CSE 4/521 Introduction to Operating Systems

Lecture 5 – Threads

(Overview, Multicore Programming, Multithreading Models, Thread Libraries, Implicit Threading, Operating-System Examples)

Summer 2018

Overview

Objective:

1. To introduce the notion of a thread
2. Examine issues related to multicore programming, and multithreaded models
3. Discuss implicit threading strategies
4. To cover operating system support for threads on Linux and Windows

- Overview
- Multicore Programming
- Multithreaded Models
- Thread Libraries
- Implicit Threading
- Operating-System Examples

Recap

- Process Concept
  - Structure of process, state of process, PCB
- Process Scheduling
  - Scheduling queues, long/mid/short-term schedulers, context switch
- Operations on Processes
  - Process creation (`fork()`), process termination
- Interprocess Communication
  - Shared memory, message passing, producer-consumer problem
- Examples of IPC Systems
  - POSIX, Mach, and Hybrid

Questions

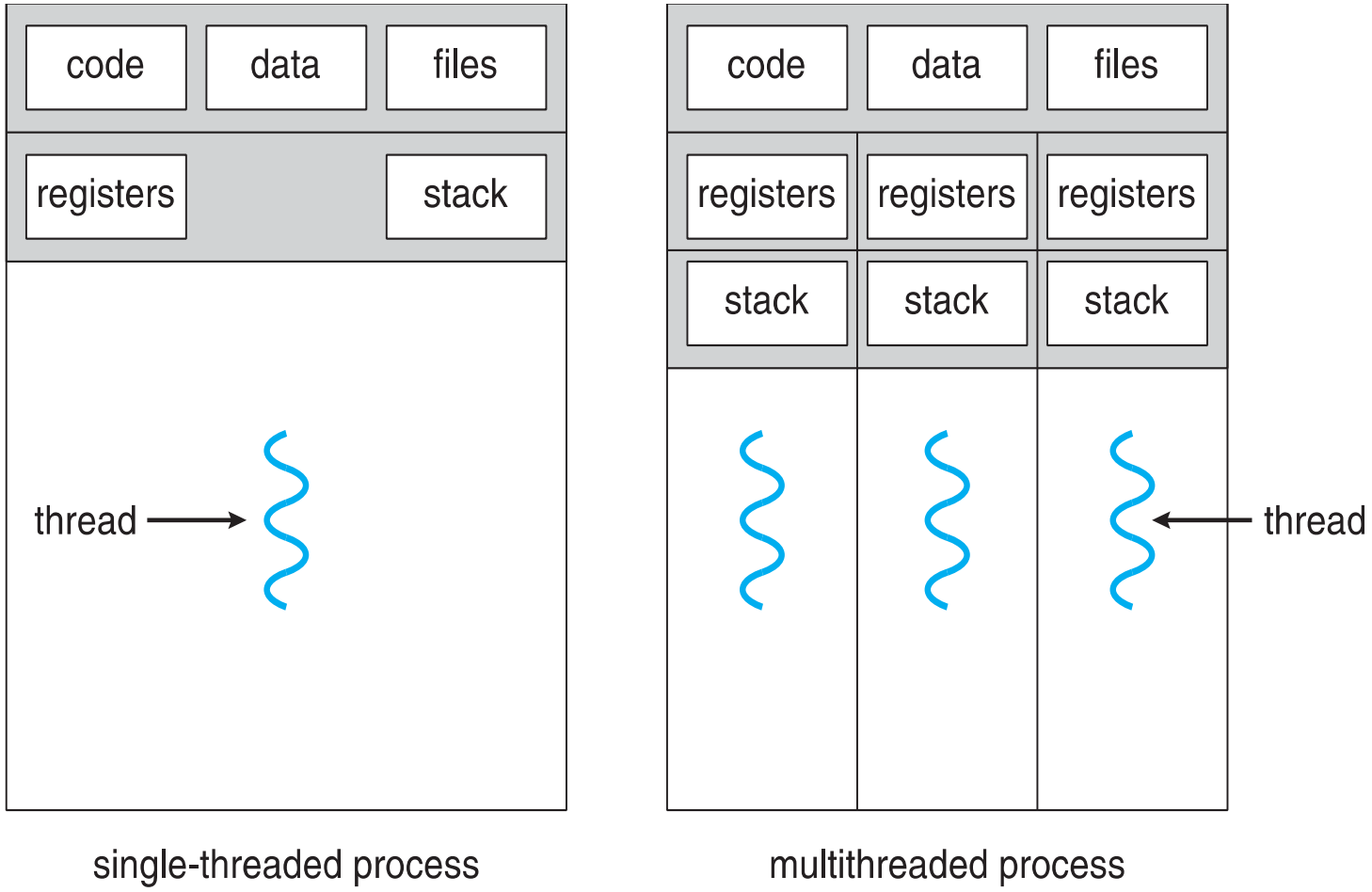
1. What is the return address of `fork()` for child process and parent process ? (Easy)
2. Why should a parent process wait() for a child process? (Medium)
3. What is the output at LINE A? (Hard)

```
#include <unistd.h>
int value = 5;
int main() {
          pid_t pid;
          pid = fork();
          if (pid == 0) {
     value += 15;
     return 0;
          }
          else if (pid > 0) {
            wait(NULL);
            printf("PARENT: value = %d", value);        /*LINE A*/
            return 0;
          }
}
```

