OPERATING SYSTEMS

Threads

OPERATING SYSTEM
Threads

What Is In This Chapter?

- Overview
- Multithreading Models
- Threading Issues
- Pthreads
- Windows XP Threads
- Linux Threads
- Java Threads

THREADS

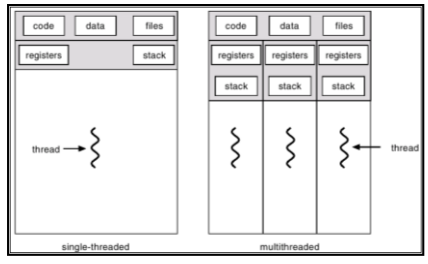
A thread is a basic unit of CPU utilization
It contains:

- **thread ID**
- **Program counter**
- **register set**
- **stack**

-Traditional process has a single thread
- A process with multiple threads can perform more than one task at a time.

THREADS

Single and Multithreaded Processes



THREADS

Eg. Web server with many clients – single process with many threads

A word processor with separate thread each to display graphics, accept keystrokes, spell checking in the background.

THREADS

Benefits

- Responsiveness – multithreaded web browser can load an image in one thread while surfing in another thread
- Resource Sharing – share memory and other resources belonging to a particular process. A program can have different threads sharing the same address space
- Economy – allocating resources for process creation is costly. More economical to create threads (since it shares memory and resources)
- Utilization of MP Architectures – Multi-threading on multi-CPU machines increases concurrency. Threads can run in parallel in multiprocessors

THREADS

User Threads

- Thread management done by user-level threads library
- Examples
  - POSIX *Pthreads*
  - Mach *C-threads*
  - Solaris *threads*

Kernel Threads

- Supported by the Kernel
- Examples
  - Windows 95/98/NT/2000
  - Solaris
  - Tru64 UNIX
  - BeOS
  - Linux

THREADS

Multithreading Models

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

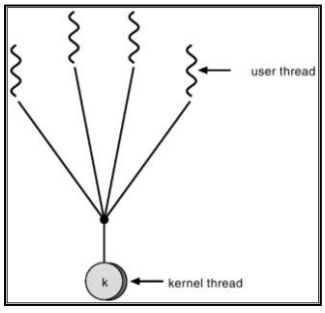
How do user and kernel threads map into each other?

THREADS

Many-to-One

- Many user-level threads mapped to single kernel thread.
- Used on systems that do not support kernel threads.
- Examples:
  Solaris Green Threads
  GNU Portable Threads

- One thread can access the kernel at a time. multiple threads are unable to run in parallel in multiprocessors

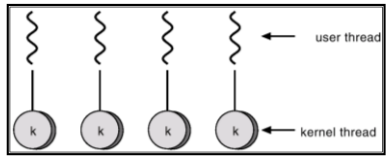


THREADS

One-to-One

- Each user-level thread maps to kernel thread.
- Examples
  - Windows 95/98/NT/2000
  - Linux
- Allows multiple threads to run in parallel in multi processors

- A kernel thread must be created to have a user thread - costly



THREADS

Many-to-many

many user threads can be created for many kernel threads that can run parallel in a multi processor architecture

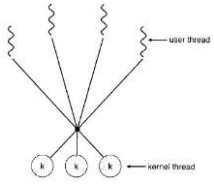


Figure 4.4 Many-to-many model.

THREADS

Thread libraries

- Thread libraries - API for creating and managing threads.

Three main thread libraries

1. **POSIX Pthreads**
2. **Win 32**
3. **Java**

THREADS

P threads

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 (Solaris, Linux, Mac OS X)

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)

THREADS

Basic thread creation

Design a multithreaded program to perform the summation in a separate thread

$$sum = \sum_{i=0}^{N} i$$

THREADS

P threads

- thread do to summation

-parent thread is created for the main () function.
-child thread is created for runner() function

-Parent thread wait until the child thread is completed.

