Instituto Tecnológico de Buenos Aires

72.11 - Sistemas Operativos

Segundo Cuatrimestre 2023 - Grupo 5

Primer entrega

Autores:

Luca Valentin Bloise (63004)
Cristian Nicolás Tepedino (62830)

1. Decisiones Tomadas

Para recibir el output de md5sum desde el proceso slave, se optó por abrir un pipe, y realizar un fork y execv para llamar a md5sum como proceso hijo y pasar el output por el pipe.

Los esclavos funcionan recibiendo los path a los archivos vía stdin, e imprimiendo el resultado por stdout. Antes de realizar un execve desde el proceso aplicación, se utiliza close y dup para cambiar stdin y stdout por los respectivos extremos de los pipes a utilizar. Esto resulta en que el proceso esclavo pueda funcionar de forma independiente a la aplicación si se desea, además de que mantiene todo el código relacionado al manejo del pipe en application.c.

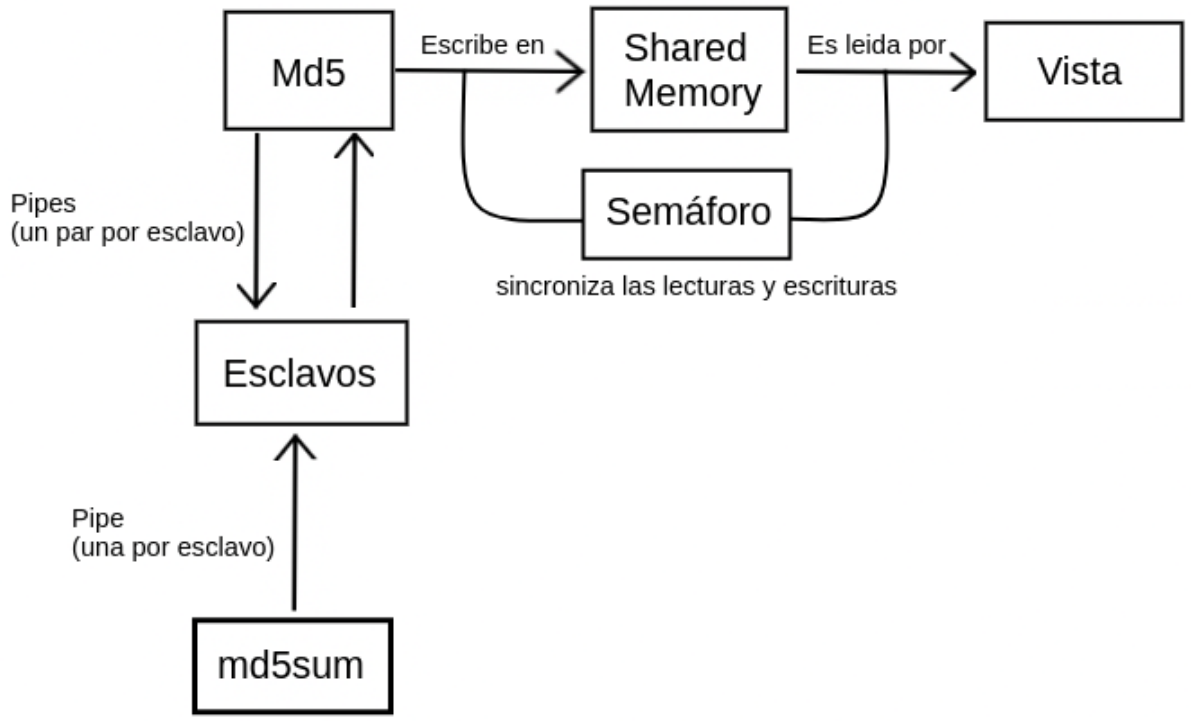
La cantidad de procesos esclavos y de archivos iniciales a enviar se deciden en función de la cantidad de archivos a procesar.

Se definió un struct que contiene los file descriptors de los pipes de un proceso esclavo, además de la cantidad de archivos pendientes que tiene. En el proceso aplicación, se utiliza un array de este struct para manejar los esclavos.

