


Contents

1. Introduction
2. System Structures
3. Process Concept
-  4. Multithreaded Programming
5. Process Scheduling
6. Synchronization
7. Deadlocks
8. Memory-Management Strategies
9. Virtual-Memory Management
10. File System
11. Mass-Storage Structures
12. I/O Systems
13. Protection, Security, Distributed Systems 1

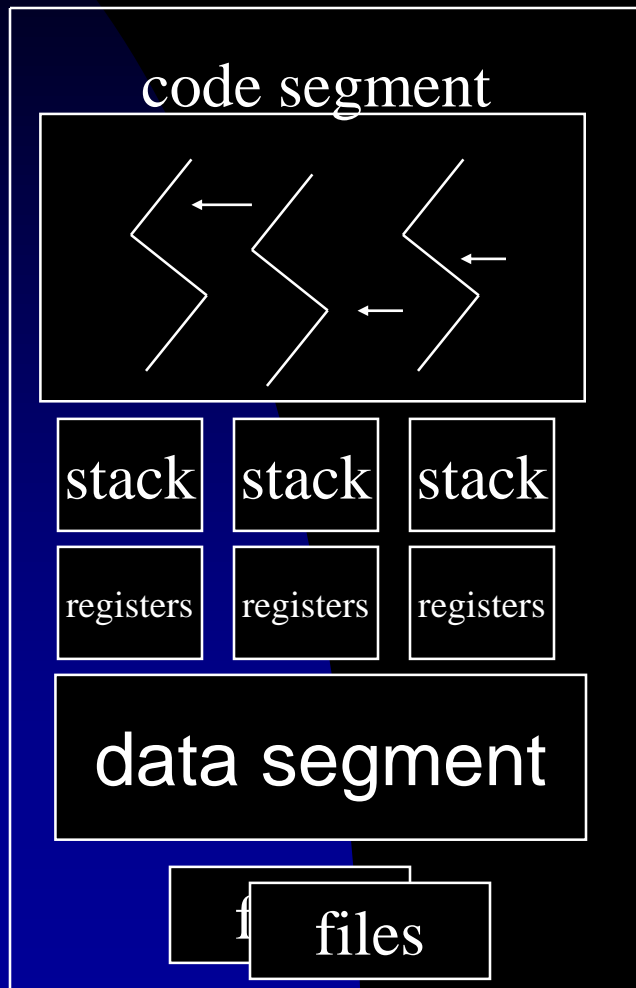
Chapter 4

Multithreaded Programming

Threads

- Objectives:
 - Concepts and issues associated with multithreaded computer systems.
- Thread – Lightweight process(LWP)
 - a basic unit of CPU utilization
 - A thread ID, program counter, a register set, and a stack space
 - Process – heavyweight process
 - A single thread of control

Threads



- Motivation
 - A web browser
 - Data retrieval
 - Text/image displaying
 - A word processor
 - Displaying
 - Keystroke reading
 - Spelling and grammar checking
 - A web server
 - Clients' services
 - Request listening

Threads

- Benefits
 - Responsiveness
 - Resource Sharing
 - Economy
 - Creation and context switching
 - 30 times slower in process creation in Solaris 2
 - 5 times slower in process context switching in Solaris 2
 - Scalability/Utilization of Multiprocessor Architectures

