System Programming

6. Thread Programming
condition variable

Seung-Ho Lim

Dept. of Computer & Electronic Systems Eng.

Condition variables

- Used for synchronization between threads using a mutex
  - a thread may wait for a condition (i.e. blocked)
  - if the blocking thread continue to hold a mutex?
    - it would be better to wait after releasing the mutex for the waiting time
  - when a condition for the blocking thread(s) is satisfied, wake up the waiting thread(s) by signaling
- A condition variable is coupled with a mutex.

Creating a condition variable

```
/* condition variable creation & destruction */
int pthread_cond_init (*condition, *attr);
int pthread_cond_destroy (*condition);
/* condition attribute creation & destruction */
int pthread_condattr_init (*attr);
int pthread_condattr_destroy (*attr);

pthread_cond_t *condition
pthread_condattr_t *attr
```

- Condition variable initialization.
  - static
    `pthread_cond_t` myconvar = **PTHREAD_COND_INITIALIZER**;
  - dynamic
    `pthread_cond_init (*condition, *attr);`

pthread_condattr_t

- condition variable attributes
  - *process-shared*
    - allows the condition variable to be shared by other processes.
  - set NULL to use default attributes

Waiting/Signalling on CV (1)

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

pthread_cond_t *condition
pthread_condattr_t *attr
```

- returns
  - 0 if Ok, or an error value on error
- pthread_cond_wait()
  - wait until the condition is satisfied (blocked)
  - when being blocked, it release (unlocks) the specified mutex so that another thread can enter the critical section and make a signal
  - when the blocking thread is awakened (signaled by another thread), mutex is acquired (locked) again

Waiting/Signalling on CV (2)

- pthread_cond_signal()
  - wakes up the first of threads waiting for the condition.
- pthread_cond_broadcast()
  - wakes up all threads waiting for the condition.
- When calling pthread_cond_signal()
  - this call has no effect if there is no waiting threads

  i.e. the `pthread_cond_signal()` is lost (not saved)!

CV example (1)

cond-var.c

```
#include <pthread.h>
#include <stdio.h>

#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_MUTEX_INITIALIZER;
pthread_cond_t count_threshold_cv = PTHREAD_COND_INITIALIZER;
```

CV example (2)

```
void *inc_count(void *idp)
{   int i, j ;
    double result=0.0;
    int *my_id = idp;

    for (i=0; i < TCOUNT; i++)  {
          pthread_mutex_lock(&count_mutex);
          count++;
          /* Check the value of count and signal waiting thread when condition is reached.
                    Note that this occurs while mutex is locked. */
          if (count == COUNT_LIMIT) {
                    pthread_cond_signal(&count_threshold_cv);
                    printf("inc_count(): thread %d, count = %d
                              Threshold reached.\n", *my_id, count);
          }
          printf("inc_count(): thread %d, count = %d,
          unlocking mutex\n", *my_id, count);
          pthread_mutex_unlock(&count_mutex);
          /* Do some work so threads can alternate on mutex */
          for (j=0; j < 1000; j++)
                    result = result + (double)random();
    }
    pthread_exit(NULL);
}
```

CV example (4)

```
void *watch_count(void *pid)
{
   int *my_id = pid;
   printf("Starting watch_count(): thread %d\n", *my_id);

   pthread_mutex_lock(&count_mutex);

   while (count < COUNT_LIMIT) {
        pthread_cond_wait(&count_threshold_cv, &count_mutex);
        printf("watch_count(): thread %d Condition signal
                        received.\n", *my_id);
   }
   pthread_mutex_unlock(&count_mutex);
   pthread_exit(NULL);
}
```

CV example (5)

```
int main (int argc, char *argv[]) {
   int i, rc;
   pthread_t threads[3];

   pthread_create(&threads[0], NULL, inc_count, (void *)&thread_ids[0]);
   pthread_create(&threads[1], NULL, inc_count, (void *)&thread_ids[1]);
   pthread_create(&threads[2], NULL, watch_count, (void*)&thread_ids[2]);

   for (i = 0; i < NUM_THREADS; i++)
                pthread_join(threads[i], NULL);
   printf ("Main(): Waited on %d threads. Done.\n", NUM_THREADS);
   pthread_exit(NULL);
}
```

CV example (6)

