Laboratorio 2: Semáforos en XV6

Programando nuestras propias llamadas al sistema

Sistemas Operativos 2022 - FaMAF - UNC

Original 2014: Rafael Carrascosa

Versión 2017: Matías David Lee

Versión 2018: Ignacio Moretti

Versión 2022: Facundo Bustos

Introducción

En este laboratorio se implementará un sistema de Semáforos para espacio de usuario, que sirven como mecanismo de sincronización entre procesos. Se implementarán en XV6 (sistema operativo con fines académicos) en espacio de kernel y deberá proveer syscalls accesibles desde espacio de usuario.

Como los semáforos vienen en varios estilos, en este laboratorio vamos a implementar sólo un estilo de semáforos llamado semáforos nombrados inspirándose en los semáforos nombrados que define POSIX.

Características de los semáforos nombrados:

- Son administrados por el kernel.
- Están disponibles globalmente para todos los procesos del SO (i.e., no hay semáforos "privados").
- Su estado se preserva mientras el SO esté activo (i.e., se pierden entre reinicios).
- Cada semáforo tiene un "nombre" que lo identifica con el kernel, en nuestro caso los nombres son números enteros entre 0 y un límite máximo (idea similar a los file descriptors).

Instalación de XV6 y puesta en marcha

1. Clonar repositorio de xv6: git clone https://github.com/mit-pdos/xv6-riscv.git
2. Copiar los archivos clonados al repo de Bitbucket asignado a su grupo.
3. Hacer el primer commit+push del proyecto.
4. Instalar qemu: sudo apt-get install qemu-system-riscv64
5. Compilar e iniciar xv6: posicionarse en el directorio raíz de xv6 de su repo (debe haber un archivo “Makefile”) y ejecutar el comando “make qemu”.

Comandos y shortcuts útiles de XV6

ls: lista todos los programas de usuario disponibles del sistema.
<CRTL-p>: Lista todos los procesos que están corriendo en el sistema.
<CRTL-a> x: Sale del sistema operativo.

Su trabajo

1. Implementar almenos 4 syscalls: sem_open(), sem_up(), sem_down(), sem_close().
2. Testear la implementación de las syscalls con los programas de usuario: barrier_init, barrier_echo, barrier_rise y barrier_release provistos por la cátedra.
3. Implementar un programa de espacio de usuario “pingpong” que funcione de la “manera natural”.
4. Hacer un pequeño informe sobre lo hecho.

Las Syscalls

int sem_open(int sem, int value) → Abre y/o inicializa el semáforo “sem” con un valor arbitrario “value”.

int sem_close(int sem) →Libera el semáforo “sem”.

int sem_up(int sem): →Incrementa el semáforo ”sem” desbloqueando los procesos cuando su valor es 0.

int sem_down(int sem) →Decrementa el semáforo ”sem” bloqueando los procesos cuando su valor es 0.

Para todas las syscalls el valor “sem” es un entero entre 0 y un número máximo a definir por ustedes. Cada una de las funciones anteriores devuelven 0 en caso de error. Analizar y definir cuales son los posibles errores para cada una de ellas. Posiblemente surjan preguntas como “que sucede si ejecutamos sem_close() de un semáforo que aún está siendo utilizado” o “que sucede si ejecutamos un sem_up() de un semáforo cuando su valor es igual al valor de inicialización”. Ustedes deben decidir qué se hace en estos casos, definiendo así las políticas de su sistema de semáforos.

Para implementar las syscalls deberán usar acquire() release(), wakeup(), sleep() y argint(). Es parte del laboratorio que investiguen y aprendan sobre estas funciones del kernel (Ver las ayudas al final).

También es parte del laboratorio que:

- Lo que implementen esté libre de problemas de concurrencia y condiciones de carrera.

- Hagan validación de los argumentos de sus syscalls.

El programa “pingpong”

Deberán escribir un programa de usuario que sincroniza la escritura por pantalla de la cadena "ping" y "pong" usando nuestro sistema de semáforos.

El comando “pingpong” deberá tomar como argumento un enteror N (rally) que será la cantidad de veces que aparecerá la palabra "ping" y la palabra "pong" por pantalla.

El programa pingpong deberá hacer fork y con los dos procesos resultantes:

- Uno deberá imprimir "ping" N veces, pero nunca imprimir dos "ping" seguidos sin que haya un "pong" al medio.
- El otro deberá imprimir "pong" N veces, pero nunca imprimir dos "pong" seguidos sin que haya un "ping" al medio.
- Se debe retornar el prompt de consola solo cuando el juego haya terminado por completo (obviamente siempre que el comando pingpong se haya ejecutado en modo foreground).

Es decir, deberá haber 2N líneas con "ping" y "pong" intercalados. Esta sincronización entre procesos es posible de realizar usando semáforos, y así deberán hacerlo. Al terminar la ejecución del programa, los semáforos deberán estar liberados para nuevos usos.

El informe

El informe debe presentarse en un archivo README.md en el directorio raiz en formato markdown e incluir:

- Una muy breve explicación o introducción a lo que hicieron (no repetir el enunciado).
- Una breve explicación de qué hacen las syscalls acquire(), release(), sleep(), wakeup() y argint().
- Cualquier decisión de implementación que tomaron y amerite alguna explicación.

Ayudas

- Como el "id" de un semáforo es un número entero entre 0 y N, entonces se pueden implementar los semáforos como un arreglo dentro del espacio de kernel.
- Mini explicación de las syscalls que utilizarán:
    - acquire(): Toma un lock haciendo busy waiting hasta que esté disponible, similar al down de un semáforo. Útil para crear zonas de exclusión mutua (mutex).
    - release(): Libera un lock tomado, similar a un up de un semáforo.
    - sleep(): Pone a "dormir" el proceso que realizó la syscall hasta un wakeup().
    - wakeup(): Permite al scheduler volver a ejecutar un proceso que hizo una llamada a sleep().
    - argint: Permite leer un argumento dado en la llamada de la syscall.
    - Implementar sem_down() es la parte más desafiante y requiere una buena comprensión de sleep() y wakeup(). Pueden buscar ideas e inspiración en la implementación de pipewrite y piperead en pipe.c.
- Para escribir en markdown este visor online es útil.

Entrega

- Vía commits+push en bitbucket(se tomará como versión entregada el último commit antes de la fecha 04/10/22 16:00hs).
- Deberán usar el repositorio git asignado al grupo y crear un directorio para `xv6` dentro sobre el cual deberán hacer sus modificaciones.
- El *coding style* deberá respetar las convenciones de XV6.
- Evaluación individual/grupal vía defensa con día y horario a acordar con su respectivo docente asignado.

Bibliografía

Para leer sobre XV6:

https://pdos.csail.mit.edu/6.828/2014/xv6/book-rev8.pdf
https://github.com/mit-pdos/xv6-book
https://pdos.csail.mit.edu/6.828/2018/xv6/xv6-rev11.pdf
https://course.ccs.neu.edu/cs3650/unix-xv6/index.html