TP1 - Inter Process Communication

Sistemas Operativos 72.11

2021-2Q

Grupo: 3

Integrantes:

Baliarda Gonzalo, 61490

Pérez Ezequiel, 61475

Ye Li Valentín, 61011

Decisiones

Métodos de IPC

Utilizamos dos métodos de IPC: unnamed pipes y shared memory.

Usamos unnamed pipes en vez de named pipes para conectar el master con los slaves, ya que fueron los que implementamos en las clases teóricas y estamos más familiarizados con ellos.

La shared memory se usó para conectar los procesos master y view. Utilizamos el estándar System-V dado que lo vimos en la teórica y teníamos algunos ejemplos. Por recomendación de la cátedra, utilizamos el PID del master como key a la hora de crear la shared memory.
Dentro de la estructura de la shared memory, tenemos un array de resultados que puede almacenar hasta 25 outputs distintos. Creemos que con esa cantidad debería ser suficiente para procesar una buena cantidad de archivos en cada llamada.
Este mecanismo de “slots” simplifica la lectura, escritura y sincronización entre la view y el master.

Mecanismos de sincronización

Usamos un sólo semáforo para sincronizar la lectura y escritura de la shared memory, que es el único lugar donde se pueden generar condiciones de carrera.
El mismo es incrementado por el master cada vez que se escribe un resultado en la shared memory, y es decrementado luego por la view cuando los va leyendo.

Utilizamos el estándar POSIX dado que era mucho más simple y legible a la hora de escribir el código.

Slaves y conexión con master

Decidimos utilizar una cantidad máxima fija de 5 slaves, dado que es la cantidad que se usa de ejemplo en la consigna y consideramos que es una cantidad que permite procesar los archivos con una buena velocidad y sin malgastar recursos del computador. Si hay menos de 5 archivos, la cantidad de slaves será igual a la de archivos.

Hicimos que cada slave procese de un archivo a la vez, incluso al inicio, ya que estuvimos haciendo pruebas y encontramos que era lo más óptimo en términos de tiempo y simplicidad.

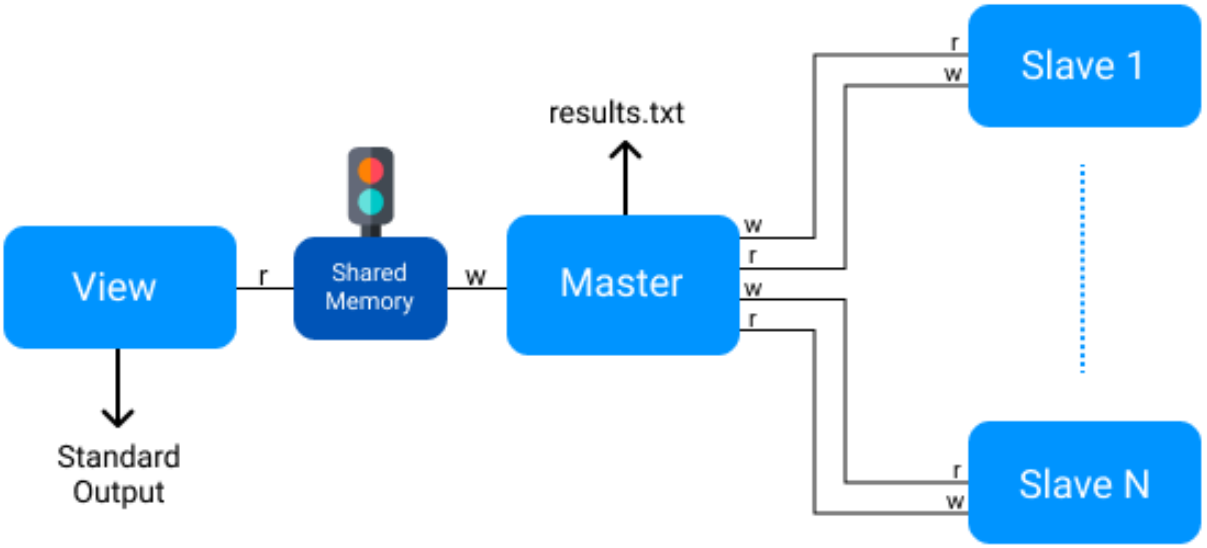
Cada slave está conectado al master mediante dos pipes, uno que le permite leer y otro escribir. De esta forma, podemos utilizar la syscall “select” desde master y obtener un identificador de los esclavos que terminaron y están listos para recibir una nueva tarea.

Uso de TADs

Implementamos 2 TADs para simplificar y unificar comportamientos relacionados. De esta forma logramos un código más limpio y seguro, además que fue recomendado por la cátedra.

- masterADT reúne el comportamiento del proceso aplicación.
- shMemHandlerADT se encarga de abstraer el uso de la shared memory junto con el semáforo.

Diagrama de conexiones entre procesos



Instrucciones de compilación y ejecución

Compilación

Ejecutar *make* o *make all* en la carpeta base para compilar los archivos.

Luego de compilar se generan los siguientes ejecutables:

- SATSolver.out
- slave.out
- view.out

Para remover los archivos creados, ejecutar *make clean* desde el mismo directorio donde se realizó *make all*.

Ejecución

Para correr el programa ejecutar desde la carpeta base alguna de las siguientes formas:

- *./SATSolver.out <CNF files>* | *./view.out*
- *./SATSolver.out <CNF files>* y durante la ejecución del programa, *./view.out* <***ID***>, donde **ID** es el número que imprime **SATSolver** por salida estándar.

Limitaciones

- Los resultados que envía cada slave tienen una longitud máxima de 256 caracteres.
- Hay en total 25 slots para resultados de archivos. Superado ese límite, el master abortará.
- Limitaciones que pueda llegar a tener minisat.

Problemas y soluciones

Tuvimos 4 problemas principalmente:

1. Intentamos utilizar un “unnamed semaphore” en un inicio, pero no logramos compartirlo entre procesos usando shared memory, a pesar de investigar en diferentes lugares.
   La solución fue cambiar a un “named semaphore”, el cual fue mucho más sencillo de utilizar.
2. Durante cada lectura y escritura en la shared memory cambiabamos el estado del semáforo para que solo se pudiera leer ó escribir en la shared memory y esto causaba busy waiting.
   La solución fue que únicamente la lectura espere en caso de no haber nada que leer, y que la escritura suceda siempre. Como la lectura y escritura suceden en distintos slots del buffer, se garantiza la exclusión mutua.
3. A la hora de pipear el master con la view en la ejecución, el master solo imprimía el ID de la shared memory al finalizar su ejecución, y así la view nunca podía acceder a la shared memory.
   La solución fue utilizar la función “setvbuf” para que la escritura en salida estándar por parte del master fuera instantánea.
4. Al momento de implementar los pipes entre el master y los slaves, consideramos implementar un único pipe para pasar los resultados al master. Al avanzar por este camino, nos dimos cuenta que no sabíamos quién era el slave que había enviado el resultado y necesitaba recibir un nuevo archivo.
   La solución fue implementar un pipe de escritura para cada slave y utilizar un select para determinar qué slave está listo para recibir un nuevo archivo.

Recursos externos consultados

- *man* de Linux.
- Códigos extraídos del libro ‘The Linux Programming Interface by Michael Kerrisk’ subidos al campus.