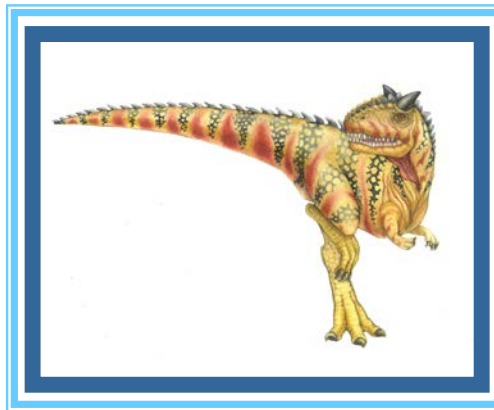


Chapter 4: Threads





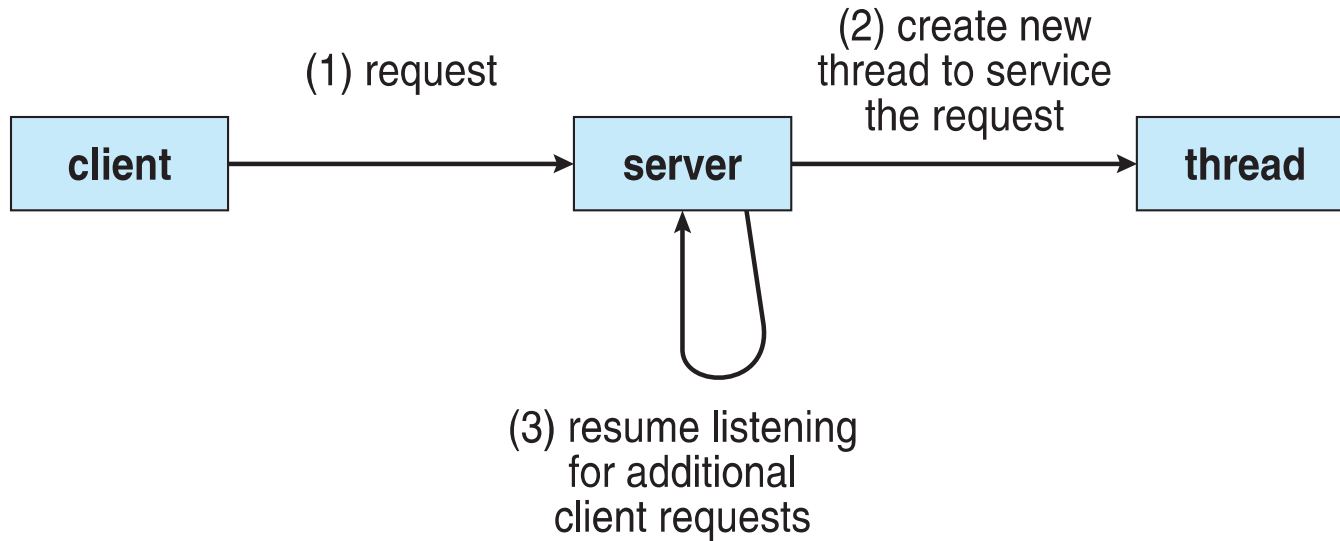
Motivation

- 👁 Most modern applications are multithreaded
- 👁 Threads run within application
- 👁 Multiple tasks with the application can be implemented by separate threads
 - 🍃 Update display
 - 🍃 Fetch data
 - 🍃 Spell checking
 - 🍃 Answer a network request
- 👁 Process creation is heavy-weight while thread creation is light-weight
- 👁 Can simplify code, increase efficiency
- 👁 Kernels are generally multithreaded





Multithreaded Server Architecture





Benefits

- 👁 **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





Multicore Programming

- 👁 **Multicore** or **multiprocessor** systems putting pressure on programmers, challenges include:
 - 🍃 **Dividing activities**
 - 🍃 **Balance**
 - 🍃 **Data splitting**
 - 🍃 **Data dependency**
 - 🍃 **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 (Cont.)

- 👁 Types of parallelism
 - 🍃 **Data parallelism** – distributes subsets of the same data across multiple cores, same operation on each
 - 🍃 **Task parallelism** – distributing threads across cores, each thread performing unique operation
- 👁 As # of threads grows, so does architectural support for threading
 - 🍃 CPUs have cores as well as **hardware threads**
 - 🍃 Consider Oracle SPARC T4 with 8 cores, and 8 hardware threads per core



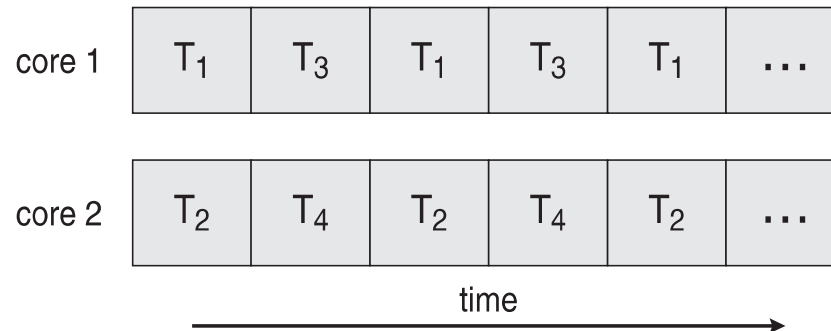


Concurrency vs. Parallelism

👁 Concurrent execution on single-core system:

