The UNIX Library for Semaphore and Shared Memory Manipulation

Programmed by B. Chandrasekara (February 2002)
Updated for C++ by E. Talagrand and J. Brinkmeier (January 2004)

1. Introduction

This library provides a set of functional calls (based on UNIX system calls) for semaphore definitions and operations on semaphores and for a shared memory definition and shared memory access.

The directory /usr/class/cis660 includes relevant files: ssem.h, ssem.c, sshm.h and sshm.c. Copy those files into your directory and obtain ssem.o and sshm.o using the following compilation commands:

```
gcc –o ssem.o –c ssem.c
gcc –o sshm.o –c sshm.c
```

For a program that uses semaphores and a shared memory, the header files ssem.h and sshm.h have to be included into the header of the source code and one of the following compilation commands is to be used (depending on a programming language):

```
gcc –o program program.c ssem.o sshm.o for C programs
```
```
g++ –o program program.cpp ssem.o sshm.o for C++ programs
```

When semaphores or sheared memories are not used in the given program, sem.o or sshm.o, respectively, may be omitted from the above commands, and ssem.h or sshm.h, respectively, may be omitted from the source code.

2. Semaphore Definition and Operations

a. int sem_create(int key, int initial_value)

This function creates a new semaphore and returns a positive integer (but only if successful). The positive integer is a semaphore identifier to be used in other semaphore functions. The parameter key has to contain a unique value, and a six digit integer is recommended. The parameter initial_value should contain the initial value for the semaphore.

b. int sem_open(int key)

This function opens an existing semaphore and it returns a positive integer (but only if successful). The positive integer is a semaphore identifier to be used in other semaphore functions. The semaphore is created by sem_create with identical value for the parameter key.
c. \texttt{int sem\_wait(int semid)}
   This function performs the wait operation on a semaphore, i.e. a value of the semaphore is decremented by 1, and if the new value is negative the issuing process is blocked. The parameter \texttt{semid} is a semaphore identifier obtained from either \texttt{sem\_create} or \texttt{sem\_open}.

d. \texttt{int sem\_signal(int semid)};
   This function performs the signal operation on a semaphore, i.e. a value of the semaphore is incremented by 1 and if the value is 0 or negative, one of the processes blocked on that semaphore is unblocked. The parameter \texttt{semid} is a semaphore identifier obtained from either \texttt{sem\_create} or \texttt{sem\_open}.

e. \texttt{int sem\_rm(int semid)}
   This function removes the semaphore from the system. The parameter \texttt{semid} is a semaphore identifier obtained from either \texttt{sem\_create} or \texttt{sem\_open}.

3. Shared Memory Definition and Operations

a. \texttt{int shm\_get(int key, int \*start\_ptr, int size)}
   This function creates a shared memory region with the specified \texttt{key} and \texttt{size} or it opens already created shared memory with the identical \texttt{key}. If creating, the parameter \texttt{key} has to contain a unique value. The starting address of the shared memory region is returned as \texttt{start\_ptr}. If successful, the returned value is a positive integer to be used in \texttt{shm\_rm}.

An illustration of creating the shared array \texttt{buffer} of size 200, with \texttt{type\_t} elements, where \texttt{type\_t} is any type such as \texttt{int}, \texttt{char} or \texttt{struct} (but pointers are not allowed):

\begin{verbatim}
    type\_t \*buffer;
    shmid = shm\_get (123456, (void**) &buffer, 200*sizeof (type\_t));
\end{verbatim}

In the program, use \texttt{buffer[i]} to access the \texttt{i}-th element in the shared region.

b. \texttt{int shm\_rm(int shmid)};
   The function removes the shared memory from the system. The parameter \texttt{shmid} is the return value from \texttt{shm\_get}.

Important: Semaphores and shared memories not explicitly removed stay in the system after a process that created them terminates and even when a user logs off. Since UNIX supports limited number of those resources, it is important to make sure that all created semaphores and shared memories are remove before logging off. The directory /usr/class/cis660 includes the script file \texttt{rsm.dat} that provides convenient way to remove all semaphores and all shared memories at once.
An individual semaphore or shared memory may be removed using UNIX command \texttt{ipcrm \_s sem\#} or \texttt{ipcrm \_m mem\#}, respectively, where \texttt{sem\#} and \texttt{mem\#} are obtained from UNIX command \texttt{ipcs}, which lists all semaphores and shared memories.