Sistema de control de procesos reencarnantes

Juan Francisco Cardona Mc'Cormick

14 de Marzo de 2012

1. Introducción

Los sistemas de control de trabajos (*Job control* `http://en.wikipedia.org/wiki/Job_control`), permite la gestión de múltiples trabajos de control teniendo como objetivo el adecuado manejo de recursos para impedir situaciones indesables como bloqueos mutuos (*deadlocks*).

Existen diferentes maneras diseñar e implementar los sistemas de control, uno de los más reconocidos es de lenguaje de control de trabajos (*Job control language JCL*). En este sistema se asume que los procesos llevan un vida ordenada: nacen, procesan y mueren, en una única vida.

En esta práctica vamos a cambiar el modelo de vida de los procesos y le vamos a permitir que los procesos vuelvan a vida según un número de veces predefinidas.

2. Modelo del sistema de control

En la figura 1 se observa la arquitectura del sistema de control propuesto. En este sistema de control cambiaremos el modelo de manejo de procesos de los sitemas de control tradicionales. El proceso llamado *consola de control* es el proceso principal que se encarga de iniciar todo el sistema de control. La configuración de arranque del sistema de control es obtenida al leer el archivo de configuración que indica, la cantidad y permanencia de los *procesos suicidas* que serán controlados.

Una vez leído el archivo de configuración e identificado cada programa a controlar, se iniciará un hilo de consola por cada proceso a controlar; este a su vez se encargará de crear un *proceso controlador*, indicandole explícitamente

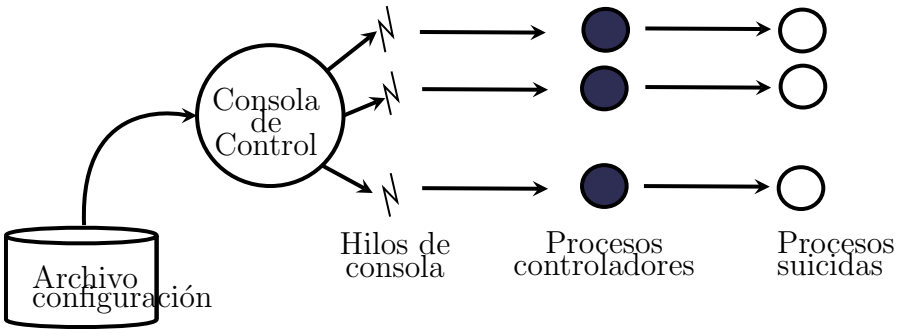

Figura 1: Modelo del sistema de control

cuál es el *proceso suicida* a controlar y por cuántas veces debe mantenerlo vivo.

2.1. Componentes

2.1.1. Archivo de configuración

El archivo de configuración describe la información pertinente al arranque del sistema. La sintaxis del archivo de configuración es la siguiente:

```
ArchCfg     → ϵ
            |  ProcesoSui ArchCfg
ProcesoSui →  'ProcesoSui' Id '{' Path '::' Filename Número'}'
```

Cada proceso es identificado de forma única con `Id` (un identificador similar a los Java); `Path`[1] identifica la ruta donde está ubicado el ejecutable `Filename`). La estabilidad de cada proceso es identificada a través de `Número` entero positivo donde 0 significa que vive por siempre y $n > 0$ es el valor del número de veces que debe vivir.

El siguiente es el contenido de un posible archivo de configuración:

```
ProcesoSui primerSuicida { /home/fcardona/bin :: ProcesoSuicida 10 }
ProcesoSui eternoSuicida { /usr/local/bin :: TendenciasSuicidas 0 }
```

[1]El separador es `/`, es independiente del sistema operativo y se debe hacer la traducción.

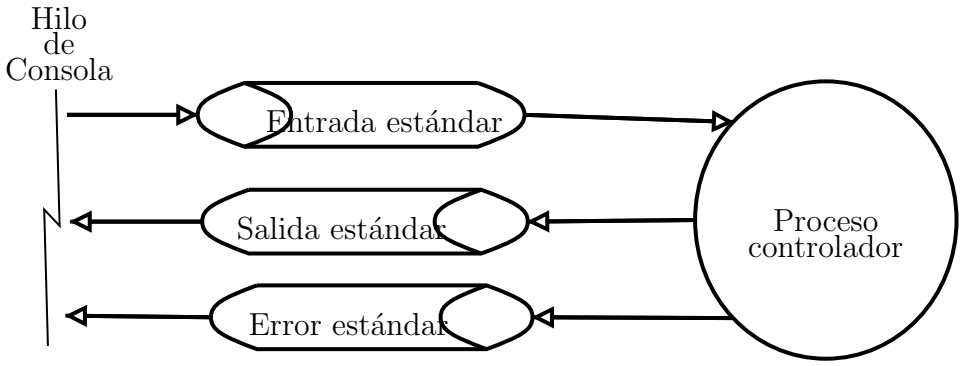

Figura 2: Conexión *hilos de control* con un *proceso de control*

El proceso identificado con `primerSuicida` está ubicado en el directorio `/home/fcardona/bin` y el nombre del archivo es `ProcesoSuicida` y se ejecutará 10 veces. El segundo proceso suicida identificado con `eternoSuicida` está ubicado en el directorio `/usr/local/bin` y tiene como nombre de archivo `TendenciasSuicidas` y lo intentará eternamente.

