Trabajo Práctico Nro. 1 (obligatorio): Inter Process Communication

Materia: Sistemas operativos

Comisión: S

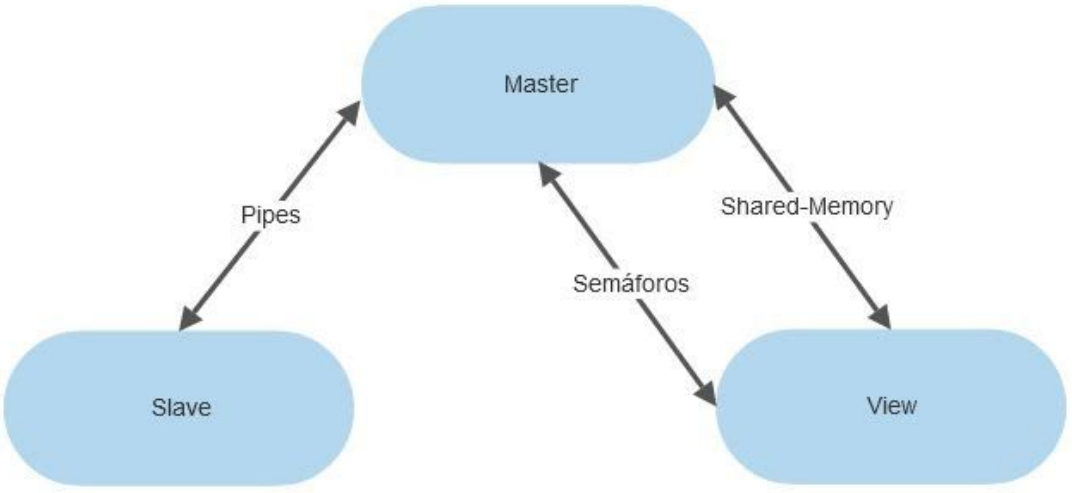
Integrantes:

- Mónaco Matías, 59102
- Lombardia Maximiliano, 56276

Decisiones tomadas durante el desarrollo:

- Para un manejo más agilizado de la memoria compartida, se decidió recorrerla en bloques fijos, el cual contempla una longitud máxima de path de archivos y longitud máxima del resultado. Para esto, se tuvo que establecer un límite para el nombre de los archivos. Para el recorrido se usó un puntero y un offset, el cual iría incrementando de N en N, siendo N el tamaño de ese bloque.
- A la hora de analizar qué parámetros se pasan a la vista se llegó a la conclusión (con consejo adicional de la cátedra) de que debería ser el tamaño de la memoria. Sin embargo, para esto existían dos opciones: pasar el tamaño real de la misma, o pasar la cantidad de archivos analizados e ir iterando en base a ese número, aprovechando que se recorría la memoria en bloques fijos. Para facilitar el manejo del programa, se optó por esta última opción.
- Para el manejo de la memoria, en un intento de "objetizar", se generó una estructura con el offset y los semáforos dentro de la misma. La mayoría del manejo de memoria se realizó con un puntero al inicio del buffer (externo a la estructura) y los "atributos" dentro del struct.
- Se decidió utilizar un par de pipes (uno de lectura y otro de escritura) por cada esclavo para comunicarse con el proceso master.
- La cantidad inicial de archivos que se le envía a los procesos esclavos se estableció que sea 5.
- Para distribuir inicialmente una cantidad de archivos significativamente menor al total entre los procesos esclavos, la cantidad de esclavos que se crean se calcula con la siguiente fórmula:
  ⌈⌈(file_qty*slave_qty_percentage/100)⌉/inicial_file_dispatch_qty⌉,
  siendo:
  - **file_qty**: cantidad total de archivos a analizar.
  - **slave_qty_percentage**: porcentaje sobre los archivos totales que se distribuirán inicialmente.
  - **inicial_file_dispatch_qty**: cantidad inicial de archivos que se le entregan a cada esclavo (en nuestro caso 5).

Diagrama de comunicación entre procesos:



Instrucciones de compilación y ejecución:

Compilación:

Para compilar los archivos master.c, slave.c y view.c podemos utilizar el 'Makefile' pedido escribiendo las instrucciones: **make clean** y **make all:**

- **make clean:** ejecuta las siguientes instrucciones para borrar los binarios y el archivo 'results.txt' de compilaciones y ejecuciones anteriores en caso que hubiera:
    - rm -rf master slave view
    - rm -rf results.txt
- **make all:** ejecuta las siguientes instrucciones para compilar con gcc los archivos master.c, slave.c y view.c utilizando los flags que requiera cada uno:
    - gcc -Wall -o master master.c utils.c -pthread -lrt -lm -std=c99 -D_XOPEN_SOURCE=500
    - gcc -Wall -o slave slave.c
    - gcc -Wall -o view view.c utils.c -pthread -lrt -lm -std=c99 -D_XOPEN_SOURCE=500

Ejecución:

Para ejecutar correctamente el programa es requisito tener instalado el programa 'minisat'. Luego, teniendo los archivos .cnf en una carpeta llamada 'files' en el mismo directorio que los binarios, hay dos formas de ejecutar el programa:

- Utilizando un pipe para iniciar el proceso aplicación y el vista: '**./master <file_paths> | ./view**', siendo <file_paths> las rutas de los archivos .cnf a procesar.
- Iniciando la aplicación y más tarde la vista: '**./master <file_paths>**' y luego el comando: '**./view <file_quantity>**', siendo <file_quantity> la cantidad de archivos que el master procesó y <file_paths> lo mismo explicado anteriormente.

Limitaciones:

- Las rutas de los archivos que se envían cómo parámetro al proceso master, no deben tener más de 128 caracteres incluyendo la terminación en null.
- El resultado entregado por 'minisat' corriendo el siguiente comando '**minisat %s | grep -o -e \"Number of.*[0-9]\\+\" -e \"CPU time.*\" -e \".*SATISFIABLE\""**', debe ser menor a 384 caracteres.
- Una vez que se ejecuta el proceso view, debe ejecutarse nuevamente el proceso master para re-instanciar a la memoria compartida, ya que sino el view buscará un fragmento de memoria que no está asignado.
- A la hora de manejo de memoria, por facilidad de evitar chequear dinámicamente cuántos archivos se procesan, se decidió crear la memoria con una cantidad de archivos fija, pero teniendo en cuenta un número que sea difícil de alcanzar. Para eso, se estableció que el programa soporta hasta 1000 archivos.

Problemas encontrados durante el desarrollo:

- Para los semáforos fue necesario discutir el manejo del mismo, ya que inicialmente fue ideado e inicialmente concebido de una manera (usando sólo un sem_post en master y sólo un sem_wait en view), pero luego de ser deliberado con el grupo se llegó a la conclusión de que faltaba agregar un sem_wait en master y un sem_post en view.
- El proceso vista intentaba consumir del buffer de memoria compartida aún cuando está estaba vacío, por lo tanto se optó por utilizar un segundo semáforo para que bloquee al proceso vista en el caso de que no haya información que consumir en el buffer.