

EECS 3221: OPERATING SYSTEM FUNDAMENTALS

Hamzeh Khazaei

Department of Electrical Engineering and
Computer Science

Week 3:
Threads & Concurrency

Assignment Project Exam Help

Feb 1, 2022

Operating System Concepts – 10th Edition

4.1

Silberschatz et al © 2018 modified by Khazaei

1

<https://powcoder.com>

Add WeChat powcoder

Chapter 4: Threads

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

Operating System Concepts – 10th Edition

4.2

Silberschatz et al © 2018 modified by Khazaei

2

Motivation

- Most modern applications are multithreaded
- Threads run within application
- Multiple tasks within 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

Assignment Project Exam Help

Operating System Concepts – 10th Edition

4.3

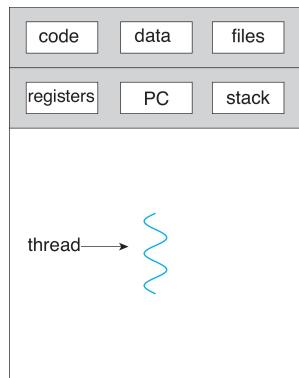
Silberschatz et al © 2018 modified by Khazaei

3

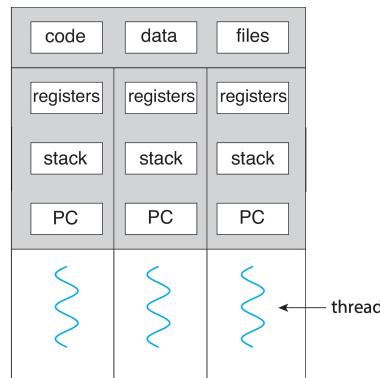
<https://powcoder.com>

Add WeChat powcoder

Single and Multithreaded Processes



single-threaded process



multithreaded process

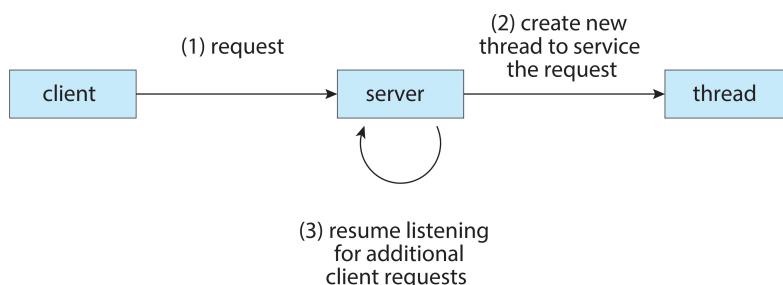
Operating System Concepts – 10th Edition

4.4

Silberschatz et al © 2018 modified by Khazaei

4

Multithreaded Server Architecture



Assignment Project Exam Help

Operating System Concepts – 10th Edition

4.5

Silberschatz et al © 2018 modified by Khazaei

5

<https://powcoder.com>

Add WeChat powcoder 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 multicore architectures
 - One process can use at most one core, while multi-threads could use all available cores or CPUs.

Operating System Concepts – 10th Edition

4.6

Silberschatz et al © 2018 modified by Khazaei

6

3

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

Assignment Project Exam Help

Operating System Concepts – 10th Edition

4.7

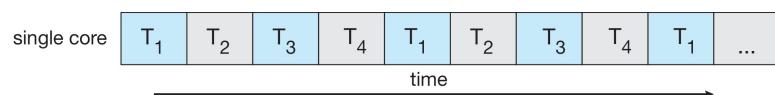
Silberschatz et al © 2018 modified by Khazaei

7

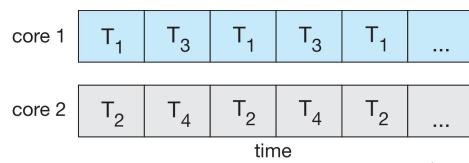
<https://powcoder.com>

Add WeChat powcoder Concurrency vs. Parallelism

- Concurrent execution on single-core system:



- Parallelism on a multi-core system:



