



Semaphore and Mutex Operations

1



Announcements

- MP1 due today (please e-mail Jin)
- MP2 will be out tonight

2



Basic Operations

```
wait (sem_t *sp)
  if (sp->value >0) sp->value--;
  else {
    Add process to sp->list;
    <block>
  }

signal (sem_t *sp)
  if (sp->list != NULL)
    remove next process from sp->list;
  else sp->value++;
```

3



Semaphores in POSIX

- `int sem_init(sem_t *sem, int pshared, unsigned value);`
- `int sem_destroy(sem_t *sem);`
- `int sem_trywait(sem_t *sem);`
- `int sem_wait(sem_t *sem);`
- `int sem_post(sem_t *sem);`

4



Mutex (Lock)

- Simplest and most efficient thread synchronization mechanism
- A special variable that can be either in
 - **locked state**: some thread *holds* or *owns* the *mutex*; or
 - **unlocked state**: no thread holds the *mutex*
- When several threads compete for a *mutex*, the losers block at that call
 - The *mutex* also has a queue of threads that are waiting to hold the *mutex*.

5



POSIX Mutex-related Functions

- int **pthread_mutex_init**(pthread_mutex_t *restrict mutex, const pthread_mutexattr_t *restrict attr);
- int **pthread_mutex_destroy**(pthread_mutex_t *mutex);
- int **pthread_mutex_lock**(pthread_mutex_t *mutex);
- int **pthread_mutex_trylock**(pthread_mutex_t *mutex);
- int **pthread_mutex_unlock**(pthread_mutex_t *mutex);

6