Sistemas Operativos

TP Nº1 - Inter Process Communication

2021 1Q

Integrantes:

- Prado Torres, Macarena legajo: 59.711
- Rodríguez, Martina legajo: 59.036
- Tarantino Pedrosa, Ana legajo: 59.147

Fecha de Entrega: 05 de Abril de 2021

Introducción

En el siguiente informe se detallan las decisiones tomadas a lo largo del trabajo para la comunicación entre procesos. El programa se encarga de distribuir tareas del tipo SAT solving entre procesos.

Limitaciones

La principal limitación al momento de realizar este trabajo fue que los archivos de extensión .cnf debían ser válidos para el programa minisat.

Problemas encontrados

Uno de los problemas encontrados fue en las funciones mmap y munmap en el proceso vista. El error cometido acá fue que se le pasaba una variable size representando el tamaño que siempre estaba en cero. Esto era porque estaba relacionado con total_files que era una variable que debía recibir del proceso solve y que no se estaba leyendo correctamente.

Otro inconveniente fue que se estaba utilizando un contador que se estaba aumentando por demás y esto llevó a que se intenten procesar más archivos que los ingresados. Por ende, se estaba intentado acceder a lugares de la memoria que no correspondían a archivos de extensión .cnf y minisat no lograba procesarlos. La manera en que se solucionó esto fue corrigiendo donde se aumentaba el contador y controlando las condiciones del ciclo.

Por último, otro dilema fue el manejo de los semáforos y la memoria compartida. Cometimos el error de realizar un sem_unlink tanto en el proceso vista como en el solve. Como el proceso solve siempre termina antes, se realizaba el unlink del semáforo antes de tiempo y el proceso vista no lograba leerlo correctamente.

Decisiones tomadas

Lo primero que se realizó en este trabajo fue la comunicación entre el master y los hijos. Se optó por hacer dos pipes para cada hijo. Uno de ellos se encarga de mandarle desde el proceso master al proceso hijo los archivos a procesar, mientras que el otro se encarga de escribir toda la información de los archivos procesados. Toda esta información, junto a la cantidad de archivos que cada hijo tiene pendiente procesar se guardó en un vector de estructuras llamadas struct_slave creado al comienzo del programa.

Al momento de elegir la cantidad de hijos que iban a realizar el trabajo, se decidió poner una cantidad fija (ocho) ya que resultó ser lo más conveniente. Con esta misma lógica se decidió fijar en cinco la cantidad de archivos que inicialmente cada hijo debía procesar.

El programa esclavo en un principio procesa los primeros archivos recibidos al iniciar el programa aplicación y luego se encarga en un ciclo ir leyendo los nuevos archivos y procesarlos. Se decidió usar el separador '\t' para diferenciar los diferentes archivos. De esta manera, el vista puede reconocer cuando hay un nuevo archivo al encontrar el caracter. Otra decisión tomada fue utilizar la ayuda de la cátedra para procesar los archivos por minisat. Mediante un *sprintf* se copió en un buffer la respuesta obtenida al hacer un pipe entre procesar el archivo con minisat y la expresión regular provista.

Códigos reutilizados de otras fuentes

```
if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
{
   perror("socket failed");
   exit(EXIT_FAILURE);
}
```

Para el manejo de errores se intentó buscar la forma más efectiva de realizarlos. Se utilizó parte de este código (con modificaciones) para poder llevarlo a cabo.

