Alumnos: Beade, Gonzalo – Castagnino, Salvador – Hadad, Santiago
Profesores: Merovich, Horacio – Godio, Ariel – Aquili, Alejo – Mogni, Guido
Materia: Sistemas Operativos
Trabajo Práctico Especial: Inter-Process Communication

Introducción

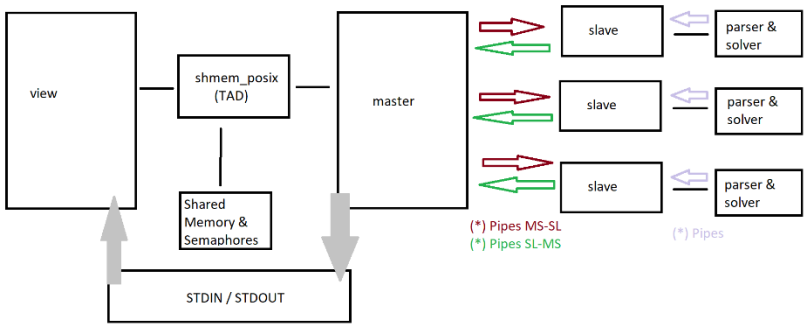
El objetivo del siguiente trabajo es explorar y analizar las ventajas de los mecanismos de comunicación entre procesos que ofrece Unix y Posix. El trabajo tenía un fin en particular: resolver problemas de SAT solving usando minisat, pero decidimos abstraernos de la problemática principal y diseñar el sistema para cualquier tarea de divisible.

Decisiones de diseño

La implementación se abstrae del problema de SAT Solving propuesto por la cátedra. De hecho, puede reutilizarse para cualquier sistema que requiera dividir el procesamiento de archivos entre varios procesos esclavos. El Makefile pide la definición de dos variables: `SOLVER` y `PARSER`, antes de compilar el ejecutable final, el usuario debe setearlos según la tarea que llevase a cabo. La primera variable es el path al ejecutable que procesará los múltiples archivos (por defecto, minisat). La segunda variables es un comando por Shell que parsea los resultados devueltos por el ejecutable. La existencia de estas variables permite una customización absoluta para correr cualquier software de procesamiento de archivos. Vale aclarar las variables están por defecto preparadas para trabajar con el minisat de la cátedra, de no ser seteadas.

Utilizamos dos mecanismos de comunicación entre procesos. El primero fue el uso de tuberías entre el proceso principal y sus esclavos. Tiene sentido que un proceso esclavo pueda existir y correr individualmente (de hecho, podríamos correr `./slave` para ver cómo podría funcionar unitaria e independientemente del resto del proyecto). Es un módulo con funcionalidad propia, que desconoce la existencia del todo que lo va a terminar rodeando y explotando. Las tuberías permiten canalizar las entradas y salidas de este proceso, para que pueda ser aprovechado y controlado desde un `master` central. En otras palabras, los procesos esclavos "*creen*" que están escribiendo y leyendo de salida estándar, pero en realidad fueron conectados por el master a tuberías que redirigen su flujo clásico. El segundo mecanismo que usamos fue la memoria compartida, entre el proceso maestro y la vista.

Diagrama del trabajo



Problemas encontrados y sus soluciones

El problema que más tiempo nos consumió fue el siguiente: no lográbamos que los pipes lleguen a un EOF. Cerrábamos sus extremos desde el master, pero el EOF nunca llegaba. El problema era que, por cómo lo implementamos, los slaves que se instanciaban últimos guardaban una referencia a los pipes de sus hermanos. Recordemos que el fork duplica el estado del proceso en su totalidad, incluida la tabla de descriptores abiertos hasta el momento. Fue bastante complicado descubrir qué ocurría, pero lo solucionamos cerrando los descriptores fraternales luego de llamar a fork().

Limitaciones

Se enumeran a continuación algunas limitaciones:

- La cantidad del bloque de memoria compartida está predefinida, pero puede modificarse dicho valor en el Makefile.
- La cantidad de archivos procesables debe ser un número de cómo mucho 5 cifras enteras.
- Los nombres de los archivos pueden ser de cómo mucho 256 caracteres.
- Los nombres de la memoria compartida y semáforos puede ser de cómo mucho 256 caracteres.
- La máxima longitud de resultado que puede devolver `PARSER` es de 300 caractéres (con el minisat no hay problema).