72.11 Sistemas Operativos

TP1 - INFORME

Grupo 22

Santiago Devesa (64223)

Tiziano Fuchinecco (64191)

Tomas Balboa (64237)

Profesores

Alejo Ezequiel Aquili

Ariel Godio

Fernando Gleiser Flores

Guido Matías Mogni

Índice

Resumen y objetivos

El objetivo de este trabajo práctico es desarrollar un programa que calcule el hash MD5 de múltiples archivos, balanceando la carga entre varios procesos esclavos. Para lograr esto, se emplean mecanismos clásicos de comunicación y sincronización entre procesos de la interfaz POSIX, como semáforos, memoria compartida y pipes.

Decisiones tomadas durante el desarrollo

Las principales características del programa son las siguientes:

- Se le asigna una cantidad constante de archivos iniciales a cada proceso esclavo, siendo esta inicialmente 2 (dos) con el fin de conseguir una distribución más equitativa.
- La cantidad de esclavos depende de los archivos que reciba el maestro como input, cada cinco archivos se creará un proceso esclavo.
- Cada vez que un esclavo devuelve un hash recibe un archivo nuevo para procesar si es que hay restantes.
- Los mecanismos de IPC utilizados fueron un par de pipes por cada esclavo (uno para que el master envíe los paths de los archivos a procesar y otro para recibir el resultado) y un buffer de memoria compartida para enviar los resultados del maestro a la vista.
- El mecanismo de sincronización utilizado fue un semáforo para controlar el acceso a la memoria compartida que simboliza cuántos resultados hay para leer en un momento dado. Esto fue planteado de esta manera porque la consigna dice: *"No es necesario un buffer circular, en su lugar se puede crear un buffer suficientemente grande para guardar todos los resultados."*, entonces no tuvimos la preocupación de que el maestro escriba en el mismo sitio en el que la vista se encuentra leyendo, dado que siempre se escribe al final del buffer. De este modo, el semáforo funciona únicamente para que la vista solo pueda leer de la memoria compartida si hay un resultado listo para leer.
- Con fines de modularizar el código y hacerlo reutilizable se encapsuló la lógica de creación, destrucción, lectura y escritura de la memoria compartida en un tipo abstracto de dato *shmManagerADT*.

Comunicación entre procesos

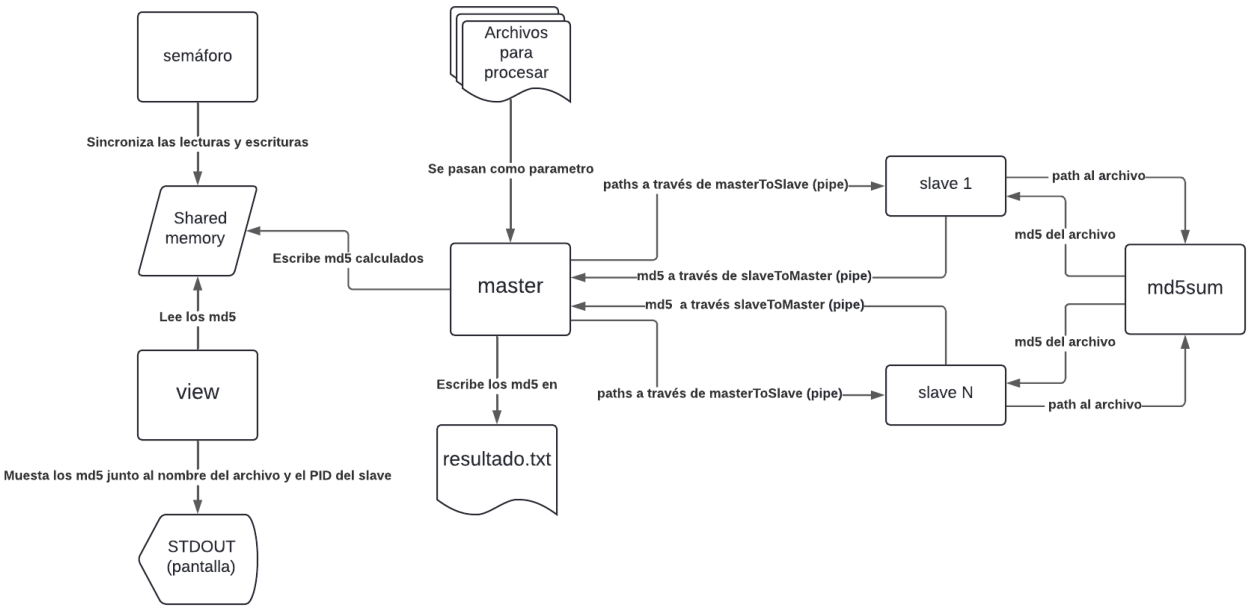

Figura 1. Diagrama de comunicación entre procesos

Como se observa en la figura 1, se utiliza un proceso maestro encargado de recibir los paths de archivos por línea de comando y los distribuye a través del pipe MasterToSlave a los diferentes procesos esclavos. Estos se comunicarán con el programa md5sum a través de un pipe temporal y devolverán los hashes MD5 por el pipe SlaveToMaster. Además, se agrega un proceso vista que puede recibir el nombre de una memoria compartida por línea de comando o utilizando un pipe con el master en la terminal que se encargará de leer de la memoria compartida para mostrar los resultados en la consola en tiempo real.

