ITBA

Sistemas Operativos

Profesores: Horacio Merovich y Ariel Godio

Trabajo Práctico 2
Construcción del Núcleo de un Sistema Operativo

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 desarrollará la propia creación de un Kernel simple, en base al trabajo práctica de la materia Arquitectura de las Computadoras. Se implementarán mecanismos de IPC, Memory Management y Scheduling.

2 Decisiones tomadas durante el desarrollo

2.1 Memory Manager

Para el Memory Manager si pidió implementar el algoritmo First Fit. Para esto, utilizamos una lista doblemente enlazada en donde en cada nodo se guarda un puntero al espacio de memoria al que apunta y el tamaño del bloque de memoria.

Cuando un proceso solicita memoria, se recorre la lista buscando el primer nodo que satisfaga que su tamaño es mayor o igual al pedido. Si el tamaño del bloque es mayor, vamos a ver si lo podemos fraccionar en otro bloque de memoria, para esto, debemos ver que el tamaño remanente sea mayor al costo de alocar un nuevo nodo en la memoria, si lo es, se crea el nuevo nodo apuntando al resto de la memoria. Si no lo es, se genera una pequeña fragmentación en la memoria.

Al liberar la memoria, se trata de unir los bloques libres para transformar (de ser posible) en un único bloque. Para esto debemos analizar el estado del nodo anterior y del nodo siguiente.

2.2 Procesos

Para la estructura de procesos se le asignó a cada proceso un id, un parent id, un state que le informa al scheduler que hacer con este proceso, un puntero a su stack que fue reservado con el Memory Manager, un puntero a función en donde se encuentra el proceso y por último un número que indica la prioridad del proceso, que va del 1 al 4, siendo 4 el que más prioridad tiene.

Para poder indicarle al scheduler cuando un proceso terminó, se hizo una funcion wrapper que basicamente lo que hace es llamar al puntero a función

que tiene el proceso y una vez finalizada la función le indica al scheduler que mate al proceso. Esta funcion wrapper es la que se llama cuando el scheduler ejecuta a cada proceso, con su respectivo puntero a función.

También, para tener procesos en background y en foreground, lo que se realizó fue tener una variable estática con que proceso se esta ejecutando en foreground, y así solamente imprimir en consola al actual proceso foreground.

2.3 Scheduler

Para el scheduler se decidió que iba a ser una variable estática, la cual iba a contener una única lista de procesos circular. Siguiendo el algorítmo Round Robin, los procesos se agregan al final de la lista circular.

El handler del timer tick realiza en cada interrupción el cambio de contexto, que este llama a una función del scheduler que elige que proceso va a ser el siguiente en ejecutarse, dependiendo de su prioridad.

2.4 Semáforos

Los semáforos se decidieron hacer named semaphores, y su estructura posee un nombre, un id, un value que siempre comienza en 0 al crear el semáforo y por último una lista simplemente encadenada que contiene la referencia a los procesos que este semáforo este bloqueando. Esta lista esta implementada en FIFO, así los procesos que queden bloqueados por el semáforo, se desbloquearan, a medida que el value lo permita, en orden de llegada.

2.5 Mutex

El mutex se realizó utilizando la instrucción atómica XCHG, como fue recomendado por las presentaciones de la cátedra. Al mutex lock se le envía por parámetro una dirección de memoria en donde se encuentra un entero y lo mismo para el unlock. Esta variable entera la primera vez que se utiliza se debe inicializar en 0 para que funcione correctamente.

2.6 IPC

La forma de enviar mensajes entre procesos es parecida al problema Productor-Consumidor. Se utiliza un solo "Mailbox", reservado en Kernel Space en donde todos los mensajes son puestos ahí. Teniendo 2 mutex (uno para escritores y otro para lectores) y un semáforo que representa la cantidad de mensajes disponibles para leer.

Cuando un proceso quiere escribir, luego de haber adquirido el permiso (para ser el único que escribe), debe fijarse si hay espacio para un lugar nuevo, de no ser así, el proceso se bloquea. Si hay espacio, escribe el mensaje en la memoria indicando el PID del proceso a quien fue dirigido el mensaje.

Cuando un proceso quiere leer, luego de haber adquirido el permiso (similar razón a la anterior) recorre la lista verificando si hay algún mensaje con un id igual a su PID (Process ID), de ser así, marca el mensaje como leído y decrementa el semáforo para indicar que hay un mensaje menos para leer.

3 Limitaciones

Tanto como los procesos, los semáforos y los IPC estan limitados a un número máximo establecido en un define de cada uno de sus archivos, para que sea mas rápida la creación y uso de los mismos.

4 Problemas encontrados en el desarrollo y posibles soluciones

4.1 Semáforos

Un problema encontrado al desarrollar los semáforos fue que cuando se bloqueaba un proceso, habia que realizar un cambio de contexto, y simplemente con llamar a la función que lo hace no funcionaba, la manera en que se soluciono fue realizando una nueva interrupción, a la cual la ubicamos en el número 70, que esta interrupción llama al context switch.

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