Table of Content

Table of Content

Overview


code
data
files
registers
stack
thread
single-threaded process
code
data
files
registers
registers
registers
stack
stack
stack
thread
multithreaded process

Overview

- Process creation is heavy-weight while thread creation is light-weight
- Kernels are generally multithreaded
- *Example for threads:* Multiple tasks within the application can be implemented by separate threads. In web-browser:
  - Update display
  - Fetch data
  - Spell checking
  - Answer a network request

Overview – Benefits of Threads

- Responsiveness – may allow continued execution if part of process is blocked, especially important for user interfaces
- Resource Sharing – threads share resources of process, easier than shared memory or message passing
- Economy – cheaper than process creation, thread switching lower overhead than context switching
- Scalability – process can take advantage of multiprocessor architectures

Table of Content

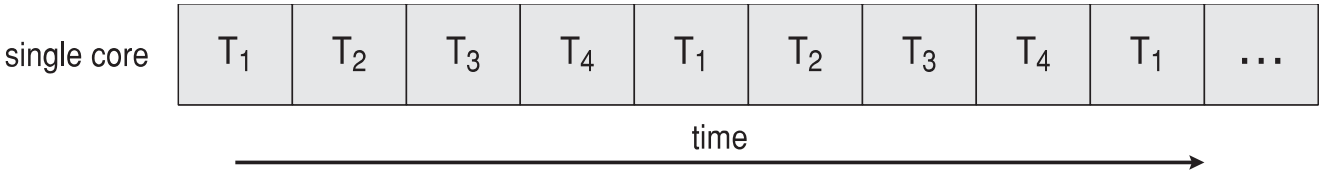
- Overview
- Multicore Programming
- Multithreaded Models
- Thread Libraries
- Implicit Threading
- Operating-System Examples

Multicore Programming

- Multicore or multiprocessor systems putting pressure on programmers, challenges include:
  - Dividing activities
  - Balance
  - Testing and debugging
- Parallelism implies a system can perform more than one task simultaneously
- Concurrency supports more than one task making progress
  - Single processor / core, scheduler providing concurrency

Multicore Programming

- Concurrent execution on single-core system:



- Parallelism on a multi-core system:

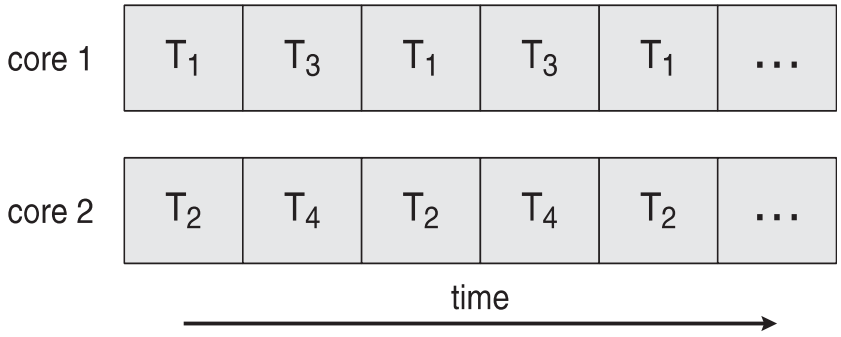


Table of Content

Multithreaded Models – User threads and Kernel threads

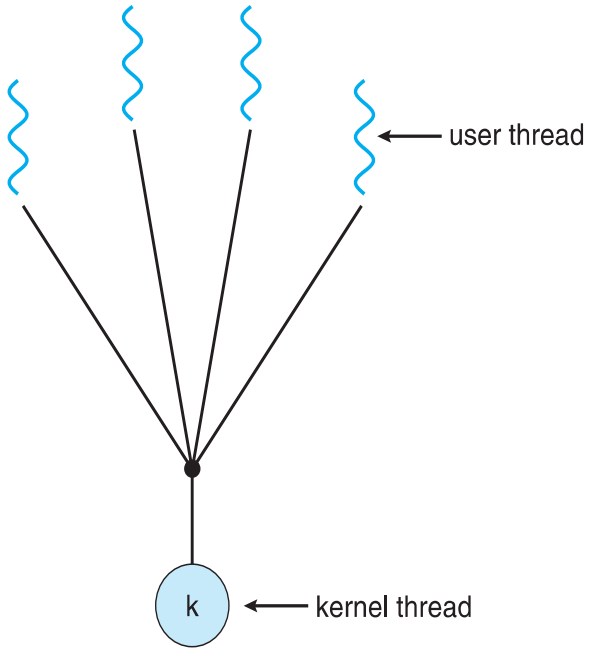
- User threads - management done by user-level threads library
- Three primary thread libraries:
  - POSIX `Pthreads`
  - Windows threads
  - Java threads
- Kernel threads - Supported by the Kernel
- Examples – virtually all general purpose operating systems:
  - Windows
  - Solaris
  - Linux
  - Tru64 UNIX
  - Mac OS X

Multithreaded Models

- 3 types:
  - Many-to-One
  - One-to-One
  - Many-to-Many

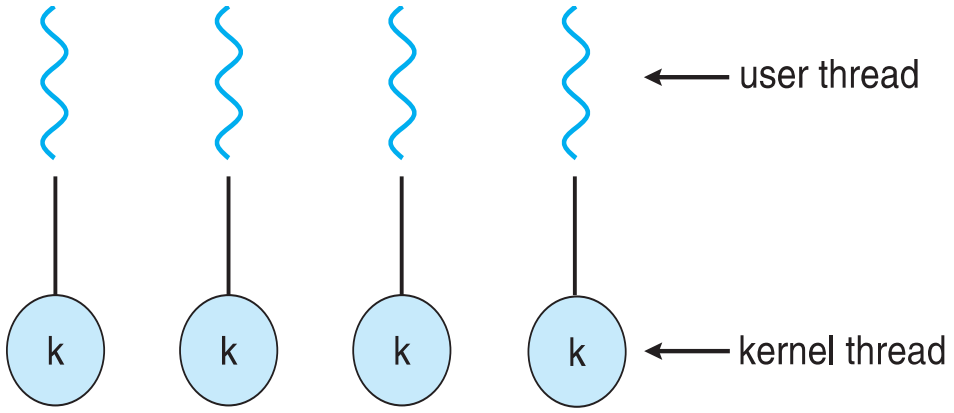
Multithreaded Models : Many-to-One

- Many user-level threads mapped to single kernel thread
- One thread blocking causes all to block
- Multiple threads may not run in parallel on multicore system because only one may be in kernel at a time
- Few systems currently use this model
- Examples:
  - Solaris Green Threads
  - GNU Portable Threads



