Trabajo Práctico 1

Sistemas Operativos

1°C 2020

Profesores:

Merovich, Horacio	hmerovich@itba.edu.ar
Godio, Ariel	agodio@itba.edu.ar

Alumnos:

Puig, Tamara	59820	tpuig@itba.edu.ar
Dell Isola, Lucas	58025	ldellisola@itba.edu.ar
Boccardi, Luciano	59518	lboccardi@itba.edu.ar

Decisiones tomadas durante el desarrollo

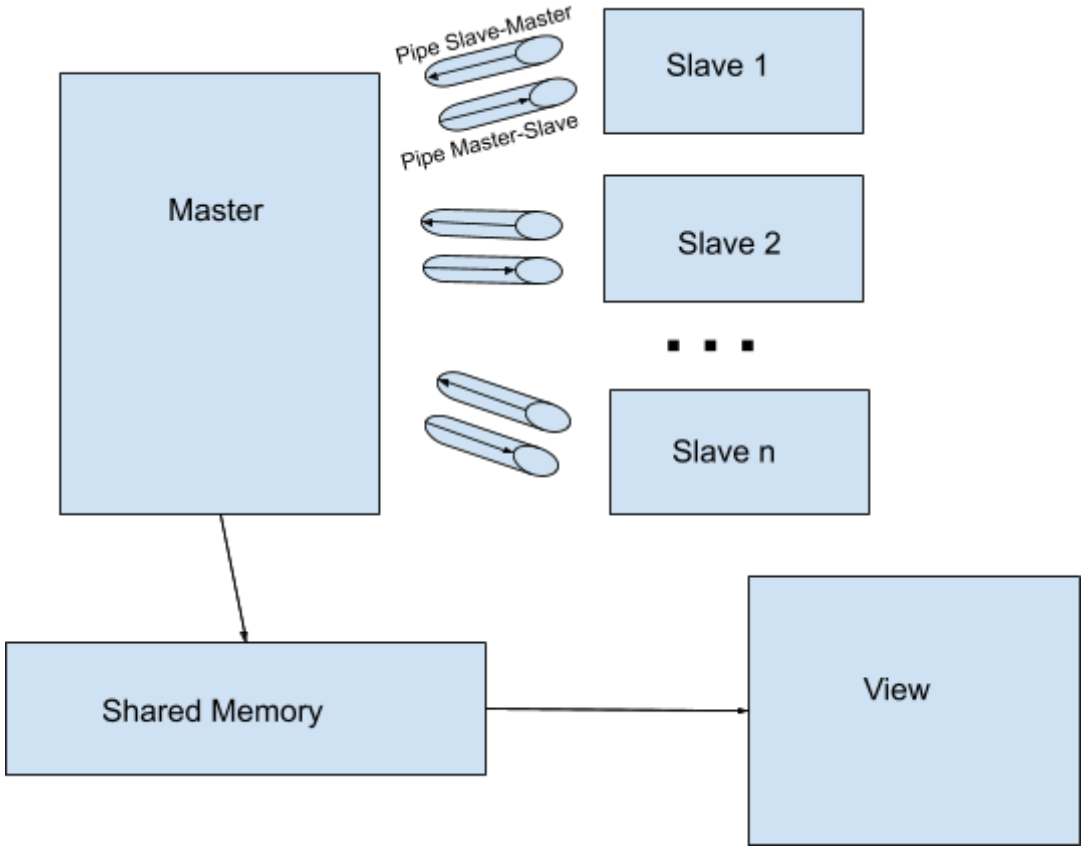
Para realizar el programa, se necesitaron diseñar dos vías de comunicación entre procesos: por un lado el Master con sus Slaves; y por el otro Master con View.

Para la primera interacción se decidió como IPC utilizar un 2 pipes (*nameless*) por cada par Master-Slave. Uno de ellos funciona como canal "Master→ Slave", por el cual se envían los nuevos archivos a analizar, mientras que el otro, "Slave→ Master", se utiliza para que una vez realizada la operación se retornen los resultados. El grupo no consideró adecuada la alternativa brindada de utilizar un único named pipe "Slave→ Master" para enviar los resultados ya que agregaría complejidad innecesaria al proyecto.

Para evitar el "busy waiting" se utiliza un `select()` en el padre, esta función bloquea al programa hasta que detecte que se enviaron datos por alguno de los file descriptor que estaba monitoreando. Como decisión de implementación, se eligió `select()` en lugar de `pselect()`, simplemente porque no fue necesario emplear señales para realizar la tarea, ya que la system call `read()` retorna el valor cero (0) cuando alcanza End of File (EOF), que en el caso de pipes se obtiene al leer de un pipe cerrado.

