CS336

 Tues Sept 16, 2008
Brief Intro to C: Part 2

- Review
  - Stack frame
  - Heap
  - Pointers
  - Malloc/free

- Strings

- Passing arguments
**pthread_create()**

```c
int pthread_create(
   pthread_t *tid,     // thread ID
   const pthread_attr_t *attr,    // thread attributes
   void *(*start_routine)(void *),   // pointer to function to execute
   void *arg                // argument to function
);
```

**Arguments:**
- The thread ID of the successfully created thread.
- The thread’s attributes, explained below; the NULL value specifies default attributes.
- The function that the new thread will execute once it is created.
- An argument passed to the start_routine().

**Return value:**
- 0 if successful. Error code from <errno.h> otherwise.

**Notes:**
- Use a structure to pass multiple arguments to the start routine.
pthread_join

```c
int pthread_join(  
    pthread_t tid,  // thread ID to wait for
    void **status   // exit status
);
```

**Arguments:**
- The ID of the thread to wait for.
- The completion status of the exiting thread will be copied into `*status` unless `status` is `NULL`, in which case the completion status is not copied.

**Return value:**
- 0 for success. Error code from `<errno.h>` otherwise.

**Notes:**
- Once a thread is joined, the thread no longer exists, its thread ID is no longer valid, and it cannot be joined with any other thread.
More Demos

- pthread_ex1.c (and try on nscc so we can look at size of pointers)
- pthread_ex2.c
Acquiring and Releasing Mutexes

```c
int pthread_mutex_lock(         // Lock a mutex
    pthread_mutex_t *mutex);
int pthread_mutex_unlock(       // Unlock a mutex
    pthread_mutex_t *mutex);
int pthread_mutex_trylock(      // Nonblocking lock
    pthread_mutex_t *mutex);
```

**Arguments:**
Each function takes the address of a mutex variable.

**Return value:**
0 if successful. Error code from `<errno.h>` otherwise.

**Notes:**
The `pthread_mutex_trylock()` routine attempts to acquire a mutex but will not block. This routine returns the POSIX Threads constant `EBUSY` if the mutex is locked.
Dining Philosophers
Deadlock

- Mutual exclusion: a resource can be assigned to at most one thread
- Hold and wait: threads both hold some resources and request other resources
- No preemption: a resource that is assigned to a thread can only be released by the thread that holds it
- Circular wait: a cycle exists in which each thread waits for a resource that is assigned to another thread