POLITECNICO DI MILANO

Thread Posix: Condition Variables

Algoritmi, Strutture Dati e Calcolo Parallelo

Daniele Loiacono

References

- The material in this set of slide is taken from two tutorials by Blaise Barney from the Lawrence Livermore National Laboratory

- Introduction to Parallel Computing
  Blaise Barney, Lawrence Livermore National Laboratory
  https://computing.llnl.gov/tutorials/parallel_comp/

- Also available as Dr.Dobb's "Go Parallel"
  Introduction to Parallel Computing: Part 2
  Blaise Barney, Lawrence Livermore National Laboratory

Overview

- Condition variables provide yet another way for threads to synchronize. While mutexes implement synchronization by controlling thread access to data, condition variables allow threads to synchronize based upon the actual value of data.
- Without condition variables, the programmer would need to have threads continually polling (possibly in a critical section), to check if the condition is met. This can be very resource consuming since the thread would be continuously busy in this activity. A condition variable is a way to achieve the same goal without polling.
- A condition variable is always used in conjunction with a mutex lock.

Mutexes vs Condition Variables

- *Mutexes* and *Condition Variables* are way for threads to synchronize
  - *Mutexes* implement synchronization by controlling thread access to data
  - *Condition Variables* allow threads to synchronize based upon the actual value of data.
- Without condition variables, threads continually poll to check if the condition is met
- This can be very resource consuming since the thread would be continuously busy in this activity
- A condition variable is a way to achieve the same goal without polling.
- A condition variable is always used in conjunction with a mutex lock.

Typical Scenario

- Declare and initialize global data/variables
- Delare and initialize a condition variable object
- Declare and initialize an associated mutex
- Create threads A and B to do work

Main

- Do work
- Lock associated mutex and check of a global variable
- If value does not meet some condition, perform a blocking wait (automatically and atomically unlocks the associated mutex)
- When signalled, wake up (mutex is automatically and atomically locked)
- Explicitly unlock mutex
- Continue

A

- Do work
- Lock mutex
- Change the value of the global variable that Thread-A is waiting upon
- If the new value met the condition desired by Thread-A, signal it toThread-A
- Unlock mutex
- Continue

B

Join / Continue

Main

Declaration and initialization

- Condition variables must be declared with type `pthread_cond_t`
- There are two ways to initialize a condition variable:
  - Statically, when it is declared. For example: `pthread_cond_t myconvar=PTHREAD_COND_INITIALIZER;`
  - Dynamically, with the `pthread_cond_init(condition,attr)` routine
    - ID of the created condition variable is returned through `condition`
    - `attr` if not NULL, permits setting condition variable object attributes
- `pthread_cond_destroy()` should be used to free a condition variable that is no longer needed.

Waiting and Signaling on Condition Variables

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

- `pthread_cond_wait()` blocks the calling thread until the specified `condition` is signalled.
- `pthread_cond_signal()` routine is used to signal (or wake up) another thread which is waiting on the condition variable.
- `pthread_cond_broadcast()` routine should be used instead of `pthread_cond_signal()` if more than one thread is in a blocking wait state.
- It is a logical error to call `pthread_cond_signal()` before calling `pthread_cond_wait()`

Waiting and Signaling on Condition Variables (2)

- `pthread_cond_wait()`
  - should be called while *mutex* is locked and it will automatically release the mutex while it waits
  - when wakes up, *mutex* will be automatically locked for use by the thread
  - programmer is responsible for unlocking *mutex* when the thread is finished with it
- `pthread_cond_signal()`
  - should be called after *mutex* is locked
  - must unlock *mutex* in order for `pthread_cond_wait()` routine to complete

Example

- The main routine creates three threads.
  - Two of the threads perform work and update a "count" variable.
  - The third thread waits until the count variable reaches a specified value

```
#define NUM_THREADS  3
#define TCOUNT 10
#define COUNT_LIMIT 12
int count = 0;
int thread_ids[3] = {0,1,2};
pthread_mutex_t count_mutex;
pthread_cond_t count_threshold_cv;

void *inc_count(void *t){…}
void *watch_count(void *t){…}
int main (int argc, char *argv[]){…}
```

Example (2)

```
int main (int argc, char *argv[]){
  int I;
  long t1=1, t2=2, t3=3;
  pthread_t threads[3];
  pthread_attr_t attr;
  pthread_mutex_init(&count_mutex, NULL);
  pthread_cond_init (&count_threshold_cv, NULL);
  pthread_attr_init(&attr);
  pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
  pthread_create(&threads[0], &attr, watch_count, (void *)t1);
  pthread_create(&threads[1], &attr, inc_count, (void *)t2);
  pthread_create(&threads[2], &attr, inc_count, (void *)t3);
  for (i=0; i<NUM_THREADS; i++) {  /* Wait for all threads */
    pthread_join(threads[i], NULL);
  }
  pthread_attr_destroy(&attr);
  pthread_mutex_destroy(&count_mutex);
  pthread_cond_destroy(&count_threshold_cv);
  pthread_exit(NULL);
}
```

Example (3)

```
void *inc_count(void *t){
  long my_id = (long)t;
  for (int i=0; i<TCOUNT; i++) {
    pthread_mutex_lock(&count_mutex);
    count++;
   if (count == COUNT_LIMIT) { pthread_cond_signal(&count_threshold_cv); }
   pthread_mutex_unlock(&count_mutex);
   sleep(1);
  }
  pthread_exit(NULL);
}
void *watch_count(void *t){
  long my_id = (long)t;
  pthread_mutex_lock(&count_mutex);
  if (count<COUNT_LIMIT) {
    pthread_cond_wait(&count_threshold_cv, &count_mutex);
    count += 125;
  }
  pthread_mutex_unlock(&count_mutex);
  pthread_exit(NULL);
}
```