Con respecto a la conexión entre View y Master, como indicaba la consigna, se utilizó una shared memory controlada por semaphores. La idea que se puso en práctica fue que al recibir un nuevo resultado que necesita ser escrito en el buffer, el Master, escribe el mensaje que recibe de los slave y mediante un semáforo permite que View lea el mensaje enviado.

Un diagrama ilustrando cómo se comunican los diferentes procesos



Instrucciones de compilación y ejecución

Para compilar el programa desde docker, solo hay que correr el comando `Make all`. Si se quiere compilar al programa fuera del entorno provisto del docker, es necesario instalar `Make` y `GCC`. Para ejecutar al programa es necesario instalado minisat, tanto para el entorno provisto por Docker como para cualquier distribución de Linux.

Para ejecutar el programa sin salida en la terminar simplemente hay que correr el siguiente comando:

```
./SatSolver files
```

Si queremos además correr el programa viendo en terminal, tenemos dos opciones:

- Podemos pipear la salida del programa principal con la vista. En este caso hay que ejecutar al siguiente comando:

```
./SatSolver files | ./SatView
```

- Podemos ejecutar a los dos programas de forma simultánea en dos terminales distintas. Para lograr esto hay que ejecutar primero a la aplicación principal y luego ejecutar a la vista. La vista al iniciar espera un código para conectarse con master, ese codigo es el mismo que master imprime en pantalla al inicializarse.

files corresponde a todos los archivos deseados, ya sea files/* o una lista file1 file 2… file n.

Limitaciones

La shared memory tiene un tamaño predefinido, por lo que hay una cantidad máxima de caracteres que se pueden almacenar en ella. En nuestro caso, tomamos la decisión de que el programa puede soportar hasta 700 archivos para que el tamaño de esta memoria no sea muy grande. Es posible que el programa pueda procesar más de 700 archivos, pero esto sucede solo si los archivos a procesar tienen pocas cláusulas, por lo que no recomendamos ejecutar a este programa con más de 700 archivos. De fallar el programa por este motivo, el siguiente error ocurrirá:

"The futex facility returned an unexpected error code"

Problemas encontrados durante el desarrollo y cómo se solucionaron

Nos encontramos con un problema a la hora de comunicarle a la vista que tiene que cerrarse. Nuestra idea inicial era que termine su ejecución cuando reciba por la shared memory un string vacío, pero esto generaba algunos falsos positivos por lo que tuvimos que repensar nuestra solución. Finalmente nos pusimos de acuerdo en que se cierre View cuando en los últimos caracteres recibe el siguiente string `&exit;`. En programación web se usa esa notación para escapar caracteres que tienen una interpretación como código además del texto (por ejemplo `&` termina siendo `&`).
Aunque no estamos completamente satisfechos con nuestra solución, podemos decir que funciona perfectamente.

Otro problema con el que tuvimos que lidiar sucedió a la hora de repartir archivos a los Slaves. Inicialmente repartiamos 2 archivos a cada Slave mediante sus argumentos y luego mientras iban respondiendo les íbamos asignando más. El problema estaba en que había ocasiones donde se le enviaban por el pipe dos archivos al mismo tiempo, y nuestro Slave no podía identificar si le enviaban un archivo o dos.

Nuestra solución fue darle al Master la capacidad de controlar cuantos archivos les mandó a cada slave. Con este mecanismo, implementamos mediante una macro que se pueda elegir la cantidad de archivos iniciales y el Master empieza a enviarle archivos al Slave cuando termina de procesar sus archivos iniciales.

También se nos ocurrió otra solución, que consiste en agregarle un carácter separador al enviar los archivos, ya que en el caso de que se envíen dos archivos rápidamente y lleguen juntos el slave podría parsear el string y procesarlos de forma correcta. Esta forma de encarar al problema parece mejor que la anterior, ya que no involucra cambios en la forma en la que se maneja Master, pero también implicaba que si queremos ejecutar el Slave solo, tendríamos que manualmente agregar el carácter separador. Por esta razón, y para mantener la independencia del Slave, decidimos resolverlo de la forma mencionada anteriormente.

Citas de fragmentos de código reutilizados de otras fuentes

No recordamos haber usado código fuentes externas. Utilizamos el Manual de Linux en el caso de funciones como `select()`, `pselect()`, y las correspondientes a las API's de semaphores y shared memory. No obstante, los ejemplos vistos en clase y la utilización del foro fueron suficientes para comprender el funcionamiento.