de largo plazo `<plpname>`, por omisión el nombre del planificador de largo plazo es `plp`. `-n` indica el número total de procesos (`N`). `-t` indica la información relativa a cada `PLP` de ejecucion, primero se indica el número de `PCP` (`npcp`) y luego el total de hilos en dicho `PCP` (`totalHilos`).

En el siguiente ejemplo se muestra un ejemplo del planificador lanzando 5 procesos de corto plazo con el planificador de largo plazo por omisión:

```
$ planificador -n 5 -t 0 3 -t 4 2 -t 2 1 -t 1 2 -t 3 2
```

Los 5 planificadores de corto plazo: $\texttt{pcp}_0$ tiene 3 hilos; $\texttt{pcp}_1$ tiene 2 hilos; $\texttt{pcp}_2$ tiene un hilo; $\texttt{pcp}_3$ tiene 2 hilos y $\texttt{pcp}_4$ tiene 2 hilos.

En el siguiente ejemplo se muestra un ejemplo de planificador lanzando 6 procesos de corto plazo con el planificador de largo indicado en la línea de comando:

```
$ planificador -l plpprofesor -n 6 -t 1 4 -t 5 5
```

El nombre del proceso planificador de largo es `plpprofesor`, se crean 6 procesos de corto plazo, solo dos de ellos son indicados explícitamente el número de hilos, el siguiente es el resumen de los proceso de corto plazo: $\texttt{pcp}_0$ tiene 3 hilos; $\texttt{pcp}_1$ tiene 4 hilos; $\texttt{pcp}_2$ tiene 3 hilos; $\texttt{pcp}_3$ tiene 3 hilos y $\texttt{pcp}_4$ tiene 3 hilos y $\texttt{pcp}_5$ tiene 5 hilos.

3.3. Error Estándar

Todos los procesos (a excepción del `planificador`) deben tener connectada su error estándar a la misma salida.

3.4. Tareas

Las tareas son los nombres ejecutables que serán ejecutados por cada `PCP`. Estos se encuentra ubicados en el directorio definido por la variable de ambiente `PLP_DIR_TAREAS` (ver 3.5.1). Los nombres de las tareas comienzan con el siguiente nombre: `tarea` y son diferenciados por la siguiente secuencia: `01`, `02`, `03`, `04`, `05` y `06`.

Las tareas son programas que simplemente se ejecutan por un tiempo determinado y retorna un código de error dependiendo del tipo. Existen 6 tipos de tareas como se observa en la figura 1.

Duración	**Retorno**	**Descripción**
1 a 5 segundos	0	Un proceso corto que siempre funciona correctamente
3 a 10 segundos	1	Un proceso corto que siempre falla
20 a 30 segundos	0	Un proceso “mediano” que siempre funciona correctamente
15 a 40 segundos	1	Un proceso “mediano” que siempre falla
45 a 65 segundos	0	Un proceso “largo” que siempre funciona correctamente
50 a 70 segundos	1	Un proceso “largo” que siempre falla

Figura 1: Descripción de las tareas según duración y valor de retorno

3.5. Variables de ambiente

Existen dos variables de ambiente que deben ser definidas para los planificadores puedan funcionar:

3.5.1. `PLN_DIR_TAREAS`

Esta variable de ambiente indica que el directorio donde están instalas las tareas.

3.5.2. `PLN_DIR_PLP`

Esta variable de ambiente indica que el directorio donde están instalados los planificadores de largo plazo.

3.5.3. `PLN_DIR_PCP`

Esta variable de ambiente indica que el directorio donde están instalados los planificadores de corto plazo.

3.6. Concurrencia

La práctica maneja concurrencia dentro de los procesos `PLP`, ya que en un momento dado tendrá variables para manejar el número total de hilos y cuales de ellos está ocupados, esto ímplica un manejo de concurrencia de manera explícita.

3.7. Implementación

La práctica, será implementada en el sistema operativo Linux. Debe ser implementada utilizando el lenguaje de programación C++[4]. Fecha final de entrega: Viernes 13 de mayo 2016 a las 18:00.

3.8. Sugerencias

- Lea, análice y asimile la prática.

[4]La práctica debe ser diseñada e implementada utilizando los conceptos orientados a objeto

- Diseñe cada una de las partes.
- Revise los puntos donde puede haber concurrencia.
- Diseñe la estrategia para manejar la concurrencia.
- No trate de hacer la prática de una sola vez.
- Aproveche el tiempo, la práctica tiene muchos detalles.

4. Requisitos

4.1. Requisitos técnicos

4.1.1. Repositorio

- Manejador de versiones git.
- Se debe utilizar el repositorio Bitbucket. Uno de los estudiantes creará el repositorio y lo compartirá con los demás estudiantes, con el profesor (usuario `fcardona`) y el monitor.
- El nombre del repositorio debe ser `257s16<nombreproyecto>`. Donde `<nombreproyecto>` es un nombre sin espacios para identificar de manera única el respositorio.
- El profesor y el monitor tendrán permisos de lectura y escritura del respositorio (no pueden tener permisos de dueño).

4.1.2. Manejador de proyectos

Se utilizara el manejador de proyectos por excelencia en C++ y C que es make.

4.1.3. Jerarquía de directorios

La siguiente es la jerarquía de directorios y ficheros que el proyecto debe contener.

```
+-- <nombreproyecto>
  |-- Install.txt
  |-- Makefile
  |-- Readme.txt
  |-- miembros.xml
  +-- bin
  +-- examples
  +-- src
```

`<nombreproyecto>` directorio del proyecto. `Makefile` fichero del constructor del proyecto. `Readme.txt` información sobre el proyecto: bibliotecas, herramientas que deben ser instaladas, miembros del grupo, etc. `Install.txt` para instalar en un directorio distinto al local. `src` directorio fuente. `bin` donde van a quedar los binarios (no se debe subir binarios al repositorio). `examples`, fichero de configuración muestras, etc.

4.2. Otros requisitos

1. Máximo tres integrantes del mismo grupo por equipo.
2. El trabajo debe ser en equipo y no entre-equipos.
3. Todo fraude en la práctica será sancionado conforme al reglamento de la universidad.
4. Debe ser hecho seleccionando al menos uno de los lenguajes de programación: C ó C++. Si por alguna razón utilizan los dos lenguajes en un mismo proyecto tenga en cuenta que todas la bibliotecas de C, están incluidas en C++ y que la biblioteca `stdio` de C y `streams` C++, no deben ser mezcladas sin que respectivamente se le indique a la biblioteca de C++ que están haciéndolo.
5. La entrega se hará a través del repositorio.
6. *No hay entregas posteriores, todas las entregas deberán hacerse el mismo día*
7. Entrega: Viernes 13 de Mayo de 2016, hora de entrega 18:00 pm.

5. Bibliografía

Referencias

[1] Michael Kerrish. The Linux Programming Interface: A Linux and Unix System Programming Handbook. No Starch Press. 2010.

[2] Kay A. Robbins, Steven Robbins. Unix Programación práctica. Prentice Hall. 1997.