Sistemas Operativos

TP2 - Construcción del Núcleo de un Sistema Operativo y estructuras de administración de recursos.

Segundo Cuatrimestre de 2019

Integrantes:

Donath, Holger Legajo: 58110
Rolandelli, Alejandro Legajo: 56644
Oliver, Felipe Legajo: 58439

Introducción

El objetivo del trabajo práctico dos es añadirle nuevas funcionalidades al trabajo práctico especial realizado en la materia Arquitectura de Computadoras (72.08) en base al kernel booteable por Pure64. Se incluyo Memory Management, Procesos, Scheduling, Mecanismos de IPC y Sincronización.

Decisiones tomadas sobre Scheduling y Administrador de Memoria

El trabajo de memory manager es manejar el uso de la memoria física, manejar su distribución para cada proceso. Nuestro memory manager arranca en 0x1000000 y tiene un tamaño de 128 megas. Tenemos dos algoritmos para el memory manager.

El primero es el first fit, este consiste en una lista de páginas, que al hacer una solicitud de memoria(malloc) recorre página por página hasta encontrar una que entre, una vez que la encuentra separa lo que sobra y devuelve la que encontró. Si no entra en ninguna página agrega las que necesite, las une, separa lo que sobre y devuelve la primera que agrego. Las páginas se agregan con un tamaño igual a la cantidad de memoria dividido la cantidad máxima de páginas.

El otro algoritmo es buddy system, este conciste en tener bloques de distinto nivel, donde cada bloque es de tamaño de dos elevado al nivel del mismo. La lista de páginas empieza con la cantidad máxima de bloques de máximo nivel que entren en la memoria. Al hacer una solicitud de memoria se chequea en qué nivel de menor tamaño entra y se divide el bloque libre de menor nivel en 2 bloques de un nivel menos y asa hasta llegar al nivel necesario para el commit.

Para la selección de que memory manager utilizar, hay que descomentar la que se quisiese usar y comentar la que no en memoryManager.c y memoryManager.h .

La tarea del scheduler es repartir el tiempo disponible en el microprocesador entre los procesos de manera equitativa para aquellos que están preparados (READY) para su ejecución. También cuentan con un nivel de prioridad, que les permite a los procesos de alta prioridad tener mas tiempo para ejecutarse. Este algoritmo es conocido como Priority-based round Robin. La idea es que cada proceso este conectado a otro proceso, en caso de ser uno, esta conectado a si mismo. Para cambiar de proceso, es gracias al timerTick que genera interrupciones, haciendo que el proceso cambie. Aunque los procesos de mayor prioridad duran mas timerTicks. También con respecto a los procesos que no están preparados para ser ejecutados, como por ejemplo procesos muertos o bloqueados, en esos casos los procesos son salteados en caso de estar bloqueados o retirados en caso de estar muertos. Se debe aclarar que el scheduler siempre debe tener procesos en su ciclo para procesar, por ende, al principio se generan procesos inservibles para que el sheduler no este vacío.

System Calls

Read	0
Write	1
Wait	2
Ball	3

Rectangle	4
Cursor	5
Malloc	6
Free	7
PrintPage	8
SemOpen	9
SemClose	10
SemWait	11
SemPost	12
MutexOpen	13
MutexClose	14
MutexLock	15
MutexUnlock	16
Set And Run Process	17
Kill Process	18
Run Process	19
Change Priority	20
Change State	21
Get Pid	22
Set Process	23
End Process	24
Print Process	25

Procesos, Context Switching y Scheduling

Para habilitar multiprocesos se implementó un sistema Scheduler que maneja una lista de procesos a ejecutar mediante el algoritmo Priority-based round robin (PBRR). Los procesos se encuentran representados en una lista de estructuras, los cuales contienen información sobre cada proceso. El Scheduler se ejecuta por cada interrupción del timerTick. Entonces en el proceso en el que se encuentra revisa si su quantum es igual a 0, en el caso de serlo pasa al siguiente proceso, de lo contrario se mantendrá en ese mismo. El quantum es cuantos timerTicks le quedan para seguir ejecutándose. El quantum se le asigna según la prioridad del proceso, alta prioridad quantum es igual a 3, baja prioridad quantum es igual a 1. También el scheduler revisa antes de ejecutar el proceso, si el mismo esta bloqueado o muerto. En el caso de estar muerto es retirado de la lista, si esta bloqueado se saltea. También se cuenta con el proceso Idle, para las situaciones en las cuales no haya ningún proceso para ejecutarse.