Se escribió un ADT para poder manejar la shared memory desde la vista y la aplicación. El ADT además cuenta con funciones para manejar un semáforo. En ambos casos (shm y semáforo) se utilizan las variantes con nombre de estos. El nombre que se usa en la aplicación y la vista siempre es el mismo, lo que permite guardarlo con un define en un .h compartido. De esta forma, la única información que puede variar es el tamaño de la shm. Para poder manejar esta información, se utiliza una constante (guardada en el .h compartido), que se multiplica por la cantidad de archivos a procesar para obtener el tamaño del buffer. El proceso aplicación imprime por stdout la cantidad de archivos, el cual puede ser utilizado por el proceso vista para también calcular el tamaño de la shm y así poder comunicarse con la aplicación.

En cuanto a la sincronización, el ADT de la shared memory inicia un semáforo con valor de 0. En el proceso aplicación, inmediatamente después de escribir se hace un up. En el proceso vista se hace un down justo antes de leer. El valor del semáforo en todo momento equivale a la cantidad de resultados de archivos que aún no se leyeron en la shared memory. De esta forma, cuando vista ya ha leído todo lo disponible, el valor del semáforo estará en 0, lo que lo forzara a esperar a que se escriban más resultados.

2. Diagrama de Procesos



3. Instrucciones de Compilación y Ejecución

Se utilizó make para facilitar la compilación. Los siguientes comandos pueden usarse desde el directorio en el que se encuentren los archivos de programa:

- make md5/ make slave/ make vista: compilan el archivo correspondiente.
- make all: compila todos los archivos.
- make clean: elimina todos los binarios y archivos objeto.

Para ejecutar únicamente el proceso aplicación, se debe correr ./md5, pasando como argumento la lista de archivos a utilizar. Esto generará un archivo “results.txt” que contiene el output. Para ejecutar el proceso aplicación con el proceso vista, es necesario pasar la información que md5 imprime en stdout a vista. Vista puede recibir esta información como argumento, o mediante stdin si no se envía ningún argumento. Esto permite varias opciones para la ejecución, por ejemplo:

- Con un pipe: ./md5 <archivos> | ./vista
- Ejecutando ./md5 <archivos> en una terminal, y luego ./vista <info> en otra
- Ejecutar md5 en background (./md5 <archivos> &), y luego vista en foreground (./vista <info>)

4. Limitaciones

A fin de poder poner un tope en la cantidad de memoria que ocupa en la shared memory el output de un archivo, se limitó la cantidad maxima de digitos que

se pueden mostrar del pid del proceso esclavo en el output a 6, y la cantidad máxima de caracteres del path a 256. Ambos números se decidieron de manera arbitraria, y pueden cambiarse simplemente modificando el valor de la constante asociada a cada tope en commons.h.

Para poder guardar la información enviada por stdin, en vista se definió una constante BUFFER_LENGTH, que indica el número de caracteres que se van a leer. Dado que el número que se debe enviar a vista es la cantidad de archivos a procesar, esto limita el número máximo de archivos a uno de BUFFER_LENGTH dígitos.

El esclavo utiliza el carácter '\n' para separar los nombres de los archivos que se le envían, por lo tanto, si un archivo contiene el carácter en el nombre, no se podrá enviar como argumento a md5sum correctamente.

5. Problemas Encontrados

Cuando se intentó enviar un archivo cuyo path contenia espacios, el programa esclavo lo enviaba al md5sum como 2 archivos separados. Esto se debe a que para enviar un argumento con espacios en linux, se debe utilizar \ antes de cada espacio, y popen ejecuta el string que se le envíe como si se escribiese en la línea de comandos. Para solucionar esto, se tuvo que reescribir la forma de llamar a md5sum, utilizando execv en lugar de popen. Esto nos permite especificar el argv, eliminando el problema con los espacios (y con otros caracteres problemáticos, como '&').

El buffer de stdout causaba problemas en aplicación y esclavo. En aplicación, causaba que la información para conectar el proceso vista se imprima después de realizar el sleep. En esclavo, causaba problemas para enviar el output a la aplicación. En ambos casos, se soluciono utilizando setvbuf para pasar el modo a unbuffered.

Al utilizar siempre el mismo nombre para la shared memory y el semáforo, y solo realizando el unlink al final del proceso aplicación, podría ocurrir que, si se mata al proceso durante la ejecución, no se realice el unlink, por lo que en la próxima ejecución, en lugar de crear una nueva shm y semáforo, se volvería a conectar a los mismos. Esto podría causar problemas, por ejemplo, si el semáforo estaba en un valor diferente a 0 cuando se mata al proceso, en la siguiente llamada, el proceso no se bloquearía al llegar al semaphoreDown. Esto fue solucionado agregando una llamada al unlink al principio de aplicación.