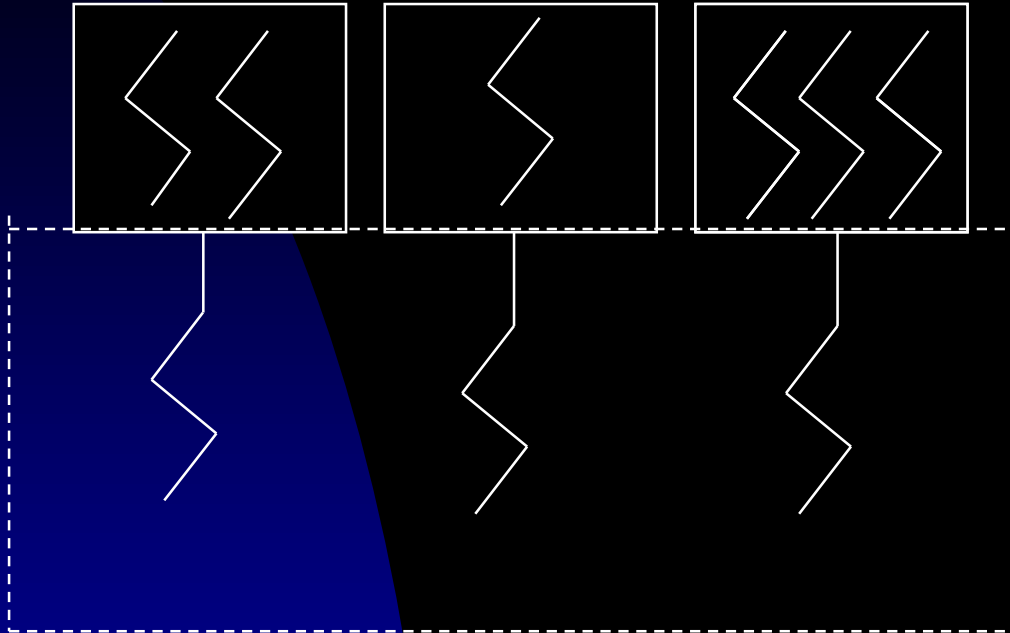
Multicore Programming

- Motivation: The popularity of multiple computing cores per system
 - Cores within a CPU chip or across CPU chips
 - Interleaving vs Parallelism
- Multithreaded Programming
 - OS Side: Scheduling and resource allocation to allow parallel executions
 - Programmer Side: Programming with proper multithreading

Multicore Programming

- Challenges in Programming
 - Identifying Tasks – Dividing Activities
 - Balance
 - Data Splitting
 - Data Dependency
 - Testing and Debugging
- Parallelism Types
 - Data Parallelism – Distribute data over cores to execute the same operation
 - Task Parallelism – Distribute work over cores over the same or different data

User-Level Threads



- Advantages

- Context switching among them is extremely fast.

- Disadvantages

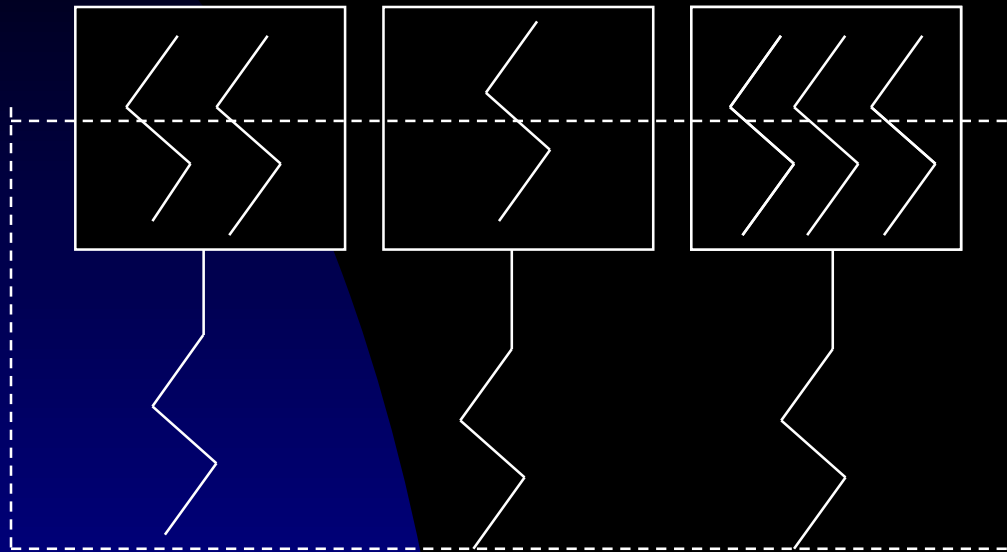
- Blocking of a thread in executing a system call can block the entire process.

- User-level threads are implemented by a thread library at the user level.

- Examples:

- POSIX Pthreads, Mach C-threads, Solaris 2 UI-threads

Kernel-Level Threads



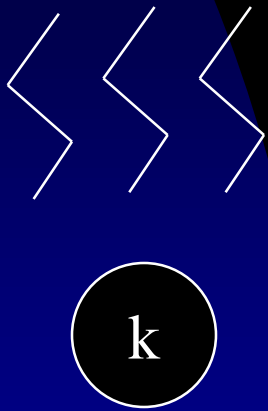
- Advantage
 - Blocking of a thread will not block its entire task.
- Disadvantage
 - Context switching cost is a little bit higher because the kernel must do the switching.

