Informe Trabajo Práctico №1:

“Inter Process Communication”

Sistemas Operativos - Segundo Cuatrimestre 2021

Docentes
Aquili, Alejo Ezequiel
Godio, Ariel
Merovich, Horacio Victor
Mogni, Guido Matías

Grupo 2
Chao, Florencia - Legajo 60054
Konfederak, Sol - Legajo 60255
Sandrini, Santiago - Legajo 61447

12 de Septiembre de 2021

Decisiones de desarrollo

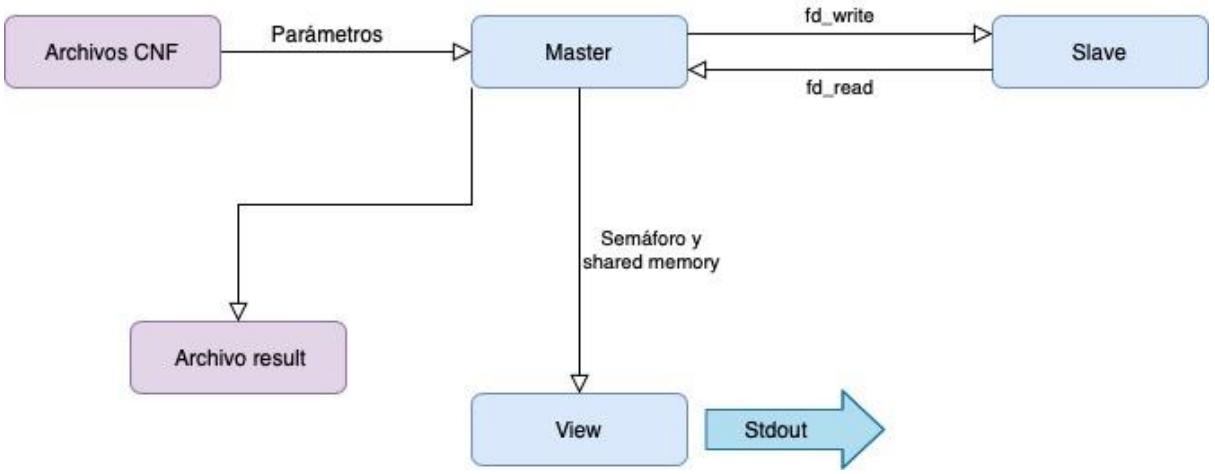
Al asignar los archivos para resolver, se decidió utilizar cinco procesos esclavos que reciben inicialmente un archivo para resolver.

Para la comunicación entre los procesos Master y Slave se decidió utilizar dos unnamed pipes para cada proceso esclavo: *fd_read* y *fd_write*. El pipe *fd_read* se utiliza para recibir las soluciones que envían los esclavos al proceso Master. El pipe f*d_write* se utiliza para asignar los archivos desde el proceso Master a los esclavos.

Para la comunicación entre Master y View, se decidió crear una shared memory. De esta manera, se logra que el proceso Master pueda escribir en un archivo llamado *result* y por otro lado, si luego se llama al proceso View, este se encarga de leer la shared memory e imprimir por salida estándar. Este funcionamiento, permite que el archivo *result* sea creado y escrito siempre, sin importar si se llama al proceso View. Cabe destacar que si el archivo *result* ya se encontraba creado, el mismo solamente se vuelve a escribir, perdiendo los resultados obtenidos anteriormente.

La conexión entre la escritura y lectura de los archivos es manejada a través de un semáforo. De esta manera, se puede coordinar los accesos de lectura y escritura así logrando evitar condiciones de carrera y deadlocks. El siguiente diagrama resume el funcionamiento del trabajo completo.

Diagrama



Limitaciones

La limitación que tiene el trabajo es el tamaño de la shared memory. La misma, está definida para manejar aproximadamente 30 archivos con nombres cortos. Como mínimo se debe enviar un archivo.

Problemas y soluciones

Un problema que encontramos a la hora de realizar el trabajo práctico fue el buffering. El principal problema se daba en el archivo result que imprimía los resultados al finalizar de procesar todos los archivos. Este problema pudo ser solucionado a través de la utilización de la función fflush().

También, tuvimos un conflicto con la función select() ya que no funcionaba como esperábamos. Cuando planteamos la forma de distribuir los archivos por primera vez consideramos que no era necesario repartir archivos de manera inicial a los esclavos pero luego entendimos que era necesario para que la función select funcione de la manera correcta.

Instrucciones de compilación y ejecución

Para la compilación, al comenzar hay que instalar el minisat con el comando apt-get install minisat. Luego, para compilar los archivos hay que ejecutar el comando make en la carpeta principal.

Para la ejecución se necesitan los archivos de extensión .cnf. Cabe destacar, que los archivos de extensión cnf deben ser agregados antes de ser ejecutados como es mencionado en la consigna. Para comenzar con la ejecución, primero hay que posicionarse en la carpeta bin donde se generan los ejecutables. Y luego hay dos formas de ejecución:

- Primera forma:

./master *pathCnfFiles*: se ejecuta el proceso padre y se le envían los archivos dentro de *pathCnfFiles*. Por consola no se mostrará ningún resultado (sólo se imprime un dato importante para el proceso view), sino que serán exportados al archivo *result* que aparecerá en la carpeta principal. Para ir viendo los resultados, se debe ejecutar el proceso view.

- Segunda forma:

./master *pathCnfFiles* | ./view: se ejecuta el proceso padre y el proceso vista, quien se conectará a un espacio de memoria compartida para leer lo que escriba el proceso padre y lo imprimirá en la salida estándar. También, se exportarán los datos al archivo *result*.

Código utilizado de otras fuentes

- Para la creación de esclavos:
  https://github.com/mit-pdos/xv6-riscv/blob/riscv/user/sh.c
- Para la creación de shared memory:
  https://github.com/WhileTrueThenDream/ExamplesCLinuxUserSpace/blob/master/sm_create.c
  https://github.com/WhileTrueThenDream/ExamplesCLinuxUserSpace/blob/master/sm_write.c