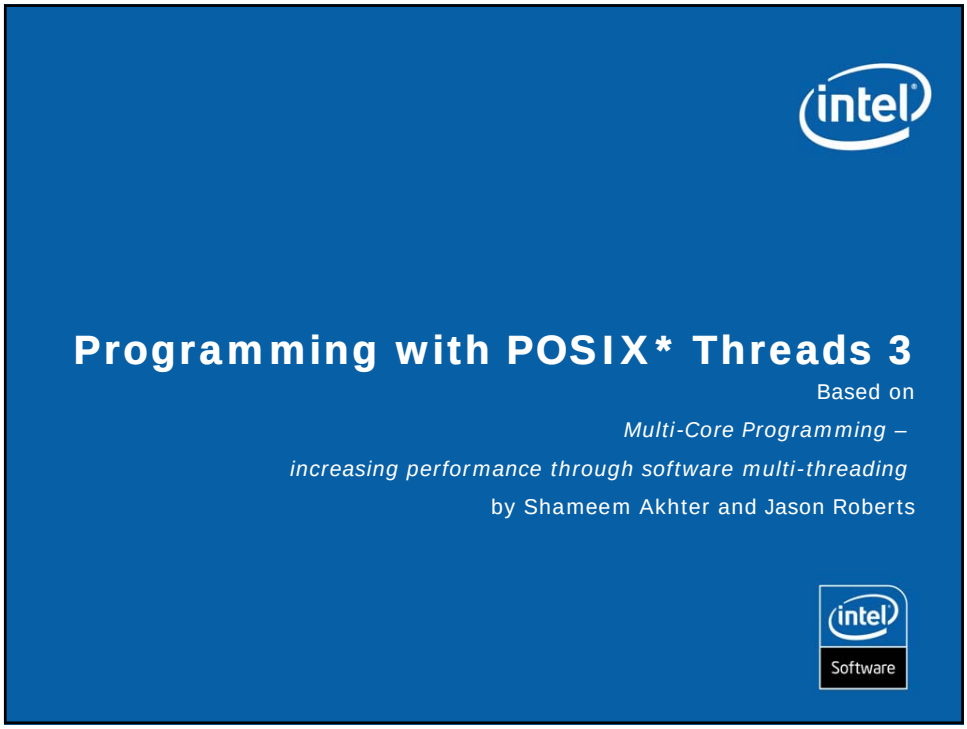

Producer/Consumer Example – Condition Variables

- Listing 5.13 from Ahkter
  - progs\Ahkter\Ch5\PthreadConditionVariables.cpp
- Typical of producer/consumer codes
- Producer threads generate data – reading the file in this case
- Consumer threads are suspended pending a signal – sent when data is ready to be consumed
  - Signal is generated in this case by condition variable mechanism
- Note that manipulation of the condition variables is protected by mutex
  - Required – can be seen by presence of mutex in parameters of pthread_cond_wait

Producer/Consumer Example – Overview

- Code reads file specified on command line
- Creates two threads – each runs PrintCountRead
  - This locks the mutex
  - Loops waiting on the value of the flag
  - Calls pthread_cond_wait to register to be awakened when the condition variable is signaled
  - The thread proceeds when the condition variable is signaled and checks the flag value and prints the number of bytes read by the main thread
- The main thread signals the condition variable when the flag value is changed to 1 using pthread_cond_broadcast
  - Ours when it has read one block of data

Producer/Consumer Example – Details

- Code creates data structure

```
typedef struct {
   pthread_mutex_t  mutex; // the mutex
   pthread_cond_t   cv;    // the condition variable
   int              data;  // the data item used as a flag.
   } flag;
```

- Initializes values of mutex and condition variable to defaults and flag to 0

```
flag ourFlag = {                  // default initialization
      PTHREAD_MUTEX_INITIALIZER,
      PTHREAD_COND_INITIALIZER,
      0 };                        // data item set to 0
```

Producer/Consumer Example – Semaphores

- Listing 5.14 from Ahkter progs\Ahkter\Ch5\PthreadSemaphoreExample.cpp
- Similar to previous example
- Pthread semaphore is counter – part of POSIX specification rather than pthreads
  - When value is 0 threads wait
  - When becomes nonzero thread is released – done in thread priority and order of attachment to semaphore
  - Semaphore value is decremented

Producer/Consumer Example – Overview

- Code reads file specified on command line
- Creates threads –runs PrintCountRead
  - This waits on the semaphore
  - The thread proceeds when the semaphore is signaled and prints the number of bytes read by the main thread
- The main thread signals the semaphore using sem_post
  - Occurs when it has read one block of data
- Note that the thread never reports 0 bytes read since it is blocked until the first block is read
  - It may report one or more blocks depending on how long it takes to wake after being signaled
- Semaphores valuable when have single consumer

sem_wait and sem_trytwait Explained

Attempts to lock semaphore

- *sem_wait()* If the semaphore value is currently zero, then the calling thread will not return from the call until it either locks the semaphore or the call is interrupted by a signal.
- *sem_trywait()* locks the semaphore only if it is currently not locked; that is, if the value is currently positive. Otherwise, it does not lock the semaphore.

EINVAL - semaphore is invalid
ENOSYS - operation not supported

EAGAIN - semaphore was already locked sem_trywait()
EDEADLK – deadlock is detected
EINTR - signal interrupted this function

sem_post Explained

Attempts to unlock the semaphore

•Increments the semaphore value

• If the semaphore value resulting from this operation is positive, then no threads were blocked waiting for the semaphore

• If the value of the semaphore resulting from this operation is zero, then one of the threads blocked waiting for the semaphore will be allowed to return successfully from its call to *sem_wait()*.

EINVAL - semaphore is invalid
ENOSYS - operation not supported

sem_timedwait Explained

• Same as **sem_wait**(), except that *abs_timeout* specifies a limit on the amount of time that the call should block if the decrement cannot be immediately performed. The *abs_timeout* argument points to a structure that specifies an absolute timeout in seconds and nanoseconds since the Epoch

EINVAL - value of *abs_timeout.tv_nsecs* is less than 0, or greater than or equal to 1000 million.
ETIMEDOUT - The call timed out before the semaphore could be locked.

EAGAIN - semaphore was already locked sem_trywait()
EINTR - signal interrupted this function
ENOSYS - operation not supported