```
$ gcc -o cond-var cond-var.c -lpthread
$ ./cond-var
Starting watch_count(): thread 2
inc_count(): thread 1, count = 1, unlocking mutex
inc_count(): thread 0, count = 2, unlocking mutex
inc_count(): thread 1, count = 3, unlocking mutex
inc_count(): thread 0, count = 4, unlocking mutex
inc_count(): thread 1, count = 5, unlocking mutex
inc_count(): thread 0, count = 6, unlocking mutex
inc_count(): thread 1, count = 7, unlocking mutex
inc_count(): thread 0, count = 8, unlocking mutex
inc_count(): thread 1, count = 9, unlocking mutex
inc_count(): thread 0, count = 10, unlocking mutex
inc_count(): thread 0, count = 11, unlocking mutex
inc_count(): thread 1, count = 12 Threshold reached.
inc_count(): thread 1, count = 12, unlocking mutex
watch_count(): thread 2 Condition signal received.
inc_count(): thread 0, count = 13 Threshold reached.
inc_count(): thread 0, count = 13, unlocking mutex
inc_count(): thread 1, count = 14 Threshold reached.
inc_count(): thread 1, count = 14, unlocking mutex
inc_count(): thread 1, count = 15 Threshold reached.
inc_count(): thread 1, count = 15, unlocking mutex
inc_count(): thread 0, count = 16 Threshold reached.
inc_count(): thread 0, count = 16, unlocking mutex
inc_count(): thread 0, count = 17 Threshold reached.
inc_count(): thread 0, count = 17, unlocking mutex
inc_count(): thread 1, count = 18 Threshold reached.
inc_count(): thread 1, count = 18, unlocking mutex
inc_count(): thread 1, count = 19 Threshold reached.
inc_count(): thread 1, count = 19, unlocking mutex
inc_count(): thread 0, count = 20 Threshold reached.
inc_count(): thread 0, count = 20, unlocking mutex
Main(): Waited on 3 threads. Done.
```

Producer/Consumer Example (1)

prod-cons.c

```
#include <stdio.h>
#include <pthread.h>

void *producer(void*);
void *consumer(void*);
#define MAX_BUF     100
//shared variables
int buffer[MAX_BUF];
int count = 0;
int in = -1, out = -1;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t buffer_has_space = PTHREAD_COND_INITIALIZER;
pthread_cond_t buffer_has_data = PTHREAD_COND_INITIALIZER;

int main(void)
{   int i;
    pthread_t threads[2];
    pthread_create (&threads[0], NULL, producer, NULL);
    pthread_create (&threads[1], NULL, consumer, NULL);
    for (i=0; i< 2; i++)
        pthread_join(threads[i], NULL);
    return 0;
}
```

Producer/Consumer Example (2)

```
void *producer (void *v)
{
   int i;

   for (i =0; i < 1000; i++)  {
      pthread_mutex_lock(&mutex);
      if (count == MAX_BUF) // buffer full !
         pthread_cond_wait(&buffer_has_space, &mutex);
      in = in++ % MAX_BUF;
      buffer[in] = i; // simple data
      count++;
      pthread_cond_signal(&buffer_has_data);
      pthread_mutex_unlock(&mutex);
   }
}
```

Producer/Consumer Example (3)

```
void *consumer (void *v)
{
   int i, data;

   for (i =0; i < 1000; i++)  {
      pthread_mutex_lock(&mutex);
      if (count == 0)  // buffer empty !
          pthread_cond_wait(&buffer_has_data, &mutex);
      out = out++ % MAX_BUF;
      data = buffer[out];
      count--;
      pthread_cond_signal(&buffer_has_space);
      pthread_mutex_unlock(&mutex);
      printf("data = %d\n",data);
   }
}
```