Sistema de control de procesos reencarnantes

Juan David Pineda Cárdenas
Juan Francisco Cardona Mc'Cormick

24 de Octubre de 2016

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 indeseable 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 ésta práctica vamos a cambiar el modelo de vida de los procesos y vamos a permitir que los procesos vuelvan a vida según una política que determine el número de veces que vivirá.

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 sistemas 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 fichero de configuración que indica la cantidad y permanencia de los *procesos suicidas* que serán controlados.

Una vez leído el fichero de configuración e identificado cada programa a controlar, se iniciará un hilo de consola por cada proceso a controlar; este a su

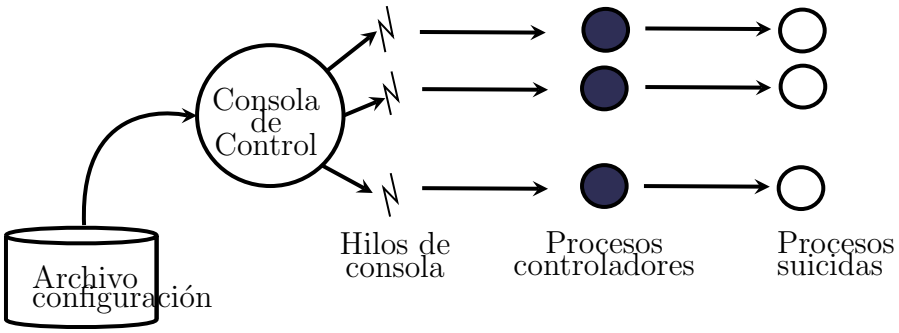

Figura 1: Modelo del sistema de control

vez se encargará de crear un *proceso controlador*, indicándole explícitamente cuál es el *proceso suicida* a controlar y por cuántas veces debe mantenerlo vivo.

2.1. Componentes

2.1.1. Fichero de configuración

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

```
FichCfg     → ϵ
            |  ProcesoSui FichCfg
ProcesoSui →  'ProcesoSui' ID '{' RUTAFICHERO '::' Fichero  NUMERO'}'
```

Cada proceso es identificado de forma única con `ID` (un identificador similar a los Java); `RUTAFICHERO`[1] identifica la ruta donde está ubicado el ejecutable `FICHERO`). La estabilidad de cada proceso es identificada a través de `NUMERO` 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 fichero de configuración:

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

[1]El separador es `/`.

El proceso identificado con `primerSuicida` está ubicado en el directorio `/home/fcardona/bin` y el nombre del fichero 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 fichero `TendenciasSuicidas` y lo intentará eternamente.

2.1.2. Consola de control `conctrl`

`conctrl` es el encargado de leer el fichero de configuración interpretarlo y crear el sistema de control completo con respecto al fichero de configuración. El nombre del ejecutable `conctrl`.

La sinopsis del programa es la siguiente:

```
conctrl [--ficheroconfiguracion=<rutaficherocfg>]
        [--semaforo=<id>]
        [--memoriacompartida=<id>]
```

La opción `ficheroconfiguracion` indica la ruta y el nombre del fichero de configuración, si ésta opción no es indicada, el nombre del fichero de configuración por omisión es `conctrl.cfg`. La opción `semaforo` indica el identificador del semáforo que será utilizado para controlar la memoria compartida, si ésta opción no es indicada, el nombre del fichero de configuración por omisión es `conctrlsem`. La opción `memoriacompartida` indica el identificador del segmento de memoria compartida que será utilizada para comunicar la consola de control `conctrl` y los procesos de control `procesoctrl` (ver 2.1.4), si ésta opción no es indicada, el nombre del fichero de configuración por omisión es `conctrlmem`.

