72.11 Sistemas operativos - 2do Cuatrimestre 2023

INTER PROCESS COMMUNICATION

Grupo 13

Rocio D´Intino - 62341 - rdintino@itba.edu.ar

Brian D´Arco - 56413 - bdarco@itba.edu.ar

Matías Ignacio Luchetti - 62337 - mluchetti@itba.edu.ar

Índice

Introducción

El siguiente trabajo tiene como principal objetivo el manejo de diversos tipos de IPCs presentes en un entorno POSIX. Lo anteriormente mencionado se logró a través del desarrollo de un sistema en C, encargado de distribuir el cálculo del hash md5 de varios archivos entre múltiples procesos.

En base a este fin, se utilizaron 3 procesos diferentes: aplicación, vista y esclavos, cuya representación se muestra a continuación.

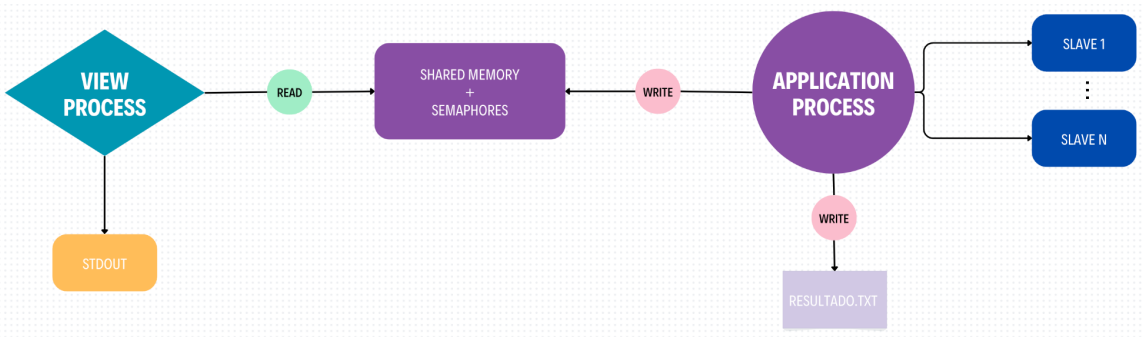

Imagen 1. Diagrama ilustrativo de la conexión entre procesos.

Es posible apreciar que el sistema cuenta con N esclavos, los cuales se comunican con la aplicación a través de los pipes masterToSlave y slaveToMaster. La aplicación es la encargada de escribir el hash md5 de los archivos recibidos por parámetro en 2 lugares diferentes, tanto en el archivo *resultado.txt* como en la memoria compartida. Es a través del último mecanismo mencionado que el proceso vista logra imprimir la información correspondiente por salida estándar.

Instrucciones de compilación y ejecución

1. Compilación:

```
$ docker pull agodio/itba-so:2.0
$ cd TPE1-SO
$ docker run -v "${PWD}:/root" --security-opt
seccomp:unconfined -ti agodio/itba-so:2.0
$ cd root
$ make all
```

2. Ejecución:

   Opción 1: uso de Pipe

   ```
   ./md5 <files>  | ./view
   ```

   Opción 2: uso de 2 terminales

   1. Terminal A

      ```
      ./md5 <files>
      ```

   2. Terminal B

      ```
      ./view <arg1> <arg2> <arg3>
      ```

   El proceso aplicación imprimirá el valor de los argumentos necesarios para ejecutar el proceso vista correctamente.

Decisiones tomadas durante el desarrollo del trabajo

Con el fin de obtener un sistema lo más claro y simple posible, se decidió agrupar la información relevante en diversas estructuras según fuera necesario. Así se definieron TADS para guardar los atributos de un hash md5, un proceso esclavo, una memoria compartida y un semáforo.

En cuanto a los mecanismos de IPC utilizados, se crearon pipes anónimos para comunicar el proceso aplicación con los procesos esclavos, y viceversa. Asimismo, al obtener el resultado del hash md5, se utilizó un proceso subesclavo con el objetivo de buscar una gestión eficiente de los recursos del sistema, redireccionando su salida estándar a la entrada estándar del proceso esclavo.

En adición, para permitir la comunicación entre el proceso aplicación y el proceso vista, se utilizó la memoria compartida, al igual que semáforos, con el propósito de obtener una compartición de datos eficiente y una sincronización ordenada. De este modo, mediante dos semáforos se logró controlar el acceso concurrente a la memoria compartida, evitando problemas como condiciones de carrera y garantizando que los datos se leyeran y escribieran de manera coherente.

Problemas encontrados durante el desarrollo del trabajo

Fue necesario investigar el uso de las funciones relacionadas con pipes, semáforos y memoria compartida. Al principio se encontraron dificultades para comprender con claridad su uso y funcionamiento, por lo que se debió consultar la documentación, al igual que diversos sitios web.

En reiteradas ocasiones la ejecución del programa se colgó, es decir, no lograba terminar como era esperado. Tras depurar y analizar el código, se notó que el obstáculo se encontraba relacionado con el mal manejo de semáforos.

Siguiendo con el problema planteado, se hallaron inconvenientes a la hora de acceder a la memoria compartida. Se observó que al ejecutar el proceso vista, no se lograban capturar correctamente los nombres de la shared memory y los semáforos, obteniendo errores a la hora de intentar acceder y abrir los mismos. Como este comportamiento del proceso vista depende del proceso aplicación, se llegó a la conclusión de que el problema se encontraba en la sincronización de los procesos.

Por esta razón, fue necesario hacer un seguimiento minucioso del uso de los semáforos para resolver el bloqueo y la falta de sincronización, aumentando y disminuyendo sus valores donde fuese necesario.

Por último, se encontró un problema con la declaración de las funciones *pwrite* y *pread*, cuya causa se desconoce. Las mismas forman parte de la librería unistd.h, la cual fue debidamente incluída en utils.h. Asimismo, al compilar con la opción -lrt, se espera el enlace con la biblioteca librt, la cual contiene funciones relacionadas con el tiempo real, presentes en un sistema POSIX. Por otro lado, mediante la opción -std=c99 se indica la compilación en el estándar C99, la cual debería ser compatible con las funciones. Las consideraciones mencionadas se han tenido en cuenta para intentar solucionar los warnings generados, pero de todas formas no lograron solucionarlos.

Limitaciones

En primer lugar, el proceso aplicación crea y configura un bloque de memoria compartida, junto con los respectivos semáforos, por más de que no se ejecute el proceso vista en ningún momento. Si únicamente quisiera obtener la información del hash md5 a través del archivo resultado.txt, se estarían utilizando innecesariamente recursos.

En segundo lugar, si bien se ha creado un buffer lo suficientemente grande para guardar los resultados, aún así existe la posibilidad de que se produzca un overflow si los datos generados exceden el tamaño previamente asignado.

Conclusión

En conclusión, el presente trabajo permitió adquirir conocimientos sobre la importancia de la sincronización adecuada en aplicaciones multiproceso y la gestión eficiente de recursos compartidos. Se destacó la necesidad de optimizar la asignación de recursos según las necesidades del sistema para evitar el uso innecesario de memoria, entre otros. De este modo, el proyecto proporcionó una comprensión más sólida de los conceptos de IPC y su aplicación en entornos POSIX.