SE3910 – REAL TIME SYSTEMS

Designing Multithreaded Software For the Beaglebone

OBJECTIVES

- List the methods commonly used in POSIX to manage pthreads and explain their purposes.
- Explain what a friend function is in C++.
- Explain what the purpose of a start method is in a Java thread.

FOR THE GOOD OF THE ORDER

- Lab tomorrow
  - Need to come in with circuit design drawn out in advance.

OBJECTIVES

- Explain the purpose for a watchdog timer
  - Finish up from Monday
- Explain how to design a simple multithreaded application using POSIX
  - Review from CS3844
- Explain the concept of conditional compilation
  - How can we use conditional compilation to our benefit

QUESTION

- How do you start designing a multithreaded application in C using POSIX?

WHY DO WE MULTITHREAD

HTTPS://COMPUTING.LLNL.GOV/TUTORIALS/PTHREADS/

Platform	fork()			pthread_create()		
	real	user	sys	real	user	sys
Intel 2.6 GHz Xeon E5-2670 (16 cores/node)	8.1	0.1	2.9	0.9	0.2	0.3
Intel 2.8 GHz Xeon 5660 (12 cores/node)	4.4	0.4	4.3	0.7	0.2	0.5
AMD 2.3 GHz Opteron (16 cores/node)	12.5	1.0	12.5	1.2	0.2	1.3
AMD 2.4 GHz Opteron (8 cores/node)	17.6	2.2	15.7	1.4	0.3	1.3
IBM 4.0 GHz POWER6 (8 cpus/node)	9.5	0.6	8.8	1.6	0.1	0.4
IBM 1.9 GHz POWER5 p5-575 (8 cpus/node)	64.2	30.7	27.6	1.7	0.6	1.1
IBM 1.5 GHz POWER4 (8 cpus/node)	104.5	48.6	47.2	2.1	1.0	1.5
INTEL 2.4 GHz Xeon (2 cpus/node)	54.9	1.5	20.8	1.6	0.7	0.9
INTEL 1.4 GHz Itanium2 (4 cpus/node)	54.5	1.1	22.2	2.0	1.2	0.6

Source fork_vs_thread.txt

POSIX THREADS (PTHREADS) INTERFACE

- *Pthreads*: Standard interface for ~60 functions that manipulate threads from C programs
  - Creating and reaping threads
    - `pthread_create, pthread_join`
  - Determining your thread ID
    - `pthread_self`
  - Terminating threads
    - `pthread_cancel, pthread_exit`
    - `exit` [terminates all threads], `return` [terminates current thread]
  - Synchronizing access to shared variables
    - `pthread_mutex_init, pthread_mutex_[un]lock`
    - `pthread_cond_init, pthread_cond_[timed]wait`

THE PTHREADS "HELLO, WORLD" PROGRAM

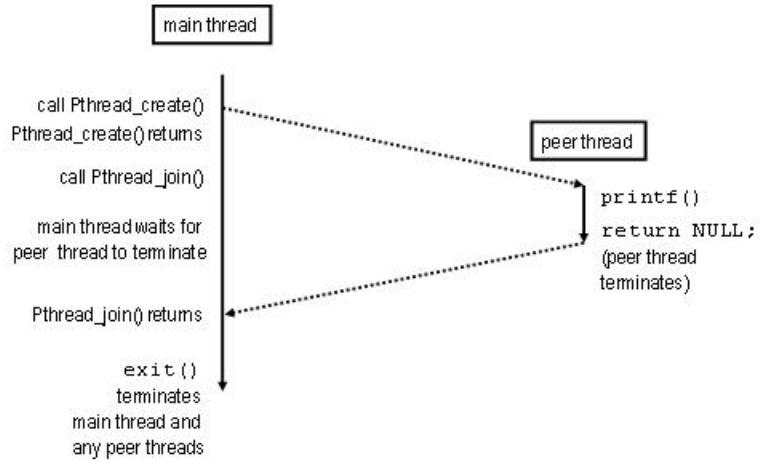
```
/*
 * hello.c - Pthreads "hello, world" program
 */
#include "csapp.h"

void *howdy(void *vargp);

int main() {
  pthread_t tid;

  Pthread_create(&tid, NULL, howdy, NULL);
  Pthread_join(tid, NULL);
  exit(0);
}

/* thread routine */
void *howdy(void *vargp) {
  printf("Hello, world!\n");
  return NULL;
}
```

Thread attributes (usually NULL)

*Thread arguments (void *p)*

*return value (void **p)*

EXECUTION OF THREADED "HELLO, WORLD"



C++ POP QUIZ

- Explain to me what a friend function is.

FRIEND FUNCTION

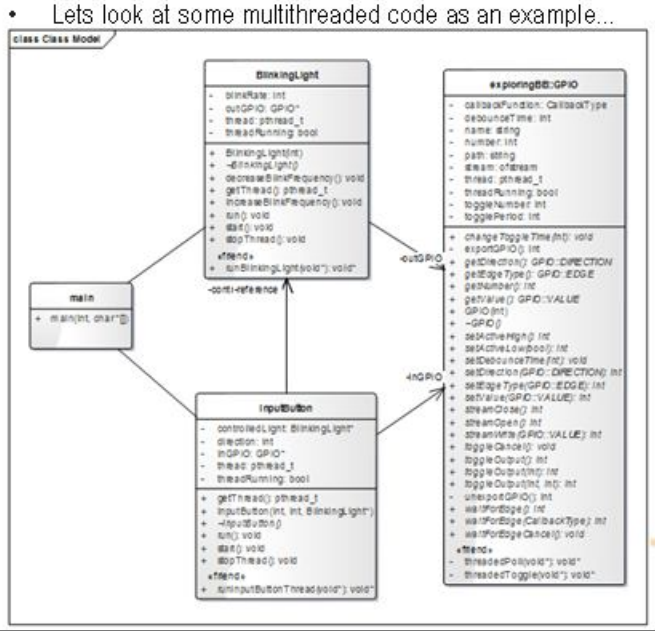
- A friend function of a class is defined outside that class' scope but it has the right to access all private and protected members of the class. Even though the prototypes for friend functions appear in the class definition, friends are not member functions.

