PThreads

Introduction

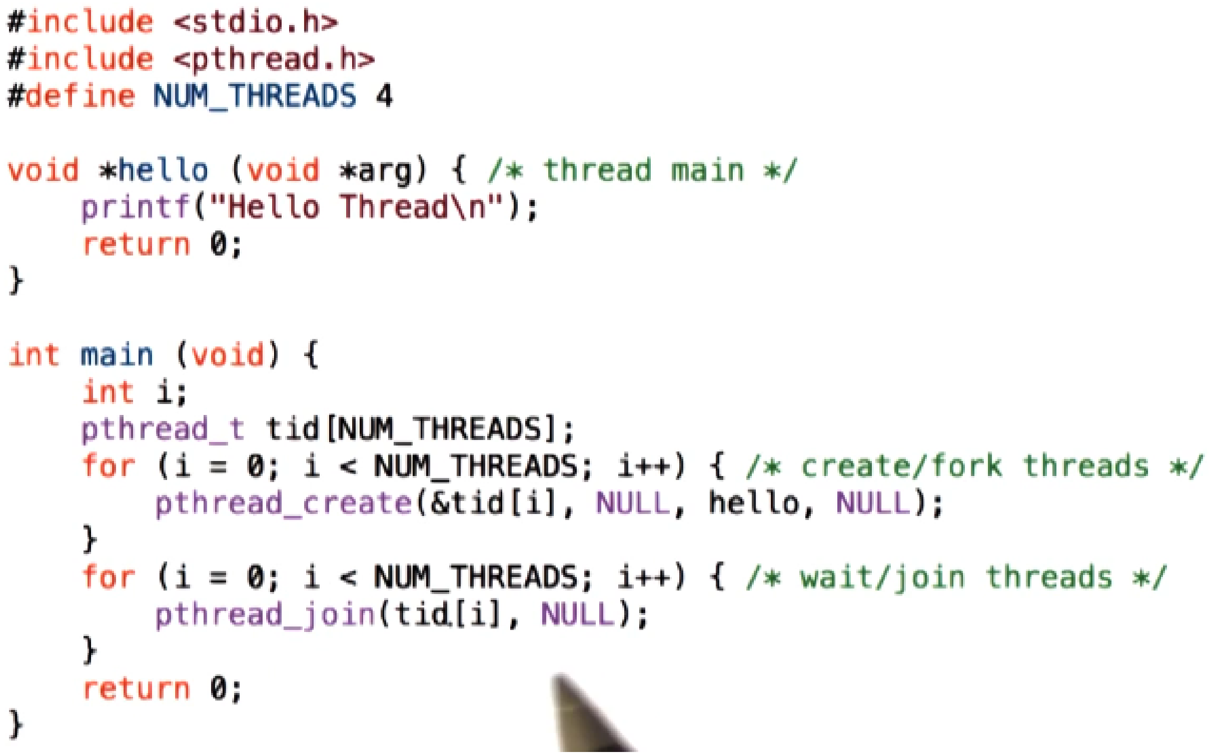
1. PThreads: POSIX Threads (Portable Operating System Interface)
    - POSIX specifies syntax and semantics of the operations
2. PThreads Creation
    - pthread_t aThread; // type of thread
    - int pthread_create(pthread_t*, const pthread_attr_t*, void* (*start_routine)(void*), void* arg);
    - int pthread_join(pthread_t thread, void** status);
3. Pthread Attributes
    - Stack Size
    - Inheritance
    - Joinable
    - Scheduling Policy
    - Priority
    - System/Process Scope
    - Joinable Threads - Parent thread won't terminate until children complete
    - Detachable Threads - Child threads can't be rejoined, continue if parent exits
    - Functions

```
int pthread_attr_init(pthread_attr_t* attr);
int pthread_attr_destroy(pthread_attr_t* attr);
int pthread_attr_set(attribute);
int pthread_attr_get(attribute);
```

Compiling PThreads

1. #include <pthread.h>
2. Compile with -lpthread
3. Check return values of common functions