2.1.2. Consola de control

Es el encargado de leer el archivo de configuración interpretarlo y crear el sistema de control completo con respecto al archivo de configuración. El nombre del ejecutable es `conctrl`[2].

2.1.3. Hilos de consola

Son los encargados de comunicarse con los programas *procesos controladores*. Ellos se encargan de mostrar los mensajes enviados por los procesos controladores en la terminal donde la consola de control corre.

Cada *hilo de consola* se conecta directamente con un *proceso controlador* utilizando tuberías (*pipes*) como la figura 2 lo muestra. Se puede observar que el hilo de control se comunica por medio de tres tuberías que conectan con los respectivas entrada, salida y error estándar de este proceso.

El *hilo de consola* se comunica con el proceso de controlador por medio

[2]Sin la extensión

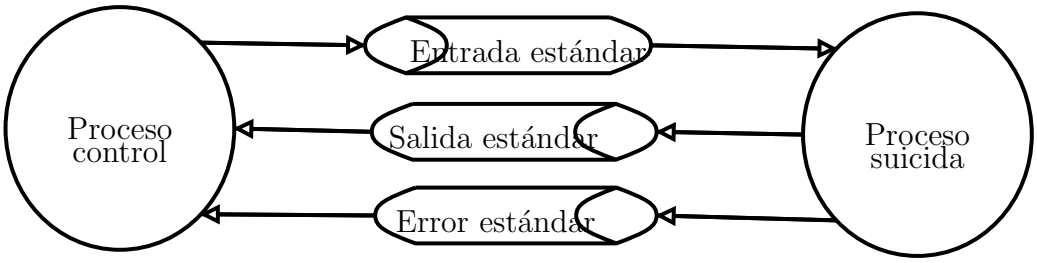

Figura 3: Conexión proceso de control con el proceso suicida

de la entrada estándar para enviar comandos[3]. Recibe las respuestas del proceso controlador[4] y las imprime en la terminal asociada a la consola de control, lo mismo sucede con los mensajes enviados por el error estándar. Todos los mensajes que son recibidos por parte del proceso controlador son inmediamente recibidos e impresos.

2.1.4. Proceso de control

El proceso de control se encarga de controlar un proceso suicida, cada vez que este proceso suicida termine por la razón que sea debe informar a su respectivo hilo de control la causa del desceso a través de la salida estándar:

```
Proceso suicida xxxx termino por causa 0   -- Proceso Control 10, vidas restantes: 3
Proceso suicida yyyy termino por causa 240 -- Proceso Control 11, vidas restantes: Infinitas
```

Luego de informar debe reiniciar el proceso suicida decrementando el valor de vidas restantes que le queda hasta que llegue a cero, en cuyo caso, el proceso de control de termina, informando que su labor fue llevada a cabo.

Cada proceso de control está identificado con un número único (además del PID[5]).

Para controlar los procesos suicidas los procesos de control están conectados completamente a ellos a través de tuberías como la figura 3 muestra:

Los procesos de control tiene la siguiente línea de comandos para iniciar:

```
procesoctrl --filepath=<path del ejecutable>
```

[3]En esta entrega no están definidos

[4]En esta entrega no están definidos.

[5]Process Id

```
--filename=<nombre del ejecutable>
--reencarnacion=<número de reencarnaciones>
<número del proceso de control>
```

2.1.5. Procesos suicidas

Los procesos son un grupo de programas que siempre que los inicien terminarán por las más diversas razones.

3. Requerimientos adicionales

3.1. Grupo

1. Grupos de dos estudiantes. Cualquier grupo individual tiene la misma carga de trabajo que la del grupo de dos estudiantes.

2. Cada grupo debe escoger un nombre que identifique al grupo, este nombre utiliza la misma sintaxis que los identificadores en Java.

3. Deben crear un proyecto privado en el manejador de repositorios `https://bitbucket.org` con el mismo nombre del grupo escogido e invitando al profesor a participar del proyecto.

4. Los miembros pueden ser de cualquiera de los grupos

3.2. Lenguajes de programación

Los lenguajes aceptados para la práctica son C o C++. Se puede llamar funciones de C desde C++, pero en el sentido inverso, al hacerlo recuerde identificar correctamente el espacio de nombres de las funciones de C.

