Multi-Threading

CS 425
Distributed Systems
What is MultiThreading?

- **Server**
  - Thread 1: Client Database Lookup
  - Thread 2: Listen for Incoming Requests
  - Thread 3: Handle Client Request
  - Thread 4: Logging
  - Thread 5: Send Control Messages
POSIX Threads

Portable Operating System Interface (POSIX)

➢ Thread management
  ○ Creating, detaching, joining, etc. Set/query thread attributes

➢ Mutex and Semaphore
  ○ Synchronization

➢ Condition variables
  ○ Communications between threads that share a mutex

Compiling on GNU Linux Platform: gcc MyProgram.c -o MyProgram -lpthread
Thread Management: Creating and Terminating a Thread

```c
int pthread_create (pthread_t* tid, pthread_attr_t* attr,
                    void*(functionA), void* arg);
```

- **pthread_create()** takes a pointer to a function as one of its arguments
  - `functionA` is called with the argument specified by `arg`
  - `functionA` can only have one parameter of type `void *`
  - Complex parameters can be passed by creating a structure and passing the address of the structure
  - The structure can't be a local variable
Example: Creating and Terminating a Thread

```c
#include <pthread.h>
#define NUM_THREADS 5

int main (int argc, char *argv[]) {
    pthread_t threads[NUM_THREADS];
    int rc, t;
    for(t=0; t < NUM_THREADS; t++) {
        printf("Creating thread %d\n", t);
        rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
        if (rc) {
            printf("ERROR; pthread_create() return code is %d\n", rc);
            exit(-1);
        }
    }
    pthread_exit(NULL);
}

void *PrintHello(void *threadid) {
    printf("\n%d: Hello World!\n", threadid);
    pthread_exit(NULL);
}
```

Thread Management: Joining and Detaching Threads

- **pthread_join** (pthread_t ID, void **value_ptr):
  - Blocks the calling thread until the specified thread ID terminates.

- **Joinable Threads:**
  - System retains information about the joinable threads after the thread ends, so that other threads can join later.

- **pthread_detach** (pthread_t ID):
  - Marks the thread identified by ID as detached. When a detached thread terminates, its resources are automatically released back to the system.
Thread Management: Joining and Detaching Threads

```c
#include <pthread.h>

void* functionA (void* arg)
{
    pthread_mutex_lock (&mutexA);
    counter++;
    pthread_mutex_unlock (&mutexA);
    return 0;
}

int main ()
{
    pthread_t thread_id [10];
    for (int i = 0; i < 10; i++)
    {
        pthread_create (&thread_id [i], NULL, functionA, NULL);
    }
    for (int j = 0; j < 10; j++)
    {
        pthread_join (thread_id [j], NULL);
    }
    printf("\n\nFinal counter value: %d\n", counter);
    return 0;
}
```

Mutex and Semaphore

Mutex:

➢ Only one thread can access the critical section of a code
➢ Locks must be released by the thread that acquired the lock

Semaphore:

➢ Counting Semaphore
➢ Binary Semaphore
➢ A good example: “Producer-Consumer Problem”
POSIX Mutex

- `int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr);`
- `int pthread_mutex_destroy(pthread_mutex_t *mutex);`
- `pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;`
- `pthread_mutex_lock (&mutexA)`
- `pthread_mutex_unlock (&mutexA);`
**Mutex**

```c
#include <pthread.h>

void* functionA (void* arg)
{
    pthread_mutex_lock (&mutexA);
    counter++;
    pthread_mutex_unlock (&mutexA);
    return 0;
}

int main ()
{
    pthread_t thread_id [10];
    for (int i = 0; i < 10; i++)
    {
        pthread_create (&thread_id [i], NULL, functionA, NULL);
    }
    for (int j = 0; j < 10; j++)
    {
        pthread_join (thread_id [j], NULL);
    }
    printf("\n\nFinal counter value: %d\n",counter);
    return 0;
}
```
Barrier

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <time.h>
#include <pthread.h>
#include <sys/neutrino.h>

pthread_barrier_t   barrier; // barrier synchronization object

int main () {
    time_t  now;
    // create a barrier object with a count of 3
    pthread_barrier_init (&barrier, NULL, 3);
    pthread_create (NULL, NULL, thread1, NULL);
    pthread_create (NULL, NULL, thread2, NULL);
    time (&now);
    printf ("main() waiting for barrier at %s", ctime (&now));
    pthread_barrier_wait (&barrier);
    // after this point, all three threads have completed.
    time (&now);
    printf ("barrier in main() done at %s", ctime (&now));
    pthread_exit (NULL);
    return (EXIT_SUCCESS);
}

void *
thread1 (void *not_used) {
    time_t  now;
    time (&now);
    printf ("thread1 starting at %s", ctime (&now));
    // do the computation
    sleep (20);
    pthread_barrier_wait (&barrier);
    // after this point, all three threads have completed.
    time (&now);
    printf ("thread1() done at %s", ctime (&now));
}

void *
thread2 (void *not_used) {
    time_t  now;
    time (&now);
    printf ("thread2 starting at %s", ctime (&now));
    // do the computation
    sleep (40);
    pthread_barrier_wait (&barrier);
    // after this point, all three threads have completed.
    time (&now);
    printf ("thread2() done at %s", ctime (&now));
}
```
Condition Variables

Creating and Destroying Condition Variables:

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

Waiting and Signaling on Condition Variables

- `pthread_cond_wait (condition,mutex)`
- `pthread_cond_signal (condition)`
- `pthread_cond_broadcast (condition)`
Example: Condition Variable

int main (int argc, char *argv[])
{
    int i, rc;
    long t1=1, t2=2, t3=3;
    pthread_t threads[3];
    pthread_attr_t attr;

    /* Initialize mutex and condition variable objects */
    pthread_mutex_init(&count_mutex, NULL);
    pthread_cond_init (&count_threshold_cv, NULL);

    /* For portability, explicitly create threads in a joinable state */
    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);

    /* Wait for all threads to complete */
    for (i=0; i<NUM_THREADS; i++) {
        pthread_join(threads[i], NULL);
    }

    /* Clean up and exit */
    ......................

    void *watch_count(void *t)
    {
        long my_id = (long)t;
        printf("Starting watch_count(): thread %ld\n", my_id);

        pthread_mutex_lock(&count_mutex);
        count++;
        if (count == COUNT_LIMIT) {
            pthread_cond_signal(&count_threshold_cv);
        }
        pthread_mutex_unlock(&count_mutex);
        sleep(1);
    }

    void *inc_count(void *t)
    {
        int i;
        long my_id = (long)t;

        for (i=0; i<TCOUNT; i++) {
            pthread_mutex_lock(&count_mutex);
            while (count<COUNT_LIMIT) {
                pthread_cond_wait(&count_threshold_cv, &count_mutex);
                printf("watch_count(): thread %ld Condition signal received.\n", my_id);
                count += 125;
                printf("watch_count(): thread %ld count now = %d.\n", my_id, count);
            }
            pthread_mutex_unlock(&count_mutex);
        }
        pthread_mutex_unlock(&count_mutex);
        sleep(1);
    }

    pthread_exit(NULL);
}
References

- https://computing.llnl.gov/tutorials/pthreads/#PthreadsAPI
- http://linux.die.net/man/3/pthread_detach