PThread Example 1

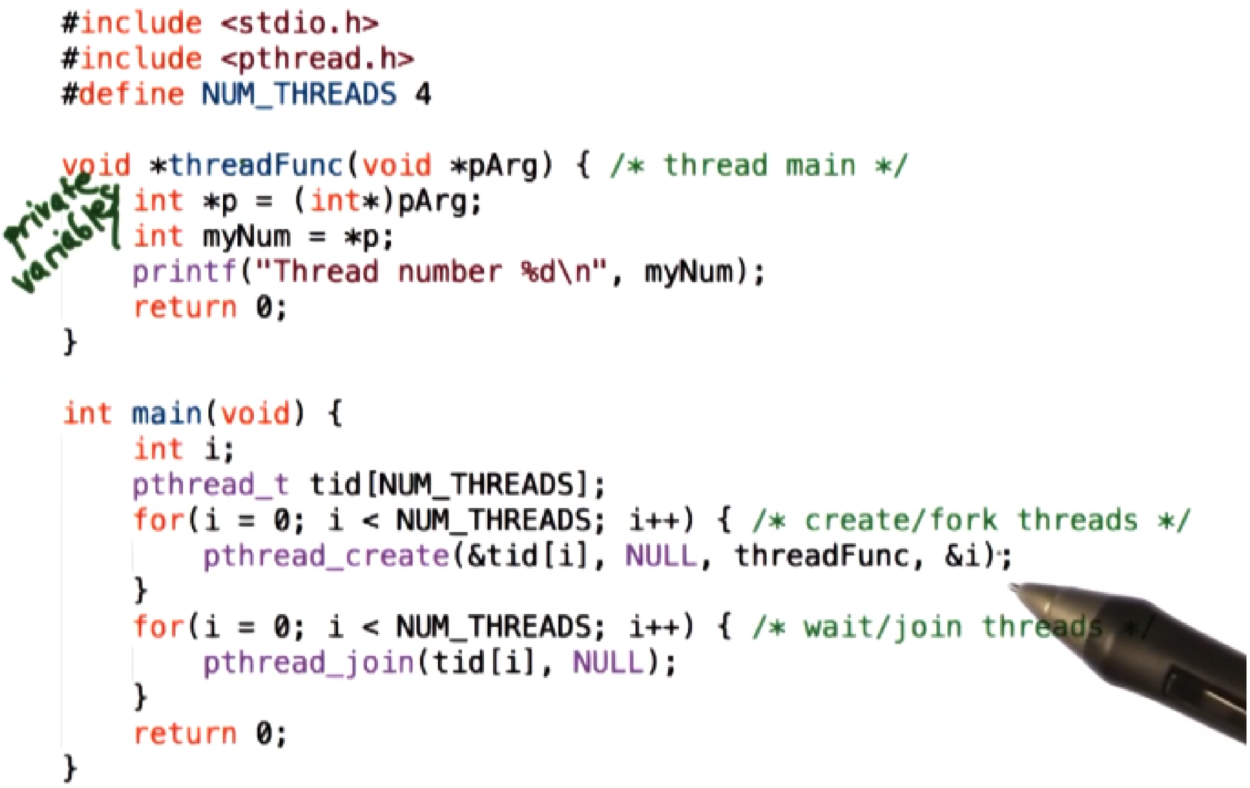


```
#include <stdio.h>
#include <pthread.h>
#define NUM_THREADS 4

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;
}
```

