

UNIT II

IMPLICIT THREADING , THREADING ISSUES





Implicit Threading

- Growing in popularity as numbers of threads increase, program correctness more difficult with explicit threads
- Creation and management of threads done by compilers and run-time libraries rather than programmers
- Three methods explored
 - Thread Pools
 - OpenMP
 - Grand Central Dispatch
- Other methods include Microsoft Threading Building Blocks (TBB),
java.util.concurrent package

- Create a number of threads in a pool where they await work
- **Advantages:**
 - Usually slightly faster to service a request with an existing thread than create a new thread
 - Allows the number of threads in the application(s) to be bound to the size of the pool
 - Separating task to be performed from mechanics of creating task allows different strategies for running task
- Windows API supports thread pools:

```
DWORD WINAPI PoolFunction(AVOID Param) {  
    /*  
    * this function runs as a separate thread.  
    */  
}
```



sns
INSTITUTIONS

Grand Central Dispatch

- Apple technology for Mac OS X and iOS operating systems
- Extensions to C, C++ languages, API, and run-time library
- Allows identification of parallel sections
- Manages most of the details of threading
- Block is in “`^{} - ^{ printf("I am a block"); }`”
- Blocks placed in dispatch queue
 - Assigned to available thread in thread pool when removed from queue



sns
INSTITUTIONS

Grand Central Dispatch

- **Two types of dispatch queues:**

- serial – blocks removed in FIFO order, queue is per process, called **main queue**
 - Programmers can create additional serial queues within program
- concurrent – removed in FIFO order but several may be removed at a time
 - Three system wide queues with priorities low, default, high

```
dispatch_queue_t queue = dispatch_get_global_queue  
    (DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);
```

```
dispatch_async(queue, ^{ printf("I am a block."); });
```




sns
INSTITUTIONS

Threading Issues

- Semantics of **fork()** and **exec()** system calls
- Signal handling
 - Synchronous and asynchronous
- Thread cancellation of target thread
 - Asynchronous or deferred
- Thread-local storage
- Scheduler Activations



sns
INSTITUTIONS

Semantics of `fork()` and `exec()`

- Does **fork()** duplicate only the calling thread or all threads?
 - Some UNIXes have two versions of fork
- **exec()** usually works as normal – replace the running process including all threads



sns
INSTITUTIONS

Signal Handling

- **Signals** are used in UNIX systems to notify a process that a particular event has occurred.
- A **signal handler** is used to process signals
 1. Signal is generated by particular event
 2. Signal is delivered to a process
 3. Signal is handled by one of two signal handlers:
 1. default
 2. user-defined
- Every signal has **default handler** that kernel runs when handling signal
 - **User-defined signal handler** can override default
 - For single-threaded, signal delivered to process



sns
INSTITUTIONS

Signal Handling (Cont.)

- Where should a signal be delivered for multi-threaded?
 - Deliver the signal to the thread to which the signal applies
 - Deliver the signal to every thread in the process
 - Deliver the signal to certain threads in the process
 - Assign a specific thread to receive all signals for the process



sns
INSTITUTIONS

Thread Cancellation

- Terminating a thread before it has finished
- Thread to be canceled is **target thread**
- Two general approaches:
 - **Asynchronous cancellation** terminates the target thread immediately
 - **Deferred cancellation** allows the target thread to periodically check if it should be cancelled
- Pthread code to create and cancel a thread:

```
pthread_t tid;

/* create the thread */
pthread_create(&tid, 0, worker, NULL);

. . .

/* cancel the thread */
pthread_cancel(tid);
```



Thread Cancellation (Cont.)

- Invoking thread cancellation requests cancellation, but actual cancellation depends on thread state

Mode	State	Type
Off	Disabled	–
Deferred	Enabled	Deferred
Asynchronous	Enabled	Asynchronous

- If thread has cancellation disabled, cancellation remains pending until thread enables it
- Default type is deferred
 - Cancellation only occurs when thread reaches **cancellation point**
 - I.e. `pthread_testcancel()`
 - Then **cleanup handler** is invoked
- On Linux systems, thread cancellation is handled through signals



sns
INSTITUTIONS

Thread-Local Storage

- **Thread-local storage (TLS)** allows each thread to have its own copy of data
- Useful when you do not have control over the thread creation process (i.e., when using a thread pool)
- Different from local variables
 - Local variables visible only during single function invocation
 - TLS visible across function invocations
- Similar to **static** data
 - TLS is unique to each thread