14. Critical Sections and Semaphores

SOA, Deemed to be University
ITER, Bhubanewar

Book(s)

Text Book(s)

Kay A. Robbins, & Steve Robbins

Unix™ Systems Programming

Communications, concurrency, and Treads

Pearson Education

Reference Book(s)

Brain W. Kernighan, & Rob Pike

The Unix Programming Environment

PHI

Introduction

- Programs that manage shared resources must execute portions of code called **critical sections** in a mutually exclusive manner.

- POSIX semaphores allow **processes** and threads to synchronize their actions.

- POSIX semaphores come in two forms: **named semaphores** and **unnamed semaphores**.

Semaphores

- A semaphore is an integer whose value is never allowed to fall below zero.
- Two operations can be performed on semaphores:
  - Increment the semaphore value by one (**`sem_post`**)
  - Decrement the semaphore value by one (**`sem_wait`**)
- If the value of a semaphore is currently zero, then a **`sem_wait()`** operation will block until the value becomes greater than zero.
- Other names for **`wait`** are **down**, **P** and **lock**.
- Other names for **`signal`** are **up**, **V**, **unlock** and **post**.

POSIX:SEM Unnamed Semaphores

- A POSIX:SEM semaphore is a variable of type **`sem_t`** with associated atomic operations for initializing, incrementing and decrementing its value.
- The difference between unnamed and named semaphores is analogous to the difference between ordinary pipes and named pipes (FIFOs).
- The following code segment declares a semaphore variable called sem.

```
#include <semaphore.h>

sem_t sem;
```

Unnamed Semaphore Initalization

```
#include <semaphore.h>

int sem_init(sem_t *sem, int pshared, unsigned
    value);
```

```
Returns:

(1) If successful, sem_init initializes sem. sem_init()
    returns 0 on success.

(2) If unsuccessful, sem_init returns -1 and sets errno.
```

The `sem_init` Parameters

☞: POSIX:SEM semaphores must be initialized before they are used.

☞: The **`sem_init`** function initializes the unnamed semaphore referenced by **`sem`** to **`value`**. The **`value`** parameter cannot be negative.

`pshared = 0`: means the semaphore can be used only by threads of the process that initializes the semaphore.

`pshared > 0`: If **`pshared`** is nonzero, any process that can access **`sem`** can use the semaphore.

`fork()`: simply forking a child after creating the semaphore does not provide access for the child. The child receives a copy of the semaphore, not the actual semaphore.

Unnamed Semaphore getvalue

```
#include <semaphore.h>

int sem_getvalue(sem_t *sem, int *sval);
```

```
Returns:

(1) If successful, sem_getvalue returns 0.

(2) If unsuccessful, sem_getvalue returns -1 and sets
errno.
```

Unnamed Semaphore `Wait` Operation

```
#include <semaphore.h>

int sem_wait(sem_t *sem);

int sem_trywait(sem_t *sem);
```

```
Returns:

(1) If successful, these functions return 0.

(2) If unsuccessful, these functions return -1 and set
    errno.
```

Unnamed Semaphore `Signal` Operation

```
#include <semaphore.h>

int sem_post(sem_t *sem);
```

```
Returns:

(1) If successful, sem_post returns 0.

(2) If unsuccessful, sem_post returns -1 and sets errno.
```

Unnamed Semaphore `Signal` Operation

The **`sem_destroy`** function destroys a previously initialized unnamed semaphore referenced by the **`sem`** parameter.

```
#include <semaphore.h>

int sem_destroy(sem_t *sem);
```

```
Returns:

(1) If successful, sem_destroy returns 0.

(2) If unsuccessful, sem_destroy returns -1 and sets errno.
```

POSIX:SEM Named Semaphores

Introduction

- POSIX:SEM named semaphores can synchronize processes that do not share memory.
- Named semaphores have a name, a user ID, a group ID and permissions just as files do.
- A named semaphore is identified by a name of the form `/somename`. that is, a null-terminated string of characters (up to 251) consisting of an initial slash, followed by one or more characters, none of which are slashes.

Creating and Opening Named Semaphores

```
#include <fcntl.h>              /* For O_* constants */
#include <sys/stat.h>           /* For mode constants */
#include <semaphore.h>

sem_t *sem_open(const char *name, int oflag);

sem_t *sem_open(const char *name, int oflag,mode_t mode,
    unsigned int value);
```

```
Returns:

(1) If successful, the sem_open function returns the
    address of the semaphore.

(2) If unsuccessful,sem_open returns SEM_FAILED and sets
    errno.
```

Parameters of `sem_open`

- The **`name`** parameter is a string that identifies the semaphore by name.
- The **`oflag`** parameter determines whether the semaphore is created or just accessed by the function.
  - The **`oflag`** parameter is either 0, `O_CREAT`, or `O_CREAT` | `O_EXECL`.
  - If the `O_CREAT` is specified, the **`sem_open`** requires **two** more parameters: a mode parameter of type **`mode_t`** giving the permissions and a value parameter of type **`unsigned`** giving the initial value of the semaphore.
  - If both the `O_CREAT` and `O_EXCL` bits of **`oflag`** are set, the **`sem_open`** returns an `error` if the semaphore already exists.
  - If the semaphore already exists and `O_CREAT` is set but `O_EXCL` is not set, the semaphore ignores `O_CREAT` and the additional parameters.

Named Semaphore getvalue

```
#include <semaphore.h>

int sem_getvalue(sem_t *sem, int *sval);
```

```
Returns:

(1) If successful, sem_getvalue returns 0.

(2) If unsuccessful, sem_getvalue returns -1 and sets
errno.
```

Named Semaphore `Wait` Operation

```
#include <semaphore.h>

int sem_wait(sem_t *sem);

int sem_trywait(sem_t *sem);
```

```
Returns:

(1) If successful, these functions return 0.

(2) If unsuccessful, these functions return -1 and set
    errno.
```

Named Semaphore `Signal` Operation

```
#include <semaphore.h>

int sem_post(sem_t *sem);
```

```
Returns:

(1) If successful, sem_post returns 0.

(2) If unsuccessful, sem_post returns -1 and sets errno.
```

Closing Named Semaphore

```
#include <semaphore.h>

int sem_close(sem_t *sem);
```

```
Returns:

(1) If successful, sem_close returns 0.

(2) If unsuccessful, sem_close returns -1 and sets errno
```

Unlinking Named Semaphore

```
#include <semaphore.h>

int sem_unlink(const char *name);
```

```
Returns:

(1) If successful, sem_unlink returns 0.

(2) If unsuccessful, sem_unlink returns -1 and sets errno.
```

Implement the following

Two concurrent processes **P** and **Q** are accessing their critical sections by using boolean variables **S** and **T** as follows;

Proces P

```
while(true)
{
//Entery section ?

print(1);
print(1);

// Exit section ?
}
```

Proces Q

```
while(true)
{
//Entery section ?

print(0);
print(0);

// Exit section ?
}
```

Complete the entry section and exit section of process **P** and **Q** with suitable semaphore operations using the two boolean semaphores **S** and **T**. Also suggest the initial values of **S** and **T**, such that the execution of the processes will print the sequence 00110011.....