Informe - Trabajo Práctico Obligatorio 1

Integrantes

- Ignacio Villanueva (Legajo: 59000)
- Rodrigo Fera (Legajo: 58079)
- Luca La Mattina (Legajo 57093)

Decisiones tomadas durante el desarrollo:

Master

El proceso master es el encargado de enviar las tareas a todos los slaves de la forma más eficiente. Por esta razón se decidió hacer que una vez inicializado el máster, se inicializan una cantidad de slaves (fijada por una constante), y se le envía una única tarea a cada slave. Cuando todos los slaves están trabajando, el master espera una respuesta mediante la función select(), lo cual evita un busy waiting. Una vez finalizada la tarea de un slave y devuelto el resultado al master, el master pushea el mismo resultado al shared memory del view, aumenta el semáforo de nuevos resultados, y le envia una nueva tarea al slave o lo mata si es que no hay más tareas. Para comunicarse con los slaves, se generan dos pipes unidireccionales para cada slave. De esta forma el slave y el máster saben si hay una nueva tarea o resultado, dependiendo si es que hay información en el pipe. Si es que se cierra el pipe del lado del máster, el slave sabe que su deber finalizó y puede finalizar el proceso. Si se cierra el pipe del lado slave, previo al del master, el master sabe que hubo un error en el slave, y puede reaccionar acorde. Una vez finalizado el máster, se limpia toda la memoria que fue utilizada en el proceso.

Slave

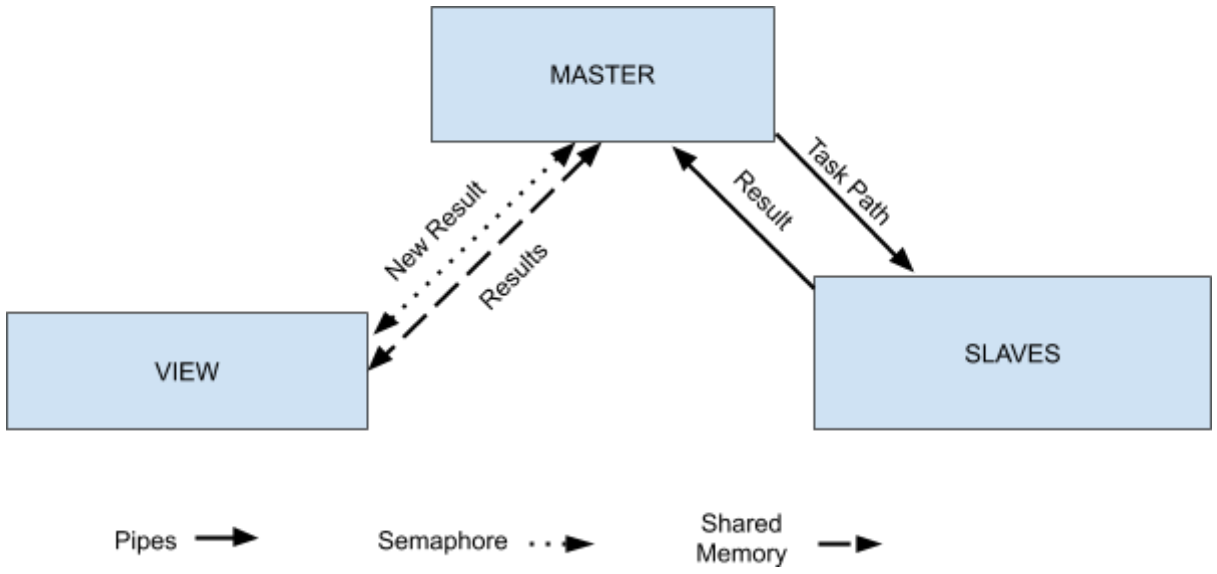
El proceso slave es relativamente simple. Al arrancar el slave espera hasta recibir una única dirección a un archivo CNF. Una vez que recibe la dirección ejecuta el programa minisat utilizando el archivo como parámetro, y el resultado lo parsea para adecuarse a lo pedido por el enunciado, mediante grep, argxs y sed. Todo esto se hace en una línea, utilizando popen(). Ahora el resultado de esto se retorna por stdout, y se loopea todo desde el principio. Al recibir cero bytes del read inicial, significa que el pipe del lado del master fue cerrado, lo cual el slave asume que no hay más tareas para procesar, y procede a limpiar la memoria y finalizar el proceso.

View

Al ser el view el encargado de imprimir los resultados de cada tarea se necesitaba de alguna manera saber cuántas tareas había pendientes para imprimir. Al inicio del proceso master se envia por stdout la cantidad total de tareas a procesar, lo que implica la cantidad total de resultados que el view debe esperar antes de finalizar. Se implementó un contador de tareas para saber cuántos resultados el proceso view imprimió, y al llegar a la cantidad notificada por el master, el proceso view finaliza.

Para poder acceder a la información generada por el master y los slaves, se creó un segmento de memoria compartida, para poder comunicar al proceso master con el view, donde cada línea del mismo se considera un resultado. Al mismo tiempo, se implementó un semáforo para saber cuando hay tareas nuevas. El mismo es aumentado por el master cada vez que empuja un resultado a la memoria compartida y el view hace un sem_wait() previo a imprimir un resultado nuevo, de esta forma el view sabe que hay un resultado para imprimir y se evita un busy waiting.

Relación de procesos:



Instrucciones de compilacion y ejecucion

Para compilar el se debe simplemente correr el siguiente comando en el working directory:

```
make
```

Una vez compilado el código, se deberían generar tres archivos binarios: *master.o*, *slave.o* y *view.o*.
El archivo master.o y view.o seran los archivos que se podrán ejecutar. Para ejecutar el máster simplemente hacemos:
./master.o [Archivos a Ejecutar]
dónde [Archivos a ejecutar] son todo los archivos que se desee que procesen los slaves. Al inicializar el máster se retorna un número por stdout, el cual se debe enviar al programa view, ya sea mediante un parámetro o utilizando un pipe de la siguiente forma:

```
./view [Resultado de master]
./master.o [Archivos a ejecutar] | ./view
```

Limitaciones

Si el proceso vista fuese cerrado y vuelto a abrir el contador de tareas interno de este se perdería, por lo tanto el proceso vista no sabrá la cantidad de tareas ya leídas hasta el momento, y por lo tanto no sabrá cuándo cortar la lectura del buffer, lo que podría generar que el proceso no funcione correctamente.

El proceso master es capaz de manejar, y continuar funcionando en el caso de que una cantidad menor a la totalidad de los slaves fallen. En el caso de que todos los slaves fallen, el master permanecerá un loop infinito, y ni el view, ni el master podrán finalizar sin una interrupción.

Problemas encontrados y sus soluciones

Al estar trabajando con conceptos nuevos, hubo momentos en donde se debió consultar varias al manual de linux para entender con mayor precisión cómo implementar ciertos aspectos. En sí, aquellos conceptos que mostraron mayor dificultad fueron: Pipes, Shared Memory, Semaphores y precisamente como implementarlos adecuadamente.