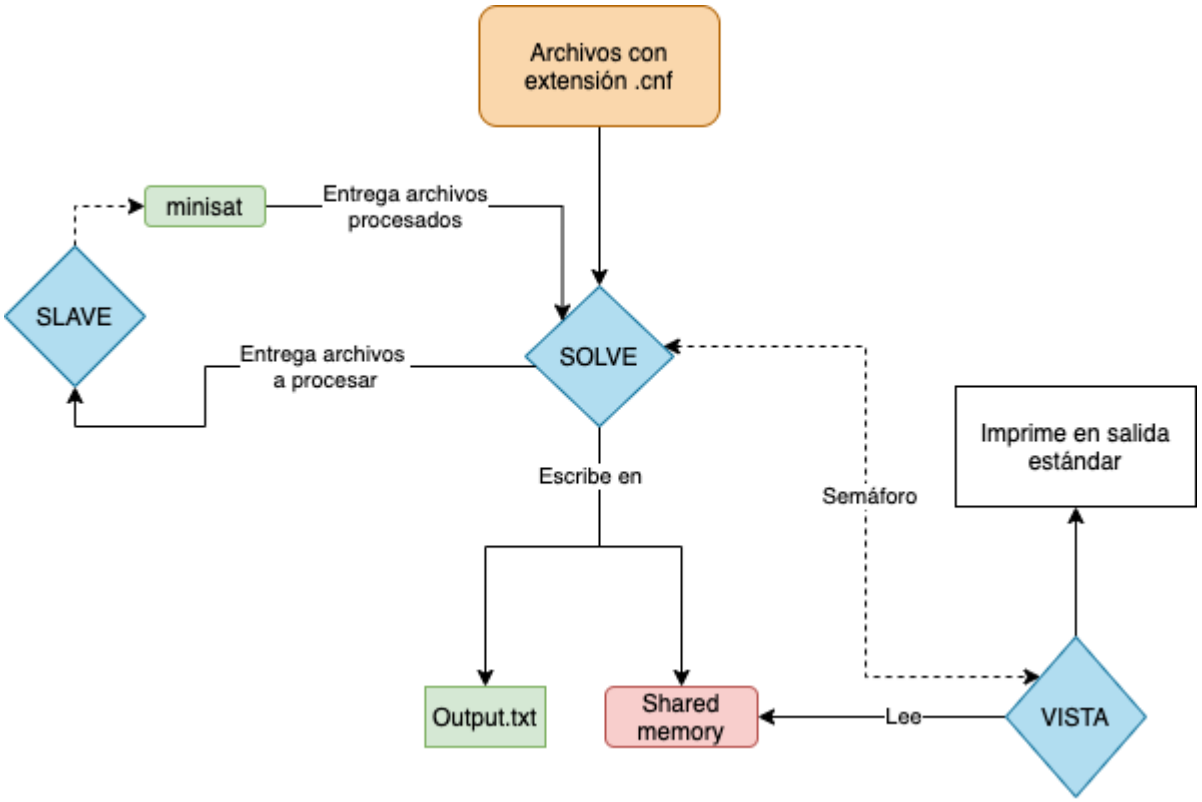
```
int socket_fd, result;
fd_set readset;
...
/* Socket has been created and connected to the other party */
...

/* Call select() */
do {
   FD_ZERO(&readset);
   FD_SET(socket_fd, &readset);
   result = select(socket_fd + 1, &readset, NULL, NULL, NULL);
} while (result == -1 && errno == EINTR);

if (result > 0) {
   if (FD_ISSET(socket_fd, &readset)) {
      /* The socket_fd has data available to be read */
      result = recv(socket_fd, some_buffer, some_length, 0);
      if (result == 0) {
         /* This means the other side closed the socket */
         close(socket_fd);
      }
      else {
         /* I leave this part to your own implementation */
      }
   }
}
else if (result < 0) {
   /* An error ocurred, just print it to stdout */
   printf("Error on select(): %s\", strerror(errno));
}
```

Este fragmento de código ayudó a entender cómo era el correcto uso de la función select.

Diagrama



Instrucciones de compilación y ejecución

Toda la información necesaria para la compilación del código se encuentran en el archivo README.md provisto.

Referencias

https://es.stackoverflow.com/questions/303411/exit-failure-en-c

https://www.geeksforgeeks.org/error-handling-c-programs/

https://gist.github.com/iago64/9f6acf5959846dc2a0f548ae3af7d34d

https://www.gnu.org/software/libc/manual/html_node/Feature-Test-Macros.html