Using semaphores and waitqs in kernel

Semaphores

1. Declaration of Semaphore Variables

```
static DEFINE_SEMAPHORE(semVar); // Declare and initialize a semaphore to 1
```

2. Initialization of counting semaphores

```
sema_init(&semVar, 3);//semVar initialized to 3
```

3. Use

```
down_interruptible(&semVar); //DON'T USE down(&semVar1)…why?
up(&semVar);
```

Other versions of down operations can be found here:
http://lxr.free-electrons.com/source/include/linux/semaphore.h
http://lxr.free-electrons.com/source/kernel/locking/semaphore.c

- `down_killable`
- `down_trylock`
- `down_timeout`

Mutex

- initialize the mutex as unlocked
```
static DEFINE_MUTEX(mutVar);
```

- re-initialize the mutex as unlocked
```
mutex_init(&mutVar)
```

- Usage
```
mutex_lock_interruptible(&mutVar);
//Critical Section
mutex_unlock(&mutVar);
```

- More versions of mutex_lock can be found here:
  - http://lxr.free-electrons.com/source/include/linux/mutex.h
  - http://lxr.free-electrons.com/source/kernel/locking/mutex.c
  - `mutex_lock_killable`
  - `mutex_lock_trylock`

Sample Code

```
static DEFINE_SEMAPHORE(sem1);
static DEFINE_SEMAPHORE(sem2);
static DEFINE_MUTEX(mut);

//inside init_module() function
sema_init(&sem1, N);
sema_init(&sem2, 0);
mutex_init(&mut); //redundant because DEFINE_MUTEX also initializes mut


…
down_interruptible(&sem1);
mutex_lock_interruptible(&mut);
Critical Section
mutex_unlock(&mut);
up(&sem2);
…
```

IMPORTANT: Don’t forget to check for error return from *_interruptible() functions. Why?

wait queues (waitq)

- See: http://lxr.free-electrons.com/source/include/linux/wait.h
- A waitqueue is a queue of processes that are waiting for a specific event.
- Declaration

```
static wait_queue_head_t wq;
```

- Initialization

```
init_waitqueue_head(&wq);
```

- OR Declare and Initialize in one shot

```
static DECLARE_WAIT_QUEUE_HEAD(wq);
```

- Making the current process wait for a condition to be true

```
wait_event_interruptible(wq, condition);
```

- Preferred over `wait_event(…)`
- Waking up a waiting process upon an event

`wake_up(&wq)`

  - wake up all processes waiting on this wait queue. Woken processes check the condition. If condition is false, they go back to sleep.

`wake_up_interruptible (&wq)`

  - wakes up only the processes that are in interruptible waits. Any process that in non-interruptible waits will continue to sleep.

Sample code: using mutex and waitqs together

```
static DECLARE_WAIT_QUEUE_HEAD(wq);
static DEFINE_MUTEX(mut);


mutex_lock_interruptible(&mut); //enter critical section
…
//wait for an event somewhere in the critical section
while(!condition) {

   //release mutex as you can't sleep holding it
   mutex_unlock(&mut);

   //sleep on condition
   wait_event_interruptible(wq,condition);

   //reacquire mutex before re-entering critical section
   mutex_lock_interruptible(&mut);

   //now go back check condition again.
   //Why?
   //because return from wait and re-locking are not atomic
   //So condition may have changed again
}
…
mutex_unlock(&mut)//exit critical section
```

- And someone has to wake up the sleepers

```
wake_up(&wq);

OR

wake_up_all(&wq);
```

Useful Header Files

```
#include <linux/module.h>
#include <linux/miscdevice.h>
#include <linux/fs.h>
#include <asm/uaccess.h>
#include <linux/slab.h>

#include <linux/wait.h>
#include <linux/semaphore.h>
#include <linux/mutex.h>
```