Threads

Amir H. Payberah
amir@sics.se

Amirkabir University of Technology
(Tehran Polytechnic)

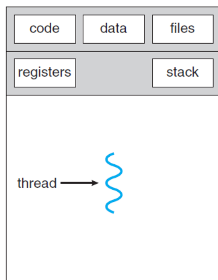
Motivation

Thread

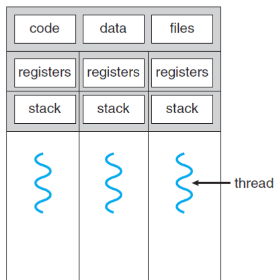
A basic unit of CPU utilization.

Threads (1/3)

- A traditional process: has a single thread.



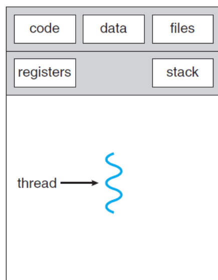
single-threaded process



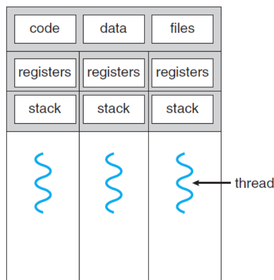
multithreaded process

Threads (1/3)

- A traditional process: has a single thread.
- Multiple threads in a process: performing more than one task at a time.



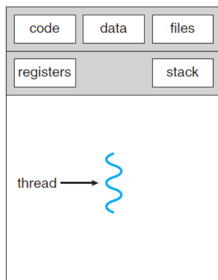
single-threaded process



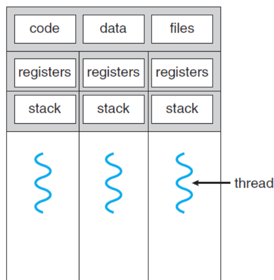
multithreaded process

Threads (1/3)

- A traditional process: has a single thread.
- Multiple threads in a process: performing more than one task at a time.
- Threads in a process share code section, data section, and other OS resources, e.g., open files.



single-threaded process



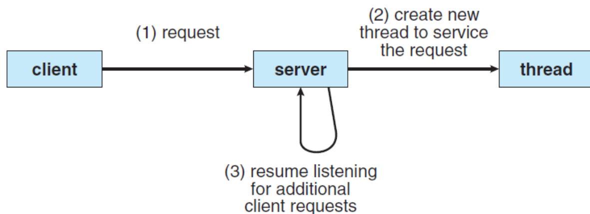
multithreaded process

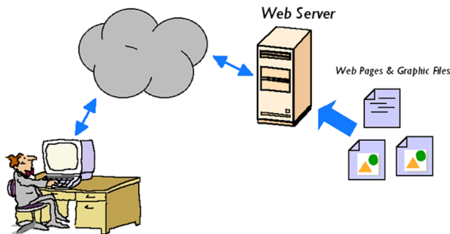
Threads (2/3)

- Multiple tasks of an application can be implemented by separate threads.
  - Update display
  - Fetch data
  - Spell checking
  - Answer a network request

Threads (3/3)

- Multi-threaded web-server architecture





Threads Benefits (1/2)

- Responsiveness: may allow continued execution if part of process is blocked, especially important for user interfaces.

Threads Benefits (1/2)

- 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.

Threads Benefits (2/2)

- Economy: cheaper than process creation, thread switching lower overhead than context switching.

Threads Benefits (2/2)

- Economy: cheaper than process creation, thread switching lower overhead than context switching.

- Scalability: process can take advantage of multiprocessor architectures

Context Switching vs. Threads Switching

- Inter-process switching: context switching from process-to-process.

Context Switching vs. Threads Switching

- Inter-process switching: context switching from process-to-process.
- Intra-process switching: switching from thread-to-thread.

Context Switching vs. Threads Switching

- Inter-process switching: context switching from process-to-process.
- Intra-process switching: switching from thread-to-thread.
- The cost of intra-process switching is less than the cost of inter-process switching.

Multicore Programming

Multiprocessor and Multicore Systems (1/2)

- Users need more computing performance: single-CPU $\rightarrow$ multi-CPU

Multiprocessor and Multicore Systems (1/2)

- Users need more computing performance: single-CPU $\rightarrow$ multi-CPU
- A similar trend in system design: place multiple computing cores on a single chip.
  - Each core appears as a separate processor.
  - Multi-core systems.

Multiprocessor and Multicore Systems (2/2)

- Multi-threaded programming
  - More efficient use of multiple cores.
  - Improved concurrency.

Concurrency vs. Parallelism (1/2)

- Parallelism
  - Performing more than one task simultaneously.

Concurrency vs. Parallelism (1/2)

- Parallelism
  - Performing more than one task simultaneously.

- Concurrency
  - Supporting more than one task by allowing all the tasks to make progress.

Concurrency vs. Parallelism (1/2)

- Parallelism
  - Performing more than one task simultaneously.