Operating System Concepts – 10th Edition

4.8

Silberschatz et al © 2018 modified by Khazaei

8

Multicore Programming

■ Types of parallelism

- **Data parallelism** – distributes distinctive chunk of data across multiple cores, same operation on each
 - ▶ Any example?
- **Task parallelism** – distributing threads across cores, each thread performing unique operation

Assignment Project Exam Help

Operating System Concepts – 10th Edition

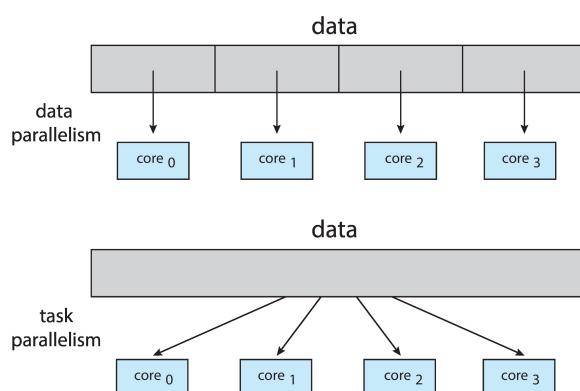
4.9

Silberschatz et al © 2018 modified by Khazaei

9

<https://powcoder.com>

Add WeChat powcoder



Operating System Concepts – 10th Edition

4.10

Silberschatz et al © 2018 modified by Khazaei

10

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



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

$\xrightarrow{\text{cancel } (1-S)}$ $\xrightarrow{\text{cancel } N}$

$$= \frac{1}{S}$$

- 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

- But does the law take into account contemporary multicore systems?

Operating System Concepts – 10th Edition

4.11

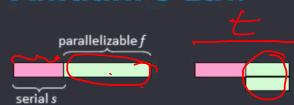
Silberschatz et al © 2018 modified by Khazaei

11

<https://powcoder.com>

Add WeChat powcoder

Amdahl's Law



$$S = \frac{1}{T}$$

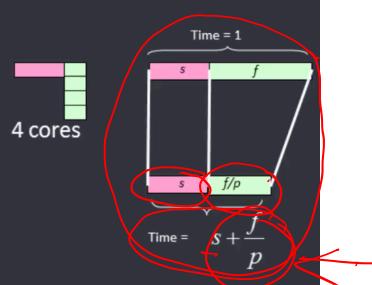
$$S = \frac{1}{\frac{s}{p} + \frac{f}{p}} = \frac{1}{1 - f + \frac{f}{p}}$$

$S = \text{speedup}$

$s = 1 - f = \text{serial_fraction}$

$f = 1 - s = \text{parallel_fraction}$

$p = \# \text{processor_cores}$



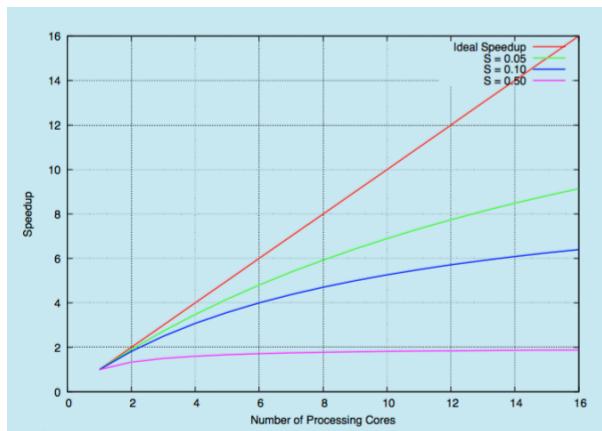
Operating System Concepts – 10th Edition

4.12

Silberschatz et al © 2018 modified by Khazaei

12

Amdahl's Law



Assignment Project Exam Help

Operating System Concepts – 10th Edition

4.13

Silberschatz et al © 2018 modified by Khazaei

13

<https://powcoder.com>

Add WeChat powcoder *User Threads and Kernel Threads* *up to here*

- **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
 - Linux
 - Mac OS X
 - iOS
 - Android