3.3. Sistemas operativos

La práctica debe ser realizada en los sistemas operativos Windows (A partir de Windows Vista) y Linux (cualquier distribución). En Windows debe llamar a las funciones de subsistema Win32 (Win64). En Linux a las llamadas al sistema correspondiente.

3.4. Estructura del repositorio

La siguiente es la estructura de los directorios que tener el respositorio para manejar el proyecto.

```
+
|= - miembros.xml
|  + src
|  += + Windows
|  += + Linux
|  += + Common
|  - MakeLinux
|  - MakeWindows.mk
|  - Readme
|  - Install
```

`miembros.xml` archivo que contiene la definición de los miembros del grupo. El siguiente es el contenido de un posible grupo.

```
<grupo>
   <nombre>ElectrochoquesCtrlSA</nombre>
   <repositorio>https://bitbucket.org/CtrlEsquizofrenicos</repositorio>
   <miembro>
      <nombre>Juan Francisco Cardona McCormick</nombre>
      <codigo>198610250010</codigo>
      <grupo>031</grupo>
      <email>fcardona@eafit.edu.co</email>
   </miembro>
   <miembro>
      <nombre>Alejandro Gómez Londoño</nombre>
      <codigo>201010001010</codigo>
      <grupo>033</grupo>
      <email>alegomez544@eafit.edu.co</email>
   </miembro>
</grupo>
```

3.5. Primera entrega

16 de Abril a las 8:00 a.m. hora oficial de Colombia. En ese mismo instante un procedimiento automático bajara los repositorios correspondientes. Deben actualizar sus repositorios antes. Grupo que no haya invitado al profesor no será tenido en cuenta.

3.6. Práctica de concurrencia

La práctica está divida en dos partes: manejo de la salida y el error estándar en la consola de control; y la gestión de estadísticas.

3.6.1. Salida concurrente

El diseño de la consola de control tiene problemas, por que al utilizar hilos, estos comparten el envio de mensajes a la salida y el error estándar simúltaneamente, haciendo que en algunas ocasiones ocurra condiciones de concurso para mezclar la salida de dos o más hilos.

Esto debe evitarse utilizando semáforos, puesto que solamente un hilo puede escribir, ya sea en la salida estándar o en el error estándar.

3.6.2. Estadísticas

El Gobierno Nacional y el DANE requieren de estadísticas de la mortandad de los procesos suicidas. Se requiere saber el número de veces exactas que el proceso suicida ha muerto en cualquier instante de tiempo.

Para lograr esto vamos a realizar varios cambios a nuestra implementación.

1. Los encargados de llevar a cabo la recolección de la información estadística serán los procesos controladores.

2. Todos los procesos controladores compartirán una región de memoria donde se guardará la información estadística.
   Esto implica que el proceso controlador se le añade una opción a la línea de comandos.

   ```
   procesoctrl --filepath=<path del ejecutable>
               --filename=<nombre del ejecutable>
               --reencarnacion=<número de reencarnaciones>
               --idMemoria=IdentificadorMemoriaCompartida
               --idSemaforoMemoria=IdentificadorSemaforo
               <número del proceso de control>
   ```

   El `idMemoria` identifica la región de memoria donde que comparte todos los procesos controladores y `idSemaforoMemoria` es el identificador

del semáforo que se encargará de controlar el acceso a la memoria compartida.

3. La memoria compartida se tendrá la siguiente estructura de datos:

```
struct MemoriaCompartida {
   int n; // Número de procesos controladores
   long int valSeq;
   struct InfoMuerte muertes[n]; // Cada entrada identifica la información
                                 // de cada proceso suicida.
};
```

4. La estructura donde se almacena la información de los decesos es la siguiente:

```
struct InfoVida {
   long int seq;
   int nDecesos;
};
```

   El valor de `seq` se obtiene de incrementar por cada proceso de control el valor de `valSeq`. Cada vez que un proceso suicida muere cada proceso controlador debe incrementar el valor de `valSeq` y incrementar también el número de muertes de que lleva su suicida a cargo y registrar en el campo correspondiente la información actualizada.

3.6.3. Entrega

1. Cada equipo trabajará está práctica bajo el sistema operativo Linux.
2. El objetivo de la práctica es implementar la parte estadística.
3. Una vez implementada la parte estadística puede implementar el control de salida y error estándar. Esto 0.8 unidades en cualquier parcial. No es acumulable.
4. Durante el fin de semana, deben registrar la práctica en el repositorio. Traten de hacer acualización muy frecuentemente.
5. Hora limite última actualización. Lunes 30 de Abril, 8:00 am.