POP QUIZ #2

How do we use multithreading in Java?

TUTORIAL

- Lets look at some multithreaded code as an example...



C++ THREADS

- C++11 threads are the new kid on the block
- std::thread class now part of standard C++ library
- std::thread is an abstraction — maps to local platform threads
  - (POSIX, Windows, etc.)

C++11 THREAD EXAMPLE

```
#include <thread>
#include <iostream>

void func()
{
  std::cout << "**Inside thread "
   << std::this_thread::get_id() << "!" << std::endl;
}

int main()
{
  std::thread  t( func );


  t.join();
  return 0;
}
```

A simple function for thread to do…

Create & schedule thread to execut

Wait for thread to finish…

C++11 THREAD CONSTRUCTORS

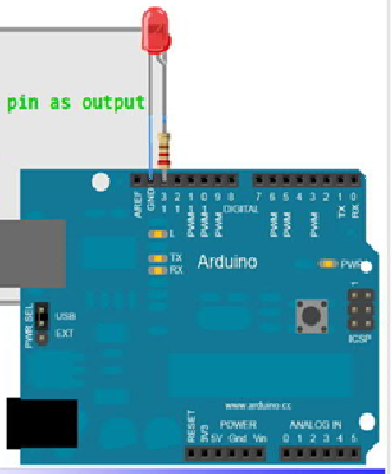
- class thread
- {
- thread(); // creates new thread object that does *not* represent a thread (i.e. not joinable)
- thread(std::Function&& f, Args&&... args); // creates new thread to execute f
- thread(thread&& other); // *move* constructor
- thread(thread& other); // *copy* constructor --- not available

EXAMPLE ON THE BEAGLEBONE

MULITHTHREADED EMBEDDED SYSTEM

```
// adapted from the Arduino Blink Tutorial (*)
void main() {
    pinMode(LED_PIN, OUTPUT); // setup: set the LED pin as output
    while (1) {  // endless loop
        digitalWrite(LED_PIN, HIGH); // turn LED on
        delay(1000); // wait for 1000ms
        digitalWrite(LED_PIN, LOW); // turn LED off
        delay(1000); // wait for 1000ms
    }
}
```

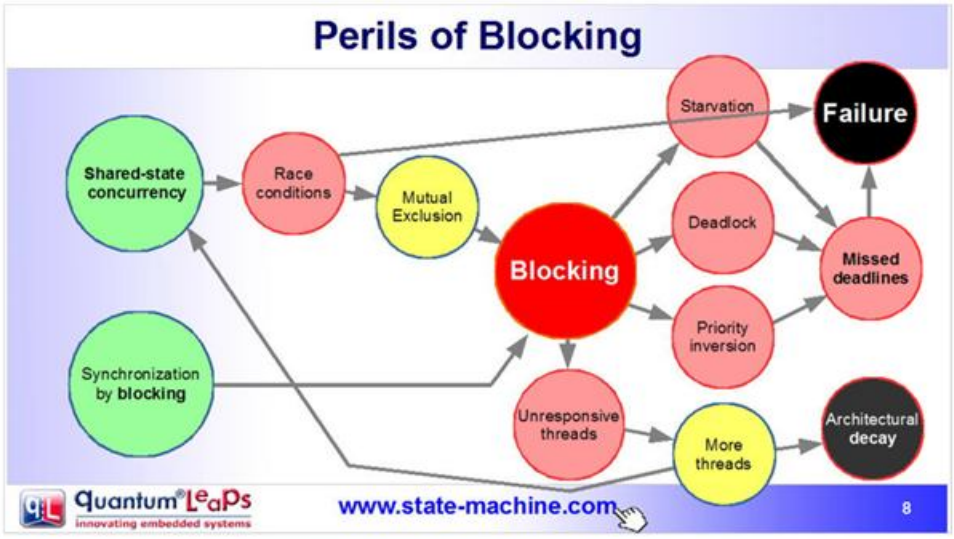
(*) Arduino Blink Tutorial: http://www.arduino.cc/en/Tutorial/Blink



RTOS BENEFITS

- Divide and Conquer strategy
  - Multiple threads make it easier to develop code
- More efficient CPU usage
  - Threads that are blocked do not consume CPU cycles
- Threads can be decoupled in the time domain
  - Changes in low priority threads have no impact on the timing of high priority threads
  - RMA can be applied

PROBLEMS: COMMUNICATION



BEST PRACTICES

- Don't specifically block inside of code
  - Communicate and synchronize threads asynchronously via event objects
- Don't share data or resources amongst threads
  - Keep data isolated and bound to threads
    - Strict encapsulation
- Structure threads as message pumps
  - Event queue + event loop