PThread Example 1

1. This program prints Hello world four times.

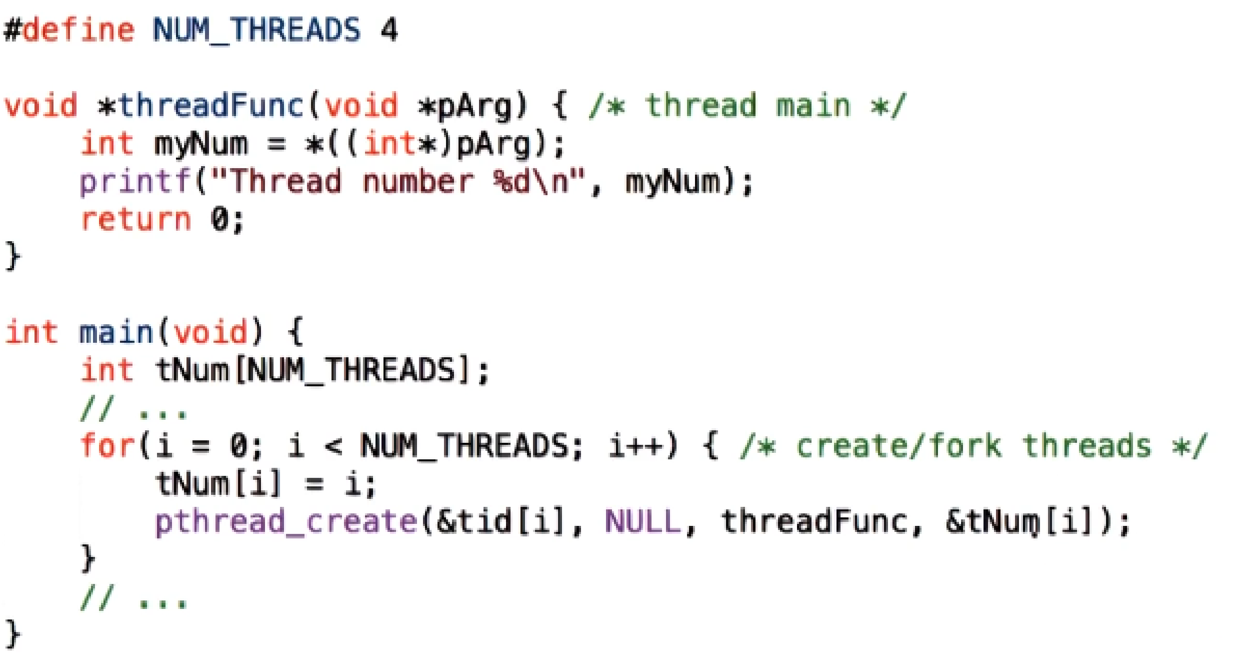
PThread Example 2



```
#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;
}
```

PThread Example 2

1. The output of this program is indeterminate.
    - Passing the address of i means that the value could be updated before the thread prints.
    - This is a race condition between the reader and writer threads.
    - This is fixed in the following example.



```
#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]);
    }
    // ...
}
```

PThread Example 2 Fixed

2. The output of this program is 0, 1, 2, 3 in an indeterminate order.

PThread Mutexes

```
* Method to solve mutual exclusion problems among concurrent threads
```

```
pthread_mutex_t aMutex; // mutex type
int pthread_mutex_lock(pthread_mutex_t* mutex); // explicit lock
int pthread_mutex_unlock(pthread_mutex_t* mutex); // explicit unlock
int pthread_mutex_init(pthread_mutex_t* mutex, const pthread_mutex_attr_t* attr);
int pthread_mutex_trylock(pthread_mutex_t* mutex); // returns if mutex is locked
int pthread_mutex_destroy(pthread_mutex_t* mutex);
```

Mutex Safety Tips

1. Shared data should always be accessed through a single mutex
2. Mutex scope must be visible to all
3. Globally order locks (lock mutexes in the same order for all threads)
4. Always unlock the correct mutex

