ITBA

Sistemas Operativos

Profesores: Horacio Merovich y Ariel Godio

Trabajo Práctico 1
Inter Process Communication

Alumnos:

Lautaro Pinilla	57504
Micaela Banfi	57293
Joaquín Battilana	57683
Tomás Dorado	56594

8 de Abril de 2019

1 Introduccion

El siguiente trabajo práctico consiste en aprender a utilizar los distintos tipos de IPCs presentes en un sistema POSIX. Para ello se implementará un sistema que distribuirá tareas de cálculo pesadas entre varios pares.

2 Especificaciones técnicas

Las instrucciones de compilación y ejecución pueden encontrarse en el README. El trabajo práctico fue siempre compilado y ejecutado con la ayuda de un archivo makefile incluído en el repositorio como parte del trabajo práctico

3 Decisiones tomadas durante el desarrollo

3.1 Memoria Compartida

Se decidió utilizar una memoria compartida para el paso de información entre la aplicación y la vista, ya que asi lo requeria el trabajo práctico. Sin embargo, presenta varias ventajas, por ejemplo, la rápida escritura sobre la misma, una vez que está definida. A su vez presenta limitaciones, descriptas en la correspondiente sección.

3.2 Pipes

Para la comunicación entre la aplicación y sus esclavos se utilizaron tuberías o pipes. Se crearon un par de pipes de ida y de vuelta por cada esclavo, por lo que la comunicación con cada uno es completamente independiente. Application al crear al slave le manda por el pipe de ida un número (carga inicial) para que sepa cuantos archivos van a venir primero y luego le manda esos archivos. Después empieza a mandar de a 1. A su vez el hijo primero consume el número y sus archivos y despues comienza a leer de a una, para lo que manda un -1 al padre cada vez que esta listo para recibir mas archivos. Para que el padre pueda saber cuando los hijos ya procesaron el archivo que les mando y lo mandaron por el pipe utilizamos select. Cuando el padre queda sin archivos para mandar, envía un 0 a cada hijo para que al recibirlo termine su proceso.

3.3 Semáforos

Utilizamos semáforos para sincronizar la aplicación y la vista. El principal problema que teníamos era que necesitábamos una forma eficiente de avisarle a la vista cuando hay nuevos elementos para que lea en la memoria compartida. Entonces fue cuando decidimos utilizar un semáforo que permite que el proceso vista quede bloqueado hasta que la aplicación escriba un nuevo elemento y accione el semáforo que va a permitir que la vista pase a ser estar corriendo de nuevo.

4 Limitaciones

4.1 Memoria compartida

Una de las limitaciones que presenta el trabajo práctico es que soporta hasta 1000 hashes. Esto se debe a que, por facilidad, en vez de crear el tamaño de la memoria compartida en función de la cantidad de archivos, se crea un tamaño fijo.

4.2 Select

Una de las limitaciones de select es que los file descriptor que se puede usar tienen que ser menores a FD_SETSIZE, definido en la libreria. Para no tener esta limitacion en vez de select se puede utilizar poll, pero no nos pareció necesario para este trabajo práctico debido a que FD_SETSIZE en docker vale 1024 y para llegar a este número tendriamos que tener mas o menos 500 esclavos, cosa que no va a pasar.

5 Problemas encontrados en el desarrollo y posibles soluciones

5.1 Uso compartido de punteros entre procesos

Un problema con el que nos encontramos fue el uso compartido de punteros entre procesos. Inicialmente, por cuestiones de eficiencia espacial y para simplificar otros cálculos, creíamos que la mejor manera de manejarse con los strings de formato hash nombre archivo era guardando un puntero char *

que contuviera el string.

Esto causaba segmentation faults ya que el proceso Vision estaba accediendo a un puntero que correspondía a una zona de memoria fuera de la que le correspondía.

La solución que encontramos para este problema fue de abandonar la idea de compartir punteros y simplemente hacer una "copia profunda" del string.

5.2 Uso compartido de punteros en zona compartida

Otro problema que tuvimos, similar al anteriormente mencionado fue que queríamos tener un puntero al próximo elemento a agregar para facilitar la escritura y lectura. Pronto nos dimos cuenta que cuando nunca íbamos a poder crear este puntero ya que cada proceso accede a la memoria compartida virtualmente, con lo que sus direcciones a la memoria compartida son distintas.

La solución que encontramos para esto fue utilizar un valor de tipo size_t como offset de la memoria compartida, permitiéndonos la facilidad de acceder al último elemento sin los problemas mencionados.

6 Citas a códigos externos usados

Algunos de los archivos extraídos de internet fueron modificados para adaptarlos al uso que se les requería

Archivo	Link
Queue.c	https://codereview.stackexchange.com/questions/141238/implementing-a-generic-queue-in-c
QueueTest.c	https://codereview.stackexchange.com/questions/141238/implementing-a-generic-queue-in-c
Tasteful.c	https://github.com/lpinilla/Tasteful