Trabajo Practico 1 de Sistemas Operativos

Integrantes: Tobias Brandy, Ignacio Sagüés, Faustino Pannunzio

La información necesaria para compilar y ejecutar se encuentran en el archivo readme en el repositorio,

Limitaciones

- Los archivos SAT tienen que ser válidos para el programa minisat.
- El path de los archivos SAT a resolver tiene como máximo 200 caracteres.
- Si hay un error en el medio de la ejecución no se liberan las zonas de memoria utilizadas.

Problemas encontrados

- Hacer qué view no haga busy waiting. Dado que este proceso tiene que ejecutarse sólo cuando hay nueva información para imprimir a pantalla, su tiempo de ejecución debería ser muy poco.
- Llevar el control de la cantidad de tareas enviadas y recibidas por master. Para poder saber si un hijo ya estaba libre para recibir tareas o no. Esto afecta a master porque necesita saber si ya le puede enviar un nuevo paquete de tareas.
- A la hora de enviar múltiples tareas, es necesario que el hijo pueda identificar dónde termina el path de un archivo y donde empieza otro.
- El proceso principal del archivo master quedó acoplado y fue difícil de modularizar. Había muchas variables de estado que se interconectaban y complicaba separar tareas.

Decisiones tomadas

Respecto a los problemas ya mencionados, las siguientes acciones se llevaron a cabo para solucionarlos.

- Para evitar el busy waiting se utilizó un semáforo que acompaña a la memoria compartida en la comunicación entre master y view. La semántica del semáforo es la cantidad tareas que están disponibles para leer. Para que funcione, es necesario saber la cantidad de tareas recibidas. Para esto se contaron la cantidad de \n, los cuales marcan el final de una respuesta.
- El proceso master, cada vez que envía una tarea guarda el valor de tareas pendientes que tiene el hijo en cuestión. Cuando recibe respuestas de este hijo, cuenta la cantidad de \n para saber cuántas respuestas fueron.

- En el caso de querer enviar múltiples tareas, el proceso hijo revisa la cantidad de \n en el mensaje recibido para saber cuantas tareas debe procesar antes de esperar nuevas tareas del padre.
- Respecto a la modularización se hizo lo mejor posible pero reconocemos que todavía es mejorable.

Por otro lado, las siguientes decisiones se tomaron a nivel general.

- Se decidió que la cantidad máxima de hijos seria 5. Este valor fue puesto de manera arbitraria y se podría subir de ser necesario. Lo mismo sucede con la cantidad máxima de argumentos que puede recibir un hijo cuando se lo crea.

Por último, la cantidad de tareas enviadas por ciclo se fijó en 1 a pedido de la consiga.

- Todos los paths fueron fijados. tanto el path al ejecutable del hijo, el archivo en el cual se deja la salida y los puntos de referencia en el filesystem para el semáforo y la memoria compartida.
- Se fijó un máximo para la longitud del path del archivo a resolver. Este fue de 200 para que, junto al tamaño de los datos, aproximadamente 50, pueda entrar en un buffer. Y, de esta forma, garantizar que la escritura sea atómica.
- Se utilizó una estructura para almacenar los file descriptors de los hijos, cuantas tareas se les ha enviado y su pid, pero finalmente este último dato no fue usado.
- Tanto la cantidad de hijos como la cantidad de tareas que reciben cuando empieza la ejecución se calcula en base a la cantidad de tareas. Elegimos un coeficiente del 1% para la cantidad de hijo y para la cantidad de tareas. En caso de ser muy chico, el mínimo valor es 1.
- Al comenzar el programa master, envía por salida estándar la cantidad de tareas que se deberán resolver para que el programa view pueda conocer este dato y pueda realizar el mmap. A su vez, master realiza un sleep de dos segundos para asegurarse que el programa view se haya configurado correctamente.
- El programa, principalmente, consiste de un while que se ejecuta hasta que todas las tareas sean procesadas. En cada iteración del ciclo:
    - Se realiza un select para encontrar algún hijo que tenga una respuesta. La ventaja de select es que nos permite bloquear mientras esperamos y no hacer busy waiting.
    - En base a los file descriptors se hace un read y se cuentan la cantidad de respuestas. Si ya no le quedan tareas pendientes a este hijo se le envía un paquete (en base a la variable ya mencionada) de nuevas tareas a resolver.
    - Se envían las respuestas recibidas al archivo de texto y se escriben en la zona de memoria compartida para que pueda leer el proceso view.
- En el caso de las variables request (child) y fdAvailable (master), cppcheck indica que se puede reducir el scope de las mismas. Nosotros decidimos dejarlas como están para mantener el código un poco más claro.

Proceso View

- Recibe la información necesaria por entrada estándar o por argumento. Esto permite que se ejecute utilizando un pipe o desde otro terminal.
- Utiliza el valor recibido para hacer el mmap y para saber la cantidad de tareas que va a recibir.
- Teniendo en cuenta la semántica del semáforo, el proceso view se desbloqueara solo si tiene una tarea para leer. Para saber que leyó todo la respuesta utiliza los \n que marcan el final de la misma, contandolos para saber cuantas tareas proceso.

Proceso hijo

- Recibe una cantidad inicial de tareas como parámetro.
- Cuando se queda sin tareas realiza un read para esperar nuevas tareas sin hacer busy waiting.
- Utiliza popen para correr minisat y grep. De esta forma se obtiene el resultado ya parseado.