- Kernel-level threads are provided a set of system calls similar to those of processes

- Examples

- Windows XP, Solaris 2, True64UNIX

Multithreading Models



- Many-to-One Model
 - Many user-level threads to one kernel thread
 - Advantage:
 - Efficiency
 - Disadvantage:
 - One blocking system call blocks all.
 - No parallelism for multiple processors
 - Example: Green threads for Solaris 2

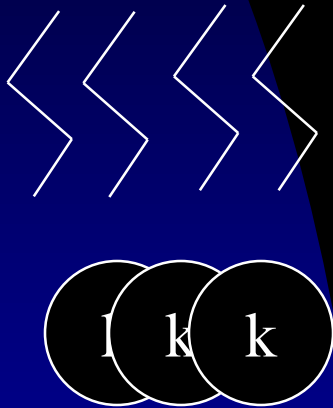
Multithreading Models

- One-to-One Model
 - One user-level thread to one kernel thread
 - Advantage: One system call blocks one thread.
 - Disadvantage: Overheads in creating a kernel thread.
 - Example: Windows NT, Windows 2000, XP, Linux, OS/2, Solaris 9



k

Multithreading Models



- Many-to-Many Model
 - Many user-level threads to many kernel threads
 - Advantage:
 - A combination of parallelism and efficiency
 - Example: Solaris 2 & 9, IRIX, HP-UX, Tru64 UNIX

Thread Libraries

- Goal: Provide an API for creating and managing threads!
 - Asynchronous or Synchronous Threading
- Two Approaches:
 - User Thread Library
 - Kernel-Level Thread Library
- Well-Known Examples
 - POSIX Pthreads – User or Kernel Level
 - Windows threads – Kernel Level
 - Java threads – Level Depending on the Thread Library on the Host System

Pthreads

- Pthreads (IEEE 1003.1c)
 - POSIX API Specification for Thread Creation and Synchronization
 - UNIX-Based Systems, Such As Solaris 2.
- User-Level Library (??)
- Header File: <pthread.h>
- pthread_attr_init(), pthread_create(), pthread_exit(), pthread_join(), etc.

Pthreads

```
#include <pthread.h>
main(int argc, char *argv[]) {
    ...
    pthread_attr_init(&attr);
    pthread_create(&tid, &attr, runner, argv[1]);
    pthread_join(tid, NULL);
    ... }

void *runner(void *param) {
    int i, upper = atoi(param);
    sum = 0;
    if (upper > 0)
        for(i=1;i<=upper;i++)
            sum+=i;
    pthread_exit(0);
}
```

Windows Threads

■ Kernel-Level Threads

```
DWORD Sum; /* shared by threads */
DWORD WINAPI Summation(LPVOID Param) {
    DWORD Upper = *(DWORD *) Param;
    for (DWORD i = 0; i <= Upper; i++)
        Sum += i;
    return 0; }

Int main(int argc, char *argv[]) {
    ...
    Param = atoi(argv[1]);
    ...
    ThreadHandle = CreateThread(
        NULL, // default security attributes
        0, // default stack size
        Summation, // thread function
        &Param, // parameter
        0, // default creation flags
        &ThreadID);
    ...
    WaitForSingleObject(ThreadHandle, INFINITE);
    ... }
```


Java

- Thread Support at the Language Level
 - Mapping of Java Threads to Kernel Threads on the Underlying OS?
 - Windows 2000: 1:1 Model
- Thread Creation
 - Create a new class derived from the Thread class or define a class that implements the Runnable interface
 - Call its start method
 - Allocate memory and initialize a new thread in the JVM
 - start() calls the run method, making the thread eligible to be run by the JVM.

Java

class Summation implements Runnable

```
{ ... public Summation(int upper, Sum sumValue) {  
    this.upper=upper; this.sumValue=sumValue; }  
    public void run() {  
        int sum = 0;  
        for (int i=0; i<=upper; i++) sum+ = i;  
        sumValue.setSum(sum); }  
    ...}
```

public class Driver

```
{ ...  
    Thread thrd = new Thread(new Summation(upper,  
    sumObject));  
    thrd.start();  
    ...  
    thrd.join();  
    ...}
```

Implicit Threading

- Objective: Transfer the creation and management of the threading from the application developers to compilers and run-time libraries.
- Thread Pools
 - Motivations
 - Dynamic creation of threads
 - Limit on the number of active threads
 - Awake and pass a request to a thread in the pool

Implicit Threading

- Benefits of a Thread Pool
 - Faster for service delivery and limit on the # of threads
- Dynamic or static thread pools

QueueUserWorkItem(&PoolFunction, NULL, 0);

- It causes a thread from the thread pool to invoke PoolFunction (under Windows API)

Implicit Threading

- OpenMP

- A set of compiler directives and an API to support parallel programming in shared memory environment.

```
#include <omp.h>
main(int argc, char *argv[]) {
    ...
    #pragma omp parallel for
    for (i = 0; i < N; i++) {
        c[i] = a[i] + b[i] ; }
    ...
}
```

- Threads with divided workload are created automatically based on the number of cores or a set bound.

