P-thread Library

Leila Sharifi

December 2017

```c
pthread_t aThread; // type of thread
```

```c
int pthread_create(pthread_t *thread,
                   const pthread_attr_t *attr,
                   void * (*start_routine)(void *),
                   void *arg));
```

```c
int pthread_join(pthread_t thread,
                 void **status);
```

Thread Attributes

- joinable
- stack size
- priority
- inheritance
- scheduling policy
- system/process scope

Default is NULL!

```
int pthread_attr_init(pthread_attr_t *attr);
int pthread_attr_destroy(pthread_attr_t *attr);
pthread_attr_{set/get}{attribute}
```

Detaching Thread

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

void *foo (void *arg) { /* thread main */
    printf("Foobar!\n");
    pthread_exit(NULL);
}

int main (void) {

    int i;
    pthread_t tid;

    pthread_attr_t attr;
    pthread_attr_init(&attr); /* required!!! */
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
    pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM);
    pthread_create(NULL, &attr, foo, NULL);
                .

    return 0;
}
```

Example

```c
#include <stdio.h>
#include <pthread.h>
#define NUM_THREADS 4

void *hello (void *arg) { /* thread main */
    printf("Hello Thread\n");
    return 0;
}

int main (void) {
    int i;
    pthread_t tid[NUM_THREADS];
    for (i = 0; i < NUM_THREADS; i++) { /* create/fork threads */
        pthread_create(&tid[i], NULL, hello, NULL);
    }
    for (i = 0; i < NUM_THREADS; i++) { /* wait/join threads */
        pthread_join(tid[i], NULL);
    }
    return 0;
}
```

Example2

```c
#include <stdio.h>
#include <pthread.h>
#define NUM_THREADS 4

void *threadFunc(void *pArg) { /* thread main */
    int *p = (int*)pArg;
    int myNum = *p;
    printf("Thread number %d\n", myNum);
    return 0;
}

int main(void) {
    int i;
    pthread_t tid[NUM_THREADS];
    for(i = 0; i < NUM_THREADS; i++) { /* create/fork threads */
        pthread_create(&tid[i], NULL, threadFunc, &i);
    }
    for(i = 0; i < NUM_THREADS; i++) { /* wait/join threads */
        pthread_join(tid[i], NULL);
    }
    return 0;
}
```

Example2- Modified

```c
#define NUM_THREADS 4

void *threadFunc(void *pArg) { /* thread main */
    int myNum = *((int*)pArg);
    printf("Thread number %d\n", myNum);
    return 0;
}

int main(void) {
    int tNum[NUM_THREADS];
    // ...
    for(i = 0; i < NUM_THREADS; i++) { /* create/fork threads */
        tNum[i] = i;
        pthread_create(&tid[i], NULL, threadFunc, &tNum[i]);
    }
    // ...
}
```

Mutual Exclusion

Mutex Construct

```
pthread_mutex_t aMutex; // mutex type
```

```
                                    // explicit lock
int pthread_mutex_lock(pthread_mutex_t
                                        *mutex);
                                  // explicit unlock
int pthread_mutex_unlock(pthread_mutex_t
                                      *mutex);
```

Mutex Construct (Count.)

```
int pthread_mutex_init(pthread_mutex_t *mutex,
                       const pthread_mutexattr_t *attr);
// mutex attributes == specifies mutex behavior when
// a mutex is shared among processes
```

```
int pthread_mutex_trylock(pthread_mutex_t *mutex);
```

```
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

Condition Variable

```
pthread_cond_t aCond; // type of cond variable
```

```
int pthread_cond_wait(pthread_cond_t *cond,
                      pthread_mutex_t *mutex);
```

```
int pthread_cond_signal(pthread_cond_t *cond);
```

```
int pthread_cond_broadcast(pthread_cond_t *cond);
```

Condition Variable (cont.)

```
int pthread_cond_init(pthread_cond_t *cond,
                      const pthread_condattr_t *attr);
// attributes -- e.g., if it's shared
```

```
int pthread_cond_destroy(pthread_cond_t *cond);
```

Producer/ Consumer Example

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

#define BUF_SIZE 3          /* size of shared buffer */

int buffer[BUF_SIZE];       /* shared buffer */
int add = 0;                /* place to add next element */
int rem = 0;                /* place to remove next element */
int num = 0;                /* number elements in buffer */

pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;      /* mutex lock for buffer */
pthread_cond_t c_cons = PTHREAD_COND_INITIALIZER;   /* consumer waits on cv */
pthread_cond_t c_prod = PTHREAD_COND_INITIALIZER;   /* producer waits on cv */

void *producer (void *param);
void *consumer (void *param);
```

Producer/ Consumer Example (Cont.)

```c
int main(int argc, char *argv[]) {

    pthread_t tid1, tid2;  /* thread identifiers */
    int i;

    if (pthread_create(&tid1, NULL, producer, NULL) != 0) {
        fprintf (stderr, "Unable to create producer thread\n");
        exit (1);
    }

    if (pthread_create(&tid2, NULL, consumer, NULL) != 0) {
        fprintf (stderr, "Unable to create consumer thread\n");
        exit (1);
    }

    pthread_join(tid1, NULL); /* wait for producer to exit */
    pthread_join(tid2, NULL); /* wait for consumer to exit */
    printf ("Parent quiting\n");
}
```

Producer/ Consumer Example (Cont.)

```
void *consumer (void *param) {

    int i;

        e (1) {

        pthread_mutex_lock (&m);
            if (num < 0) { /* underflow */
                exit (1);
            }
            while (num == 0) { /* block if buffer empty */
                pthread_cond_wait (&c_cons, &m);
            }
            i = buffer[rem]; /* buffer not empty, so remove element */
            rem = (rem+1) % BUF_SIZE;
            num--;
        pthread_mutex_unlock (&m);.

        pthread_cond_signal (&c_prod);
        printf ("Consume value %d\n", i); fflush(stdout);
    }
}
```

Final Project

Priority Readers and Writers

Write a multi-threaded C program that gives readers priority over writers concerning a shared (global) variable. Essentially, if any readers are waiting, then they have priority over writer threads -- writers can only write when there are no readers. This program should adhere to the following constraints:

- Multiple readers/writers must be supported (5 of each is fine)
- Readers must read the shared variable X number of times
- Writers must write the shared variable X number of times
- Readers must print:
- The value read
- The number of readers present when value is read
- Writers must print:
- The written value
- The number of readers present were when value is written (should be 0)
- Before a reader/writer attempts to access the shared variable it should wait some random amount of time
- Note: This will help ensure that reads and writes do not occur all at once
- Use pthreads, mutexes, and condition variables to synchronize access to the shared variable

Simple Socket

Client

Write a simple C program that creates, initializes, and connects a client socket to a server socket. You should provide a way to specify the connecting server address and port. This can be hardcoded or passed via the command line.

Server

Write a simple C program that creates and initializes a server socket. Once initialized, the server should accept a client connection, close the connection, and then exit.

You should provide a way to specify the server's listening port. This can be hardcoded or passed via the command line.

The Echo Protocol

In C, write a server and client that implement the fictitious "echo protocol". To implement the protocol, the server should accept any string from the client, and then return that string with all letters capitalized (if letters exist).

Echo Protocol Example:

- Client sends "Hello, wOrlD"
- Server echoes "HELLO, WORLD"

As soon as the server responds to a client, it may close. And, as soon as the clients receives a response, it may close.