



Threads and Concurrency

Operating Systems

Yu-Ting Wu

(with slides borrowed from Prof. Jerry Chou)

1

Operating Systems 2022

Outline

- Thread introduction
- Multi-threading models
- Threaded case study
- Threading issues

2

2

Operating Systems 2022

Thread Introduction

3

3

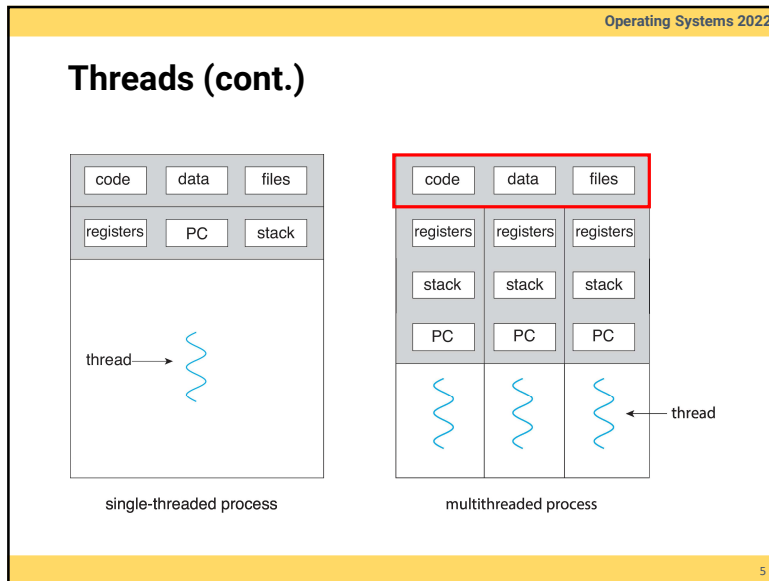
Operating Systems 2022

Threads

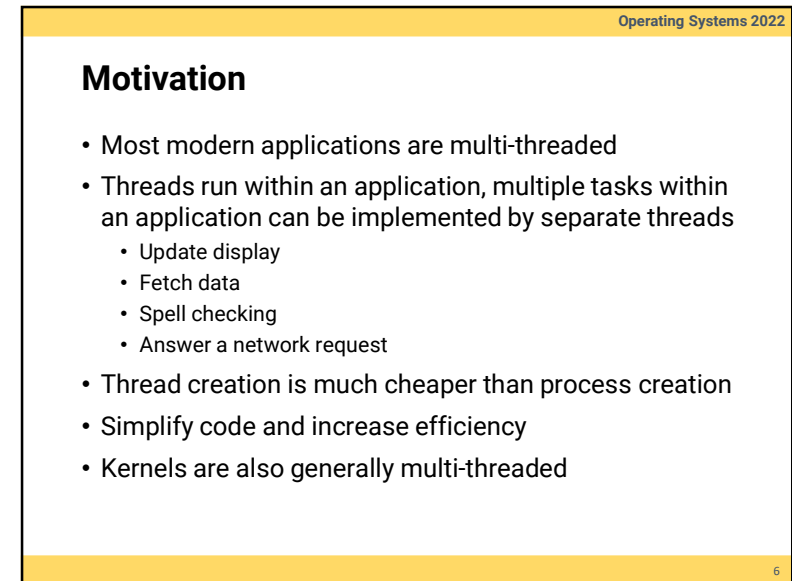
- A thread is a **lightweight** process
 - Basic unit of CPU utilization
- All threads belonging to the same process **share**
 - Code section
 - Data section (including dynamic allocated variables)
 - OS resources (e.g., open files)
- **But each thread has its own** (thread control block)
 - Thread ID
 - Program counter
 - Register set
 - Stack

