POSIX Thread Programming

Topics to be Covered

- Objective
- POSIX threads
- What are Threads?
- Creating threads
- Using threads
- Summary

Creating threads

- Always include pthread library:

  #include <pthread.h>

  int pthread_create (pthread_t *threadp, const pthread_attr_t * attr, void *(* start routine)(void *), void *arg);
- This creates a new thread of control that calls the function start_routine.
- It return a zero if the creation is successful, and thread id in threadp (first parameter).
- attr is to modify the attributes of the new thread. If it is NULL, default attributes are used.
- *arg* is to pass arguments to the thread function.

Using threads

1. Declare a variable of type *pthread_t*
2. Define a function to be executed by the thread.
3. Create the thread using *pthread_create*

   Make sure creation is successful by checking the return value.
4. Pass any arguments need through' arg (packing and unpacking arg list if needed)
6. Compile: *cc –o xyz xyz.c*

{+ options such as -lpthread}

Thread's local data

- Variables declared within a thread (function) are called local data.
- Local (static) data associated with a thread are allocated on the stack. So these may be deallocated when a thread returns.
- So don't plan on using locally declared variables for returning arguments. Plan to pass the arguments thru argument list passed from the caller or initiator of the thread.

Thread termination (destruction)

Implicit : Simply returning from the function executed by the thread terminates the thread. In this case thread's completion status is set to the return value.

- Explicit : Use *thread_exit*.

  void thread_exit(void *status);

The single pointer value in status is available to the threads waiting for this thread.

Waiting for thread exit

int pthread_join (pthread_t tid, void **statusp);

- A call to this function makes a thread wait for another thread whose *thread_id* is specified by tid in the above prototype.
- When the thread specified by *tid* exits its completion status is stored(returned) in *statusp*.

Other pthread functions

pthread_t pthread_self(void)

- get the “handle” for itself.
- Synchronization:

  pthread_mutex_t
  lock=PTHREAD_MUTEX_INITIALIZER;

  pthread_mutex_lock (&lock);

  pthread_mutex_unlock(&lock);

Synchronization (ctd)

```
pthread_cond_t c;
int pthread_cond_init();
int pthread_cond_wait();
int pthread_cond_signal();
```

Other pthread functions

- Posix Semaphores

```
#include <semaphore.h>
sem_t s;
sem_init(sem_t *s, int shared, uint_t value);
sem_wait(sem_t *s);
sem_post(sem_t *s);
sem_getvalue(sem_t *s);
sema_destroy(sem_t *s);
```