Una vez iniciada la consola esta se queda en modo de comando (Si `conctrl` reside en el directorio `bin`)

```
$ bin/conctrl
conctrl>
```

En el modo comando acepta los siguientes comandos:

```
ConctrlComando  → EOF
                | EOL
                | Comando EOL
Comando → 'listar' IdProcCtrl
        | 'sumar' IdProcCtrl  NUMERO
        | 'restar' IdProcCtrl NUMERO
        | 'suspender' IdProcCtrl
        | 'restablecer' IdProcCtrl
        | 'indefinir' IdProcCtrl
        | 'definir' IdProcCtrl NUMERO
        | 'terminar' IdProcCtrl
IdProcCtrl → '*'
           | ID
```

`IdProcCtrl` indica cual proceso de control `procesoctrl` se le hará algún comando, hay dos posibilidades: Todos `'*'` o un `procesoctrl` particular (`ID`).

Los comando son los siguientes. Fin de entrada: `EOF` termina la ejecución de la `conctrl` y todos los `procesoctrl` (Es como si ejecutara el comando `terminar con todos los ficheros`). Ignorar: `EOL` un comando vacío. Listar (`listar`) los `procesoctrl` que se encuentra activos y sus valores. Sumar (`sumar`) a los `procesoctrl` un valor `NUMERO` al número de reencarnaciones, si el valor es infinito no es afectado por la suma. Restar (`restar`) a los `procesoctrl` un valor `NUMERO` al número de reencarnaciones, si el valor es infinito no es afectdo por la resta. Suspender (`suspender`), suspende la ejecución de `procesoctrl`, es decir que no se ejecutarán más procesos reencarnantes hasta un comando `restablecer`. Restablecer (`restablecer`) a los procesos `procesoctrl` que fueron suspendidos por un comando `suspender`. Indefinir (`indefinir`), establece que los procesos suicidas tendrán un valor infinito. Definir (`definir`), define que un proceso tenga un valor `NUMERO` de reencarnaciones. Terminar (`terminar`), termina `procesoctrl`, si todos los procesos de control `procesoctrl` son terminados, esto implica que la consola de control (`conctrl`) no funcionará más.

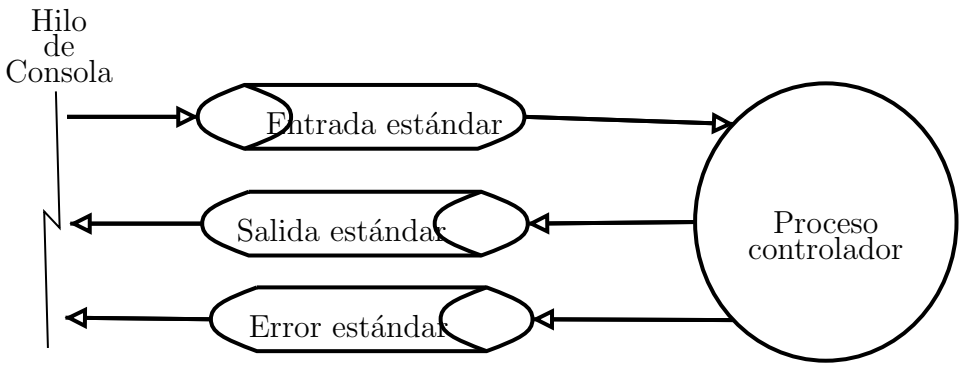

Figura 2: Conexión *hilos de control* con un *proceso de control*

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 de la entrada estándar para enviar comandos[2]. Recibe las respuestas del proceso controlador[3] 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 `procesoctrl`