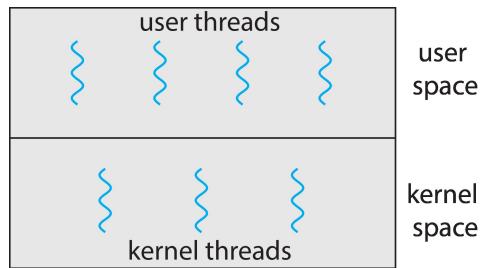
Operating System Concepts – 10th Edition

4.14

Silberschatz et al © 2018 modified by Khazaei

14

User and Kernel Threads



Q: Do we need to have a relationship
between user threads and kernel
threads?

Assignment Project Exam Help

Operating System Concepts – 10th Edition

4.15

Silberschatz et al © 2018 modified by Khazaei

15

<https://powcoder.com>

Add WeChat powcoder

Multithreading Models

- Relationship between user threads and kernel threads:
 - Many-to-One
 - One-to-One
 - Many-to-Many

Operating System Concepts – 10th Edition

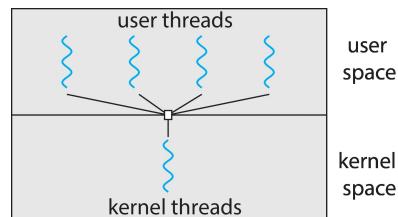
4.16

Silberschatz et al © 2018 modified by Khazaei

16

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



Assignment Project Exam Help

Operating System Concepts – 10th Edition

4.17

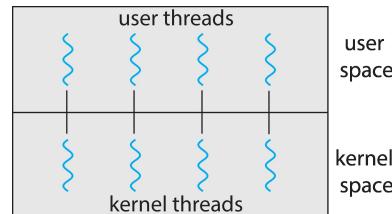
Silberschatz et al © 2018 modified by Khazaei

17

<https://powcoder.com>

Add WeChat powcoder 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
- Many threads at kernel may result in poor performance (overhead)
 - Number of threads per process sometimes restricted due to overhead
- Examples
 - Windows
 - Linux



Operating System Concepts – 10th Edition

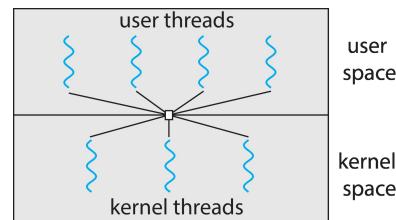
4.18

Silberschatz et al © 2018 modified by Khazaei

18

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
- Windows with the *ThreadFiber* package
- Otherwise not very common



Assignment Project Exam Help

Operating System Concepts – 10th Edition

4.19

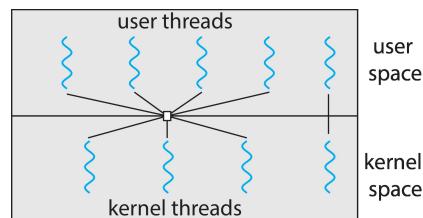
Silberschatz et al © 2018 modified by Khazaei

19

<https://powcoder.com>

Add WeChat, powcoder

- Similar to M:M, except that it allows a user thread to be **bound** to kernel thread



Operating System Concepts – 10th Edition

4.20

Silberschatz et al © 2018 modified by Khazaei

20

Question?

- Why a user thread must always be mapped to a specific kernel thread?
 - Users' programs make their own "threads" and simulate context-switches to switch between them. However, these threads aren't kernel threads. Each user thread can't actually run on its own, and the only way for a user thread to run is if a kernel thread is actually told to execute the code contained in a user thread.
 - In other words, the reason that user threads have to be associated with kernel threads is that by itself a user thread is just a bunch of data in a user program. Kernel threads are the real threads in the system, so for a user thread to make progress the user program has to have its scheduler take a user thread and then run it on a kernel thread.

