Sistemas Operativos

Trabajo Práctico Nro. 2: 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 armarán su propio kernel simple. Para ello implementarán mecanismos de IPC, memory management, scheduling y memory protection.

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 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 la generación de un cross compiler de 64 bits, pero pueden utilizar el mismo toolchain (pipeline de desarrollo) que usaron en Arquitectura de Computadoras. Pueden encontrar más información al respecto acá: http://wiki.osdev.org/GCC_Cross-Compiler.

Requerimientos

El kernel debe implementar en 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 no trivial 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. Quien 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), que sea externo a los procesos en sí, es decir, sin que los mismos se enteren (pre-emptivamente) y con alguna estructura/algoritmo que permita seleccionar al siguiente proceso (scheduler).

Cada proceso en el scheduler tiene que tener al menos tres estados: ejecutando, dormido, pausado. Solo habrá un proceso ejecutando, los procesos dormidos son ignorados por el scheduler a la hora de seleccionar el próximo proceso a ejecutar, los demás se encontrarán pausados esperando a tener tiempo de procesador.

Además, por otro lado, el scheduler tiene que llevar de alguna forma el control de qué proceso está en el “foreground”, este proceso debería ser el único que puede leer el input de teclado.

No confundir el estado “ejecutando” con el proceso “foreground”, todos los procesos no dormidos reciben tiempo de CPU, solo uno a la vez (el foreground) puede recibir input de teclado. El sistema no requiere contar con múltiples terminales.

Syscalls involucradas

- Crear un proceso
- Finalizar un proceso
- Listar procesos

Opcionales (pueden servirles para implementar IPCs más adelante)

- En este paso posiblemente tengan que reescribir parte de la system call read (keyboard input) para que mande el proceso a dormir hasta que reciba el foreground y el usuario introduzca un enter, “\n”, en el buffer de teclado.
- Dormir el proceso actual hasta que suceda un evento x.
- Enviar una señal a un proceso x (lo despierta si está dormido)
- Ceder el procesador al siguiente proceso (yield)

Memory Protection

Utilizando el page allocator desarrollado y el sistema de de paginación de 64 bits de Intel, deberán conseguir proteger Kernel Space. Para esto será necesario que los procesos no puedan escribir en la memoria del Kernel (lo que sucede con un proceso que lo intenta es determinado por ustedes), para esto será necesario atender a la excepción Page Fault. No es necesario que los procesos estén protegidos entre sí, recuerden que todas las secciones de

código de los procesos de Userspace van a estar en el mismo binario, como en el TP original, por lo cual separarlos para protegerlos entre sí es difícil.

Además cada proceso tendrá inicialmente 4 KiB físicos de stack, mapeados en alguna dirección de memoria virtual, que podrá crecer hasta un máximo de 8 MiB. Cada vez que el stack crezca más allá de su límite actual, el kernel otorgará páginas de 4 KiB físicas (pero mapeadas contiguamente en memoria virtual) para satisfacer el nuevo requerimiento. Para ello deberán nuevamente atender a la excepción Page Fault.

Esencialmente Page Fault tendrá dos casos:

- El proceso accedió a memoria virtual dentro de sus 8 MiB de stack. (otorgar más stack)
- El proceso accedió a memoria virtual fuera de Userspace (manejar el error).

Recuerde que para que las secciones de código (user y kernel) funcionen apropiadamente, su ubicación en el virtual address space debe ser la misma con la cual el código se linkeditó! En el proyecto de ejemplo original, esta dirección era la etiqueta **.text** en los archivos **kernel.ld** y **sampleCodeModule.ld**. Una forma muy simple de lograr esto es hacer “Identity mapping” en las páginas del kernel, es decir, mapear las direcciones virtuales a las mismas direcciones físicas.

IPCs

Se debe implementar al menos un mecanismo de IPC, a elección, con las syscalls correspondientes. El mecanismo elegido debe contar con las primitivas para poder sincronizar y comunicar los procesos, por ejemplo, si implementan shared memory, deberán también implementar alguna primitiva para sincronizar los procesos, como semáforos.

Syscalls involucradas

Dependen del IPC elegido, considerar que algunas de estas system calls posiblemente deberán dormir el proceso que las llama, y el kernel deberá despertarlo cuando se cumpla una cierta condición.

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.

- **ipcs:** muestra la lista de estructuras de IPC creadas en el sistema
- **help:** muestra una lista con todos los comandos disponibles

Aplicaciones sugeridas

- Envío de mensajes en formato string entre dos procesos, por ejemplo: **msg “a message” <pid>**
- Solución de algún problema de sincronización, como el problema de los filósofos, o producer-consumer.
- Un stack bomb o algún programa que permita al usuario reservar memoria manualmente para mostrar el funcionamiento de la alocación automática del stack.

Agregue sus propias aplicaciones para demostrar el funcionamiento del sistema.

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.

Fecha de entrega

El trabajo deberá entregarse completo, funcionando y con la documentación completa el día 23 de noviembre de 2015.

Consideraciones finales

- El trabajo práctico es grande y no especifica limitaciones, elija sus propias limitaciones para simplificar el desarrollo del sistema, en particular, para evitar tener que programar un sistema con un diseño demasiado dinámico! Un ejemplo de esto es: el sistema solo soporta 512 procesos de hasta 512 MiB. Elija sus limitaciones cuidadosamente y compartalas con la cátedra por mail para verificarlas. A esto se le llama “Hacer asumpciones”.
- Pueden utilizar el sistema de 32 bits MTask desarrollado y provisto por la cátedra como ejemplo e incluso tomar código del mismo. Pueden encontrar MTask en la página de la cátedra en IOL (material didáctico, carpeta Teoría).
- Todo el código tomado de cualquier otro sistema debe ser comentado y citado en el informe!
- Se aconseja diseñar un mapa de memoria físico global, un mapa de memoria virtual para cada aplicación, y un mapa de memoria virtual para el kernel, con las secciones de memoria correspondientes.
- Se sugiere tener dos consolas: una para la salida y entrada de los procesos, y una consola de debug para la salida del kernel, se puede alternar entre ambas con una tecla o combinación de teclas. La consola de debug del kernel ayuda a mostrar el funcionamiento del mismo!

- No es necesario que el sistema considere los múltiples cores del CPU, si los tiene (es una tarea extremadamente compleja).
- Recuerde que para que el kernel pueda seguir funcionando sus páginas de código **siempre** tienen que estar presentes al menos en modo **read**, de otra forma el sistema fallará ni bien se ejecute una interrupción (de hardware o de software), ya que las mismas están manejadas por el kernel.
- Implementar un algoritmo O(log(n)) para reservar de memoria y O(1) amortizado para liberarla, o un Scheduler de prioridades dinámicas en base a estadísticas es excelente y sube la nota final... **siempre y cuando todo lo demás funcione!** Prioricen el funcionamiento de los requerimientos por encima de la eficiencia.
- Mantenga su diseño simple en base a las limitaciones elegidas, Keep It Simple Student.
- Al igual que en Arquitecturas, no es posible acceder a Kernel Land desde User Land a menos que sea a traves de una System Call.
- El sistema **no requiere** múltiples terminales virtuales. Dependiendo de su diseño, es posible que manejar el foreground de una aplicación se resuma a simplemente otorgar foreground a un proceso especial (el shell) cada vez que una aplicación en foreground termine.
- El sistema **no requiere** un filesystem. Dependiendo de su diseño y del IPC elegido, deberá tomar alguna convención para poder identificar apropiadamente un recurso de IPC usado por dos procesos.