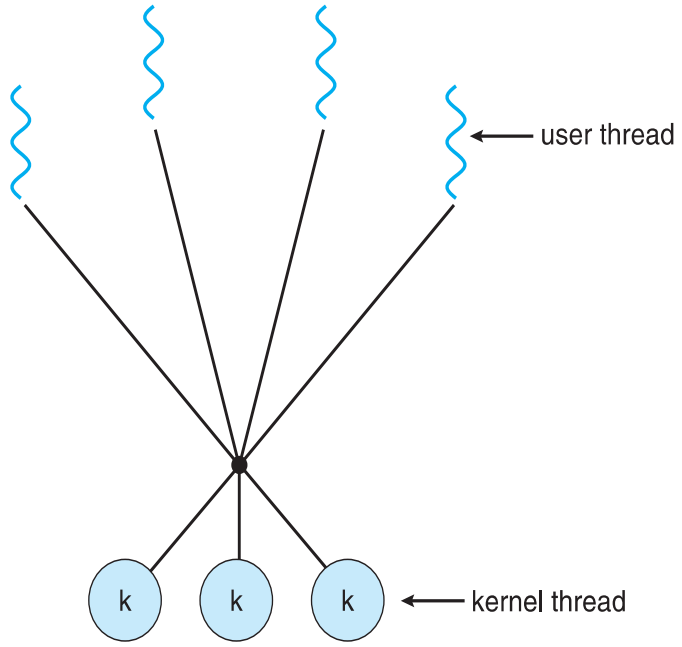
Multithreaded Models : One-to-One

- Each user-level thread maps to unique kernel thread
- Creating a user-level thread creates a kernel thread
- More concurrency than many-to-one
- Number of threads per process sometimes restricted due to overhead
- Examples
  - Windows
  - Linux
  - Solaris 9 and later



Multithreaded Models : Many-to-Many

- Allows many user-level threads to be mapped to many kernel threads
- Allows the operating system to create a sufficient number of kernel threads
- Examples
  - Solaris prior to version 9



Multithreaded Models: Two-level

- Similar to M:M, except that it allows a user thread to be bound to kernel thread
- Examples
  - IRIX
  - HP-UX
  - Tru64 UNIX
  - Solaris 8 and earlier

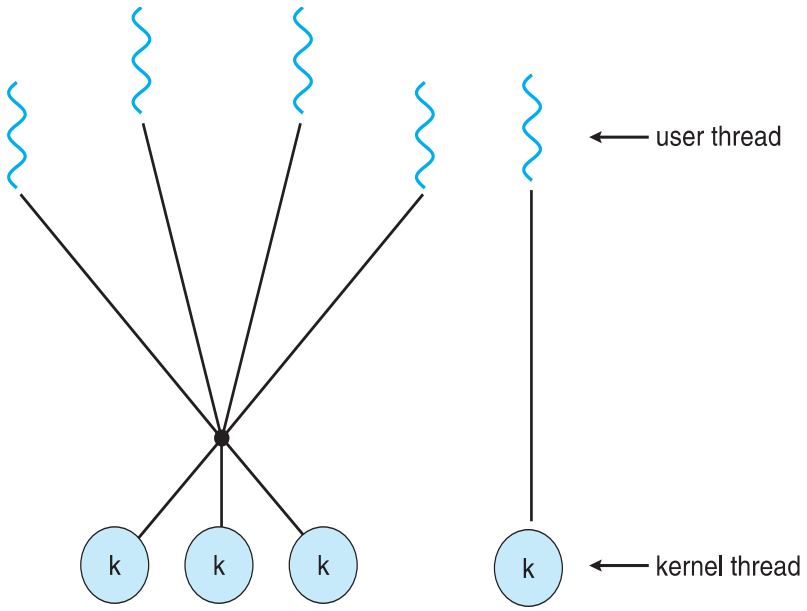


Table of Content

- Overview
- Multicore Programming
- Multithreaded Models
- Thread Libraries
- Implicit Threading
- Operating-System Examples

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
- Examples:
  - Pthreads
  - Java threads

Thread Libraries : Pthreads

Pthreads in UNIX operating systems (Solaris, Linux, Mac OS X)

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

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

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);
}
```

Thread Libraries : Pthreads

Pthread code for joining 10 threads

```
#define NUM_THREADS 10

/* an array of threads to be joined upon */
pthread_t workers[NUM_THREADS];

for (int i = 0; i < NUM_THREADS; i++)
   pthread_join(workers[i], NULL);
```

Thread Libraries : Java Threads

- Java threads are managed by the JVM
- Typically implemented using the threads model provided by underlying OS
- Java threads may be created by:

```
public interface Runnable
{
    public abstract void run();
}
```

  - Extending Thread class
  - Implementing the Runnable interface

Table of Content

Implicit Threading

- Creation and management of threads done by compilers and run-time libraries rather than programmers
- 2 methods explored
  - Thread Pools
  - OpenMP
- Other methods include Microsoft Threading Building Blocks (TBB), `java.util.concurrent` package

Implicit Threading : Thread Pools

- Create a number of threads in a pool where they await work
- Advantages:
  - 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
- Windows API supports thread pools:

```
DWORD WINAPI PoolFunction(AVOID Param) {
    /*
    * this function runs as a separate thread.
    */
}
```

Implicit Threading : OpenMP

- Set of compiler directives and an API for C, C++, FORTRAN
- Provides support for parallel programming in shared-memory environments
- Identifies parallel regions – blocks of code that can run in parallel

```
#pragma omp parallel
```

Creates as many threads as there are cores

```
#pragma omp parallel for
  for(i=0;i<N;i++) {
     c[i] = a[i] + b[i];
}
```

Above code runs for loop in parallel

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

int main(int argc, char *argv[])
{
   /* sequential code */

   #pragma omp parallel
   {
      printf("I am a parallel region.");
   }

   /* sequential code */

   return 0;
}
```

Table of Content

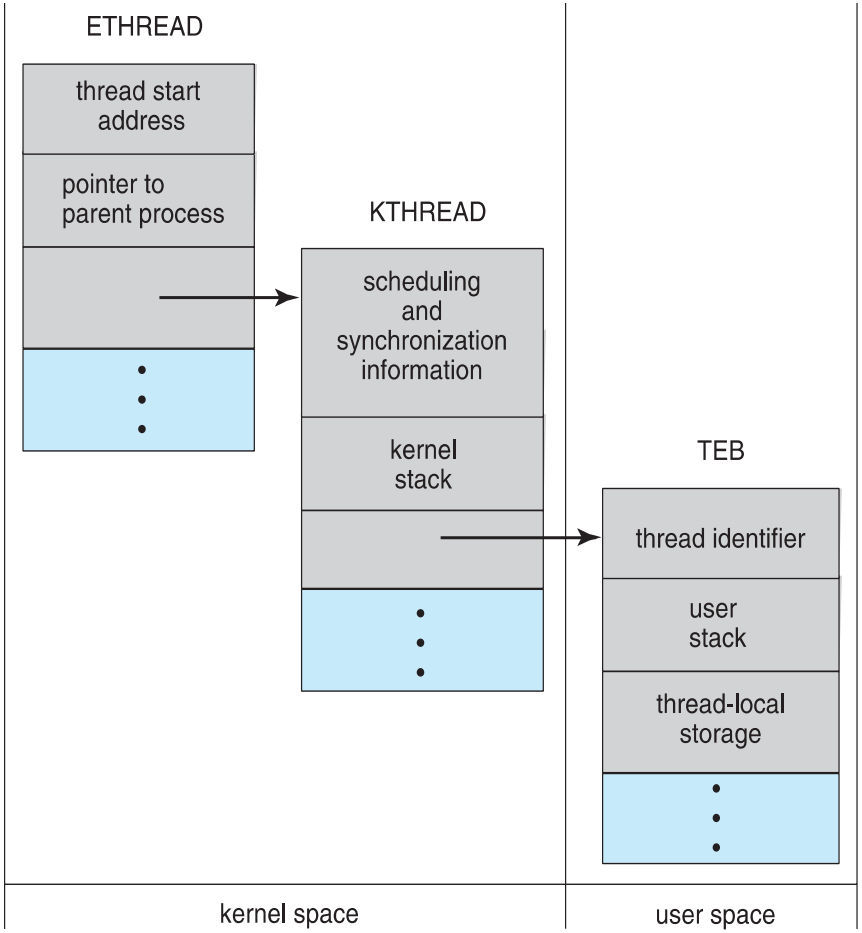
- Overview
- Multicore Programming
- Multithreaded Models
- Thread Libraries
- Implicit Threading
- Operating-System Examples

Operating-System Examples

- Windows Threads
- Linux Threads

Operating-System Examples : Windows Threads

- Implements the one-to-one, kernel-level mapping.
- The primary data structures of a thread include:
  - ETHREAD (executive thread block) – includes pointer to process to which thread belongs and to KTHREAD, in kernel space
  - KTHREAD (kernel thread block) – scheduling and synchronization info, kernel-mode stack, pointer to TEB, in kernel space
  - TEB (thread environment block) – thread id, user-mode stack, thread-local storage, in user space



Operating-System Examples : Linux Threads

- Thread creation is done through `clone()` system call
- `clone()` allows a child task to share the address space of the parent task (process)
  - Flags control behavior (unlike `fork()` where behavior cannot be controlled)

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.

Credential for slides

Silberschatz, Galvin and Gagne