Chapter 4: Threads

COURSE INSTRUCTORS:
DR. MUHAMMAD NASEEM, ENGR. FARHEEN QAZI, MS. FALAK SALEEM

COURSE : OPERATING SYSTEMS (SWE-204)
BATCH : 2019 FALL : 2020

DEPARTMENT OF SOFTWARE ENGINEERING
SIR SYED UNIVERSITY OF ENGINEERING & TECHNOLOGY

1

Today's Agenda

- Overview
- Multithreading Models
- Thread Libraries
- Threading Issues
- Operating System Examples
- Windows XP Threads
- Linux Threads

2

Objectives

- To introduce the notion of a thread — a fundamental unit of CPU utilization that forms the basis of multithreaded computer systems
- To discuss the APIs for the Pthreads, Win32, and Java thread libraries
- To examine issues related to multithreaded programming

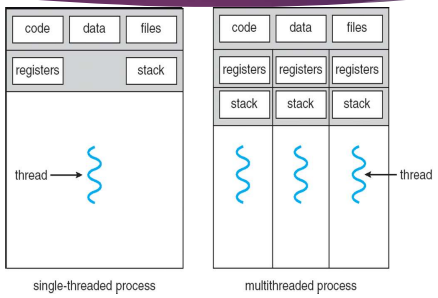
3

Thread Overview

- Threads are mechanisms that permit an application to perform multiple tasks concurrently.
- Thread is a basic unit of CPU utilization
  - Thread ID
  - Program counter
  - Register set
  - Stack
- A single program can contain multiple threads
  - Threads share with other threads belonging to the same process
    - Code, data, open files…….

4

Single and Multithreaded Processes



heavyweight process **lightweight** process

Traditional **(heavyweight)** process has a single thread of control

5

Threads

Threads share….

- Global memory
- Process ID and parent process ID
- Controlling terminal
- Process credentials (user)
- Open file information
- Timers
- ……..

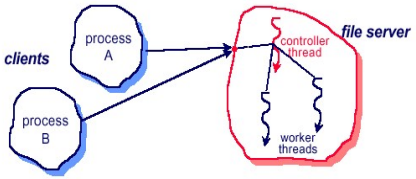
Threads specific Attributes….

- Thread ID
- Thread specific data
- CPU affinity
- Stack (local variables and function call linkage information)
- ……

6

Thread Example # 01

A good example of an application that could make use of threads is a file server on a local area network (LAN).

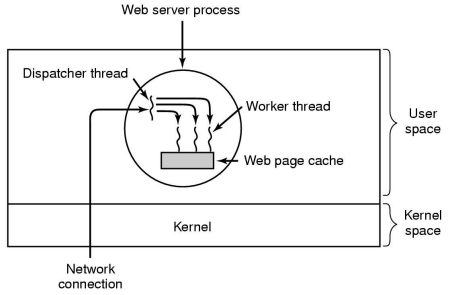


A ‘‘controller’’ thread accepts file service requests and spawns a ‘‘worker’’ thread for each request, therefore may handle many requests concurrently. When a worker thread finishes servicing a request, it is destroyed.

7

Contd…

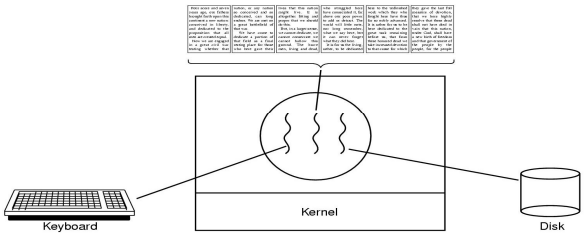
- Thread Usage



A multithreaded Web server

8

Thread Example # 02



Word processor may have a thread for;

1. Displaying graphics.
2. Reading keystrokes from the user.
3. Performing spelling and grammar checking in background.

9

Benefits

- **Responsiveness**
  Interactive application can delegate background functions to a thread and keep running
- **Resource Sharing**
  Several different threads can access the same address space
- **Economy**
  Allocating memory and new processes is costly. Threads are much 'cheaper' to initiate.
- **Scalability**
  Use threads to take advantage of multiprocessor architecture

10

Multithreading Models

- Support provided at either
  - User level -> **user threads**
    Supported above the kernel and managed without kernel support
  - Kernel level -> **kernel threads**
    Supported and managed directly by the operating system

What is the relationship between user and kernel threads?