-int sum; is shared by both threads

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

int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* the thread */

int main(int argc, char *argv[])
{
  pthread_t tid; /* the thread identifier */
  pthread_attr_t attr; /* set of thread attributes */

  if (argc != 2) {
    fprintf(stderr,"usage: a.out <integer value>\n");
    return -1;
  }
  if (atoi(argv[1]) < 0) {
    fprintf(stderr,"%d must be >= 0\n",atoi(argv[1]));
    return -1;
  }

  /* get the default attributes */
  pthread_attr_init(&attr);
  /* create the thread */
  pthread_create(&tid,&attr,runner,argv[1]);
  /* wait for the thread to exit */
  pthread_join(tid,NULL);

  printf("sum = %d\n",sum);
}

/* The thread will begin control in this function */
void *runner(void *param)
{
  int i, upper = atoi(param);
  sum = 0;

  for (i = 1; i <= upper; i++)
    sum += i;

  pthread_exit(0);
}
```


THREADS

Windows Threads

Implements the one-to-one mapping

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*

THREADS

Win 32

-DWORD data type declared globally
-summation () function to be executed in a separate thread

- CreateThread is used to create a thread

```
#include <windows.h>
#include <stdio.h>
DWORD Sum; /* data is shared by the thread(s) */
/* the thread runs in this separate function */

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[])
{
  DWORD ThreadId;
  HANDLE ThreadHandle;
  int Param;
  /* perform some basic error checking */
  if (argc != 2) {
    fprintf(stderr,"An integer parameter is required\n");
    return -1;
  }
  Param = atoi(argv[1]);
  if (Param < 0) {
    fprintf(stderr,"An integer >= 0 is required\n");
    return -1;
  }

  // create the thread
  ThreadHandle = CreateThread(
    NULL, // default security attributes
    0, // default stack size
    Summation, // thread function
    &Param, // parameter to thread function
    0, // default creation flags
    &ThreadId); // returns the thread identifier

  if (ThreadHandle != NULL) {
    // now wait for the thread to finish
    WaitForSingleObject(ThreadHandle,INFINITE);

    // close the thread handle
    CloseHandle(ThreadHandle);

    printf("sum = %d\n",Sum);
  }
}
```

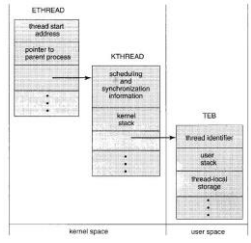
Figure 4.7 Multithreaded C program using the Win32 API.

THREADS

Win 32

-primary data structure of a windows thread

•Ethread – executive thread block
•Kthread – kernel thread block
•TEB – thread environment block (user space data)



THREADS

Java threads

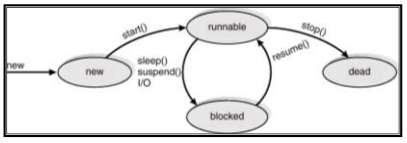
Java Threads

Java threads may be created by:

Extending Thread class

Implementing the Runnable interface

Java threads are managed by the JVM.



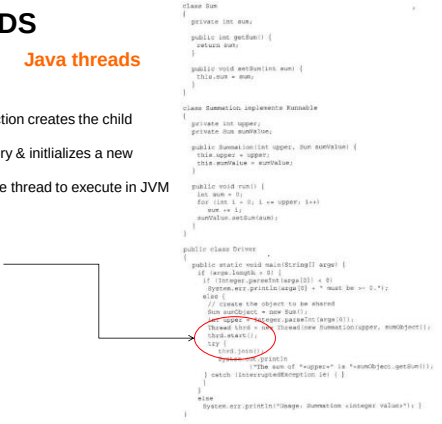
THREADS

Java threads

-thrd.start () function creates the child thread:

-allocates memory & initlializes a new thread in JVM

-calls run() for the thread to execute in JVM



THREADS

Threading Issues

Semantics of fork() and exec() system calls

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

Thread cancellation

- Terminating a thread before it has finished (eg. Database search with multiple threads, one threads finds the result. User presses the stop button while a webpage is being loaded).

- Two general approaches:
  - **Asynchronous cancellation** one thread terminates the target thread (*thread to be cancelled*) immediately
  - **Deferred cancellation** allows the target thread to periodically check if it should be cancelled (cancellation points)

THREADS

Threading Issues

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. The delivered Signal is handled
- Options (in multithreaded programs)
  - 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

THREADS

Threading Issues

Signal handling:

1 Synchronous signal

- **eg. Illegal memory access or division by 0**
- **If a running process performs /0, a signal is generated, it is *delivered to the same process* that performed the operation that caused the signal**

2. Asynchronous signal

- **eg. Pressing ^C to terminate a process**
- **Signal is generated externally to a running process**

THREADS

Threading Issues

Thread pools

- A web server may get exhausted if unlimited number of threads has to be created.

When a process is started, 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

-If there are no threads in the pool, wait until a thread becomes free

THREADS

Threading Issues

Thread specific data

- Threads belonging to a process share data of that process
- 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)

Threads

WRAPUP

We've looked in detail at how threads work. Specifically we've looked at:

- Multithreading Models
- Threading Issues
- Pthreads
- Windows XP Threads
- Linux Threads
- Java Threads