Informe TP01 - **Sistemas Operativos**

Grupo 2

Integrantes

- Roberto José **Catalán**, 59174
- María Victoria **Conca**, 58661
- Gian Luca **Pecile**, 59235

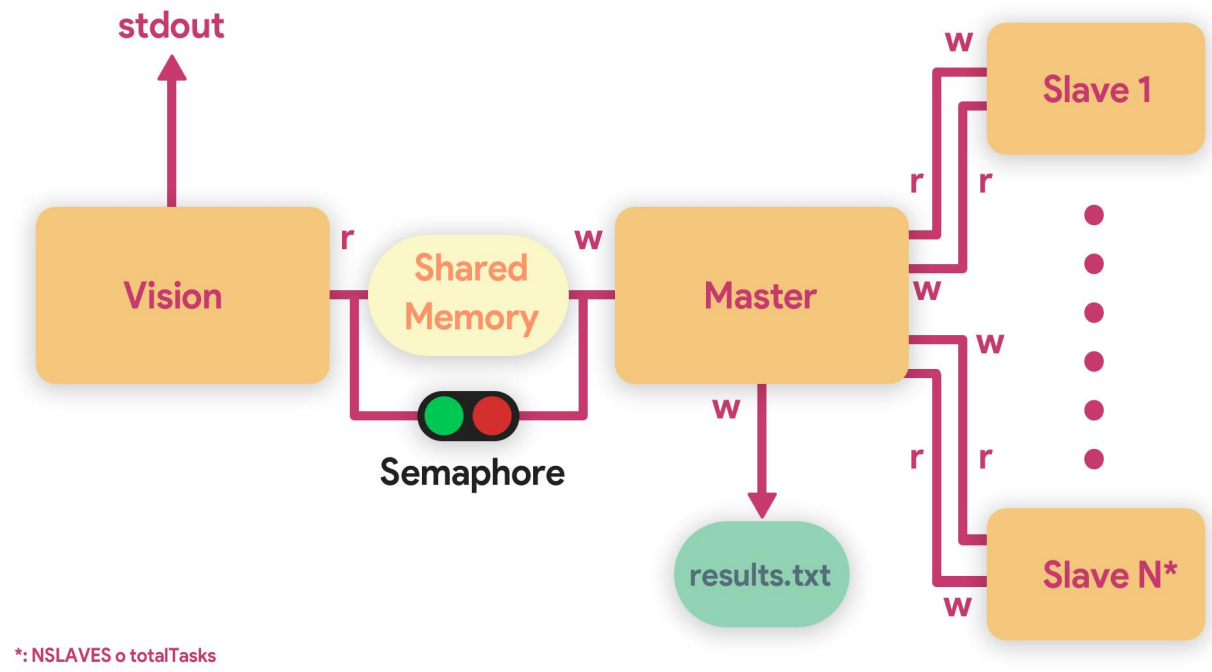
Decisiones

En el desarrollo de la aplicación se dió uso de dos medios de IPC, *pipes* y *shared memory*. No se optó por el uso de *named pipes* debido principalmente a la mayor comodidad del uso de pipes anónimos y la posibilidad de crear código que resulte más confuso por el nombramiento de los mismos, los unnamed *pipes utilizados* son unidireccionales.

Se decidió utilizar una cantidad fija de esclavos, dada por la variable *NSLAVES* donde fue definida como 5 en base a lo discutido en clase aunque si la cantidad de tareas es menor a la misma se crea la misma cantidad de esclavos que la cantidad de tareas. Además se le asignan dos pipes para cada esclavo en vez de un único pipe debido a las dificultades con la lectura y escritura atómica de los datos. Los esclavos poseen una cantidad fija de tareas, en este caso es una única.

Para las funcionalidades de *semaphores* y *shared memory* se decidió diseñar una librería para cada uno a fin de evitar la repetición de código. En el caso de *shm* se utiliza el modelo *POSIX* en vez del *SYSTEM-V*. Siguiendo la misma idea y preservando el mismo modelo a lo largo del TP se decidió utilizar el modelo *POSIX* para los mecanismos de sincronización. Los mismos son *named* debido a su facilidad de uso en comparación con los *unnamed* que se encuentran basados en memoria. Para la sincronización de los semáforos entre *Vision* y *Master* se optó por una lectura desfasada donde sólo uno hace el post y el otro hace el *wait*. Con este método se evita el bloqueo del *Master* ya que únicamente hace un *post* tal como se vio en clase.

Diagrama



Instrucciones de compilación y ejecución

Las instrucciones de compilación y ejecución están presentes en el readme dentro del repositorio de github.

Limitaciones

La posibilidad de hacer que *Slave.c* funcione de manera independiente es una de las limitaciones que se encontró a lo largo del proceso, en primera instancia funcionaba de manera independiente analizando únicamente los archivos que se pasaban por parámetro pero luego al implementar una espera utilizando *read* a causa de archivos que no se leían, se perdió el comportamiento independiente. Al consultar

con la cátedra por mail se nos sugirió la posibilidad de usar *getline* a modo de solución pero por limitaciones de tiempo no llegó a ser implementado.

Se asume que *minisat* no falla en cuanto a salidas de errores. Esto se debe a que recibe archivos de tipo cnf y se dificulta la validación de los mismos, lo cual se considera que extiende el alcance teórico del trabajo práctico.

Problemas y Soluciones

Se encontraron con algunas dificultades a la hora de implementar el select en el master. El problema residía luego de enviar EOF con read y que select continuaba seleccionando a dicho file descriptor. Se probó en primera instancia eliminarlo del set utilizando FD_CLR (macro sugerida por el manual) y no funcionó. Se logró solucionar el problema utilizando un flag dentro del struct del esclavo.

En cuanto a las tareas asignadas a cada esclavo se optó por una única tarea inicial por cada uno debido a que en el caso donde se ejecuta la aplicación con una gran cantidad de archivos se vio que generaba problemas en la lectura y escritura de memoria.

El buffer de *stdout* generaba problemas al enviar la respuesta entonces se utilizó *setvbuf* en tanto *Master* como *Slave* y *Vision* para el correcto funcionamiento.

Al diseñar *Vision* se encontraron problemas con su ejecución mediante un *pipe* donde la salida del mismo era errónea y prohibía la generación de los datos procesados en *results.txt* a causa de mal manejo de memoria. A la vez se encontró que la implementación del mecanismo de sincronización del semáforo estaba produciendo *busy waiting* en cuanto a la escritura por salida estándar.

Bibliografía

- Manual Linux, o bien *man*.
- Shared Memory y Semáforos.