Assignment Project Exam Help

Operating System Concepts – 10th Edition

4.21

Silberschatz et al © 2018 modified by Khazaei

21

<https://powcoder.com>

Add WeChat powcoder

- **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

Operating System Concepts – 10th Edition

4.22

Silberschatz et al © 2018 modified by Khazaei

22

11

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 (Linux & Mac OS X)

Assignment Project Exam Help

Operating System Concepts – 10th Edition

4.23

Silberschatz et al © 2018 modified by Khazaei

23

<https://powcoder.com>

Add WeChat powcoder

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

#include <stdlib.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 */

    /* set the default attributes of the thread */
    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);
}
```

Operating System Concepts – 10th Edition

4.24

Silberschatz et al © 2018 modified by Khazaei

24

Pthreads Example (cont)

```
/* The thread will execute 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);
}
```

Assignment Project Exam Help

Operating System Concepts – 10th Edition

4.25

Silberschatz et al © 2018 modified by Khazaei

25

<https://powcoder.com>

Add WeChat powcoder

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);
```

Operating System Concepts – 10th Edition

4.26

Silberschatz et al © 2018 modified by Khazaei

26

Windows Multithreaded C Program

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

/* The thread will execute in this function */
DWORD WINAPI Summation(LPVOID Param)
{
    DWORD Upper = *(DWORD*)Param;
    for (DWORD i = 1; i <= Upper; i++)
        Sum += i;
    return 0;
}
```

Assignment Project Exam Help

Operating System Concepts – 10th Edition

4.27

Silberschatz et al © 2018 modified by Khazaei

27

<https://powcoder.com>

Add WeChat powcoder Windows Multithreaded C Program (Cont.)

```
int main(int argc, char *argv[])
{
    DWORD ThreadId;
    HANDLE ThreadHandle;
    int Param;

    Param = atoi(argv[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 */

    /* now wait for the thread to finish */
    WaitForSingleObject(ThreadHandle, INFINITE);

    /* close the thread handle */
    CloseHandle(ThreadHandle);

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

Operating System Concepts – 10th Edition

4.28

Silberschatz et al © 2018 modified by Khazaei

28

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:
 - Extending **Thread** class
 - Implementing the **Runnable** interface
- Standard practice is to implement Runnable interface

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

Assignment Project Exam Help

Operating System Concepts – 10th Edition

4.29

Silberschatz et al © 2018 modified by Khazaei

29

<https://powcoder.com>

Add WeChat powcoder
Java Threads

Implementing Runnable interface:

```
class Task implements Runnable  
{  
    public void run() {  
        System.out.println("I am a thread.");  
    }  
}
```

Creating a thread:

```
Thread worker = new Thread(new Task());  
worker.start();
```

Waiting on a thread:

```
try {  
    worker.join();  
}  
catch (InterruptedException ie) { }
```

Operating System Concepts – 10th Edition

4.30

Silberschatz et al © 2018 modified by Khazaei

30

Java Executor Framework

- Rather than explicitly creating threads, Java also allows thread creation around the Executor interface:

```
public interface Executor
{
    void execute(Runnable command);
}
```

- The Executor is used as follows:

```
Executor service = new Executor;
service.execute(new Task());
```

Assignment Project Exam Help

Operating System Concepts – 10th Edition

4.31

Silberschatz et al © 2018 modified by Khazaei

31

<https://powcoder.com>

Add WeChat powcoder

```
import java.util.concurrent.*;

class Summation implements Callable<Integer>
{
    private int upper;
    public Summation(int upper) {
        this.upper = upper;
    }

    /* The thread will execute in this method */
    public Integer call() {
        int sum = 0;
        for (int i = 1; i <= upper; i++)
            sum += i;

        return new Integer(sum);
    }
}
```

Operating System Concepts – 10th Edition

4.32

Silberschatz et al © 2018 modified by Khazaei

32

Java Executor Framework (cont)

```
public class Driver
{
    public static void main(String[] args) {
        int upper = Integer.parseInt(args[0]);

        ExecutorService pool = Executors.newSingleThreadExecutor();
        Future<Integer> result = pool.submit(new Summation(upper));

        try {
            System.out.println("sum = " + result.get());
        } catch (InterruptedException | ExecutionException ie) { }
    }
}
```

