Sistemas Operativos

Trabajo Práctico N°1

Comunicación Interproceso

22 de Agosto de 2018

Agustina Osimani 57526

Nicolás Barrera 57694

Ezequiel Keimel 58325

Marcos Lund 57159

Estructura General

La solución fue desarrollada para Linux y cualquier otro sistema operativo que implemente las llamadas de sistema definidas en el estándar POSIX.

La funcionalidad de la aplicación se distribuye entre un proceso maestro, un proceso vista de ejecución opcional, y *S* procesos esclavos, instanciados por el maestro. El número *S* de esclavos es el resultado de la función siguiente, donde *n* es el número de archivos válidos a procesar:

$$S(n) = \begin{cases} \frac{n}{10} & \text{si } n \leq 150 \\ 15 & \text{si } n > 150 \end{cases}$$

Los valores que definen el ratio archivos/esclavo fueron establecidos por convención, siempre y cuando fueran considerados adecuados para una correcta y eficiente distribución del trabajo. El equipo entiende, sin embargo, que la definición más correcta de los valores discutidos excede los objetivos del trabajo.

Al ejecutarse, la aplicación maestro evalúa que los archivos especificados mediante parámetro sean válidos para el cálculo del hash MD5 y, si lo son, imprime sus rutas de acceso en un pipe. A dicho pipe son redirigidas las entradas estándar de los *S* procesos esclavos que el maestro instancia a continuación. Adicionalmente, la salida estándar de dichos procesos se redirecciona a un segundo pipe.

Los procesos esclavos leen del pipe la ruta de un archivo a procesar y, aprovechando el programa *md5sum* provisto por Linux, recuperan el hash del mismo. A continuación, imprimen el path y el hash generado en el formato requerido por el enunciado. Cabe aclarar que la integridad de los datos en el pipe es garantizada por un semáforo que asegura que solo un proceso pueda leer o escribir del pipe en un momento dado. El mismo es inicializado con un valor de uno por el master, y recuperado por cada esclavo para este fin.

Mientras los esclavos procesan los archivos, el maestro recupera del pipe de salida las cadenas de texto generadas, las imprime en un archivo según los requerimientos especificados, y posteriormente en un buffer ubicado en un espacio de memoria compartida creado previamente, para ser leídas por un eventual proceso vista.

Como el extremo de escritura del pipe asociado a la entrada estándar de los esclavos es cerrada por el maestro inmediatamente después de terminar de escribir, se garantiza que, al haberse leído todas las rutas para su procesamiento, los esclavos restantes levantarán un EOF, indicando que sus servicios no son ulteriormente requeridos y terminando su ejecución. Además, se incorporó al maestro llamadas de espera (*waitpid*) cerca del final de su ejecución, que levanten el valor de retorno de los esclavos a fin de evitar que degeneren en zombies.

A su vez y como fue mencionado, el proceso maestro genera un espacio de memoria compartida que albergará una estructura que define una matriz de cadenas de texto, un índice de lectura, y uno de escritura, para guardar lo recuperado de los esclavos inmediatamente después de escribirlos en el archivo de salida. La estructura se compartirá con un eventual proceso vista que, de conectarse, leerá de la matriz e imprimirá en pantalla las rutas de los archivos y los hashes. Para evitar esperas activas se utilizó un semáforo, que reanuda la ejecución de la vista exclusivamente cuando existe una cadena de texto pendiente de impresión. Cuando el maestro termina de escribir la última salida en la

memoria compartida y la vista termina de imprimirla, ésta se encarga de liberar los recursos que le corresponden, considerando que terminará su ejecución después de que el maestro lo haya hecho. Caso contrario, es el propio maestro el responsable de la limpieza.

Fundamentación de IPCs utilizados

Se optó por usar solo dos pipes, de entrada y salida respectivamente, para comunicarse con el esclavo, pues se consideró que garantiza una mejor distribución de las tareas, y libera al proceso maestro de la responsabilidad de repartir el trabajo. Esta implementación busca emular la funcionalidad provista por las Message Queues, a las que el grupo reafirma como una alternativa mucho más adecuada para la interacción entre el maestro y los esclavos, así también como entre el maestro y la vista. El equipo reconoce que la opción de crear dos pipes, de entrada y salida, por cada esclavo trae aparejada la posibilidad de evitar las system calls que nuestra sincronización con semáforos implica, pero mantiene que el impacto en performance real será mínimo, y reafirma su convicción en la superioridad de la implementación considerándose todos los factores.

Se obvió la impresión por salida estándar del PID del maestro por considerarse innecesaria. Se decidió que una nomenclatura para los recursos compartidos precedida por un dominio adecuado bastará para garantizar que los mismos no serán usurpados por agentes externos.

Se incluyeron tiempos de espera al inicio de la ejecución del maestro y de la vista para facilitar su ejecución simultánea. Debido a la necesidad de limpiar los recursos, la vista no corre si el maestro ya finalizó su ejecución. Estos tiempos fueron surgiendo en base a la conveniencia en el uso durante el desarrollo por parte de los programadores, y no pretenden sugerirse ideales.

Un problema que surgió en la implementación de pipes entre el proceso maestro y los esclavos consistió en que, utilizando sólamente dos pipes de lectura y escritura, la escritura era pisada y como consecuencia se recibía información errónea, como por ejemplo nombres de archivos que no existían. Como solución, se implementó un semáforo para prevenir el acceso múltiple al pipe. Una solución que también se planteó fue utilizar un lock cuyo efecto hubiera sido el mismo, así como también se propuso crear pipes de lectura y escritura para cada proceso esclavo lo cual parecía ineficiente.

Instrucciones de compilación y ejecución

Para compilación se debe navegar al directorio que contiene el código fuente y ejecutar el siguiente comando por consola:

```
$ make
```

Para ejecución se debe inicializar los procesos maestro y, de manera opcional, vista, aportando como parámetro los paths con los que se desea trabajar, de la siguiente manera:

```
$ ./Binaries/hash “path” &
$ ./Binaries/view
```

O también se pueden correr utilizando un pipe:

```
$ ./Binaries/hash “path” | ./Binaries/view
```

Para ejecutar los tests del programa, en donde se verifica la creación correcta de pipes, el agregado de archivos a los mismos y el cálculo de los hashes md5, se utiliza un flag y ningún parámetro:

```
$ ./Binaries/hash -t
```

Se ejecutó el programa en el entorno Docker solicitado por la cátedra, sin encontrar problemas de ningún estilo. **Se recomienda** correr los comandos de ejecución por separado en distintas shell con el fin de apreciar claramente lo que imprime el proceso maestro, que será el responsable de reportar errores o advertir al usuario si el caso lo ameritase.