Sistemas Operativos

Trabajo Práctico 1

Grupo 3

Ian Franco Tognetti - 61215 - itognetti@itba.edu.ar
Lautaro Farias - 60505 - lfarias@itba.edu.ar
Axel Castro Benza - 62358 - acastrobenza@itba.edu.ar

Índice

Introducción

El siguiente trabajo práctico tiene como objetivo aprender a utilizar los distintos tipos de *IPC*s presentes en un sistema *POSIX*. Esto se consigue mediante la implementación de un sistema que distribuye el cálculo del *md5* de múltiples archivos entre varios procesos.

El siguiente diagrama ilustra la interconexión entre los procesos y mediante qué funciones se acceden a cada uno de ellos:

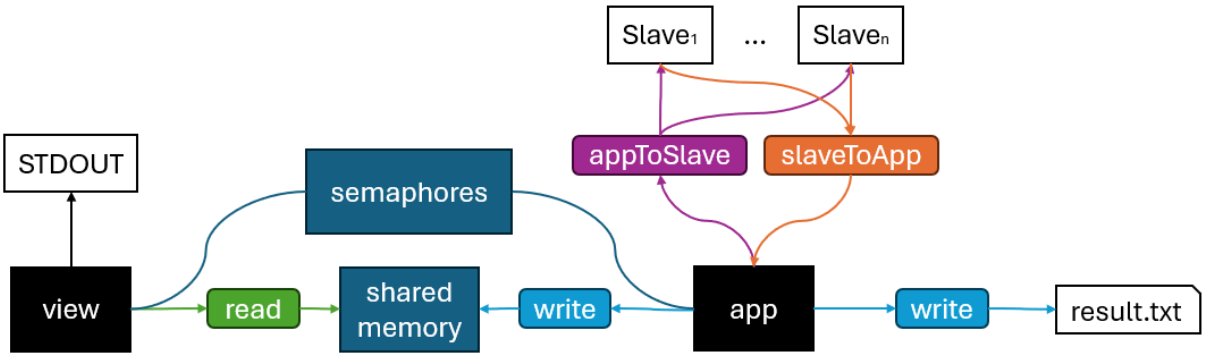

Diagrama 1: *Diagrama de la conexión entre procesos.*

Como se puede observar, los N procesos *esclavos* se conectan con la *aplicación* mediante los pipes *slaveToApp* y *appToSlave*. Donde, previo a lectura de los pipes *slaveToApp*, se realiza un *select* para determinar si el pipe debe ser leído o no. Por otro lado, la aplicación es la encargada de escribir el md5 de los archivos recibidos y este resultado se puede ver tanto en el archivo *result.txt* como en la memoria compartida, que es utilizada por el proceso *view* para imprimir la información por salida estándar. Es importante destacar que los semáforos son utilizados como prevención para la race condition entre la lectura y escritura de la shared memory.

Instrucciones de compilación y ejecución

Dentro de una terminal de linux se deberán correr los siguientes comandos:

1- Para la compilación:

```
$ cd TP-SO
$ docker run -v "${PWD}:/root" --privileged --rm -ti agodio/itba-so:2.0
$ cd root
$ make all
```

2- Para la ejecución:

Opción **A** en una terminal:

```
$ ./md5 files/*
```

Opción **B** en una terminal:

```
$ ./md5 files/* | ./view
```

Opción **C** en dos terminales:

En la primera terminal:

```
$ ./md5 files/*
```

En la segunda terminal (conectada al docker container id de la primera):

```
$ ./view <arg1>
```

Decisiones tomadas durante el desarrollo

En el diseño del sistema, se optó por organizar la información relevante en diferentes estructuras según fuera necesario. En cada una de ellas se guardaron los atributos específicos de cada elemento, como el hash md5, los procesos esclavos, la memoria compartida y los semáforos.

En cuanto a los mecanismos de comunicación utilizados entre los diferentes componentes del sistema, se implementaron pipes anónimos. Estos permiten la comunicación bidireccional entre la aplicación y los procesos esclavos. Además, con el objetivo de manejar el resultado del md5, se empleó un proceso subesclavo. Este se encarga de redirigir su salida estándar de los procesos subesclavos hacia la entrada estándar de los procesos esclavos, logrando una mejor optimización de los recursos del sistema.

Adicionalmente, para facilitar la comunicación entre app y view, se utilizó la memoria compartida junto con semáforos. La memoria compartida permite que ambos procesos accedan a los mismos datos de forma eficiente. Los semáforos, por su parte, se emplearon para controlar el acceso concurrente a la shared memory. Esto ayuda a evitar problemas como las condiciones de carrera y garantiza que los datos se lean y escriban de manera coherente y ordenada.

Problemas durante el desarrollo

Uno de los problemas más simples y a su vez más destructivos que ocurrieron durante el desarrollo de este trabajo práctico fueron los errores de semántica o constantes globales. Un ejemplo claro de esto es la incorrecta utilización del STDIN_FD cuando en realidad corresponde el *STDOUT_FD* y viceversa.

Por otra parte, se tuvo que investigar el funcionamiento de varias funciones con el fin de lograr una correcta utilización y manejo en caso de errores. Especialmente aquellas relacionadas a la creación de memoria compartida y semáforos.

Además, al compilar el trabajo mediante el makefile creado, se generaban warnings con respecto a la declaración de las funciones *pread* y *pwrite* propias de la librería unistd.h. Esto se resolvió mediante la adición de la línea *#define _XOPEN_SOURCE 500* previo a la inclusión de la librería.

También se debió explorar el uso de PVS-Studio. Se encontraron dificultades iniciales para entender completamente cómo funciona esta herramienta. Para superar este obstáculo, se recurrió a la documentación oficial y se consultaron diferentes recursos en línea para obtener una comprensión más clara de su uso y funcionamiento

Por añadidura, al comienzo del desarrollo, se observó que el programa *view* ejecutado en una segunda terminal suspendia su ejecución durante un tiempo indeterminado. Esto fue ocasionado principalmente por el mal manejo de los argumentos que lo acompañaban.

Una última observación, al analizar el proyecto con PVS-Studio, se encontraron tres warnings en el archivo *view.c* relacionadas con el uso de la función *scanf*. Se señaló un posible uso incorrecto que podría resultar en un overflow del buffer. Sin embargo, se decidió no actuar en consecuencia ya que cuando se declara la variable *string* con el formato *char[N]* se asegura que dichas cadena no tenga tamaños dinámicos.

Limitaciones

En caso de querer ejecutar el proyecto utilizando dos terminales es necesario que ambas estén conectadas con el mismo id del container de docker.

El proceso *app* genera los semáforos y la shared memory aún en el caso en el que no se esté ejecutando el proceso *view*. Por lo que se están utilizando recursos innecesarios en caso de solamente querer obtener los resultados en el archivo *result.txt*.

Por otra parte, debido a que la shared memory tiene un tamaño determinado, el programa tiene una cantidad limitada de archivos máximos a procesar.

Conclusión

Para concluir, este trabajo práctico permitió explorar los distintos tipos de IPCs en un entorno POSIX, mediante la implementación de un proceso que distribuye el cálculo del md5 de varios archivos entre varios procesos. Se adquirió experiencia en el uso de mecanismos como pipes, shared memory y semáforos con el fin de lograr una comunicación eficiente y coordinada entre procesos. Como así también la importancia de la gestión correcta de los recursos del sistema.