Assignment Project Exam Help

Operating System Concepts – 10th Edition

4.33

Silberschatz et al © 2018 modified by Khazaei

33

<https://powcoder.com>

Add WeChat powcoder

Lambda Expression in Java

Beginning with Version 1.8 of the language, Java introduced Lambda expressions, which allow a much cleaner syntax for creating threads. Rather than defining a separate class that implements Runnable, a Lambda expression can be used instead:

```
Runnable task = () -> {
    System.out.println("I am a thread.");
};

Thread worker = new Thread(task);
worker.start();
```

Lambda expressions—as well as similar functions known as **closures**—are a prominent feature of functional programming languages and have been available in several nonfunctional languages as well including Python, C++, and C#. As we shall see in later examples in this chapter, Lambda expressions often provide a simple syntax for developing parallel applications.

Operating System Concepts – 10th Edition

4.34

Silberschatz et al © 2018 modified by Khazaei

34

Implicit Threading

- Growing in popularity as numbers of threads increase, program correctness more difficult with explicit threads
- Creation and management of threads done by compilers and run-time libraries rather than programmers
- Five methods explored
 - Thread Pools
 - Fork-Join
 - OpenMP
 - Grand Central Dispatch
 - Intel Threading Building Blocks

Assignment Project Exam Help

Operating System Concepts – 10th Edition

4.35

Silberschatz et al © 2018 modified by Khazaei

35

<https://powcoder.com>

Add WeChat powcoder

- 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(VOID Param) {
    /*
     * this function runs as a separate thread.
     */
}
```

Operating System Concepts – 10th Edition

4.36

Silberschatz et al © 2018 modified by Khazaei

36

18

Java Thread Pools

- Three factory methods for creating thread pools in Executors class:

- static ExecutorService newSingleThreadExecutor()
- static ExecutorService newFixedThreadPool(int size)
- static ExecutorService newCachedThreadPool()

Assignment Project Exam Help

Operating System Concepts – 10th Edition

4.37

Silberschatz et al © 2018 modified by Khazaei

37

<https://powcoder.com>

Add WeChat powcoder

```
import java.util.concurrent.*;

public class ThreadPoolExample
{
    public static void main(String[] args) {
        int numTasks = Integer.parseInt(args[0].trim());

        /* Create the thread pool */
        ExecutorService pool = Executors.newCachedThreadPool();

        /* Run each task using a thread in the pool */
        for (int i = 0; i < numTasks; i++)
            pool.execute(new Task());

        /* Shut down the pool once all threads have completed */
        pool.shutdown();
    }
}
```

Operating System Concepts – 10th Edition

4.38

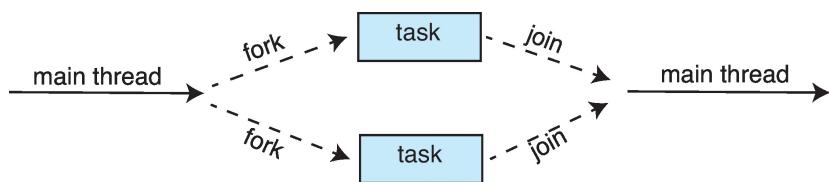
Silberschatz et al © 2018 modified by Khazaei

38

19

Fork-Join Parallelism

- Multiple threads (tasks) are **forked**, and then **joined**.



Assignment Project Exam Help

Operating System Concepts – 10th Edition

4.39

Silberschatz et al © 2018 modified by Khazaei

39

<https://powcoder.com>

Add WeChat powcoder

- General algorithm for fork-join strategy:

