Trabajo Práctico Especial n° 1

Inter Process Communication

72.08 - Sistemas Operativos

2020 1Q

VÁZQUEZ, Ignacio (59.309)

LEGAMMARE, Joaquín (59.376)

MANFREDI, Ignacio (58.513)

Introducción:

En el presente informe se exhiben las decisiones tomadas, los problemas encontrados y las soluciones al establecer comunicaciones entre procesos. Para llevar a cabo esta tarea se elaboró un programa que distribuye tareas de SAT solving entre varios procesos.

Limitaciones:

La única limitación encontrada fue que los archivos que recibe el proceso master deben ser correctos con extensión .cnf y no deben tener el carácter '%' al final.

Problemas encontrados:

Uno de los problemas encontrados ocurrió cuando se creaban los esclavos y se les asignaba las tareas. Teniendo en cuenta que las tareas resueltas eran devueltas por un pipe al master, en algunos casos se obtenían ambas tareas juntas en el pipe lo que dificultaba procesar si se trataba de una o más tareas. Además, traía como consecuencia que el tamaño del buffer donde se guardaba la información leída del pipe no era tan grande como para almacenarla es por esto que se decidió que el buffer sea del máximo tamaño en el cual el write todavía es anatómica, evitando así problemas de concurrencia.

Por otra parte, hubo que prestar atención al utilizar shared memory entre el proceso master y el proceso vista, dado que podía pasar que el primero intente escribir cuando el segundo estaba leyendo o viceversa, además, nos encontramos con otro problema a la hora de utilizar printf ya que la función por defecto escribe a la salida estándar sólo cuando se completa una línea (line buffered) y para solucionarlo tuvimos que especificarle a nuestros procesos que no se utilizara ningún buffer a la hora de realizar acciones de I/O mediante la función setvbuf.

Decisiones tomadas y solución a los problemas:

Para lograr la comunicación entre procesos en primer lugar, se debió establecer una conexión entre el proceso master y los procesos esclavos. Se crearon dos pipes por cada slave: El pipe llamado "pipeMtoS" se utilizó para que Master le envíe a Slave los archivos a procesar. Luego, a través del pipe llamado "pipeStoM" el Slave escribe la información de los archivos procesados por el comando minisat.

Para solucionar el problema de obtener más de una tarea en el pipe, se decidió que la información de cada archivo procesado termine con un '\n'. Por lo tanto, hubo que asegurarse que el único carácter especial correspondiera al del final. De esta manera, el Master puede parsear la información y sabe a cuántas tareas corresponden. Además se agrandó el buffer para que sea capaz de contener información de más de una tarea.

Se utilizó Shared memory para que la información obtenida por los esclavos a través del pipe, pueda ser compartida entre el proceso Master y el proceso Vista. Master se encarga de escribir la información procesada en el archivo llamado IPC_Information y posteriormente Vista se encarga de leerla e imprimirla por salida estándar.

Además para que el proceso Vista sepa cuántos archivos tiene que leer, el Master imprime en su salida estándar la cantidad de archivos que recibe por línea de comandos. De esta manera se puede pipear la salida de Master a Vista y obtener un correcto funcionamiento.

Para llevar registro de cuantas tareas tiene pendiente cada proceso esclavo y cual es su correspondiente file descriptor, se utilizó un array de estructuras de tipo "slave_t" que se inicializa al principio del programa junto a su creación. De esta forma, cuando un hijo tiene cero tareas pendientes, el proceso Master se encarga de enviarle una nueva.

Se decidió que el número de hijos sería cinco y que al principio, si hay una cantidad de tareas suficientes, se le envíe a cada uno de ellos dos tareas. Luego, una vez que no le quede ninguna pendiente, el hijo recibirá de a una tarea hasta que se terminen los archivos pendientes.

Finalmente, para poder lograr sincronización entre el proceso master y vista se utilizó un semáforo. De esta forma, el proceso Vista se desbloquea cuando el Master le informa que ya escribió en IPC_Information y puede leerlo sin pisarse entre sí. El semáforo indica cuantas tareas tiene para leer. Además estas se encuentran separadas por un '\0' entre ellas.

Instrucciones de compilación y ejecución:

Las mismas se encuentran detalladas en el README.md.

Citas de fragmentos de código reutilizados de otras fuentes:

Se utilizó el siguiente fragmento de código, al cual se le realizaron algunas modificaciones, para chequear que no se haya cometido algún error en el llamado de una función[1]:

```
#define CHECK(x) do { \
  int retval = (x); \
  if (retval != 0) { \
    fprintf(stderr, "Runtime error: %s returned %d at %s:%d", #x, retval, __FILE__, __LINE__); \
    return /* or throw or whatever */; \
  } \
} while (0)
```

Consideraciones:

- El archivo report.tasks marca un warning diciendo que charsRead puede ser -1 pero esto es verificado por la macro ERROR_CHECK. En caso de ser -1, es indicio que se produjo un error y se concluye con la finalización del programa.
- Dado que la escritura del slave en el pipe no implica una cantidad de caracteres muy grande, asumimos que el write que se realiza sobre el pipe es atómico. La función write es atómica siempre y cuando la cantidad de caracteres no supere a la definida en PIPE_BUF(4096 bytes).

[1] https://stackoverflow.com/questions/6932401/elegant-error-checking

Diagrama de funcionamiento del sistema: