The pthread Library

11/22/16
Thread operations

- **create**
  - Starts a new thread, calling a specified function.
  - Returns the thread’s ID.

- **join**
  - Block until a specified thread terminates.
  - Gives access to the thread function’s return value.

- **lock/acquire**
  - Block until the mutex is available, then claim it.

- **unlock/release**
  - Release a mutex.

- **barrier_wait**
  - Block until a specified number of threads reach the barrier.
Some pthread library functions

pthread_create

pthread_join

pthread_mutex_lock

pthread_mutex_unlock

pthread_barrier_wait
pthread_create

Returns zero on success, nonzero on error.

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

First arg is a thread ID pointer.
Second arg is usually NULL.
Third arg is the thread function.
Fourth arg is a pointer to the function’s args.
void*

int pthread_create(…,  void*  args);

void*: a pointer to any type (a generic pointer)

• all addresses are the same number of bytes
  char *cptr; int *ptr; // store 4 byte addresses

• can pass the address of any type as a void *
  pthread_create( …,  &x);  // addr of an int
  pthread_create(…,  &ch);  //addr of a char

• cannot de-reference a void * pointer
  *args = 6;  // store 6 in 1 byte? 2 bytes? 4 bytes?

• re-cast first before dereference
  *((int *)args) = 6;  // store 6 in 4 bytes
Returns zero on success, nonzero on error.

First arg is a thread ID to wait for.

Second arg is a pointer to be filled with the return value.
Example

/* pthreads "hello, world" program */
#include "pthreads.h"

void *hello(void *arg);

int main() {
    pthread_t tid[2];

    pthread_create(&tid[0], NULL, hello, NULL);
    pthread_create(&tid[1], NULL, hello, NULL);
    pthread_join(tid[0], NULL);
    pthread_join(tid[1], NULL);
    exit(0);
}

void *hello(void *arg) {
    printf("Hello, world!\n");
    return NULL;
}
How can you pass multiple args to a function with pthread_create?

You’d like to call this function when you start your thread:

```c
int find_max(int* array, int size);
```

But the start routine has to have this signature:

```c
void *(*start_routine) (void *);
```

How can you rewrite find_max as a start routine?
How can you pass multiple args to a function with pthread_create?

```c
struct max_args{
    int* arr;
    int size;
};

void* find_max(void* arg){
    int* arr = ((struct max_args*)arg)->arr;
    ...
}
```
```c
pthread_mutex_t m; // should be global

// two ways to initialize (only do one)
• m = PTHREAD_MUTEX_INITIALIZER;
• pthread_mutex_init(&m, NULL);

pthread_mutex_lock(&mutex);
// critical section code
pthread_mutex_unlock(&mutex);

pthread_mutex_destroy(&mutex);
```
pthread_barrier_t

pthread_barrier_t b; // should be global

// initialize with number of threads
pthread_barrier_init(&b, NULL, n_threads);

// section of thread parallel code
pthread_barrier_wait(&b);

pthread_barrier_destroy(&b);
In-class example of hello.c

```c
main:
    pthread_create(&tids[i], 0, thread_hello, &tid_args[i]);
    // creates a thread (thread_hello is function it will run)

thread_hello:  // each spawned thread’s “main” function
    count += i;  // count: a global var, all threads can access
    // i is local: each tid gets copy on
    // its private stack
```

```sh
cd ~/cs31
mkdir inclass
cd inclass
cp -r ~bryce/public/cs31/inclass/w12/* .
cd 11
make
./hello 5  # run a few times & try with diff num
```

vim hello.c
More pthread library functions

• Exit a thread (can also return from thread function)
  
  `pthread_exit`

• Wait until another thread sends a signal
  
  `pthread_cond_wait`
  `pthread_cond_signal`

  • These are tricky. We’ll do an example next week.
Exercise: implement your parallel algorithm for max.

Write C code using pthreads for main and a thread function that uses pthread_create and pthread_join.

- Array size M
- N threads
- Version 1: each thread returns its local max
Exercise: update your max solution to find the K largest items.

Write c code using pthreads for main and a thread function that uses pthread_create, pthread_join, and appropriate synchronization.

- Array size M
- N threads
- Fill an array with the K largest items