```
Task(problem)
    if problem is small enough
        solve the problem directly
    else
        subtask1 = fork(new Task(subset of problem))
        subtask2 = fork(new Task(subset of problem))

        result1 = join(subtask1)
        result2 = join(subtask2)

    return combined results
```

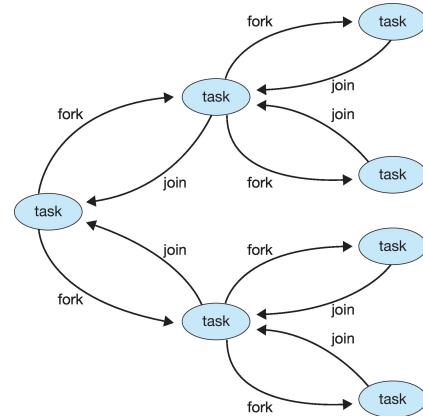
Operating System Concepts – 10th Edition

4.40

Silberschatz et al © 2018 modified by Khazaei

40

Fork-Join Parallelism



Assignment Project Exam Help

Operating System Concepts – 10th Edition

4.41

Silberschatz et al © 2018 modified by Khazaei

41

<https://powcoder.com>

Add WeChat powcoder
Fork-Join Parallelism in Java

```
ForkJoinPool pool = new ForkJoinPool();
// array contains the integers to be summed
int[] array = new int[SIZE];

SumTask task = new SumTask(0, SIZE - 1, array);
int sum = pool.invoke(task);
```

Operating System Concepts – 10th Edition

4.42

Silberschatz et al © 2018 modified by Khazaei

42

Fork-Join Parallelism in Java

```
import java.util.concurrent.*;  
  
public class SumTask extends RecursiveTask<Integer>  
{  
    static final int THRESHOLD = 1000;  
  
    private int begin;  
    private int end;  
    private int[] array;  
  
    public SumTask(int begin, int end, int[] array) {  
        this.begin = begin;  
        this.end = end;  
        this.array = array;  
    }  
  
    protected Integer compute0 {  
        if (end - begin < THRESHOLD) {  
            int sum = 0;  
            for (int i = begin; i <= end; i++)  
                sum += array[i];  
  
            return sum;  
        } else {  
            int mid = (begin + end) / 2;  
  
            SumTask leftTask = new SumTask(begin, mid, array);  
            SumTask rightTask = new SumTask(mid + 1, end, array);  
  
            leftTask.fork();  
            rightTask.fork();  
  
            return rightTask.join() + leftTask.join();  
        }  
    }  
}
```

```
int _pow_recursion(int x, int y)  
{  
    if (y < 0)  
    {  
        return (-1);  
    }  
    if (y == 0)  
    {  
        return (1);  
    }  
    return (_pow_recursion(x, y - 1) * x);  
}
```

Assignment Project Exam Help

Operating System Concepts – 10th Edition

4.43

Silberschatz et al © 2018 modified by Khazaei

43

<https://powcoder.com>

Add WeChat powcoder



- 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

```
#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;  
}
```

Operating System Concepts – 10th Edition

4.45

Silberschatz et al © 2018 modified by Khazaei

45

- Run the for-loop in parallel

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

Assignment Project Exam Help

Operating System Concepts – 10th Edition

4.46

Silberschatz et al © 2018 modified by Khazaei

46

<https://powcoder.com>

Add WeChat powcoder Grand Central Dispatch

- Apple technology for macOS and iOS operating systems
- Extensions to C, C++ and Objective-C languages, API, and run-time library
- Allows identification of parallel sections
- Manages most of the details of threading
- Block is in “^{}” :

```
^{ printf("I am a block"); }
```
- Blocks placed in dispatch queue
 - Assigned to available thread in thread pool when removed from queue

Operating System Concepts – 10th Edition

4.47

Silberschatz et al © 2018 modified by Khazaei

47

Grand Central Dispatch

- Two types of dispatch queues:
 - **serial** – blocks removed in FIFO order, queue is per process, called **main queue**
 - ▶ Programmers can create additional serial queues within program
 - **concurrent** – removed in FIFO order but several may be removed at a time
 - ▶ Four system wide queues divided by quality of service:
 - QOS_CLASS_USER_INTERACTIVE
 - QOS_CLASS_USER_INITIATED
 - QOS_CLASS_USER.Utility
 - QOS_CLASS_USER_BACKGROUND

Assignment Project Exam Help

Operating System Concepts – 10th Edition

4.48

Silberschatz et al © 2018 modified by Khazaei

48

<https://powcoder.com>

Add WeChat powcoder

- For the **Swift** language, a task is defined as a closure – similar to a block, minus the caret symbol
- Closures are submitted to the queue using the `dispatch_async()` function:

```
let queue = dispatch_get_global_queue
           (QOS_CLASS_USER_INITIATED, 0)

dispatch_async(queue, { print("I am a closure.") })
```

Operating System Concepts – 10th Edition

4.49

Silberschatz et al © 2018 modified by Khazaei

49

Intel Threading Building Blocks (TBB)

- Template library for designing parallel C++ programs
- A serial version of a simple for-loop

```
for (int i = 0; i < n; i++) {  
    apply(v[i]);  
}
```

- The same for loop written using TBB with `parallel_for` statement:

```
parallel_for (size_t(0), n, [=](size_t i) {apply(v[i]);});
```

Assignment Project Exam Help

Operating System Concepts – 10th Edition

4.50

Silberschatz et al © 2018 modified by Khazaei

50

<https://powcoder.com>

Add WeChat powcoder Threading Issues

- Semantics of `fork()` and `exec()` system calls
- Signal handling
 - Synchronous and asynchronous
- Thread cancellation of target thread
 - Asynchronous or deferred
- Thread-local storage
- Scheduler Activations

Operating System Concepts – 10th Edition

4.51

Silberschatz et al © 2018 modified by Khazaei

51

25

Semantics of fork() and exec()

- Does `fork()` duplicate only the calling thread or all threads?
 - Some UNIXes have two versions of fork
- `exec()` usually works as normal – replace the running process including all threads

Assignment Project Exam Help

Operating System Concepts – 10th Edition

4.52

Silberschatz et al © 2018 modified by Khazaei

52

<https://powcoder.com>

Add WeChat powcoder

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. Signal is handled by one of two signal handlers:
 1. default
 2. user-defined
- Every signal has **default handler** that kernel runs when handling signal
 - **User-defined signal handler** can override default
 - For single-threaded, signal delivered to process

Operating System Concepts – 10th Edition

4.53

Silberschatz et al © 2018 modified by Khazaei

53

Signal Handling (Cont.)

- Where should a signal be delivered for multi-threaded?
 - 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

Assignment Project Exam Help

Operating System Concepts – 10th Edition

4.54

Silberschatz et al © 2018 modified by Khazaei

54

<https://powcoder.com>

Add WeChat powcoder

- Terminating a thread before it has finished
- Thread to be canceled is **target thread**
- Two general approaches:
 - **Asynchronous cancellation** terminates the target thread immediately
 - **Deferred cancellation** allows the target thread to periodically check if it should be cancelled
- Pthread code to create and cancel a thread:

```
pthread_t tid;  
/* create the thread */  
pthread_create(&tid, 0, worker, NULL);  
  
...  
  
/* cancel the thread */  
pthread_cancel(tid);  
  
/* wait for the thread to terminate */  
pthread_join(tid, NULL);
```

Operating System Concepts – 10th Edition

4.55

Silberschatz et al © 2018 modified by Khazaei

55

Thread Cancellation (Cont.)

- Invoking thread cancellation requests cancellation, but actual cancellation depends on thread state

Mode	State	Type
Off	Disabled	—
Deferred	Enabled	Deferred
Asynchronous	Enabled	Asynchronous

- If thread has cancellation disabled, cancellation remains pending until thread enables it
- Default type is deferred
 - Cancellation only occurs when thread reaches **cancellation point**
 - i.e. `pthread_testcancel()`
 - Then `cleanup handler` is invoked
- On Linux systems, thread cancellation is handled through signals

Operating System Concepts – 10th Edition

4.56

Silberschatz et al © 2018 modified by Khazaei

56

<https://powcoder.com>

Add WeChat powcoder

Thread Cancellation in Java

- Deferred cancellation uses the `interrupt()` method, which sets the interrupted status of a thread.