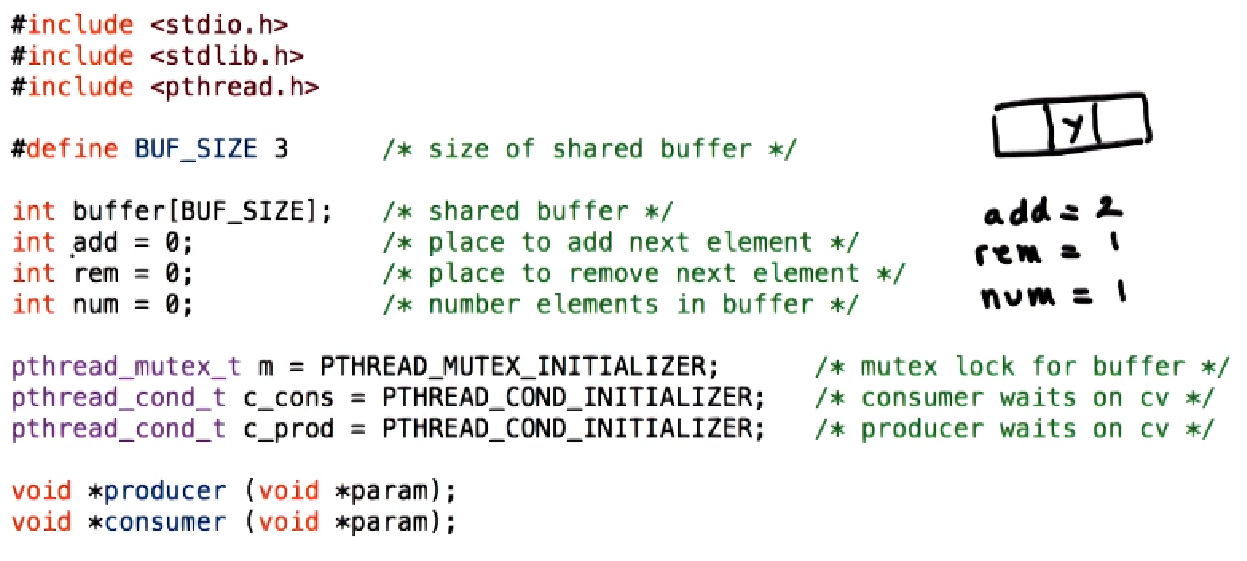
Condition Variables

```
pthread_cond_t aCond; // type of condition 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);
int pthread_cond_init(pthread_cond_t* cond, const pthread_condattr_t* attr);
int pthread_cond_destroy(pthread_cond_t* cond);
```

Condition Variables Safety Tips

1. Don't forget to notify waiting threads
   - Predicate change -> signal/broadcast correct condition variable
2. When in doubt, use broadcast (incurs performance penalty)
3. You don't need a mutex to signal/broadcast (wait until after mutex is unlocked)

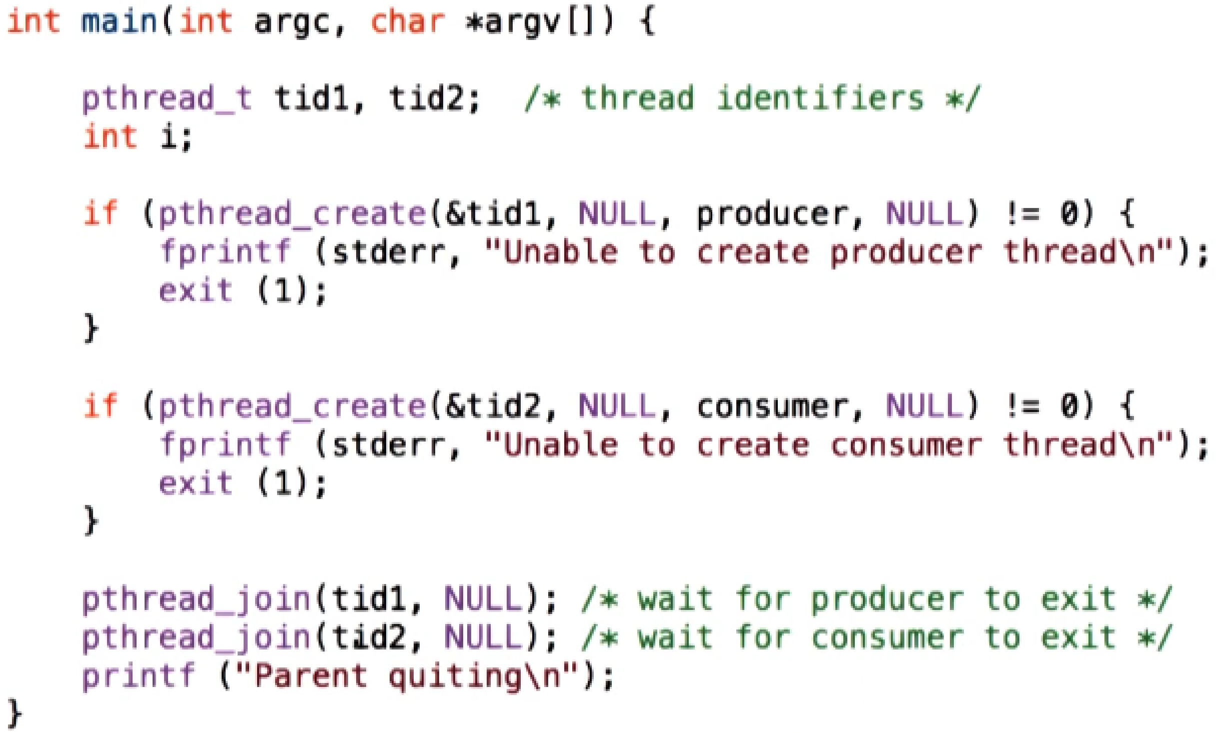
Producer/Consumer Example Using PThreads



```
#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);
```

