API Semaphores

Ricardo Rocha

Department of Computer Science

Faculty of Sciences

University of Porto

For more information please consult

'Advanced Programming in the UNIX® Environment, 3rd Edition,

W. Richard Stevens and Stephen A. Rago, Addison Wesley'

Section 15.10

Semaphores

- POSIX semaphores are available in two flavors:
  - **Named semaphores**
  - **Unnamed semaphores**

- Named/unnamed semaphores differ in how they are created and destroyed, otherwise work the same
  - Named semaphores are **accessed by name** and thus they can be used by any process that **know their names**
  - Unnamed semaphores **only exist in memory** and thus they can be used only by processes that have **mapped that same memory extent into their address spaces**

Opening a Named Semaphore

```
#include <semaphore.h>

sem_t *sem_open(char *name, int oflag);
sem_t *sem_open(char *name, int oflag, mode_t mode, int value);
// * creates a new named semaphore or uses an existing one
// * to use an existing semaphore, we should set oflag to 0
// * to create a new named semaphore, we should set the oflag
//   to O_CREAT and, if the named semaphore already exists,
//   it is still opened but with no additional initialization
// * to ensure that we are creating the semaphore, we can set
//   the oflag with both O_CREAT and O_EXCL which will cause
//   sem_open to fail if the named semaphore already exists
// * returns a semaphore if successful, SEM_FAILED on error
```

Opening a Named Semaphore

```
#include <semaphore.h>

sem_t *sem_open(char *name, int oflag);
sem_t *sem_open(char *name, int oflag, mode_t mode, int value);
// * when creating a new semaphore (O_CREAT flag), we need to
//   specify also the access permissions (mode argument) and
//   the initial value (value argument) for the semaphore
// * to promote portability, we must follow certain conventions
//   when selecting a semaphore name:
//   - the first character in the name should be a slash (/)
//   - the name should contain no other slashes
```

Closing a Named Semaphore

```
#include <semaphore.h>

int *sem_close(sem_t *sem);
// * closes a semaphore by releasing the resources associated
//   with it (the semaphore value is unaffected)
// * by default, all pending open semaphores are closed
//   automatically by the kernel when a process terminates
// * returns 0 if successful, -1 on error
```

Removing a Named Semaphore

```
#include <semaphore.h>

int *sem_unlink(char *name);
// * removes the name of the semaphore and, if there are no
//   open references to it, then it is also destroyed
// * otherwise, destruction is deferred until the last open
//   reference is closed
// * regardless of whether the semaphore is still in use,
//   the semaphore's name is immediately removed so that
//   sem_open() can no longer be used to open it
// * returns 0 if successful, -1 on error
```

Decrementing a Named Semaphore

```
#include <semaphore.h>

int sem_wait(sem_t *sem);
int sem_trywait(sem_t *sem);
// * decrements the value of a semaphore
// * with sem_wait(), it won’t return until it has decremented
//   the semaphore count, i.e., it blocks while the semaphore
//   count is 0
// * with sem_trywait(), it avoids blocking and returns an
//   error instead of blocking
// * both return 0 if successful, -1 on error
```

Incrementing a Named Semaphore

```
#include <semaphore.h>

int sem_post(sem_t *sem);
// * increments the value of a semaphore
// * returns 0 if successful, -1 on error
```

Named Semaphores: Example

```
#define SEM_NAME   "/mysem"
#define SEM_FLAGS  S_IRUSR | S_IWUSR

int main() {
  sem_t *sem;  // semaphore pointer
  sem = sem_open(SEM_NAME, O_CREAT | O_EXCL, SEM_FLAGS, 1);
  ...
  // use sem_wait()/sem_post() to increment/decrement semaphore
  ...
  sem_close(sem);  // close semaphore
  sem_unlink(SEM_NAME);  // destroy semaphore name
}
```

Creating an Unnamed Semaphore

```
#include <semaphore.h>

int sem_init(sem_t *sem, int pshared, int value);
// * creates an unnamed semaphore to be shared between
//   threads (pshared = 0) or between processes (pshared != 0)
// * to use the semaphore between threads, it should be located
//   at some address that is visible to all threads
// * to use the semaphore between processes, it should be
//   located in a region of shared memory
// * returns 0 if successful, -1 on error
```

Destroying an Unnamed Semaphore

```
#include <semaphore.h>

int sem_destroy(sem_t *sem);
// * destroys an unnamed semaphore
// * destroying a semaphore that other threads or processes
//   are still using produces undefined behavior
// * using a semaphore that has been destroyed produces
//   undefined results, unless we reinitialize it by calling
//   sem_init() again
// * returns 0 if successful, -1 on error
```

Unnamed Semaphores: Example

```
sem_t sem;  // unnamed semaphore

int main() {
  sem_init(&sem, 0, 1);  // create semaphore
  ...
  // use sem_wait()/sem_post() to increment/decrement semaphore
  ...
  sem_destroy(&sem);  // destroy semaphore
}
```