4

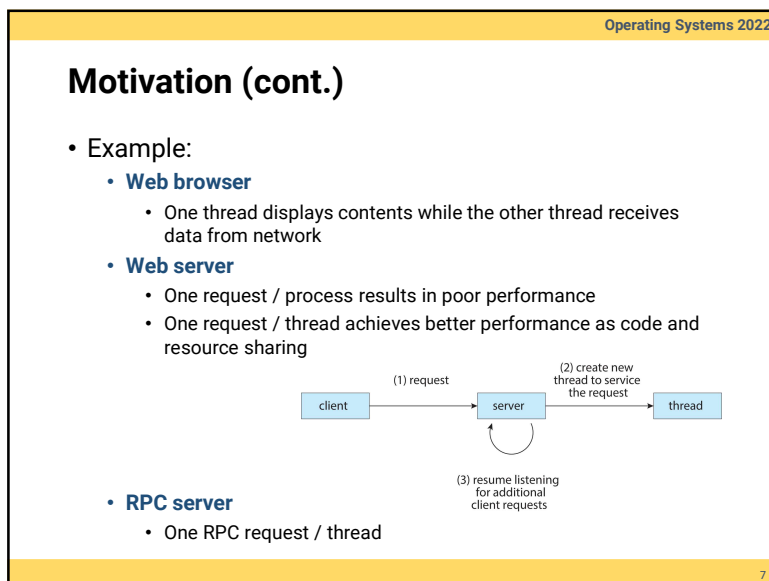
4



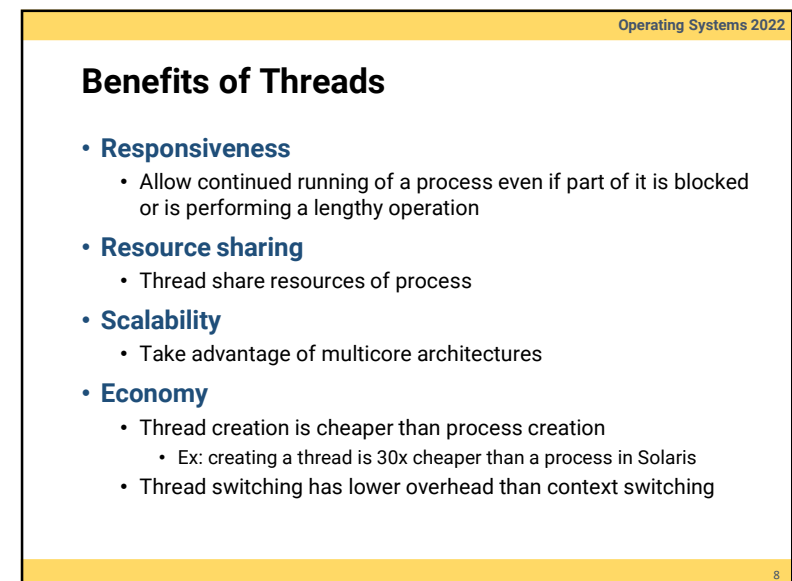
5



6



7



8

Benefits of Threads (cont.)

- Lower creation / management cost v.s. process

platform	fork()	pthread_create()	speedup
AMD 2.4 GHz Opteron	17.6	1.4	15.6x
IBM 1.5 GHz POWER4	104.5	2.1	49.8x
INTEL 2.4 GHz Xeon	54.9	1.6	34.3x
INTEL 1.4 GHz Itanium2	54.5	2.0	27.3x

- Faster inter-process communication v.s. MPI

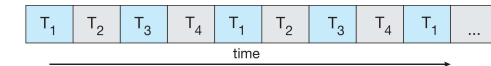
platform	MPI Shared Memory BW (GB/sec)	Pthreads Worst Case Memory-to-CPU BW (GB/sec)	speedup
AMD 2.4 GHz Opteron	1.2	5.3	4.4x
IBM 1.5 GHz POWER4	2.1	4	1.9x
INTEL 2.4 GHz Xeon	0.3	4.3	14.3x
INTEL 1.4 GHz Itanium2	1.8	6.4	3.6x

9

Multi-Core Programming

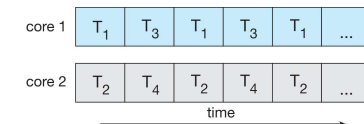
• Concurrency

- Supports more than one task making progress
- Single processor/core with scheduler can provide concurrency



• Parallelism

- A system can perform more than one task simultaneously



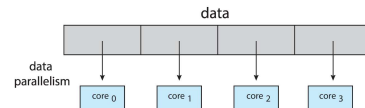
10

10

Parallelism

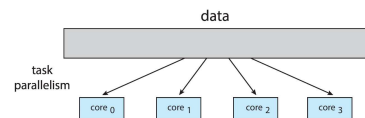
• Data parallelism

- Distribute subsets of the same data across multiple cores
- Same operation on each



• Task parallelism

- Distribute threads across cores, each thread performing unique operation



11

11

Amdahl's Law

- Identify performance gains from adding additional cores to an application that has both serial and parallel components
- Assume S is the serial portion and the system has N processing cores