11

User Threads

- Supported above the kernel and are implemented by thread library at the user level.
- The library supports thread creating, scheduling, and management with no support from the kernel.
- Kernel is unaware of user-level threads, all thread creation and scheduling are done in user space. Hence, they are generally fast.
- Three primary thread libraries:
  - POSIX Pthreads
  - Win32 threads
  - Java threads

12

User Threads Advantages

- **Simple Representation**:
  Each thread is represented simply by a PC, registers, stack and a small control block, all stored in the user process address space.
- **Simple Management:**
  Means that creating a thread, switching between threads and synchronization between threads can all be done without intervention of the kernel.
- **Fast and Efficient:**
  Thread switching is not much more expensive than a procedure call.

13

User Threads Disadvantages

- There is a lack of coordination between threads and operating system kernel. Therefore, process as whole gets one time slice irrespective of whether process has one thread or 1000 threads within. It is up to each thread to relinquish control to other threads.
- Any user-level thread performing a blocking system call will cause the entire process to block, even if other threads are available to run.

14

Kernel Threads

- Supported directly by the OS.
- Kernel performs thread creation, scheduling and management in kernel space.
- Kernel threads are generally slower.
- If a thread performs a blocking system call, the kernel can schedule another thread.
- Examples
  - Windows XP/2000
  - Solaris
  - Linux
  - Tru64 UNIX
  - Mac OS X

15

Kernel Threads Advantages

- Because kernel has full knowledge of all threads, Scheduler may decide to give more time to a process having large number of threads than process having small number of threads.
- Kernel-level threads are especially good for applications that frequently block.
- In multiprocessor system, kernel can schedule threads on different processors.

16

Kernel Threads Disadvantages

- The kernel-level threads are slow and inefficient. For instance, threads operations are hundreds of times slower than that of user-level threads.
- Since kernel must manage and schedule threads as well as processes. It require a full thread control block (TCB) for each thread to maintain information about threads. As a result there is significant overhead and increased in kernel complexity.

17

Multithreading Models

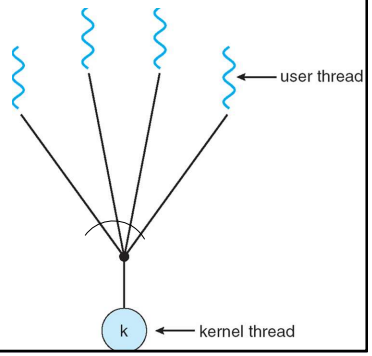
User Thread – to - Kernel Thread

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

18

Many-to-One

Many user-level threads mapped to single kernel thread

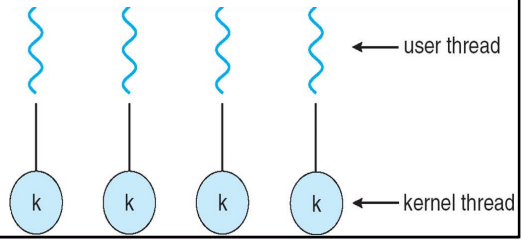


19

One-to-One

Each user-level thread maps to kernel thread

- Examples
  - Windows NT/XP/2000
  - Linux



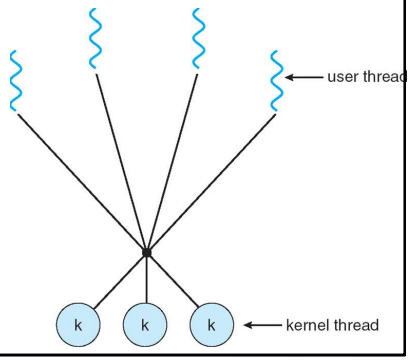
20

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

- Example
  - Windows NT/2000 with the *ThreadFiber* package



21

Thread Libraries

- Thread library provides programmer with API for creating and managing threads
- Two primary ways of implementing
  - Library entirely in user space
  - Kernel-level library supported by the OS
- Three main thread libraries in use today:
  - POSIC Pthreads
  - Win32
  - Java

22

Threading Issues

- Semantics of **fork()** and **exec()** system calls
- Thread cancellation.
- Signal handling
- Thread pools
- Thread safety
- Thread-specific data
- Scheduler activations

23

Semantics of *fork() ,exec(), exit()*

Does **fork()** duplicate only the calling thread or all threads?

- Threads and ***exec()***

  With *exec()*, the calling program is replaced in memory. All threads, except the once calling exec(), vanish immediately. No thread-specific data destructors or cleanup handlers are executed.

- Threads and ***exit()***

  If any thread calls *exit()* or the main thread does a return, ALL threads immediately vanish. No thread-specific data destructors or cleanup handlers are executed.

24

Semantics of *fork() ,exec(), exit()*

- Threads and ***fork()***

  When a multithread process calls *fork()*, only the calling thread is replicated. All other threads vanish in the child. No thread-specific data destructors or cleanup handlers are executed.