```
Thread worker;  
.  
.  
/* set the interruption status of the thread */  
worker.interrupt()
```

- A thread can then check to see if it has been interrupted:

```
while (!Thread.currentThread().isInterrupted()) {  
    .  
}
```

Operating System Concepts – 10th Edition

4.57

Silberschatz et al © 2018 modified by Khazaei

57

Thread-Local Storage

- Thread-local storage (TLS) 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)
- Different from local variables
 - Local variables visible only during single function invocation
 - TLS visible across function invocations
- Similar to `static` data
 - TLS is unique to each thread

Assignment Project Exam Help

Operating System Concepts – 10th Edition

4.58

Silberschatz et al © 2018 modified by Khazaei

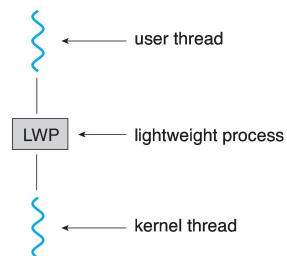
58

<https://powcoder.com>

Add WeChat powcoder

Scheduler Activations

- Both M:M and Two-level models require communication to maintain the appropriate number of kernel threads allocated to the application
- Typically use an intermediate data structure between user and kernel threads – **lightweight process (LWP)**
 - Appears to be a virtual processor on which process can schedule user thread to run
 - Each LWP attached to kernel thread
 - How many LWPs to create?
- Scheduler activations provide **upcalls** - a communication mechanism from the kernel to the **upcall handler** in the thread library
- This communication allows an application to maintain the correct number kernel threads



Operating System Concepts – 10th Edition

4.59

Silberschatz et al © 2018 modified by Khazaei

59

Operating System Examples

- Windows Threads
- Linux Threads

Assignment Project Exam Help

Operating System Concepts – 10th Edition

4.60

Silberschatz et al © 2018 modified by Khazaei

60

<https://powcoder.com>

Add WeChat powcoder Windows Threads

- Windows API – primary API for Windows applications
- Implements the one-to-one mapping, kernel-level
- Each thread contains
 - A thread id
 - Register set representing state of processor
 - Separate user and kernel stacks for when thread runs in user mode or kernel mode
 - Private data storage area used by run-time libraries and dynamic link libraries (DLLs)
- The register set, stacks, and private storage area are known as the **context** of the thread

Operating System Concepts – 10th Edition

4.61

Silberschatz et al © 2018 modified by Khazaei

61

30

Windows Threads (Cont.)

- 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

Assignment Project Exam Help

Operating System Concepts – 10th Edition

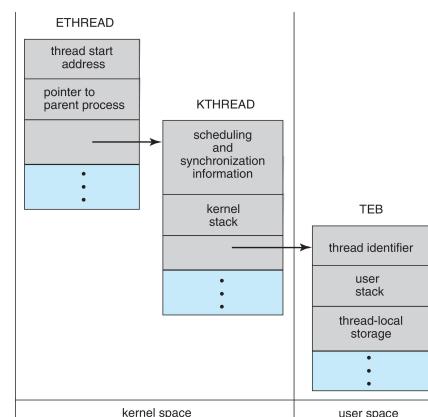
4.62

Silberschatz et al © 2018 modified by Khazaei

62

<https://powcoder.com>

Add WeChat powcoder Windows Threads Data Structures



Operating System Concepts – 10th Edition

4.63

Silberschatz et al © 2018 modified by Khazaei

63

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)
 - Flags control behavior

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

- `struct task_struct` points to process data structures (shared or unique)

Assignment Project Exam Help

Operating System Concepts – 10th Edition

4.64

Silberschatz et al © 2018 modified by Khazaei

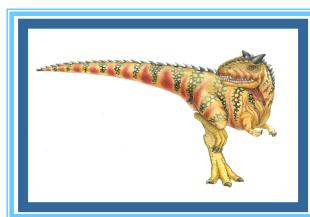
64

<https://powcoder.com>

Add WeChat powcoder

End of Chapter 4

Any Question?



Operating System Concepts – 10th Edition

Silberschatz et al © 2018 modified by Khazaei@2020

65