Trabajo Práctico 1

Sistemas Operativos

Integrantes:
Alfredo Santiago (66601)
Ramiro Luis Garcia (61133)
Simón Marabi (61626)

Año: 2024-1C

Grupo: 12

Profesores: *Ariel Godio , Alejo Ezequiel Aquili, Fernando Gleiser Flores, Guido Matias Mogni*

Introducción

Este trabajo práctico tiene como propósito implementar un gestor de memoria, procesos, planificación, mecanismos de comunicación entre procesos (IPC) y sincronización en un kernel básico, basado en el Trabajo Práctico Especial de la materia Arquitectura de Computadoras.

El informe incluirá las instrucciones para la compilación y ejecución, una descripción de los comandos implementados y su funcionamiento, y un análisis de las decisiones de diseño, modificaciones realizadas, problemas enfrentados y limitaciones encontradas durante el desarrollo. Finalmente, se ofrecerá una breve conclusión.

Instrucciones de compilación y ejecución

Dentro de una terminal de linux se deberán correr los siguientes comandos: **1-** Para la compilación:

$ cd TPSO/

$ docker run -v “${PWD}:/root” --privileged --rm -ti agodio/itba-so:2.0

$ cd root
$ cd Toolchain $ make all
$ cd ..
$ make all

Para la ejecución: $ ./run.sh

Para la limpieza de archivos binarios: $ make clean

Decisiones tomadas durante el desarrollo

En el desarrollo del Memory Manager y el Scheduler, hemos tomado decisiones cruciales para garantizar un funcionamiento eficiente y seguro del sistema operativo.

Para el Memory Manager, optamos por implementar el algoritmo de "buddy system", aprovechando su eficiencia en la asignación y liberación de bloques de memoria. Decidimos utilizar una lista enlazada para mantener un seguimiento de los bloques de memoria libres disponibles y garantizar la alineación adecuada de la memoria asignada. Además, implementamos un manejo básico de errores para casos de asignación insuficiente de memoria.

En cuanto al Scheduler, elegimos una estructura de datos basada en listas enlazadas para administrar los procesos listos y bloqueados, junto con información detallada sobre cada proceso. Optamos por un algoritmo de planificación de prioridades para determinar qué procesos ejecutar en qué momento, priorizando aquellos con mayor prioridad. Consideramos la interacción con el administrador de memoria para asignar recursos necesarios y garantizar la robustez y seguridad del sistema frente a posibles condiciones de carrera o errores inesperados.

Estas decisiones fueron fundamentales para el diseño y la implementación exitosa de ambos componentes del sistema operativo, asegurando un rendimiento eficiente y una ejecución segura de los procesos en el sistema.

Problemas encontrados durante el desarrollo

El tamaño y la complejidad del proyecto también presentaron un desafío significativo. La implementación de un kernel básico con todas sus funcionalidades, incluida la gestión de memoria, procesos, planificación, IPC y sincronización, resultó ser un proyecto de gran envergadura. Gestionar y coordinar todos estos elementos, asegurando que trabajaran de manera coherente y eficiente, requirió un considerable esfuerzo de diseño y codificación, así como una buena organización del trabajo y una gestión efectiva del tiempo.

La sincronización de procesos resultó ser un aspecto particularmente complicado. Garantizar que múltiples procesos pudieran acceder y modificar recursos compartidos sin causar condiciones de carrera o bloqueos requirió una planificación cuidadosa y una implementación rigurosa de mecanismos de sincronización, como semáforos y monitores. A pesar de los esfuerzos, surgieron problemas relacionados con la prioridad inversa y otros aspectos de la sincronización que complicaron el desarrollo.