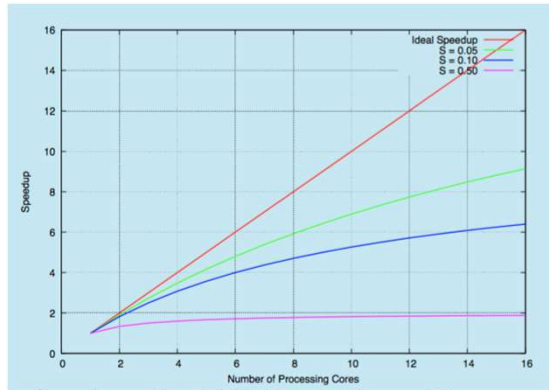
$$speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$$

- Example: 75% parallel / 25% serial, moving from 1 to 2 cores results in a speedup of 1.6 times
- As N approaches infinity, speedup approaches $1/S$

12

12

Amdahl's Law (cont.)



13

Challenges in Multi-Core Programming

- **Dividing activities**
 - Divide program into concurrent tasks
- **Data splitting**
 - Divide data accessed and manipulated by the tasks
- **Data dependency**
 - Synchronize data access
- **Balance**
 - Evenly distribute tasks to cores
- **Testing and debugging**

14

Multi-Thread Models

15

User Threads and Kernel Threads

- **User threads**
 - Thread management done by user-level threads library
 - Ex: POSIX Pthreads, Win32 threads, Java threads
- **Kernel threads**
 - Supported by kernel (OS) directly
 - Ex: Windows 2000 (NT), Solaris, Linux, Tru64 UNIX
- Programmers create **user threads** using thread APIs and then the threads are bounded to **kernel threads** by the OS

16

User Threads and Kernel Threads (cont.)

• User threads

- Thread library provides support for thread creation, scheduling, and deletion
- Generally fast to create and manage
- If the kernel is single-threaded, a user thread blocks → entire process blocks even if other threads are ready to run

• Kernel threads

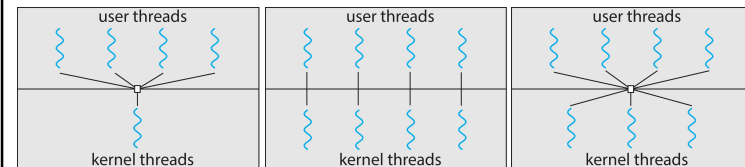
- The kernel performs thread creation, scheduling, etc.
- Generally slower to create and manage
- If a thread is blocked, the kernel can schedule another thread for execution

17

17

Multi-Threading Models

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

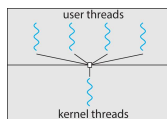


18

18

Many-to-One

- Many user-level threads mapped to a single kernel thread
- Used on systems that do not support kernel threads
- The entire process will block if a thread makes a blocking system call
- Only one thread can access the kernel at a time
 - Multiple threads are unable to run in parallel on multi-processors

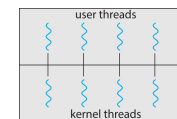


19

19

One-to-One

- Each user-level thread maps to a kernel thread
 - Creating a user-level thread creates a kernel thread
 - There could be a limit on number of kernel threads (high overhead)
- More concurrency
- Examples
 - Windows
 - Linux
 - Solaris 9 and later

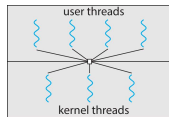


20

20

Many-to-Many

- Multiplexes many user-level threads to a smaller or equal number of kernel threads
- Allow the developer to create as many user threads as wished
- The corresponding kernel threads can run in parallel on a multi-processor
- When a thread performs a blocking call, the kernel can schedule another for execution



21

21

Thread Case Study

22

22

Case Study

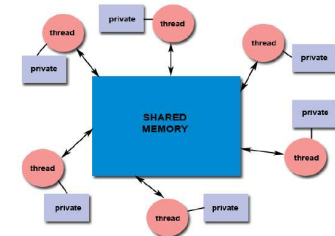
- Thread libraries
 - Pthreads
 - Java threads
- OS examples
 - Linux

23

23

Shared-Memory Programming

- **Definition**
 - Processes communicate or work together with each other through a shared memory space which can be accessed by all processes
 - Usually faster and more efficient than message passing
- **Issues**
 - Synchronization
 - Deadlock
 - Cache coherence
- **Programming techniques**
 - Parallelizing compiler
 - UNIX processes
 - Threads (Pthreads, Java)



