2019

Sistemas Operativos

Informe TP N°1:
Inter Process Communication

Integrantes: Britos, Nicolás Ignacio (59529)

Mail: nbritos@itba.edu.ar

Ignacio Grasso (57610)

Agustín Roca (59160)

Fecha de entrega: 16 de septiembre 2019

ITBA

Informe TP N°1:

Inter Process Communication

1. Consigna:

1.1. Objetivo:

El trabajo práctico consiste en utilizar distintos tipos de IPCs presentes en un sistema POSIX en la implementación de un sistema que distribuye tareas de SAT solving entre varios procesos.

1.2. Requerimientos:

El sistema de SAT solving debe utilizar tres procesos distintos, de los cuales uno (la Vista) no es necesario para el correcto funcionamiento del mismo. Comenzando con el proceso Aplicación, este debe recibir por línea de comandos los archivos a procesar, debe ocuparse de instanciar una cantidad tal de procesos Esclavos con el fin de paralelizar el trabajo, de compartir información tanto con ellos como con el proceso Vista (eventualmente) y guardar los resultados de SAT en un archivo. El proceso Esclavo es el encargado de resolver cada archivo y devolverle la información al proceso Aplicación. Finalmente, el proceso Vista se encarga de mostrar los resultados de estas tareas por línea de comandos.

1.3. Restricciones:

Los procesos Esclavo deberán utilizar pipes (o named pipes) para conectarse con el proceso Aplicación y el proceso Vista deberá utilizar Shared Memory y semáforos para comunicarse con el mismo. Todo el sistema deberá de estar libre de deadlocks, race conditions y busy waiting.

2. Desiciones tomadas durante el desarrollo del trabajo

2.1. IPCs:

Para comunicarse entre si se usaron distintas tecnicas. Entre el proceso Aplicación y cada proceso Esclavo se utilizan dos pipes. En uno de ellos la aplicación escribe los archivos (filenames) que un determinado Esclavo debe procesar y este mismo los lee. En el otro, un determinado Esclavo escribe los resultados del archivo procesado con minisat y la Aplicación los recibe.

Con el fin de sincronizar las lecturas y de evitar busy waiting, se utiliza un semáforo entre el proceso Aplicación y cada uno de estos Esclavos, permitiéndoles a estos verificar que haya información disponible para leer en el pipe antes que intenten leer de él, como así también permitiendo que se bloqueen en caso de que no lo haya.

En el caso de la comunicación entre proceso Aplicación y Vista, inicialmente el proceso primer mencionado imprime su PID, el cual será necesario si se quiere correr el proceso Vista. Esto no solo permite que un eventual usuario le indique el PID a la Vista (como argumento o escrito en su entrada estándar), sino que permite que se ejecuten ambos procesos con sus entradas y salidas estandar pipeados.

Aparte de este detalle menor, la comunicación entre ambos se da con dos Shared Memory distintas. Esto se debe a que en una se guardan los resultados SAT en forma de struct en una (donde se incluyen todos los detalles importantes como el SlaveID, el nombre del archivo, el resultado, la cantidad de clausulas y variables, entre otros) y, en otra, todos los filenames. Esto es necesario puesto que no es posible almacenar un puntero en una Shared Memory ya que puede apuntar a cualquier lugar en un proceso diferente al manejar diferentes direcciones de memoria. Para almacenar estos filenames guardamos el tamaño de cada uno de ellos en los structs anteriormente mencionados, el proceso Aplicación agranda el Shared Memory donde se almacenan estos nombres de manera tal que todo este string entre y, finalmente, la Vista los imprime por pantalla.

Para que el proceso Vista tenga conocimiento de la cantidad de archivos que se procesaron (y que, por lo tanto, sus resultados SAT se encuentran en los Shared

Memory) utilizamos un semáforo del proceso Aplicación a Vista que incrementa su valor cada vez que un resultado SAT es registrado, y lo decrementa cada vez que la Vista imprime un resultado por pantalla.

Es necesario notar que, debido a esta naturaleza, los resultados son registrados en orden de llegada.

