pthreads

CS449 Fall 2015
Quiz 2

![Quiz 2 Chart]

- 10 or less: 0
- 11 to 20: 0
- 21 to 30: 2
- 31 to 40: 9
- 41 to 50: 26
Quiz 2 Statistics

- Max: 50
- Min: 23
- Average: 43.38
- Median: 46
Pthreads

• Pthreads (POSIX threads) is a standard or API for doing threading

• Can be implemented using User threading or Kernel threading

• Users can be oblivious of underlying implementation
POSIX

- Portable Operating System Interface
- Standard to unify the programs and system calls that many different OSes provide.
Linux Thread Implementation

• Linux Native POSIX Thread Library
  – Native: Implemented using kernel threading
  – POSIX Thread: Follows the Pthread standard
  – Library: Implemented in the form of a library (that you have to link to your program)
  – Rely on kernel to create / schedule threads

• Compare: Windows Thread API
  – Not POSIX compliant but Pthreads ports exist
Compiling with Pthreads

• Need the –pthread option to gcc for linking and compiling

  gcc -o threadtest threadtest.c -pthread
  – Links in the library libpthread.so
  – Defines macros that enables thread-safe code using #ifdefs

• DO NOT use the –lpthread option
  – Will link in libpthread.so library but not define macros
  – Some C library calls will become thread-unsafe
    (E.g. errno global variable with file operations)
Pthread API

- In `<pthread.h>`,

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

void pthread_exit(void * value_ptr);

int pthread_join(pthread_t thread, void ** value_ptr);

int pthread_yield(void);
```
`pthread_create()`

```c
#include <stdio.h>
#include <pthread.h>

void *do_stuff(void *p) {
    printf("Hello from thread %d\n", *(int *)p);
}

int main() {
    pthread_t thread;
    int id, arg1, arg2;

    arg1 = 1;
    id = pthread_create(&thread, NULL, do_stuff, (void *)&arg1);
    arg2 = 2;
    do_stuff((void *)&arg2);
    return 0;
}
```
Output

Hello from thread 2
#include <stdio.h>
#include <pthread.h>

void *do_stuff(void *p)
{
    printf("Hello from thread \%d\n", *(int *)p);
}

int main()
{
    pthread_t thread;
    int id, arg1, arg2;

    arg1 = 1;
    id = pthread_create(&thread, NULL, do_stuff, (void *)&arg1);
    pthread_yield();
    arg2 = 2;
    do_stuff((void *)&arg2);

    return 0;
}
Output

Hello from thread 1
Hello from thread 2
#include <stdio.h>
#include <pthread.h>

void *do_stuff(void *p)
{
    printf("Hello from thread %d\n", *(int *)p);
}

int main()
{
    pthread_t thread;
    int id, arg1, arg2;

    arg1 = 1;
    id = pthread_create(&thread, NULL, do_stuff, (void *)&arg1);
    arg2 = 2;
    do_stuff((void *)&arg2);
    pthread_join(thread, NULL);
    return 0;
}
Output

Hello from thread 2
Hello from thread 1
```c
int pthread_create(
    pthread_t *restrict thread,
    const pthread_attr_t *restrict attr,
    void *(*start_routine)(void*),
    void *restrict arg
);
```

- A unique identifier for the thread
- Thread attributes or NULL for the default
- A C Function Pointer
- The argument to pass to the function
Start Routine Prototype

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

• Why void * as argument type?

• How to declare a prototype that receives a variable number of argument values?
  – Can’t use variadic functions
    • A new thread starts in its own stack
  – Declare a struct with as many fields as args
  – Pass pointer to struct in a void *
pthread_exit() / pthread_join()

void pthread_exit(void * value_ptr);
• The value that is “returned” by thread
  – Threads have separate processor registers and stacks so need a special way of “returning”
int pthread_join(pthread_t thread, void ** value_ptr);
• Unique identifier for joined thread
• The address of location that will be updated to the value that is “returned” by joined thread
• Why are the types for value_ptr different?
```
struct Value { ... };  
void* thread_func(void *p) {  
    struct Value* vall1 = malloc(sizeof(struct Value));  
    ...  
    pthread_exit(vall1);  
}  
int main() {  
    struct Value *val2;  
    ...  
    pthread_join(thread, &val2);  
    // vall1 == val2  
}  
```
class TestThread implements Runnable {
    private int x;
    public static void main(String[] args) {
        Thread t1 = new Thread(new TestThread(1));
        Thread t2 = new Thread(new TestThread(2));

        t1.start();
        t2.start();
    }

    public void run() {
        System.out.println("Hello from thread " + x);
    }

    public TestThread(int y) { x = y; }
}
Output

Hello from thread 1
Hello from thread 2