Grand Central Dispatch

Sri Teja Basava

CSCI 5528: Foundations of Software Engineering

Spring '10

New Technologies in Snow Leopard

64-bit

Grand Central Dispatch

OpenCL

QuickTime X

Grand Central Dispatch

- An Apple technology to optimize application support for systems with multicore processors
- Released with Mac OS X Snow Leopard (v10.6)
- Shifts responsibility for managing threads and their execution from applications to the operating system

Grand Central Dispatch

- Provides a new programming model consisting of *blocks* and *queues*
- GCD consists of a set of extensions to the C language, an API, and a runtime engine
- Apple released the source code for libdispatch, the library providing the implementation of GCD's services, under the Apache License on September 10, 2009

Traditional Approach

- To create an efficient application for multi-core using threads, a programmer must
  - Break each logical task down to a single thread
  - Lock data that can be modified by two or more threads at the same time
  - Implement a thread pool with as many threads as there are available cores
  - Hope that no other applications are using the processor cores

GCD Approach

- To create an efficient application for multi-core using GCD, a programmer needs to
  - Identify units of work (think *tasks*) and describe them using blocks
  - Assign blocks to different queues based on how they need to be executed
- No need to worry about threads, thread managers, or locking data!

Benefits

- Improved responsiveness
- Dynamic scaling
- Better processor utilization
- Smaller & cleaner code

Block Objects

- An extension to C, C++, and Objective-C
- Allow programmers to define self-contained units of work
- Similar to function pointers, but far more powerful
  - Block objects can be defined inline, as “anonymous functions”
  - Block objects can refer to variables defined outside of their bodies
- Internally implemented as a function pointer plus context data and optional support routines

Block Objects

- Similar to function pointers, but far more powerful
  - Block objects can be defined inline, as “anonymous functions”
  - Block objects can refer to variables defined outside of their bodies

Example 1

```
void (^blk)(void);

blk = ^{ printf("Hello World!\n"); };

blk(); /* prints Hello World! */
```

Block Objects

- Similar to function pointers, but far more powerful
  - Block objects can be defined inline, as "anonymous functions"
  - Block objects can refer to variables defined outside of their bodies

Example 2

```
int (^sum)(int, int);

sum = ^(int x, int y){ return x + y; };

printf("%d\n", sum(4, 5)); /* prints 9 */
```

The compiler infers the return type of the block literal!

Block Objects

- Similar to function pointers, but far more powerful
  - Block objects can be defined inline, as "anonymous functions"
  - Block objects can refer to variables defined outside of their bodies

Example 3

```
int (^addtovar)(int);
int var = 5;

addtovar = ^(int x){ return x + var; };

var = 6;

printf("%d\n", addtovar(4)); /* prints 9 */
```

The block captures a read-only copy of `var`.

Block Objects

- Similar to function pointers, but far more powerful
  - Block objects can be defined inline, as “anonymous functions”
  - Block objects can refer to variables defined outside of their bodies

Example 4

```
int (^addtovar)(int);
__block int var = 5;

addtovar = ^(int x){ return x + var; };

var = 6;

printf("%d\n", addtovar(4)); /* prints 10 */
```

`__block` storage type enable `var` to be edited inside the body.

Dispatch Queues

- Blocks are scheduled for execution by placing them on various system- or user-defined dispatch queues
- Blocks are added and removed from queues using atomic operations
- 3 types of dispatch queues
  - Global concurrent queues
  - Private serial queues
  - Main queue

Global Queues

- GCD provides a set of global concurrent queues to each process
- Each queue has an associated priority
- Each queue is associated with a pool of threads, created as needed based on the work to be done and the load on the rest of the operating system

Global Queues

- For each global concurrent queue with blocks
  - GCD searches for an available thread at the appropriate priority
  - If a thread is found, GCD dequeues a block (on a FIFO basis) and assigns it for execution on the thread
  - When the thread completes the work and becomes available, GCD dequeues another block (if available) for execution on the thread

Global Queues

Example 4

```
dispatch_queue_t q_default;

/* get default queue */
q_default = dispatch_get_global_queue(0, 0);

dispatch_async(q_default, ^{ work(); });
```

`dispatch_async` enqueues the specified block on the default queue and returns immediately.

Global Queues

Example 5

```
#define COUNT 128
__block double result[COUNT];
dispatch_apply(COUNT, q_default, ^(size_t i) {
    result[i] = complex_calculation(i);
});
double sum = 0;
for (int i=0; i < COUNT; i++) sum += result[i];
```

`dispatch_apply` can be used to parallelize for loops. It is synchronous.

Private Queues

"islands of serialization in a sea of concurrency"

- Programmers can create their own private serial queues to serialize access to data structures
- Blocks in a private queue are executed one after another, never concurrently
- Each private queue has an associated target global concurrent queue, initially set to the default queue

Private Queues

"islands of serialization in a sea of concurrency"

- When a developer adds a block to an empty serial queue
  - The private queue is added to the target queue
  - The private queue is treated in the same way as blocks added directly to the the target queue; it is executed using the same policy and mechanism as these blocks
  - When the private queue is executed, it dequeues each block (on a FIFO basis) and executes them one after another

Private Queues

"islands of serialization in a sea of concurrency"

Example 6

```
#define COUNT 128
__block double sum = 0;
dispatch_queue_t q_sum = dispatch_queue_create("com.example.sum", NULL);
dispatch_apply(COUNT, q_default, ^(size_t i){
   double x = complex_calculation(i);
   dispatch_async(q_sum, ^{ sum += x; });
});
dispatch_release(q_sum);
```

The private queue `q_sum` is used to serialize access to shared variable `sum`.

Main Queue

- Associated with the main thread of every process is a unique, well-known main-queue
- Main queue is always serial
- Typically associated with CFRunLoop (for Core Foundation) or NSRunLoop (for Cocoa) on the main thread. Both must drain the main queue at the end of their work cycles.

Event Sources

- Programmers can assign blocks as handlers to event sources such as timers, signals, file descriptors and network sockets
- When an event triggers, GCD schedules the associated handler on a queue if it is not currently running. GCD will coalesce pending events if it is.
- The handler is never run more than once at a time

Example

- Algorithm for computing approximate value of PI
  - Multi-threaded implementation using pthreads (shown in class before)
  - Multi-threaded implementation using GCB

Example

- Compiled using gcc -O3
- Runtime measurement using time utility

pthread

real	0m9.454s
user	0m17.976s
sys	0m0.041s

GCD

real	0m10.642s
user	0m20.479s
sys	0m0.036s

Questions?