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 deceso 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
```

[2]En esta entrega no están definidos

[3]En esta entrega no están definidos.

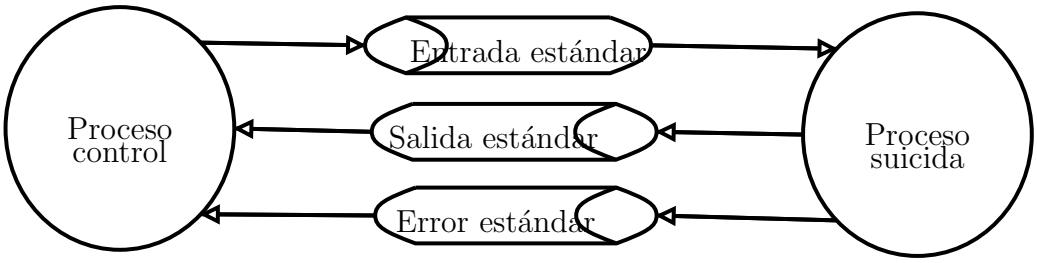

Figura 3: Conexión proceso de control con el proceso suicida

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 `ID` del proceso que se obtiene del fichero de configuración.

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=<rutafichero>
            --filename=<fichero>
            --reencarnacion=<numero>
            [--memoriacompartida=<id>]
            [--semaforo=<id>]
            <número del proceso de control>
```

Donde, <`rutafichero`> indica la ruta de camino donde está ubicado el ejecutable. <`fichero`> nombre del fichero ejecutable (proceso suicida). La opción `reencarnacion` indica el número de reencarnaciones del sistema. La opción `memoriacompartida` indica el `id` que la memoria compartida tendrá (por omisión: `conctrlmem`). La opción `semaforo` indica el `id` que el semáforo de control tendrá (por omisión: `conctrlsem`. El manejo de la memoria compartida y el semáforo se explicará en la sección 2.2.

2.1.5. Procesos suicidas

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

2.2. 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.

2.2.1. 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.

2.2.2. 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 tiene opciones de control particular (ver 2.1.4). Donde 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 InfoMuerte {
   string id;
   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. Requerimientos adicionales

3.1. Grupo

1. Grupos de dos estudiantes. Cualquier grupo menor 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++, 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[4]

[4]Esto implica también para la implementación de C#.

3.3. Sistemas operativos

La práctica debe ser realizada en el sistema operativos Linux (cualquier distribución).

3.4. Estructura del repositorio

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

```
+
|= - miembros.xml
|  + bin
|  + src
|  - Makefile
|  - Readme
|  - Install
|  + examples
```

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

```
<grupo>
   <nombre>CtrlEsquizofrenicosSA</nombre>
   <repositorio>https://bitbucket.org/CtrlEsquizofrenicosSA</repositorio>
   <miembro>
      <nombre>Juan Francisco Cardona McCormick</nombre>
      <codigo>198610999010</codigo>
      <grupo>031</grupo>
      <email>fcardona@eafit.edu.co</email>
   </miembro>
   <miembro>
      <nombre>Juan David Pineda Cárdenas</nombre>
      <codigo>200110999010</codigo>
      <grupo>032</grupo>
      <email>jpineda2@eafit.edu.co</email>
   </miembro>
</grupo>
```

`bin`. Es un directorio que es creado la primera vez que se compila, no hace parte del control de versiones.

`src`. Es un directorio donde se encuentra todos los fuentes del programa.

`Makefile`. Es un fichero para el manejo de proyecto. Este fichero tiene los comandos para compilar todo el proyecto. Esta es la lista de objetivos esperados[5]:

- `clean`. Limpia todo el proyecto.
- `all`. Genera todo el proyecto.
- `init`. Inicializa el directorio `bin`.

`Readme`. Es un fichero que tiene las instrucciones de último momento con el fichero. Como bibliotecas que se necesitan, configuración especiales, variables de entorno necesarias para los programas funcionen, etc.

`Install`. Es un fichero que tiene las instrucciones para instalar el programa en algún directorio particular.

`examples`. Es un directorio donde se tienen ejemplos de los ficheros de configuración que se han probando el programa.

3.5. Entrega

14 de Noviembre del 2016 hora oficial de Colombia. En ese mismo instante un procedimiento automático bajara los repositorios correspondientes. Deben actualizar sus repositorios antes de la hora. Grupo que no haya invitado al profesor no será tenido en cuenta.

3.5.1. 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.

[5]Existen otros objetos que son definidos por el diseño propio de la solución a ésta práctica