- Concurrency
  - Supporting more than one task by allowing all the tasks to make progress.
  - Single processor/core: scheduler providing concurrency.

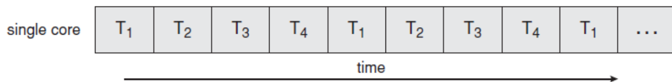
Concurrency vs. Parallelism (1/2)

- Parallelism
  - Performing more than one task simultaneously.

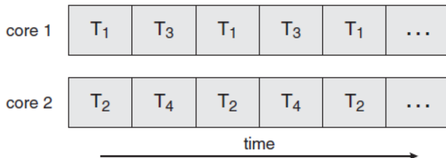
- Concurrency
  - Supporting more than one task by allowing all the tasks to make progress.
  - Single processor/core: scheduler providing concurrency.
  - It is possible to have concurrency without parallelism.

Concurrency vs. Parallelism (2/2)

- Concurrent execution on a single-core system.



- Parallelism on a multi-core system.



Types of Parallelism

- Data parallelism
  - Distributes subsets of the same data across multiple cores, same operation on each.

Types of Parallelism

- Data parallelism
  - Distributes subsets of the same data across multiple cores, same operation on each.
- Task parallelism
  - Distributes threads across cores, each thread performing unique operation.

Programming Challenges

- Dividing activities
- Balance
- Data splitting
- Data dependency
- Testing and debugging

Multi-threading Models

User Threads and Kernel Threads

- User threads: management done by user-level threads library.

- Kernel threads: supported by the Kernel.

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.

User Threads and Kernel Threads

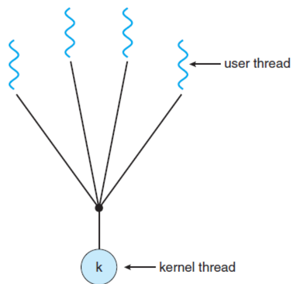
- User threads: management done by user-level threads library.
  - Three primary thread libraries:
  - POSIX pthreads
  - Windows threads
  - Java threads
- Kernel threads: supported by the Kernel.
  - All general purpose operating systems, including: Windows, Solaris, Linux, Tru64 UNIX, Mac OS X

Multi-Threading Models

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

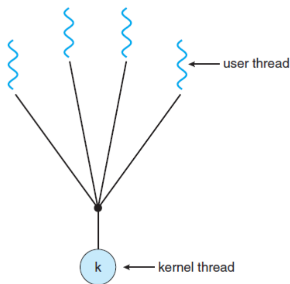
Many-to-One Model

- Many user-level threads mapped to single kernel thread.



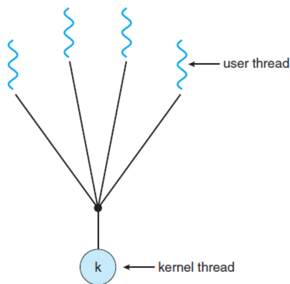
Many-to-One Model

- Many user-level threads mapped to single kernel thread.
- One thread blocking causes all to block.



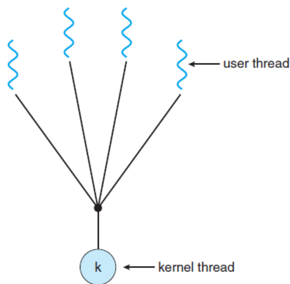
Many-to-One Model

- Many user-level threads mapped to single kernel thread.
- One thread blocking causes all to block.
- Multiple threads may not run in parallel on multicolor system because only one may be in kernel at a time.



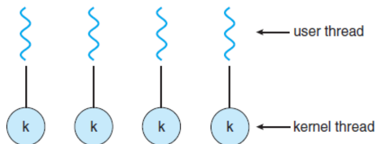
Many-to-One Model

- Many user-level threads mapped to single kernel thread.
- One thread blocking causes all to block.
- Multiple threads may not run in parallel on multicolor system because only one may be in kernel at a time.
- Few systems currently use this model.
  - Solaris green threads
  - GNU portable threads



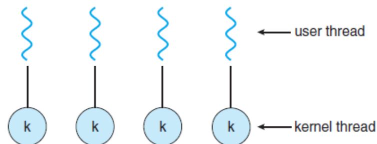
One-to-One Model

- Each user-level thread maps to a kernel thread.



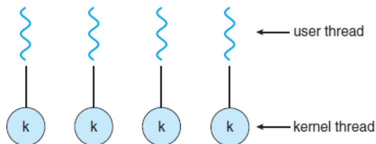
One-to-One Model

- Each user-level thread maps to a kernel thread.
- Creating a user-level thread creates a kernel thread.



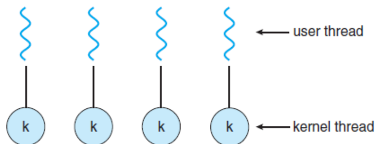
One-to-One Model

- Each user-level thread maps to a kernel thread.
- Creating a user-level thread creates a kernel thread.
- More concurrency than many-to-one.