  Problems:

  - The global data may be inconsistent:
    - Was another thread in the process of updating it?
    - Data or critical code may be locked by another thread. That lock is copied into child process, too.
  - Memory leaks
  - Thread (other) specific data not available

Recommendation: In multithreaded application, only use *fork()* after *exec()*

25

Thread cancellation

- A task of terminating a (target) thread before it has completed.
- Example: If multiple threads are concurrently searching through a database and one thread returns
- the expected result, the remaining threads might be cancelled.
- The thread to be cancelled is called the target thread.
- Example: When a user press the *stop* button on a web browser, the thread loading a web page should be cancelled.
- **Asynchronous Cancellation**
  - One thread immediately terminates the target thread.
- **Deferred Cancellation**
  - The target thread can periodically check if it should terminate.
- *Pthreads* API provides deferred cancellation

26

Signal Handling

- A signal is used in UNIX systems to notify a process that a particular event has occurred.
- **Synchronous Signal Delivery.**

  Synchronous signals are delivered to the same process that performed the operation causing the signal.
- **Example:** An illegal memory access or division by zero
- **Asynchronous Signal Delivery**

  When a signal is generated by an event external to a running process, the signal is asynchronously delivered typically to another process.
- **Example:** Terminating a process with <control> <C> or having a timer expire.

27

Contd….

- **Three Steps of Signal Processing.**
  1. A signal is generated by the occurrence of an event.
  2. A generated signal is delivered to a process.
  3. Once delivered, the signal must be handled.
- **Two Ways of Signal Handling.**
  1. A Default Signal Handler (run by kernel).
  2. A User-defined Signal Handler
- Overwrites the default handler.

28

Thread Pools

- 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

29

Thread Safety

A function is *thread-safe* if it can be safely invoked by multiple threads at the same time.

Example of a not safe function:

```
static int glob = 0;
static void Incr (int  loops)
{
   int loc, j;
   for (j = 0; j<loops; j++ {
         loc = glob;
         loc++;
         glob = loc;
   }
}
```

Employs global or static values that are shared by all threads

30

Thread Safety

Ways to render a function thread safe

- Serialize the function
  - Lock the code to keep other threads out
  - Only one thread can be in the sensitive code at a time
- Lock only the critical sections of code
  - Only let one thread update the variable at a time.
- Use only thread safe system functions
- Make function reentrant
  - Avoid the use of global and static variables
  - Any information required to be safe is stored in buffers allocated by the call

31

Thread Specific Data

- Makes existing functions thread-safe .
  - May be slightly less efficient than being reentrant
- Allows each thread to have its own copy of data
  - Provides per-thread storage for a function
- Useful when you do not have control over the thread creation process (i.e., when using a thread pool)

32

Scheduler Activations

- Both M:M and Two-level models require communication to maintain the appropriate number of kernel threads allocated to the application
- Scheduler activations provide upcalls - a communication mechanism from the kernel to the thread library
- This communication allows an application to maintain the correct number kernel threads

33

Threads vs. Processes

- Advantages of multithreading
  - Sharing between threads is easy
  - Faster creation
- Disadvantages of multithreading
  - Ensure threads-safety
  - Bug in one thread can bleed to other threads, since they share the same address space
  - Threads must compete for memory
- Considerations
  - Dealing with signals in threads is tricky
  - All threads must run the same program
  - Sharing of files, users, etc

34

Operating System Examples

- Windows XP Threads
- Linux Thread

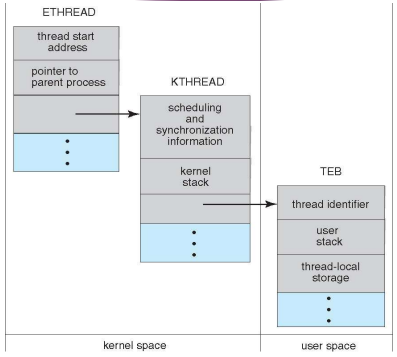
35

Windows XP Threads

- Implements the one-to-one mapping, kernel-level
- Each thread contains
  - A thread id
  - Register set
  - Separate user and kernel stacks
  - Private data storage area
- The register set, stacks, and private storage area are known as the context of the threads
- The primary data structures of a thread include:
  - ETHREAD (executive thread block)
  - KTHREAD (kernel thread block)
  - TEB (thread environment block)

36

Windows XP Threads



37

Linux Threads

- 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)

38

Linux Threads

flag	meaning
CLONE_FS	File-system information is shared.
CLONE_VM	The same memory space is shared.
CLONE_SIGHAND	Signal handlers are shared.
CLONE_FILES	The set of open files is shared.

39

End of Chapter 4

Thank you

40