UNIX System Programming
Lecture 14: Semaphores

• Outline
  - IPC Overview
  - Introduction to Semaphores
  - POSIX Semaphores

• Reference
  - Man pages:
    • sem_overview, sem_open, sem_wait, sem_post
    • sem_close, sem_unlink, sem_init, sem_destroy

Lecture 14: Semaphores
IPC Overview

• Semaphores, message ________, and shared memory are interprocess comm. facilities
  introduced in AT&T System V.2 and are now common in UNIX systems. The methods
  allow IPC between unrelated processes.

• The original IPC methods are known as System V IPC. There are several common
  features in the API to all three SYS V IPC methods (refer to the svipc man page).

Lecture 14: Semaphores
IPC Overview

• There is a newer implementation of each of these mechanisms known as POSIX IPC.
  The POSIX methods have a _______ interface, but may not be available on older systems.

• Modern Linux systems support both SYSV and POSIX IPC. Only POSIX IPC will be
  discussed. Do not confuse the man pages. For POSIX start with sem_overview,
  mq_overview, and shm_overview.

Lecture 14: Semaphores
Introduction to Semaphores

• A semaphore is a protected variable that can be used to synchronize processes or to
  ensure restricted access to shared resources.

• To access a shared resource, a process: (1) Tests the corresponding semaphore, (2) If
  the semaphore is positive, the resource is available and the process _________ the
  semaphore and uses the resource. (3) If the semaphore is 0, the process sleeps until it is
  greater than 0.

Lecture 14: Semaphores
Introduction to Semaphores

• When the process is done with the resource, it increments the semaphore.

• A regular variable can not be used as a semaphore, because the test and decrement
  steps must be performed in a single ________ operation. For this reason, semaphores are
  implemented inside the kernel.

• We've described a counting semaphore. A binary semaphore is used to control access
  to a single unit of a resource.
Lecture 14: Semaphores
Introduction to Semaphores

- Using semaphores for ________ protection:

```c
// First process
s_init(&sem, 1);
s_wait(&sem);
use_resource();
s_post(&sem)
```

```c
// Second process
s_wait(&sem);
use_resource();
s_post(&sem)
```

- For two-way (bilateral) synchronization use two semaphores.

Lecture 14: Semaphores
POSIX Semaphores

- POSIX semaphores are counting semaphores that can be incremented/decrement in steps of one. Each semaphore object represents only one semaphore. (SYSV IPC allows increments in larger steps and manipulate sets of semaphores.).
- There are two types of POSIX semaphores: named and _________. Named semaphores will be discussed first.

Lecture 14: Semaphores
POSIX Named Semaphores

- If _________ is used in flags then two additional arguments are required:

```c
sem_t *sem_open(
    const char *name, int flags,
    mode_t perms, unsigned value);
```

- perms are permissions (0600 for example).
- value is the initial value. Give careful thought to the initial value. An initial value of 0 is typical for synchronization while non-zero values are used for resource protection.

Lecture 14: Semaphores
POSIX Named Semaphores

- The sem_wait() routine is used to ________ (decrement) a semaphore while sem_post() releases it:

```c
int sem_wait(sem_t *sem);
int sem_post(sem_t *sem);
```

- sem_wait() will block until the semaphore is available. sem_trywait() and sem_timedwait() routines are also available. sem_getvalue() can be used to find the current semaphore value.

- sem_open() creates a new semaphore or opens an existing semaphore:

```c
sem_t *sem_open( 
    const char *name, int flags); 
```

- The name should be of the form "/somename". On Linux, a named semaphore exists in a ________ file system (normally /dev/shm) with a name of the form sem.somename.
Lecture 14: Semaphores
POSIX Named Semaphores

- Use `sem_close()` to close the semaphore:
  ```c
  int sem_close(sem_t *sem);
  ```
- `sem_unlink()` will __________ a named semaphore. No other `sem_open()` calls will succeed. It is not actually destroyed until all processes have closed the semaphore:
  ```c
  int sem_unlink(const char *name);
  ```
- Named semaphores have corresponding file names in the `/dev/shm` directory and can also be deleted using “rm”.

Lecture 14: Semaphores
POSIX Unnamed Semaphores

- Use `sem_init()` (instead of `sem_open()`) to __________ an unnamed semaphore.
  ```c
  sem_t sem;
  int sem_init(sem_t *sem, int pshared
              unsigned int value);
  ```
- `value` is the desired initial value of the semaphore.

Lecture 14: Semaphores
POSIX Unnamed Semaphores

- If `pshared` is 0 the semaphore is to be shared between threads and must be in an area of memory visible to all threads, i.e. a global variable or allocated on the __________.
- If `pshared` is nonzero the semaphore is to be shared between processes and must be allocated in a shared memory region.

Lecture 14: Semaphores
POSIX Unnamed Semaphores

- An unnamed semaphore is destroyed using `sem_destroy()`:
  ```c
  int sem_destroy(sem_t *sem);
  ```
- The routines `sem_open()`, `sem_close()` and `sem_unlink()` are not used with unnamed semaphores.
- `sem_wait()` and `sem_post()` are also used to acquire an unnamed semaphore.

Lecture 14: Semaphores
Named vs Unnamed

- ________ semaphores can be easily shared between related or unrelated processes.
- To share unnamed semaphores between separate processes the semaphore must be in a shared memory segment. (Shared memory will be covered in a future lecture.)
- Unnamed semaphores should be used for synchronization between threads. (Threads will also be covered in a future lecture)

Lecture 14: Semaphores
In Class Exercise

- We want to guarantee exclusive access to a function `foo()`. Download, compile and run the example programs. (Just run `parent`, it execs the `child`.) The `parent` acquires a semaphore before calling `foo()` and releases it afterward, the `child` does not. Fix the `child` so that it also acquires and releases the semaphore before and after calling `foo()`. Rerun the program.