24

24

Pthread

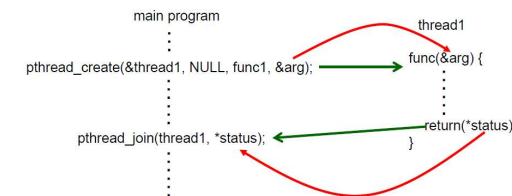
- Historically, hardware vendors have implemented their own proprietary versions of threads
- POSIX** (Portable Operating System Interface) standard is specified for portability across UNIX-like system
- Pthread is the implementation of POSIX standard for thread

25

25

Pthread Creation

- Library call: ***pthread_create(thread, attr, routine, arg)***
 - thread***: an unique identifier (token) for the new thread
 - attr***: used to set thread attributes
 - routine***: the routine that the thread will execute once it is created
 - arg***: a single argument that may be passed to routine

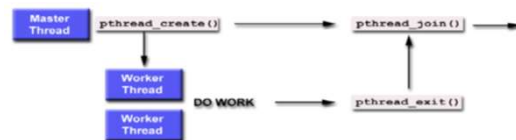


26

26

Pthread Joining and Detaching

- Library call: ***pthread_join(threadId, status)***
 - Blocks until the specified thread (threadId) terminates
 - One way to accomplish synchronization between threads
 - Example:
for (int i = 0; i < n; ++i) pthread_join(thread[i], NULL);
- Library call: ***pthread_detach(threadId)***
 - Once a thread is detached, it can never be joined
 - Detach a thread could free some system resources



27

27

Example: PThread

```

#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);
}
  
```

28

28

Example: Pthread (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);
}
```

29

29

Example: Windows Thread

```
#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;
}
```

30

30

Example: Windows Thread (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);
}
```

31

31

Java Threads

- Java threads are implemented using a thread library on the host system
 - Win32 threads on Windows
 - Pthreads on UNIX-like system
- Thread mapping depends on implementation of the java virtual machine
 - Windows 98/NT: one-to-one model
 - Solaris2: many-to-many model

32

32

Linux Kernel Threads

- Linux does not support multithreading (in the view of OS)
- Various Pthreads implementation are available for user-level
- **The fork system call**
 - Create a new process and a copy of the associated data of the parent process
- **The clone system call**
 - Create a new process and a link that points to the associated data of the parent process

33

Linux Threads (cont.)

- A set of **flags** is used in the clone system call for indication of the level of the sharing
 - None of the flags is set → clone = fork
 - All flags are set → parent and child share everything

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.

34

Threading Issues

35

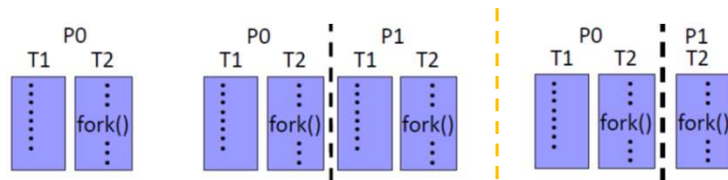
Threading Issues

- **Semantics of fork() and exec() system calls**
 - Duplicate all the threads or not
- **Thread cancellation**
 - Asynchronous or deferred
- **Signal handling**
 - Where should a signal be delivered
- **Thread pools**
 - Create a number of threads at process startup

36

Semantics of fork() and exec()

- Does **fork()** duplicate only the calling thread or all threads?
 - Some UNIX systems support two versions of fork()
- execvp()** works the same; replace the **entire** process



37

37

Thread Cancellation

- What happens if a thread terminates before it has completed?
 - Example: terminate web page loading
- Target thread:** a thread that is to be cancelled
- Two general approaches
 - Asynchronous cancellation**
 - One thread terminates the target thread immediately
 - Deferred cancellation (default option)**
 - The target thread periodically checks whether it should be terminated, allowing it an opportunity to terminate itself in an orderly fashion (canceled safely)
 - Check at cancellation point

38

38

Signal Handling

- Signals are used in UNIX systems to notify a process that a particular event has occurred
 - Synchronous:** illegal memory access
 - Asynchronous:** e.g., <control-C>
- 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

39

39

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 applications to be bound to the size of the pool
- Determine the number of threads
 - Based on number of CPUs, amount of physical memory ... etc.

40

40

Objectives Review

- Identify the basic components of a thread, and distinguish threads and processes
- Describe the benefits and challenges of designing multi-threaded applications
- Illustrate different approaches to implicit threading including fork-join and thread pools