Los problemas de compilación también fueron una constante durante el desarrollo. Dado que el kernel debía compilarse y ejecutarse en un entorno específico, cualquier pequeño error en el código podría resultar en fallos críticos que impidieran la generación del ejecutable. Además, la compatibilidad con diferentes versiones de herramientas y bibliotecas de desarrollo complicó aún más el proceso, requiriendo ajustes constantes y depuración meticulosa para asegurar que todos los componentes del sistema se compilaran correctamente y funcionaran como se esperaba.

Comandos Implementados

help: Imprime una lista de comandos disponibles junto con su funcionamiento.

divzero: Genera una excepción del estilo de división por cero.

invopcode: Genera una excepción del estilo de código de operación inválido.

time: Imprime la hora actual.

inforeg: Imprime los registros en el momento en que se haya tomado una captura de pantalla con la combinación de teclas CTRL + R.

clear: Limpia la shell.

testMM (size): Ejecuta el test de memoria proporcionado por la cátedra. El argumento "size" indica la cantidad de memoria a utilizar en MB.

testProcesses (quantity): Ejecuta el test de procesos proporcionado por la cátedra. El argumento "quantity" indica la cantidad máxima de procesos a crear.

testPriority: Ejecuta el test de prioridades proporcionado por la cátedra.

testSynchro (increments, quantity, flag): Ejecuta el test de sincronización proporcionado por la cátedra. El argumento "increments" determina la cantidad de incrementos a realizar, "quantity" la cantidad de procesos que realizarán los incrementos y "flag" determina si se utilizarán semáforos o no (1 = sí, 0 = no).

cat: Imprime el stdin tal como lo recibe.

loop: Imprime un saludo junto con el PID del proceso tras una determinada cantidad de segundos.

wc: Cuenta las líneas del stdin e imprime su resultado al recibir la señal EOF.

filter: Imprime el stdin filtrando las vocales.

kill (pid): Mata el proceso vinculado con el PID especificado.

ps: Imprime todos los procesos que se están ejecutando seguido de su información correspondiente.

phylo: Ejecuta el problema de los filósofos. Se pueden agregar filósofos al presionar la tecla A y reducir los mismos al seleccionar la tecla R.

nice (pid, priority): Asigna un valor de prioridad a un proceso vinculado con el PID especificado.

block (pid): Cambia el estado de un proceso vinculado a ese PID.

printmem: Imprime el estado de la memoria.

Limitaciones

El diseño actual del scheduler presenta algunas limitaciones significativas que pueden afectar su eficiencia y escalabilidad en entornos de sistemas operativos más complejos. Por ejemplo, el uso de una única lista enlazada circular tanto para los procesos listos como para los bloqueados puede generar ineficiencias operativas a medida que el número de procesos aumenta. Además, la falta de un mecanismo de envejecimiento podría exponer al sistema a problemas de inanición de procesos de baja prioridad. Estas limitaciones resaltan la necesidad de una revisión exhaustiva del diseño del scheduler para abordar estas deficiencias y garantizar un rendimiento óptimo en una variedad de escenarios de ejecución

Modificaciones realizadas a las aplicaciones de test provistas

Se agregó un exit para los procesos creados en test_sync, se modificó en todos los test la entrada de argc debido a que nuestra shell crea los nuevos procesos con un argumento que incluye el nombre del proceso. Se modificó el número de los arreglos de argv en los test que requieren parámetros.

Conclusión

En conclusión, este trabajo práctico ha permitido consolidar los conocimientos obtenidos en la materia de Sistemas Operativos, abarcando desde la implementación de un kernel simple hasta la gestión de recursos y la sincronización de procesos. Asimismo, el proyecto ha evidenciado la complejidad y los retos que implica el desarrollo de un sistema operativo.

Por otro lado, a lo largo del desarrollo de este trabajo práctico, se han reafirmado los conceptos estudiados en la materia de Sistemas Operativos, desde la creación de un kernel básico hasta la administración de recursos y la coordinación de procesos. Este proyecto también ha puesto de manifiesto la dificultad y los desafíos inherentes al desarrollo de un sistema operativo.