Informe Trabajo Práctico N°1

72.11 - Sistemas Operativos

1° Cuatrimestre 2018

Nicolás Benenzon(57539)

Juan Pablo Lo Coco(57313)

Guido Princ(57334)

Ariel Atar(57352)

Introducción

El trabajo consistió en poner en práctica la creación de múltiples procesos y que estos se comunicaran, por diferentes medios, para realizar una tarea en particular. La misma fue la de crear los hashes de los archivos obtenidos por un argumento pasado. Este documento va dedicado a explicar las decisiones tomadas a lo largo del trabajo y los problemas encontrados en el transcurso del mismo, entre otros asuntos.
No pudimos implementar la vista en el tp.

Decisiones tomadas durante el desarrollo

- Al comienzo íbamos a utilizar pipes para realizar la comunicación entre la aplicación y sus esclavos ya que era con algo que habíamos trabajado previamente. Pero luego de muchos problemas, como tener que guardar los pipes de ida y vuelta para ambos procesos (sus files descriptors principalmente), saber a cuál hijo teníamos que interactuar, etc., decidimos usar named pipes para usar una manera en la que los procesos podían conectarse de manera rápida y sencilla usando como referencia sus PIDs.
- A través de shared memory y semáforos realizamos la conexión entre el proceso vista y el proceso aplicación. Al no haber utilizado esto previamente, surgió la problemática de no saber cómo afrontarlo. Gracias a la explicación de compañeros y búsquedas pudimos comprender el concepto básico para su uso. La verdad usamos principalmente shared memory para incluirlo en el trabajo y llegar a probar nuestras capacidades para hacerlo.
- Para el proceso esclavo se utilizó el "pipe" como mecanismo de comunicación entre procesos ya que nos pareció la forma más sencilla de resolver los cálculos de los hashes MD5 y luego devolverlos sin necesidad de volcar a archivos en disco las soluciones a los cálculos de los hashes y luego leerlos desde ahí. Lo que se hace básicamente es que a través de un pipe y un fork, el proceso hijo va calculando los hashes MD5 de los archivos que se encuentran en la cola de pedidos y los escribe en el buffer del pipe, y luego el proceso padre lee de ese buffer el hash calculado por su hijo y lo retorna. Para la cola de pedidos implementamos una cola en C que almacena todos los nombres de los archivos, y guarda una referencia tanto al primer elemento como al último (además del tamaño actual de la lista). La referencia al último elemento la podríamos haber omitido pero lo implementamos así por

cuestiones de eficiencia temporal al momento de encolar un nuevo archivo, ya que sin la referencia al último archivo no queda otra opción que recorrer toda la cola cada vez que necesitamos encolar un nuevo archivo. Se hizo testing para el proceso esclavo y pensamos también en hacer para los métodos de la cola tales como encolar y desencolar un archivo y ver qué sucede, pero entre todos llegamos a la conclusión de que eso no iba a ser necesario ya que eran métodos bastante sencillos y ya tenemos un manejo bastante amplio de las colas.

- El proceso aplicación para la lectura de los fd de sus esclavos realiza un Select. En un primer momento se había optado por realizar una espera activa pero se comprobó que se pierden recursos y se rendía menos.
- Se crean siempre 5 esclavos porque pensamos que aunque tengamos menor cantidad de archivos, los esclavos a los que no se les pueden asignar tareas mueren.
- Se decidió usar una cola al comienzo para guardar únicamente todos los no directorios que nos llegan por lo ingresado en consola.

Conflictos/Problemas ocurridos y limitaciones

La mayor limitación que tuvimos fue la falta de conocimiento sobre la mayoría de los temas, lo que nos llevó a invertir mucho tiempo investigando y consultando.

En cuanto a conflictos, se pueden nombrar:

- Shared Memory fue un problema por no comprender nada, en un principio, acerca del tema. **No se pudo implementar en el tp.** Nos fallaba el shared memory y aunque íbamos a implementar un named pipe para la vista también, creímos más conveniente por el tiempo subir lo que teníamos.
- Los Read/Write de los FD a veces no funcionaban correctamente entre la aplicación y el esclavo. Se agregaron waits y se mejoraron las sintaxis de código.
- Ocurrieron problemas al desencolar los elementos de la cola de archivos y hashes, por lo que tuvo que ser reemplazada por falta de tiempo.
- Problemas con la eliminación de los archivos generados por el mkfifo.

Instrucciones compilación y ejecución

Se ejecutó todo en Ubuntu 16.04.
Para la compilación utilizamos el comando:

- gcc slaveProcess.c -o slaveProcess -Wall -pedantic -std=c99 -lpthread
- gcc viewProcess.c -o viewProcess -Wall -pedantic -std=c99 -lpthread
- gcc applicationProcess.c queue.c testLib.c -o hash -Wall -pedantic -std=c99 -lpthread

Por otro lado, para la ejecución se utiliza:

- ./hash [pattern]

Bibliografía

Para realizar el trabajo práctico fue utilizado el comando man para comprender la mayoría de las funciones utilizadas, así como también se utilizaron las páginas de Github y Stackoverflow para realizar búsquedas particulares sobre problemas encontrados.
A continuación listamos algunos otros links utilizados para consultar información para el trabajo:

- http://developerweb.net/viewtopic.php?id=2934
- https://github.com/Jeshwanth/Linux_code_examples/tree/master/POSIX_shared_memory