Sistemas Operativos

Primer cuatrimestre 2018

TP Nº 2: Construcción del Núcleo de un Sistema Operativo

Profesor: Horacio Merovich
Adjuntos: Ariel Godio, Rodrigo Rearden
Fecha de entrega: 9/5/2018

Introducción

Durante el transcurso de la materia aprendieron a usar la API de sistemas operativos de tipo UNIX, ahora en cambio crearán su propio kernel simple en base al trabajo práctico final de la materia anterior, Arquitectura de Computadoras. Para ello implementarán mecanismos de IPC, Memory Management, y Scheduling.

Grupos

Continuarán con los grupos ya formados.

Requisitos Previos

Es necesario contar con una versión funcional del trabajo práctico de Arquitectura de Computadoras (un kernel monolítico de 64 bits, manejo de interrupciones básico, con system calls, driver de teclado, driver de video en modo texto y binarios de Kernel Space y User Space separados). Si el sistema tenía problemas de memoria, como por ejemplo strings corruptos, es necesario arreglarlos antes de comenzar el trabajo.

En este paso se les recomienda el uso de una de dos soluciones:

- Generar un cross compiler de 64 bits, pueden usar este sitio como referencia.
- Utilizar Docker como ambiente de desarrollo host portable entre distintas máquinas y OSes (se verá en clase).

Requerimientos

Para la primera entrega el kernel debe implementar las siguientes features. Se les recomienda implementarlas en el orden en el que son enumeradas.

System calls

Las system calls son la interface entre user y kernel space. El sistema deberá proveer system calls para que los procesos (explicados más adelante) interactúen con el kernel. Utilice exactamente el mismo mecanismo desarrollado en Arquitectura de Computadoras (interrupciones de software).

Physical Memory Management

El kernel debe contar con algún sistema simple para reservar y liberar páginas de al menos 4 KiB contiguos (page allocator), note que esto no está atado a memoria virtual/paginación, el requerimiento es solamente reservar y liberar bloques de memoria física. Quién utiliza esta memoria puede ser el kernel para sus estructuras internas, o un proceso en user space.

Syscalls involucradas

- Reservar memoria para el proceso que llama
- Liberar memoria del proceso que llama

Procesos, Context Switching y Scheduling

El sistema debe contar con multitasking pre-emptivo en una cantidad variable de procesos. Para ello el sistema deberá implementar algún mecanismo que permita suspender la ejecución de un proceso y continuar la ejecución de otro (context switching), sin requerir que los mismos entreguen el control del CPU, y con alguna estructura/algoritmo que permita seleccionar al siguiente proceso (scheduler).

Syscalls involucradas

- Crear, Finalizar, y Listar procesos

IPCs

Se debe implementar envío y recepción de mensajes bloqueantes, de cantidad fija de bytes, enviados a un identificador común de tipo cadena de caracteres definida entre los procesos que van a comunicarse (puede ser una combinación de PIDs, un nombre de dominio, u otra cosa).

También deberán implementar mutexes, los mismos pueden ejecutar sus operaciones de up y down sobre un identificador acordado.

Syscalls involucradas

Send y Receive para mensajes, siendo la segunda bloqueante. Up y Down para mutexes. El diseño del scheduler debería contemplar estas syscalls.

Drivers

Teclado y video en modo texto

Use el driver de teclado implementado en el TP anterior. Si hiciera falta se requiere implementar las system calls pertinentes para aislar kernel de user space.
Use el driver de video implementado en el TP anterior.

Aplicaciones de Userspace

Para mostrar el cumplimiento de todos los requisitos anteriores, deberán desarrollar varias aplicaciones, que **muestren el funcionamiento del sistema** llamando a las distintas system calls.

Aplicaciones mandatorias

- **sh:** shell de usuario que permita ejecutar las aplicaciones. Debe contar con algún mecanismo simple para determinar si va a ceder o no el foreground al proceso que se ejecuta, por ejemplo, bash cede el foreground cuando se agrega un & al final de un comando.
- **ps:** muestra la lista de procesos con sus propiedades, PID, nombre, estado, foreground, memoria reservada, etc.
- **prodcons:** muestra una resolución para el problema productor consumidor de buffer acotado, se puede incrementar/decrementar en runtime la cantidad de consumidores y productores.
- **help:** muestra una lista con todos los comandos disponibles
- Es muy importante que agreguen sus propias aplicaciones prácticas **para demostrar el funcionamiento de cada una de las capacidades del sistema**, de otra forma la existencia de las mismas no podrá ser evaluada. Por ejemplo, para mostrar la protección de memoria, debería haber una aplicación que intente acceder a una zona inválida, y sea bloqueada por el kernel.

Consideraciones

Es necesario que programen de forma modular y desacoplada, en un siguiente TP se pedirá que reemplacen alguna parte del kernel con algoritmos de mayor complejidad. Asimismo se deberá respetar a rajatabla la separación a nivel binario y memoria entre kernelspace y userspace, las apps de usuario se comunican entre sí y con el kernel solo a través de system calls.

Armar casos de prueba, transformarlos en testeos unitarios y derivar las features del sistema operativo no es mandatorio, pero si es muy recomendado, en especial para los algoritmos más complejos.

Asimismo, si se programa de forma desacoplada y a partir de testeos unitarios, utilizar valgrind para revisar errores de acceso de memoria se vuelve trivial, esto es importante ya que debuggear un sistema operativo en runtime es posible, pero muy trabajoso, debería ser la última opción.

Informe

El grupo debe presentar un informe el día de la entrega, el mismo debe incluir explicaciones de las system calls desarrolladas y del funcionamiento de los algoritmos.

Entrega

Fecha: 9/5/2018
Entregables: Mail con id de commit y branch del entregable en un repositorio como gitlab, github o bitbucket.
Defensa del trabajo práctico: 16/5/2018