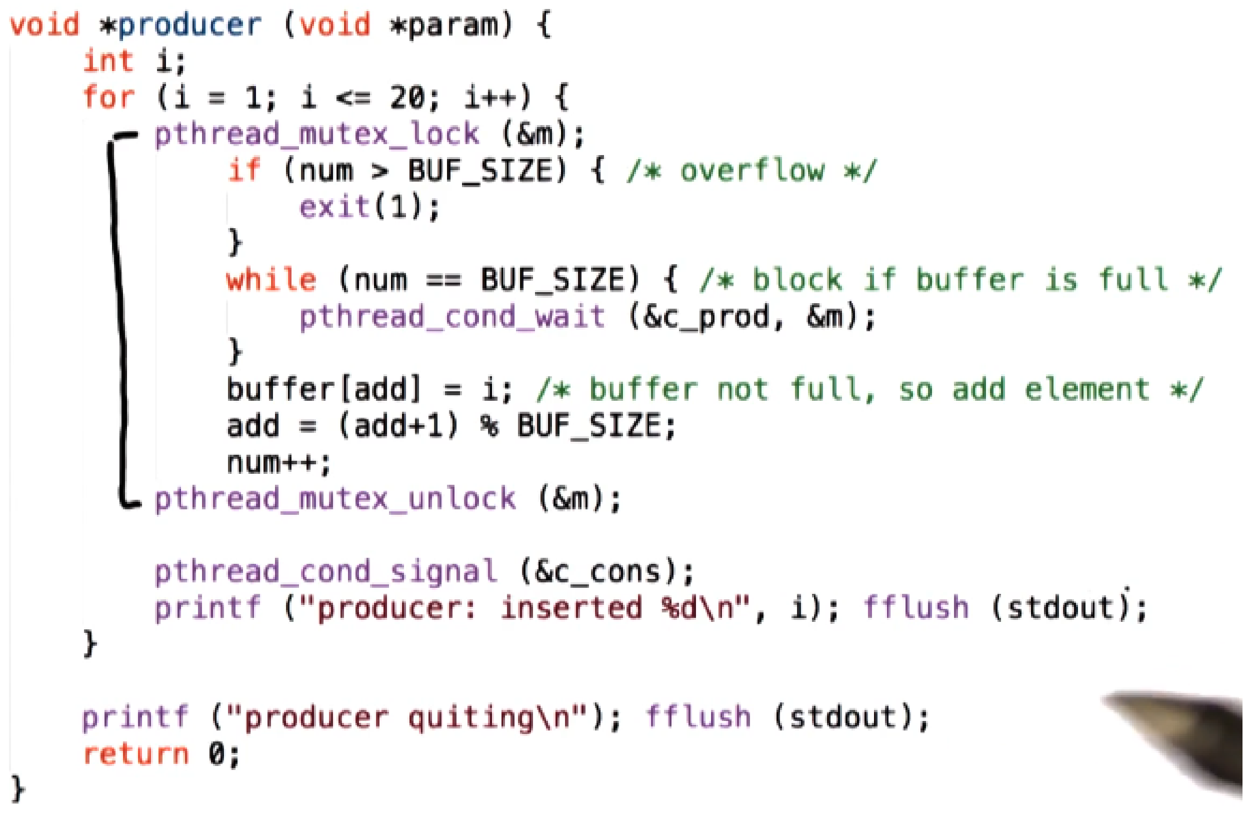
add = 2
rem = 1
num = 1

PThread Producer/Consumer Global



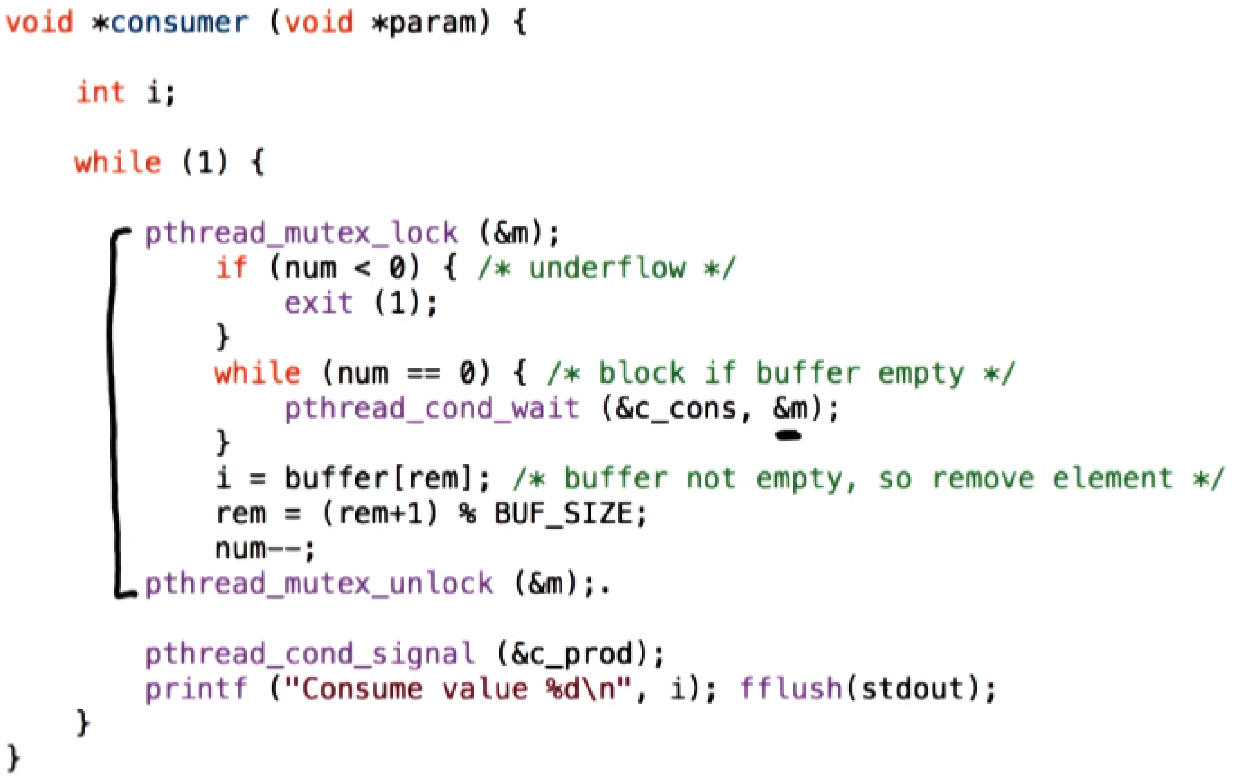
```
int main(int argc, char *argv[]) {

    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");
}
```

PThread Producer/Consumer Main



```
void *producer (void *param) {
    int i;
    for (i = 1; i <= 20; i++) {
        pthread_mutex_lock (&m);
            if (num > BUF_SIZE) { /* overflow */
                exit(1);
            }
            while (num == BUF_SIZE) { /* block if buffer is full */
                pthread_cond_wait (&c_prod, &m);
            }
            buffer[add] = i; /* buffer not full, so add element */
            add = (add+1) % BUF_SIZE;
            num++;
        pthread_mutex_unlock (&m);

        pthread_cond_signal (&c_cons);
        printf ("producer: inserted %d\n", i); fflush (stdout);
    }

    printf ("producer quiting\n"); fflush (stdout);
    return 0;
}
```

PThread Producer/Consumer Producer



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

    int i;

    while (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);
    }
}
```

PThread Producer/Consumer Consumer