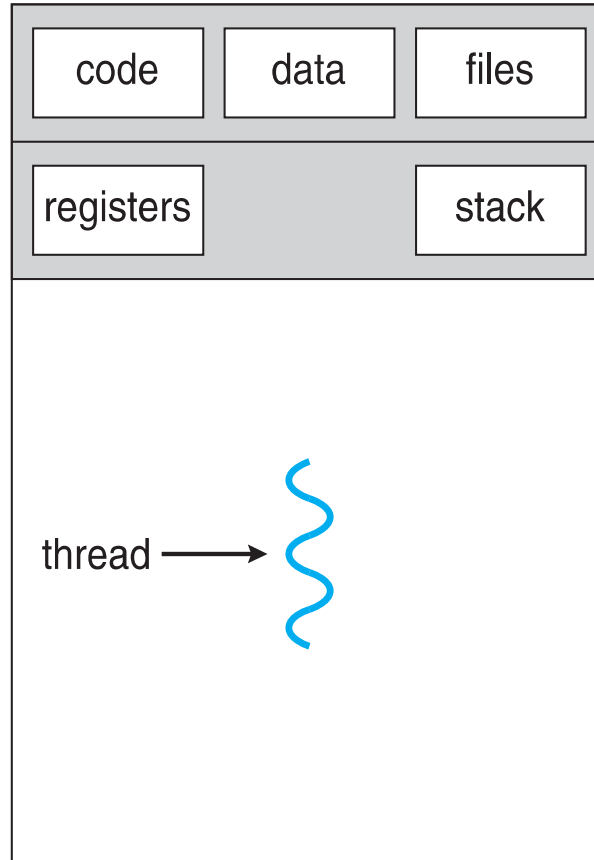


👁 Parallelism on a multi-core system:

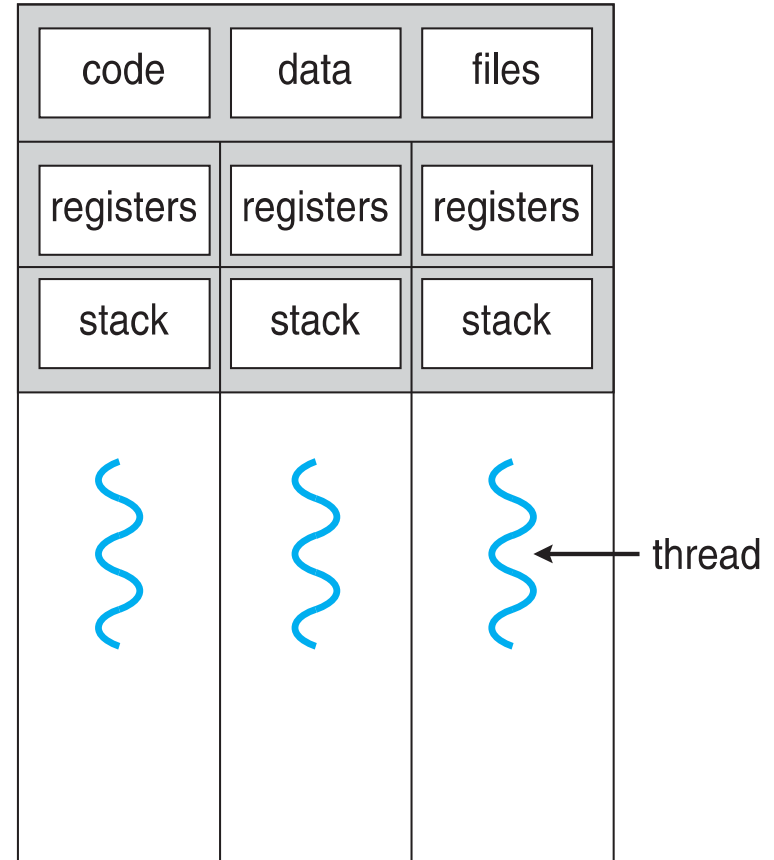




Single and Multithreaded Processes



single-threaded process



multithreaded process





Amdahl's Law

- 👁 Identifies performance gains from adding additional cores to an application that has both serial and parallel components
- 👁 S is serial portion
- 👁 N processing cores

$$speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$$

- 👁 That is, if application is 75% parallel / 25% serial, moving from 1 to 2 cores results in speedup of 1.6 times
- 👁 As N approaches infinity, speedup approaches $1 / S$

Serial portion of an application has disproportionate effect on performance gained by adding additional cores





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, including:
 - 🍃 Windows
 - 🍃 Solaris
 - 🍃 Linux
 - 🍃 Tru64 UNIX
 - 🍃 Mac OS X