Implicit Threading

- Grand Central Dispatch (GCD)
 - A Max OS X/iOS combination of extensions to the C, an API, and a run-time library to identify sections of code to run in parallel.

```
^ { printf("I am a block"); }
```

- Dispatch Queues
 - A FIFO policy where blocks of a serial queue are executed serially (per process), and blocks of concurrent queues can run by multiple threads concurrently from the thread pool (3 system-wide queues with low, default, and high priorities).

Threading Issues

- Fork and Exec System Calls
 - Fork: Duplicate all threads or create a duplicate with one thread?
 - Exec: Replace the entire process, including all threads and LWPs.
 - Fork → exec?

Threading Issues

- Signal Handling
 - Signal
 - Synchronous – delivered to the same process that performed the operation causing the signal,
 - e.g., illegal memory access or division by zero
 - Asynchronous
 - e.g., ^C or timer expiration
 - Default or user-defined signal handler
 - Signal masking

Threading Issues

- Delivery of a Signal
 - To the thread to which the signal applies
 - e.g., division-by-zero
 - To every thread in the process
 - e.g., ^C
 - To certain threads in the process
 - Assign a specific thread to receive all threads for the process
 - Solaris 2
- Asynchronous Procedure Calls (APCs)
 - To a particular thread rather than a process

Threading Issues

- Thread Cancellation
 - Target thread
 - Two scenarios:
 - Asynchronous cancellation
 - Deferred cancellation
 - *Cancellation points* in Pthread.
 - Difficulty
 - Resources have been allocated to a cancelled thread.
 - A thread is cancelled while it is updating data.

Threading Issues

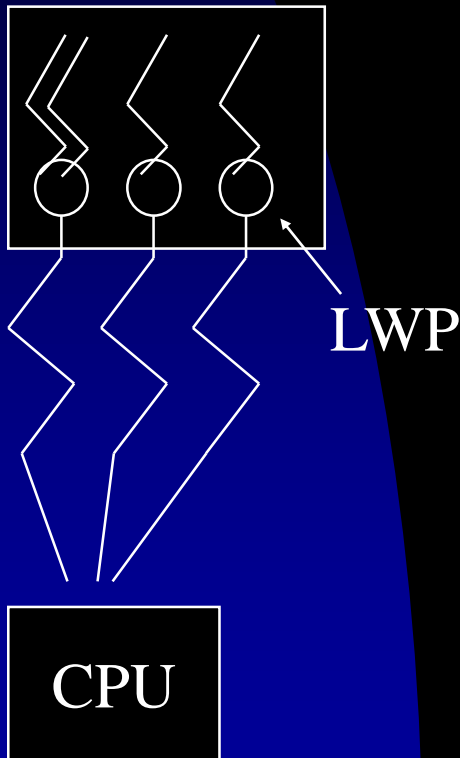
- Three Thread Cancellation Modes

Mode	State	Type
Off	Disabled	
Deferred	Enabled	Deferred
Asynchronous	Enabled	Asynchronous

```
pthread_t tid;  
...  
pthread_create(&tid, 0, worker, NULL);  
...  
pthread_cancel(tid);  
...  
pthread_testcancel();  
...
```

- Thread-local storage – Win32 & Pthreads²⁷

Scheduler Activations



- Definition: A scheme for communication between the user-thread library and the kernel
 - The kernel provides an application with a set of virtual processors, i.e., light weight processes (LWP's)
 - An upcall handler to stop or resume the execution of a thread
 - User threads on a LWP are blocked if any of the user threads is blocked!

Windows

- Windows API
 - One-to-One Model
 - Fiber Library for the M:M Model
- A Thread Contains
 - A Thread ID
 - Context: A Register Set, A User Stack, A Kernel Stack, A Private Storage Space for Run-Time Libraries and DLL's

Windows

Kernel
Space

- Data Structures

- ETHREAD (executive thread block)
 - A ptr to the process, a ptr to KTHREAD, the address of the starting routine
- KTHREAD (kernel thread block)
 - Scheduling and synchronization information, a kernel stack, a ptr to TEB
- TEB (thread environment block)
 - The thread ID, a user stack, an array for thread-specific data.

User
Space

Linux

- Threads introduced in Version 2.2
 - `clone()` versus `fork()`
 - Term task for process & thread
 - Several per-process data structures, e.g., pointers to the same data structures for open files, signal handling, virtual memory, etc.
 - Flag setting in `clone()` invocation.
 - `CLONE_FS`, `CLONE_VM`, `CLONE_SIGHAND`, `CLONE_FILES`
 - Setting → Threads or Processes