Cuando un proceso vuelve a ser ejecutado, es decir, que el ciclo de procesos completo una vuelta mínimamente, se guarda un estado del stack pointer y se devuelve el stack pointer correspondiente al nuevo y/o siguiente proceso. También en los casos de la ejecución de un nuevo proceso, se recrea el stack frame de interrupciones para que no se diferencie de un cambio de contexto.

Sincronización

Se generaron mutex, mecanismo de exclusión mutua, y semáforos, método de señalización entre procesos.

Primero se explica el mutex ya que el semáforo cuenta con un mutex dentro. La cantidad de mutex permitidos es de 50, todos aquellos mutex creados y en uso, se encuentran guardaos en un vector estático. Cada uno tiene asignado un nombre para identificarse del resto. Su funcionamiento es simple, funciona mediante un unlock y un lock.

Por el otro lado los semáforos, también cuentan con una misma cantidad (50) y están guardados en un vector estático. Cuentan con un valor previamente definido al momento de ser creado, donde un SemPost incrementa ese valor y SemWait viceversa. Su utilización es requerida en situaciones donde existe la posibilidad de acceder/modificar valores por mas de una acción al mismo tiempo, generado problemas en la ejecución de un código. Por eso el semáforo previene estos escenarios, generado que se accedan/modifiquen valores uno a la vez.

IPC

Utilizamos un vector estatico como con los semáforos y los mutex para los pipes, y una estructura para sus datos. Nuestro mayor problema fue la liberación de los pipes, dado que tomamos en cuenta que debe ser cuando dos procesos cierran el fd del pipe o cuando se mata el proceso con ese pipe. Como no tenemos una conexión con el proceso es manejada con un counter de users pero no lo consideramos muy eficiente.

Drivers

Se utilizaron los drivers de teclado y video implementados en Arquitectura de Computadoras.

Aplicaciones de User Space

Las aplicaciones que tenemos hechas son memtest, que te permite ver el un poco del funcionamiento del memory manager y la cantidad de páginas siendo usadas en ese momento. También printProcess para ver los procesos, y nice para cambiar la prioridad de un proceso. Nos faltan muchas aplicaciones que debido a quedarnos sin tiempo no pudimos realizar.

Problemas encontrados durante el desarrollo y cómo se solucionaron

Con memory manager tuvimos problemas con donde asignarle memorias a las variables, donde arrancar la memoria y donde poner las paginas, el resto de los problemas fue manejo de listas.

Una situación complicada fue al momento de enterarnos que el scheduler no podía estar vacío, generándonos problemas al principio del funcionamiento de nuestro sistema operativo ya que no se nos ocurría como hacer que el scheduler este funcionado sin algún proceso en el. El

problema estaba claramente al principio, por ende, se nos ocurrió como idea generar procesos de mantenimiento o inservibles al iniciar el kernel.

Limitaciones

Con respecto al manejo de memoria física, en el memory manager la cantidad máxima de bloques es de 1.000.000, una memoria física total de 128M y el tamaño de las paginas seria 128M/1.000.000 = 134 redondeado. Luego en el scheduler, los procesos en el algoritmo de Priority-Based Round Robin tienen 2 prioridades, high y low. Decidimos hacerlo simple a la cantidad de prioridades. El proceso de prioridad alta dura 3 timerTicks y los de prioridad baja duran 1 timerTick. También en el mutex hay una cantidad máxima, hay como mucho 50, y su nombre, en otras palabras, su id tiene un máximo de 40. Para los semáforos es un caso similar, hay como máximo 50 semáforos y su nombre como máximo 20.

Instrucciones de compilación y ejecución

Requisitos: Tener Docker y Qemu instalados

Luego se recomienda para el correcto compilado y ejecución del mismo, realizar los siguientes comandos en la terminal con todos los archivos en el mismo directorio.

Para la compilación, se cuenta con un makefile para su simplificación (en docker):

$~ make clean

$~ make all

Y luego para la ejecución…

$~ ./run.sh

Fuentes utilizadas

Round Robin Algorithm:

Link: https://www.geeksforgeeks.org/program-round-robin-scheduling-set-1/

Memory Manager:

Link: http://support.tenasys.com/INtimeHelp_62/ovw_memory.html
Link: https://www.win.tue.nl/~aeb/linux/lk/lk-9.html

Idle Process:

Link:
https://askleo.com/what_is_the_system_idle_process_and_why_is_it_using_most_of_the_cpu/