Multithreading Models

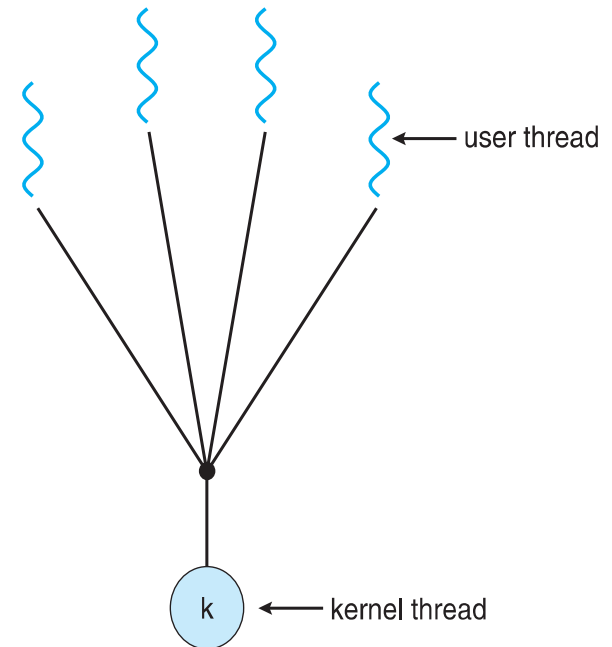
- 👁 Many-to-One
- 👁 One-to-One
- 👁 Many-to-Many





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**



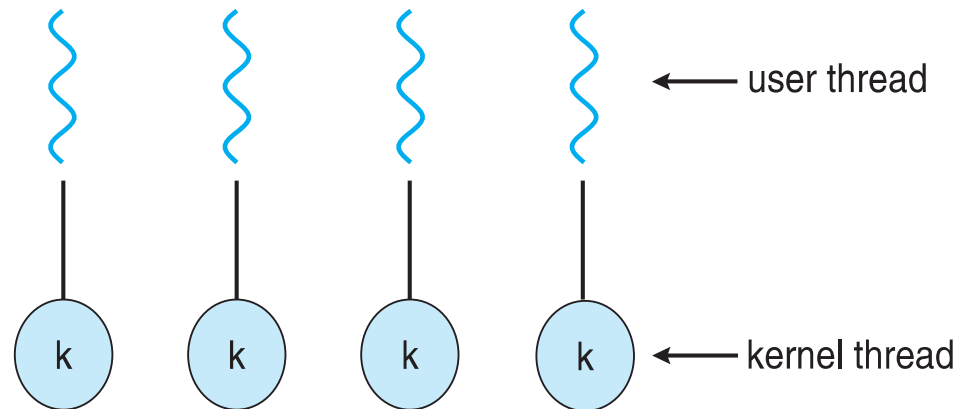


One-to-One

- 👁 Each user-level thread maps to 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

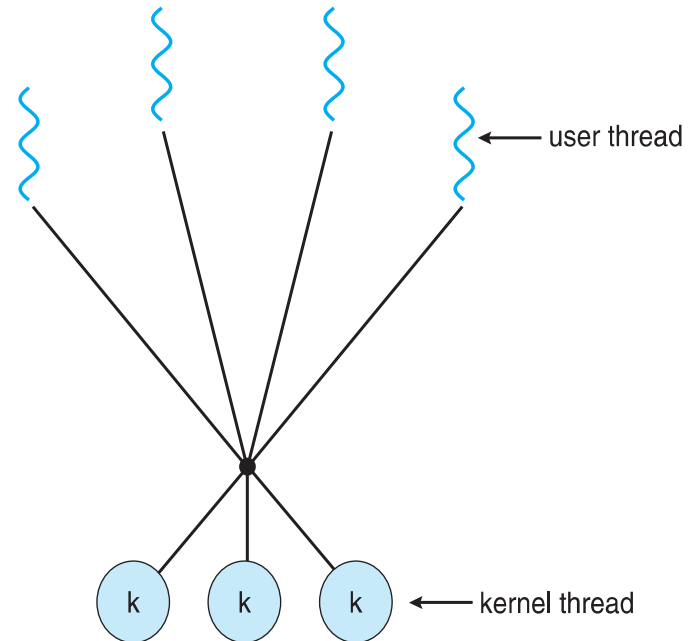
- 🍃 Windows
- 🍃 Linux
- 🍃 Solaris 9 and later





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 prior to version 9
- 👁 Windows with the *ThreadFiber* package





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





Pthreads

- 👁 May be provided either as user-level or kernel-level
- 👁 A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
- 👁 ***Specification***, not ***implementation***
- 👁 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)





Pthreads Example

```
#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;
    }
}
```





Pthreads Example (Cont.)

```
    /* 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);
}
```





Pthreads 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);
```

Figure 4.10 Pthread code for joining ten threads.





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
 - 🍃 Separating task to be performed from mechanics of creating task allows different strategies for running task
 - ▶ i.e. Tasks could be scheduled to run periodically
- 👁 Windows API supports thread pools:

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





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

Create as many threads as there are cores

#pragma omp parallel for

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

Run 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;  
}
```



End of Chapter 4

