Threads

Chapter 5
Chapter 5: Threads

- Overview
- Multithreading Models
- Threading Issues
- Pthreads
- Solaris 2 Threads
- Windows 2000 Threads
- Nachos Threads
Single and Multithreaded Processes

Thread specific Data (TSD)
User Threads

Thread management done by user-level threads library

Examples

- POSIX Pthreads
- Solaris threads
- Nachos Thread
Kernel Threads

- Supported by the Kernel

Examples
- Windows 95/98/NT/2000
- Solaris
- Tru64 UNIX
- BeOS
- Linux
Thread Models

Many to One:
- Many user-level threads mapped to single kernel thread.
- Used on systems that do not support kernel threads.

One to One
- Each user-level thread maps to kernel thread.
- Examples
  - Windows 95/98/NT/2000
  - OS/2

Many to Many
Many-to-One Model

- User thread
- Kernel thread

Diagram illustrates the relationship between a user thread and a kernel thread in a Many-to-One Model.
One-to-one Model
Many-to-Many Model

- Allows many user level threads to be mapped to many kernel threads.
- Allows the operating system to create a sufficient number of kernel threads.
- Solaris 2
- Windows NT/2000 with the *ThreadFiber* package
Many-to-Many Model

![Diagram of Many-to-Many Model with kernel thread and user thread connections]

- Many-to-Many Model diagram showing multiple nodes connected to a central node, representing the relationships between different entities.

- Nodes labeled 'k' connected to the central node, indicating multiple entities.

- Arrows indicating connections between the nodes and the central node, with labels 'user thread' and 'kernel thread.'
Thread

- Thread
- Data Structures
- Thread control methods
Threading Issues

- Thread life cycle management
  - Thread create, thread suspend, thread yield, thread resume, thread sleep, thread kill, thread start, thread stop
- Thread specific data
- Thread pools
- Signal handling
Pthreads

- a POSIX standard (IEEE 1003.1c) API for thread creation and synchronization.
- API specifies behavior of the thread library, implementation is up to development of the library.
- Common in UNIX operating systems.
- Simply a collection of C function.
Solaris 2 Threads
Solaris Process

Solaris process

process id
memory map
priority
list of open files
LWP_1  LWP_2  LWP_3  \ldots
Nachos Thread

Nachos process consists of

- An address space
  - Executable code
  - Stack space for automatic (local) variables
  - Heap space for global variables
- A single thread of control
- Other object such as file descriptors
- We allow multiple threads of control to execute concurrently.
Thread States

- JUST_CREATED
- READY
- BLOCKED
- RUNNING

States transition:
- Thread()
- Fork()
- Scheduler dispatch
- Event occurs
- Waiting for an event
Nachos Thread Operations

**Creation:**
- Thread *(char *debugName)*;
  Create a thread but no code attached yet.
- Fork *(voidfunctionPtr func, int arg)*;
  Associate a function to be executed by the thread.

**Control:**
- void Yield()
  Suspend the current thread (calling thread..self) and select a new one for execution from the Ready queue.
Nachos Thread Operation

- **void Sleep()**
  Suspend the current thread, change its state to BLOCKED; It will return to READY on when a particular event takes place.

- **void Finish()**
  Terminate current thread. Return all data structures held to the system; Finish sets the global variable threadToBeDestroyed to point to current thread, then calls sleep to effectively terminate the thread. Scheduler will start running another thread, this thread will examine threadToBeDestroyed and finish it off.
Mechanics of Thread Switching

Switching the CPU from one thread to another involves suspending the current thread, switching its state (e.g., registers), then restoring the state of the thread being switched to. Thread switch is complete the moment the program counter is loaded into the PC.
Routine switch( oldThread, newThread) actually performs a thread switch.

Routine switch is written in assembly language and therefore is machine dependent.

Semantics of switch:
1. Save all registers in oldThread’s context block.
2. Save the return address in the context block.
3. Load new values into the register from the context block of next thread.
4. Replace PC with the saved PC in the process table; now switch is done.
5. Context switch has taken place.
Threads and Scheduling

 Threads that are ready to run are kept in a ready to run queue.
 Scheduler is invoked when ever time slice is over or if the current thread wants to give up the CPU.
 Threads suspended move to the end of the ready list.
Scheduler Routines

Void ReadyToRun(Thread *thread)
Make the thread ready to run and place it in the ready to run list
ReadyToRun is invoked for example by the Fork routine.

Thread FineNextToRun()
Select a ready thread and return it.
Run() scheduler method

```c
void Run(Thread *nextThread)
Suspend the current thread and switch to the new one.
Semantics:
1. Check if the current thread overflowed its stack. If yes, reset it.
2. Change the state of new thread to RUNNING.
3. Invoke switch to switch to the next thread.
4. If the previous thread is to be destroyed then do it. (Threads cannot terminate themselves.)
```
We will discuss thread synchronization later.