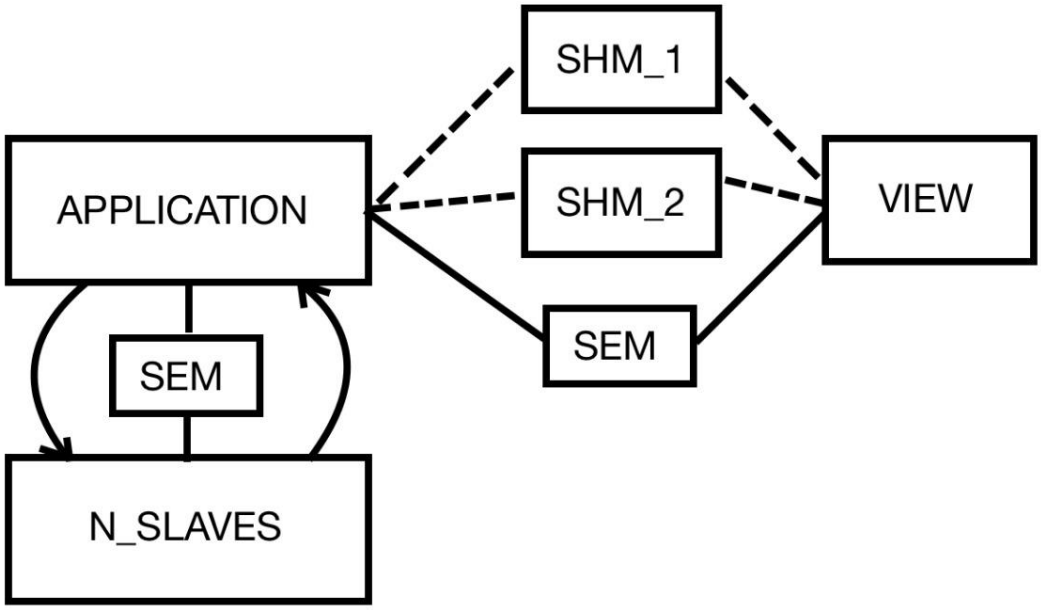

Figura 1: Diagrama de la relacion entre los distintos procesos.

3. Instrucciones de compilacion y ejecucion:

Con el comando “make” en la carpeta principal del proyecto se compila todo lo necesario para que se ejecute correctamente el mismo. Agregamos la opcion de “make install” con la finalidad de instalar minisat. No es necesario ejecutar este comando pero si es necesario que el sistema tenga instalado minisat para que el programa funcione correctamente.

Aquí cabe mencionar que, si el código es analizado con herramientas de análisis estático como PVS-Studio pueden saltar advertencias como que el archivo view.c no retorna en el main o que el puntero pasado a una llamada de strncpy en este mismo archivo puede ser NULL. Sin embargo, estas advertencias solo se deben a que estos casos se contemplan y se manejan con la función shutdown, el cual hace un exit, terminando el proceso y, por lo tanto, imposibilitando que estas líneas de código que fueron marcadas como advertencia se ejecuten.

Luego de ejecutar el comando “make” se generan varios archivos, uno de los cuales es “solve” y “view” (este último encontradose en ./src). Luego, para ejecutar el programa es necesario correrlo de la siguiente manera en una terminal de comandos (**no incluir los paréntesis**): “./solve [FILES]”. En el proyecto incluimos 2000 archivos .cnf de ejemplo en el directorio ./cnf. Para ejecutar el programa usando esos archivos se puede escribir: “./solve cnf/*.cnf”.

Tambien incluimos un comando “make clean” que elimina todos los ejecutables y archivos objeto que generó “make” y elimina, además, los pipes en /tmp que podrían haber quedado abiertos dado por algún problema de ejecución (por ejemplo, si se cortara la ejecución con CTRL + C).

Si se desea ejecutar la vista, se puede ejecutar haciendo “./solve [FILES] | ./src/view”.

4. Limitaciones

Debido al uso de memoria dinamica para el almacenamiento de cualquier informacion variable tanto de cantidad como de peso de la información, no hay limitaciones impuestas en estos términos. Aun asi, cabe mencionar que, por problemas de minisat, los archivos deben tener un filepath alfanumerico, sin espacios y/o paréntesis.

Además, si existen limitaciones presentes en, por ejemplo, la cantidad de archivos abiertos al mismo tiempo. Sin embargo, esto es una limitación del sistema donde se corre. Para tomar esto en consideración, el proceso Aplicación limita automáticamente la cantidad de Esclavos a ejecutar basado en el límite actual, teniendo en cuenta que por cada uno de ellos se necesitan abrir tres archivos (pipe de lectura, pipe de escritura y el semáforo).

5. Problemas encontrados

Durante el desarrollo del trabajo se encontraron los siguientes problemas que afectaron al funcionamiento del programa:

1. Inconsistencia con la devolucion de minisat en “CPU time” (en algunas maquinas devuelve int y en otras float) incluso corriendo desde Docker.
2. Problemas con el proceso Vista al mostrar los resultados, sin embargo, se debió a utilizar punteros a caracteres (Strings) dentro del Shared Memory compartido. Esto se soluciono con el segundo Shared Memory anteriormente mencionado.
3. Problemas al procesar más de un archivo en un Esclavo al mismo tiempo, lo que resultaba en que el proceso Aplicación leyera más de un resultado al mismo tiempo sin posibilidad de separarlos. Esto se debía a que como estos resultados son Strings en un pipe necesitan estar finalizados con el ASCII NULL. Sin embargo, por un bug este carácter final no era escrito y, por lo tanto, se leían mas de un resultado al mismo tiempo, haciendo que la aplicación finalice sin error aparente en cualquier momento. Este fue uno de los problemas que más tiempo nos ocupó.

6. Citas de fragmentos de código reutilizados de otras fuentes

Durante el desarrollo del trabajo no reutilizamos código de otras fuentes. Nos valimos de man y utilizamos algunos foros para guiar nuestras respuestas, pero el código aquí presentado es de producción enteramente nuestra.