Sistemas Operativos

Primer cuatrimestre 2019

Trabajo Práctico Nro. 2 (obligatorio): Construcción del Núcleo de un Sistema Operativo

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

Se realizará con los grupos ya establecidos.

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. Es necesario que el proyecto compile en la imagen de Docker provista por la cátedra.

Requerimientos

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, 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.

Cada grupo debera implementar uno de los siguientes algoritmos dependiendo del número de grupo:

(N Grupo % 3 == 0) Bitmap
(N Grupo % 3 == 1) Free list con first fit
(N Grupo % 3 == 2) Free list con best fit

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).

Cada grupo debera implementar uno de los siguientes algoritmos de scheduling dependiendo del número de grupo:

(N Grupo % 2 == 0) Lottery scheduling
(N Grupo % 2 == 1) Priority-based round Robin
(La prioridad de los procesos deberá ser especificada al momento de creación)

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 Arquitectura de Computadoras. Si hiciera falta se requiere implementar las system calls pertinentes para aislar kernel de user space.

Use el driver de video implementado en Arquitectura de Computadoras.

Aplicaciones de User space

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, prioridad, etc.
- **prodcons:** muestra una resolución para el problema productor consumidor de buffer acotado, se debe poder 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.

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 kernel space y user space, 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 (TDD), 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

Se deberá presentar un informe en formato pdf (NO entregarlo impreso), en el cual se desarrollen de forma breve, los siguientes puntos:

- Decisiones tomadas durante el desarrollo, por ejemplo, detalles del algoritmo de scheduling y el page alocator, etc.
- Instrucciones de compilación y ejecución.
- Limitaciones.
- Problemas encontrados durante el desarrollo y cómo se solucionaron.
- Citas de fragmentos de código reutilizados de otras fuentes.

Entorno de compilación

Es un requisito obligatorio para la compilación, utilizar la imagen provista por la cátedra:

```
docker pull agodio/itba-so:1.0
```

Evaluación

La evaluación incluye y no se limita a los siguientes puntos:

- [1] Deadline.
- [4] Funcionalidad (Mandatorio).
- [3] Calidad de código.
- [1] Tests.
- [1] Informe.
- Defensa.

Entrega

Fecha: 06/05/2019 hasta las 23:59.
Se habilitará la actividad “TP2” en el campus donde se podrá subir el material requerido. En caso de tener algún problema con la entrega en Campus, enviarlo por mail a los docentes a cargo de la clase práctica.
Entregables: Link del repositorio especificando el hash del commit y la rama correspondiente a la entrega y el informe.
Defensa del trabajo práctico: Grupal, obligatoria y con nota individual 03/10/2018.