ITBA

Sistemas Operativos

Profesores: Horacio Merovich y Ariel Godio

Trabajo Práctico 3 Estructuras de administración de recursos

Alumnos:

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

13 de Mayo de 2019

1. Introducción

En el siguiente trabajo práctico se mejorará el memory manager y scheduler ya realizados en el segundo trabajo práctico de la materia y se implementará un sistema de IPC por pipes y una solución al problema de los filósofos comensales.

2. Decisiones tomadas durante el desarrollo

2.1. Pipes

Para la implementación de Pipes se decidió implementar named pipes usando un estilo de file system. En cada proceso hay un array de file descriptors, que cuando se crea el proceso se crean con los fd principales al igual que en unix(STDIN, STDOUT, STDERR). Cada file descriptor apunta a un inode el cual tiene un path, la entry de la memoria a la que apuntaban (buffer), donde estaba parado el cursor y hasta donde podía avanzar el cursor. Así cada file descriptor tenía un tipo, que podían ser: STDIN, STDOUT, STDERR, READONLY, WRITEONLY. Depende de su tipo era si apuntaban, a la pantalla, teclado o las files creadas. Se crearon syscalls para crear files, y un wrapper para crear los named pipes. Luego se crearon syscalls para abrir o cerrar file descriptors, los cuales se usan para ingresar a la file. También se crearon syscalls para leer o escribir, las cuales recibían en que file descriptor querían escribir. En específico la syscall read(leer) se utilizó un semáforo para que el proceso que quiera leer y no tenga caracteres disponible se duerma hasta que tenga para leer. Después es responsabilidad del usuario que no se produzcan cosas como condiciones de carrera al leer.

2.2. Memory Manager

Para la implementacion del Buddy Memory Allocation utilizamos una implemetación de heap en la que guardamos los nodos dentro de la memoria con la unidad más chica de memoria de 1KB. En base a la cantidad de memoria total se calcula la cantidad de espacio que van a ocupar los nodos y se inicia la memoria ya fraccionada alocando espacio para los nodos de uso interno.

2.3. Scheduler

Se implementó un scheduler de multi nivel con prioridades para este trabajo. El mismo tiene 3 niveles de prioridades con una cola para cada uno. A cada una de estas se le asgina una cantidad de ticks. Para el manejo de las colas se creo una implementación de colas llamada queue, que se especializó para el uso particular del scheduler. Los niveles de prioridades se acomodaron de manera que 1 tiene menos prioridad y 3 tiene mas prioridad, desde userland. Tambien se implementó la llamada a sistema "nice"que se le pasa por parámetro que nivel de prioridad se desea modificar al proceso actual.

2.4. Filósofos Comensales

Para el problema de los filosofos comensales, se utilizaron un mutex para toda la mesa y un semaforo por filosofo. Al crear un nuevo filosofo, se crea un nuevo proceso y se le pasa por argumento el actual ID del filosofo. Este nuevo filosofo realiza siempre las acciones de Take Fork y Put Fork, estas se realizan entre el mutex. Para poder obtener los cubiertos, chequea si su estado es HUNGRY y si el estado de los filosofos adyacentes no es EATING. Teniendo esto en cuenta, irian comiendo los pares o los impares. El semaforo se utiliza para, si un filosofo no pudo comer, se lo pone en wait, hasta que se lo vuelva a llamar. Para la parte grafica, cada cuadrado representa un filosofo, con su respectivo color representando su estado. Estos estados se obtienen leyendo de un array creados en philosophers. Hay algunas limitaciones, ya que por ahi no refresca a tiempo o una vez impreso o lo que lee no es el correcto estado actual.

3. Limitaciones

La memoria total se recomienda que sea de un tamaño máximo de 256MB ya que sino la zona de heap que corresponde a las zonas de memoria de 1K sería demasiado grande y relentizaría todos los algoritmos. Esto se debe a que tenemos por cada tamaño una lista simplemente enlazada en donde en cada nodo se guarda el offset respecto de la base de la memoria. La cantidad de nodos crece exponencialmente a medida que se achica el tamaño (a razón de 2^n).

3.1. Procesos, Semáforos, Scheduler

Todas estas partes estan límitadas a un número máximo establecido por un define en cada uno de sus respectivos archivos, para que sea mas rápida su creación y el uso de los mismos.

3.2. Pipes

Se intento al principio implementar la función de fork, para realizar los pipes comúnes, pero se terminó implementando named pipes, para evitar implementar el fork. El limitador mas importante de Pipes es que su buffer tiene tamaño fijo y esta definido en una constante dentro de files. Como no usamos un buffer circular el pipe es lineal y puede llegar a su fin, por eso hay que tener cuidado de no pasarse. Al igual los fd tienen un maximo por proceso y hay una cantidad maxima de files que se pueden crear, todas definidas en constantes dentro de los archivos.

4. Problemas encontrados en el desarrollo y posibles soluciones

Un problema con el que nos encontramos era designar el tamaño mínimo de memoria que se podía solicitar. Inicialmente partimos de un tamaño mínimo de 4KB pero vimos que el promedio de los pedidos de memoria que íbamos realizando en otras partes del TP no superaban los 100 bytes, por lo que se generaba demasiada fragmentación interna.

Reducir el tamaño mínimo también tenía sus contras ya que ahora aumentamos considerablemente el tamaño del heap y por ende, reducimos la capacidad total de la memoria.

Aparte de eso, no pudimos encontrar el porque el intruction pointer a veces salta a una zona de memoria invalida. Creemos que es un error en alguna estructura que se propaga hacia algoritmos que la usan.

5. Citas a códigos externos usados

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

Archivo	Link
Tasteful.c	https://github.com/lpinilla/Tasteful
Idea de memory manager	Indicada en mem_manager.h