Sistemas Operativos
Segundo cuatrimestre 2022
Trabajo Práctico N 1
Grupo 14

Integrantes:

- Castro, Juan Ramiro
- Gazzaneo, Lautaro Nicolas
- Limachi, Desiree Melisa

Instrucciones de Compilación

Para compilar el programa desde docker, solo hay que correr el comando Make all. Si se quiere compilar el programa fuera del entorno provisto del docker, es necesario instalar Make y GCC.

Instrucciones de Ejecución

Asegurarse de tener instalado el programa md5sum.
Para ejecutar el programa sin salida en la terminar simplemente hay que correr el siguiente comando:
./md5 [FILES]

Si queremos además correr el programa viendo en terminal, podemos pipear la salida del programa principal con la vista. En este caso hay que ejecutar el siguiente comando:
./md5 [FILES] | *./vista*

También se puede ejecutar primero md5 y después la vista, dándole como parámetro la información que imprime md5.

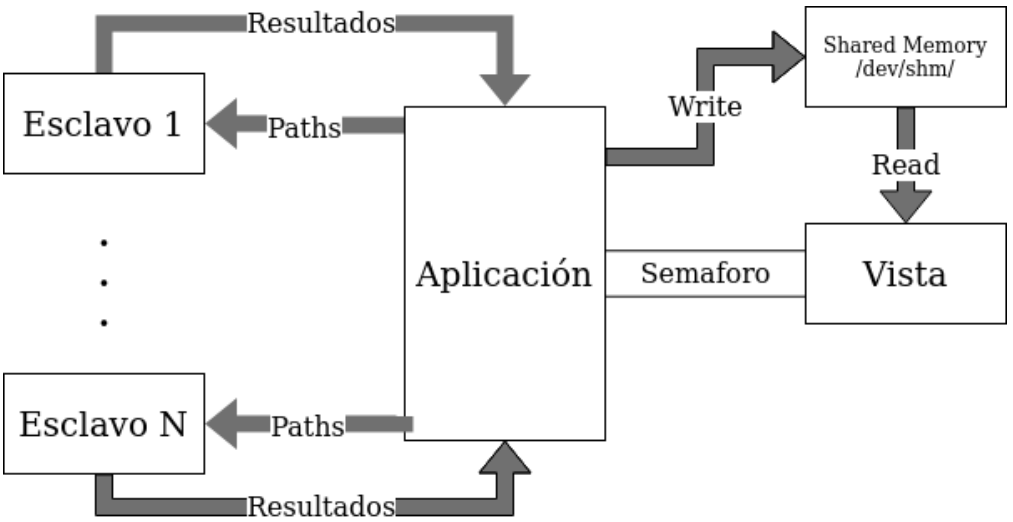
Decisiones tomadas durante el desarrollo

A la hora de enviar los paths recibidos a los esclavos, decidimos chequear anteriormente mediante el uso de stat que estos sean archivos regulares para evitar posibles errores.

Las IPCs utilizadas fueron todas las descritas en la consigna del trabajo práctico. El uso de la shared memory decidimos abstraerlo en un TAD que se encarga de crearla, conectarse, mapearla a la memoria del proceso, además del proceso de eliminarla y desmapearla.

Para crear el semáforo utilizado en la shared memory, decidimos crear uno con nombre a partir del nombre dado a la shared memory que ya es compartido con el proceso vista. Luego, cada vez que el proceso md5 escribe a la shared memory, aumenta en uno el valor del semáforo, mientras que el escritor lo reduce en uno lo que nos permite que el escritor no se trabe esperando a que el lector termine de leer. Una vez que md5 termina, realiza el unlink del semáforo y shared memory, así cuando el proceso vista los cierra son eliminados.

Decidimos poner una cantidad fija de esclavos mediante el uso de un define, así en caso de querer cambiarlo puede realizarse fácilmente editando el valor y recompilando. También, decidimos sacarle el buffer al stdout de vista, así se puede notar mejor el hecho de que está pasando todo al mismo tiempo.



Problemas encontrados

Uno de los primeros problemas que nos encontramos fue que asumimos que teníamos que encargarnos nosotros del procesamiento del wildcard en el path del argumento, cuando realmente se encarga la shell. En vez de utilizar glob para obtener los paths a los archivos, directamente usamos los strings que ya se encuentran en argv.

Otro problema que tuvimos es que cuando utilizabamos los procesos en modo pipe, md5 no escribia el nombre de la shared memory hasta que terminaba, entonces cuando el proceso vista iba a abrir la shared memory, se encontraba con que esta no existía, pues ya había sido cerrada. Esto se soluciono mediante un fflush de stdout luego de imprimir el nombre, y decidimos usar un simple fflush en vez de cambiar el buffer entero dado que es lo único que debería imprimir a stdout el proceso.

Limitaciones

La shared memory tiene un tamaño predefinido, por lo que hay una cantidad máxima de caracteres que se pueden almacenar en ella. Sabemos que no es la manera más eficiente de guardar la información, pero no representa un problema en la gran mayoría de usos. Se realiza esta implementación para facilitar la relación entre el proceso md5 y vista.

Otra limitación es el tamaño del path al archivo a hashear, que decidimos limitarlo a 1024 bytes.

PVS-Studio da una warning en el archivo vista.c, en la linea 29. Es un falso positivo, pues arriba nos fijamos que el tamaño sea mayor a 1, y si no lo es entonces hacemos un exit.