Chapter 5: Threads

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

Threads are lightweight processes – have own thread ID, PC, registers, stack

![Diagram showing single-threaded and multithreaded processes with their respective memory segments (code, data, files) and stacks (registers, stack)]
Benefits

- Responsiveness
  - Can run even if one thread blocked or busy
  - Web browser example – one thread per client

- Resource Sharing

- Economy
  - Creating and context switching threads is low cost
  - Solaris 2: creating 30x, context switch 5x slower for procs

- Utilization of MP Architectures
  - Run each thread on different CPU
User Threads

- Thread management done by user-level threads library
- No need for kernel intervention
- Drawback: all may run in single process. If one blocks, all block.
- Examples
  - POSIX Pthreads
  - Mach C-threads
  - Solaris threads
Kernel Threads

- Supported by the Kernel
- Generally slower to create than user threads
- If one blocks another in the application can be run
- Can be scheduled on different CPUs in multiprocessor
- Examples
  - Windows 95/98/NT/2000
  - Solaris
  - Tru64 UNIX
  - BeOS
  - Linux
Multithreading Models

- Many-to-One
- One-to-One
- Many-to-Many
Many-to-One

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

user thread

kernel thread
One-to-One

- Each user-level thread maps to kernel thread.

- Examples
  - Windows 95/98/NT/2000
  - OS/2
One-to-one Model

user thread

kernel thread
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
- IRIX
- HP-UX
- Tru64 Unix
- Windows NT/2000 with the *ThreadFiber* package
Many-to-Many Model

user thread

kernel thread
Threading Issues

- Semantics of fork() and exec() system calls
  - Duplicate all threads or not
  - Exec replaces all threads
  - If call exec next no need to duplicate all threads.

- Thread cancellation.
  - Asynchronous or deferred (target thread checks periodically)
  - Resource reclamation problem

- Thread pools
  - Create pool of threads to do work
  - When server receives request awakens thread. Returns on finish.
  - Advantages:
    - Faster than creating threads
    - Limits number of threads in server and hence load on CPU

- Thread specific data
Threading Issues

- Signal handling
  - Signals can be synchronous (e.g., illegal memory access) or asynchronous (e.g., i/o completion, ^C)
  - Handled by default handler or user-defined handler
  - Where should the thread be delivered?
    - To thread to which applies (synchronous signals)
    - To all threads in process
    - To certain threads in process
    - Assign a specific thread to receive all signals (Solaris 2)
Pthreads

- a POSIX standard (IEEE 1003.1c) API for thread creation and synchronization.
- API specifies behavior of the thread library, implementation is up to developer of the library.
- Common in UNIX operating systems.
Pthreads example

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

int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* the thread */
main(int argc, char *argv[])
{
    pthread_t tid; /* the thread identifier */
    pthread_attr_t attr; /* set of attributes for the thread */
    if (argc != 2) {
        fprintf(stderr,"usage: a.out <integer value>\n");
        exit();
    }
    if (atoi(argv[1]) < 0) {
        fprintf(stderr,"Argument %d must be non-negative\n",atoi(argv[1]));
        exit();
    }
    pthread_attr_init(&attr); /* get the default attributes */
    pthread_create(&tid,&attr,runner,argv[1]); /* create the thread */
    pthread_join(tid,NULL); /* now wait for the thread to exit */
    printf("sum = %d\n",sum);
}
Pthreads example (ctd.)

```c
/**
 * The thread will begin control in this function
 */
void *runner(void *param)
{
    int upper = atoi(param);
    int i;
    sum = 0;
    if (upper > 0) {
        for (i = 1; i <= upper; i++)
            sum += i;
    }
    pthread_exit(0);
}
```
Solaris 2 Threads

- User and Kernel level threads, Light weight processes (LWP)
- Process : one or more LWPs
- Each LWP has kernel thread
- One LWP is needed for each user thread that may block
- If kernel thread blocks, LWP, and user level thread also block
- If all LWPs in process block, but there are user level threads which could run, kernel creates new LWP
- Kernel “ages” LWPs and deletes unused ones after +5 min
- Kernel threads may be *bound* to particular CPU
Solaris 2 Threads

- Task 1
- Task 2
- Task 3

- Kernel thread
- User-level thread
- Lightweight process

- CPU
Solaris Process

The diagram shows a Solaris process with the following components:

- **Process id**
- **Memory map**
- **Priority**
- **List of open files**

The process consists of several LWPs (Light Weight Processes) connected in a sequence, as follows:

\[ \text{LWP}_1 \rightarrow \text{LWP}_2 \rightarrow \text{LWP}_3 \rightarrow \ldots \]

The diagram illustrates the structure of a Solaris process, highlighting its key components and the interconnected LWPs that compose it.
Windows 2000 Threads

- Implements the one-to-one mapping.
- Each thread contains
  - a thread id
  - register set
  - separate user and kernel stacks
  - private data storage area
Linux refers to them as *tasks* rather than *threads*.

Thread creation is done through clone() system call.

Clone() allows a child task to share the address space of the parent task (process).

The amount of parent process shared is determined by a set of flags passed as parameter in clone() call:
- None set, no sharing clone() is fork()
- All set, everything shared
Java Threads

- Java threads may be created by:
  - Extending Thread class
  - Implementing the Runnable interface

- Java threads are managed by the JVM.
Java Thread States

- New
- Runnable
- Blocked
- Dead

Transition:
- New -> Runnable
- Runnable -> Blocked
- Blocked -> Runnable
- Runnable -> Dead

Methods:
- start()
- sleep()
- suspend()
- I/O
- resume()
- stop()
Java Thread Example

```java
public class Summation extends Thread {
    public Summation(int n) {
        upper = n;
    }
    public void run() {
        int sum = 0;
        if (upper > 0) {
            for (int i = 1; i <= upper; i++)
                sum += i;
        }
        System.out.println("The summation of " + upper + " is " + sum);
    }
    private int upper;
}
```
Java Thread Example (ctd.)

```java
public class ThreadTester {
    public static void main(String[] args) {
        if (args.length > 0) {
            if (Integer.parseInt(args[0]) < 0)
                System.err.println(args[0] + " must be non-negative.");
            else {
                Summation thrd = new Summation(Integer.parseInt(args[0]));
                thrd.start();
            }
        } else {
            System.err.println("Usage: Summation <integer value>");
        }
    }
}
```