POSIX THREADS PROGRAMMING – 3

Emmanuel S. Pilli

Semaphore

- The pthreads library itself does not provide a semaphore
- However, a separate POSIX standard does define them
- The necessary declarations to use these semaphores are contained in semaphore.h
- To define a semaphore object, use

  sem_t *sem_name*;

Semaphore API

- **sem_init**: Initialize a new semaphore
  - The second argument denotes *how* the semaphore will be shared.
  - Passing zero denotes that it will be shared among **threads** rather than processes.
  - The final argument is the initial value of the semaphore.
- **sem_destroy**: Deallocate an existing semaphore.
- **sem_wait**: This is the Wait () operation.
- **sem_post**: This is the Signal () operation.

sem_init

- int sem_init (sem_t *sem, int pshared, unsigned int value);
  - sem address of the declared semaphore
  - pshared should be 0 (not shared with threads in other processes)
  - value the desired initial value of the semaphore

- Return: The return value is 0 if successful.

sem_wait

- To wait on a semaphore
- int sem_wait (sem_t *sem);
  - sem_wait (&sem_name);
- If the value of the semaphore is negative, the calling process blocks
- One of the blocked processes wakes up when another process calls sem_post

sem_post

- To increment the value of a semaphore
- int sem_post(sem_t *sem);
  - sem_post(&sem_name);
- It increments the value of the semaphore
- It wakes up a blocked process waiting on the semaphore, if any.

sem_getvalue

- To find out the value of a semaphore
- int sem_getvalue (sem_t *sem, int *valp);
- Gets the current value of sem
- Places it in the location pointed to by valp

sem_destroy

- To destroy a semaphore
- sem_destroy(sem_t *sem);
- Destroys the semaphore
- No threads should be waiting on the semaphore if its destruction is to succeed.

Comparison

State	Pthread	Mutex	Semaphore
Creation	**pthread_create**	**pthread_mutex_init**	**sem_init**
Destroy	**pthread_exit**	**pthread_mutex_destroy**	**sem_destroy**
Waiting	**pthread_join**	-	-
Acquisition	-	**pthread_mutex_lock**	**sem_wait**
Release	-	**pthread_mutex_unlock**	**sem_post**