

Class Notes Sept 11 2006.

See:

<http://www.llnl.gov/computing/tutorials/threads>

for more information.

Pthreads – POSIX Threads.

This is a threading package that does scheduling entirely within a single process. Despite what you may read, a new thread will not automatically be scheduled when one thread is waiting for I/O, nor is it possible for two threads to run simultaneously on two different CPUs.

To use pthreads, you must include pthread.h in your program. One thread is automatically created for you, but you have to call “pthread_self()” before the pthreads package will have any knowledge of this thread.

Other threads are created using pthread_create().

pthread_create (thread,attr,start_routine,arg)

To terminate a thread, execute pthread_exit().

Threads can be synchronized using pthread_join();

Mutexes, which are essentially binary semaphores, can be used for synchronization.

pthread_mutex_init (mutex,attr)

pthread_mutex_destroy (mutex)

pthread_mutexattr_init (attr)

pthread_mutexattr_destroy (attr)

pthread_mutex_lock (mutex)

pthread_mutex_trylock (mutex)

pthread_mutex_unlock (mutex)

Mutex variables are usually declared like this, but the init routines can be used to initialize the mutex using non-standard options.

pthread_mutex_t mymutex = PTHREAD_MUTEX_INITIALIZER;

Most synchronization also requires condition variables like this:

```
pthread_cond_init (condition,attr)
pthread_cond_destroy (condition)
pthread_condattr_init (attr)
pthread_condattr_destroy (attr)
```

```
pthread_cond_t myconvar = PTHREAD_COND_INITIALIZER;
```

```
pthread_cond_wait (condition,mutex)
pthread_cond_signal (condition)
pthread_cond_broadcast (condition)
```

Bounded Buffer:

Declarations: (Global Variables)

```
#define SIZE 5000
int SendPos=0,RecPos=0;
int SenderWaiting=0,ReceiverWaiting=0;
int Count=0;
int Buffer[SIZE];
pthread_mutex_t mymutex= PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t sendcond = PTHREAD_COND_INITIALIZER;
pthread_cond_t recond = PTHREAD_COND_INITIALIZER;
```

Sender:

```
pthread_mutex_lock(mymutex);
if (Count >= SIZE)
{
    SenderWaiting++;
    pthread_cond_wait(sendcond,mymutex);
}
Count++;
Buffer[SendPos] = msg;
SendPos = (SendPos+1)%SIZE;
if (ReceiverWaiting)
{
    ReceiverWaiting--;
    pthread_cond_signal(recond);
}
pthread_mutex_unlock(mymutex);
```

```
Receiver:
pthread_mutex_lock(mymutex);
if (Count <=0)
{
    RecieverWaiting++;
    pthread_cond_wait(recond,mymutex);
}
Count--;
rmsg = Buffer[RecPos];
RecPos = (RecPos + 1)%SIZE;
if (SenderWaiting)
{
    SenderWaiting--;
    pthread_cond_signal(sendcond);
}
pthread_mutex_unlock(mymutex);
```