Instrucciones de compilación y ejecución

Instalación

Prerrequisitos

Docker: el proyecto corre en un contenedor con todos los demás prerrequisitos ya instalados. Para instalar Docker, siga las instrucciones en este link.

Pasos a seguir

1. Clonar el repositorio:
   git clone git@github.com:smdevesa/TP1_SO.git
2. Ingresar al directorio del repositorio:
   cd TP1_SO/root
3. Descargar la imagen del contenedor de Docker utilizando el comando:
   docker pull agodio/itba-so-multi-platform:3.0
4. Correr el contenedor utilizando el comando:
   docker run -v ${PWD}:/root --privileged -ti --name SO
5. Ingresar al directorio root:
   cd root
6. Compilar el proyecto con el comando:
   make all

Ejecución

Ejecución sin vista

Este modo permitirá ejecutar el programa sin imprimir los datos en la terminal. El programa solo imprimirá el nombre identificador del buffer de memoria compartida que utiliza. Para ejecutar en este modo se debe correr el siguiente comando:
./master archivo1 archivo2 ... archivoN

Ejecución con vista a través de pipe

Este modo permitirá ejecutar el programa y visualizar los datos en tiempo real. Para esto se deberá ejecutar el siguiente comando:
./master archivo1 archivo2 ... archivoN | ./view

Ejecución con vista en dos terminales

Este modo permitirá ejecutar el proceso maestro en una terminal y visualizar los resultados en tiempo real desde otra terminal. Para esto se deberá ejecutar:

1. En la primera terminal:
   ./master archivo1 archivo2 ... archivoN

2. En la segunda terminal:
   ./view /sharedMemory

Limitaciones

El grupo considera que el programa tiene las siguientes limitaciones:

- El buffer de memoria compartida no es circular y los datos no se pisan, por lo que, si no se compila con un tamaño adecuado para almacenar todos los resultados obtenidos, podría llegar a perderse información y/o causar que el programa termine de forma repentina.
- Al distribuir los archivos entre los diferentes procesos esclavos no se realiza ningún chequeo con el tamaño de los archivos recibidos por lo que no se puede asegurar que la carga de los esclavos es equitativa, pudiendo esto hacer que el cálculo de los hashes md5 tarde más de la cuenta. Ejemplo: con seis archivos y dos procesos esclavos el esclavo 1 recibe tres archivos de 2GB cada uno y el esclavo 2 recibe archivos livianos de 3MB.
- El programa asume que todos los archivos recibidos tienen permisos suficientes para acceder y calcular su hash md5. Esta limitación causa que si se le envía un archivo con permisos insuficientes o un directorio el programa no termine correctamente.

Problemas encontrados durante el desarrollo y sus respectivas soluciones

Inicialmente el grupo no estaba lo suficientemente informados acerca del uso de pipes para la comunicación por lo que nos encontramos con problemas al momento de la recepción del master de los md5 calculados. Esto se resolvió estudiando el manual de Linux y haciendo ejercitación en un proyecto aparte. La segunda gran problemática fue al momento de el master tener que leer de múltiples pipes, exactamente saber de donde debería leer para evitar bloqueos innecesarios. Esto se resolvió utilizando la recomendación ofrecida por la consigna del trabajo, utilizando la función select(2). La sincronización del master con la vista fue otro problema, no lográbamos sincronizarlos utilizando un único semáforo y tampoco queríamos emplear recursos extra. Inicialmente queríamos hacer uso del semáforo de exclusión mutua

pero nos dimos cuenta que podíamos evitarlo utilizando un semáforo que tenga registro de la cantidad de md5 que puede leer la vista en un momento dado. Finalmente encontramos complicaciones a la hora de finalizar la ejecución del proceso vista, dado que esta no tiene acceso a la cantidad de md5 totales que debe leer (el master únicamente debe enviarle a través del pipe la información para conectarse a la shared memory). Esto se resolvió poniendo una marca de final en el buffer de memoria compartida (en nuestro caso fue la lectura de un string vacío), de modo que al encontrarse (el proceso vista) con dicha marca de final, culmine su ejecución puesto que no habrá más md5 por leer.

Conclusión

El trabajo práctico nos resultó interesante dado que es una estructura de programa que podría servir para cualquier tipo de tareas que se quieran dividir en diferentes procesos que no sean necesariamente calcular el hash md5.

Además, nos permitió conocer una interfaz muy útil como es POSIX que sirve mucho al poder ser utilizada en la gran mayoría de los sistemas operativos UNIX facilitando que el mismo código corra en diferentes entornos.

Finalmente, los temas abordados de sincronización de procesos e IPC nos permitieron conocer una parte de la programación que en anteriores materias estaba totalmente oculta como tener en cuenta que hay más procesos corriendo además del que estamos programando individualmente, así como las ventajas y problemáticas que esto puede generar.

Analisis con PVS-Studio

Al ejecutar el programa con PVS-Studio nos aparece un warning que menciona acceso a memoria (específicamente a shmName) luego de liberarlo. *"/Users/smdevesa/Desktop/TP1_SO/root/lib/shmManager.c 196 warn V774 The 'shmManager->shmName' resource descriptor was used after the resource was released."* Revisamos en profundidad el código y no encontramos ninguna ocasión en la que pareciera utilizarse "shmName" posterior a su liberación. Es por esto que decidimos desestimar este warning y tomarlo como un falso positivo.

Fuentes

https://man7.org/linux/man-pages/dir_section_2.html [manual de linux, sección 2]