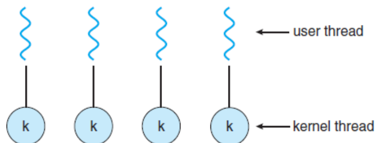
One-to-One Model

- Each user-level thread maps to a 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.



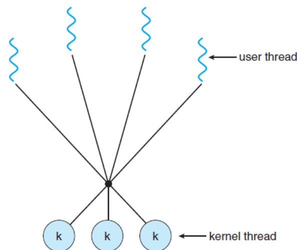
One-to-One Model

- Each user-level thread maps to a 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



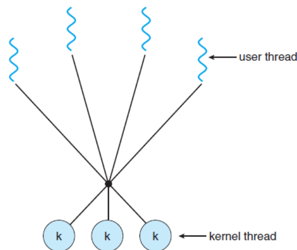
Many-to-Many Model

- Allows many user-level threads to be mapped to many kernel threads.



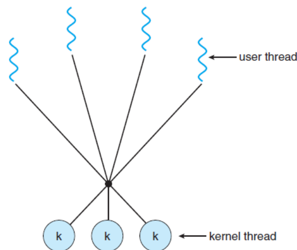
Many-to-Many Model

- Allows many user-level threads to be mapped to many kernel threads.
- Allows the OS to create a sufficient number of kernel threads.



Many-to-Many Model

- Allows many user-level threads to be mapped to many kernel threads.
- Allows the OS to create a sufficient number of kernel threads.
- Examples:
  - Solaris prior to version 9
  - Windows with the ThreadFiber package



Thread Libraries

Thread Libraries (1/2)

- Thread library provides programmer with API for creating and managing threads.

Thread Libraries (1/2)

- 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.

Thread Libraries (2/2)

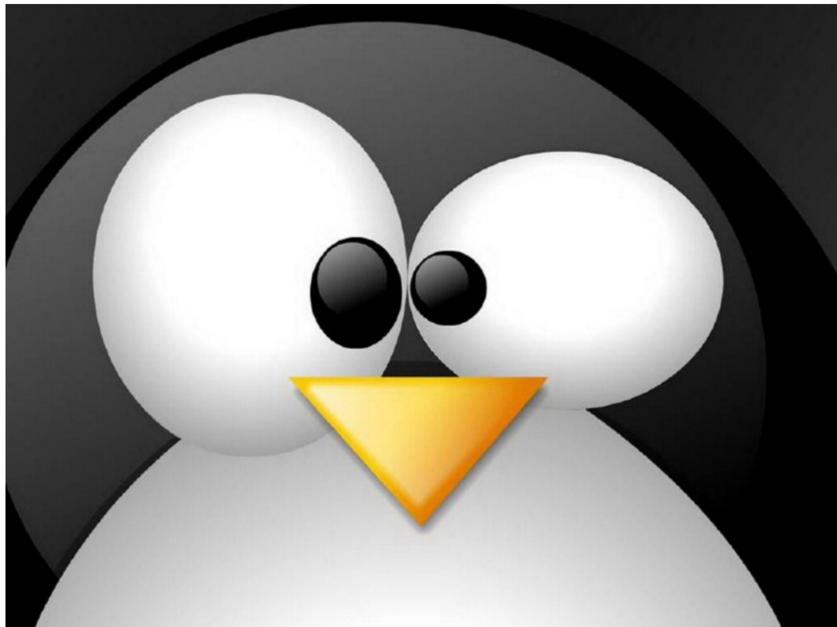
- Pthread
  - Either a user-level or a kernel-level library.

Thread Libraries (2/2)

- Pthread
  - Either a user-level or a kernel-level library.
- Windows thread
  - Kernel-level library.

Thread Libraries (2/2)

- Pthread
  - Either a user-level or a kernel-level library.
- Windows thread
  - Kernel-level library.
- Java thread
  - Uses a thread library available on the host system.



Pthreads

- A POSIX API for thread creation and synchronization.

Pthreads

- A POSIX API for thread creation and synchronization.
- Specification, not implementation.

Pthreads

- A POSIX API for thread creation and synchronization.
- Specification, not implementation.
- API specifies behavior of the thread library, implementation is up to development of the library.

Pthreads

- A POSIX 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 OSs, e.g., Solaris, Linux, Mac OS X

Thread ID

- The thread ID (TID) is the thread analogue to the process ID (PID).

Thread ID

- The thread ID (TID) is the thread analogue to the process ID (PID).
- The PID is assigned by the Linux kernel, and TID is assigned in the Pthread library.

Thread ID

- The thread ID (TID) is the thread analogue to the process ID (PID).
- The PID is assigned by the Linux kernel, and TID is assigned in the Pthread library.
- Represented by `pthread_t`.

Thread ID

- The thread ID (TID) is the thread analogue to the process ID (PID).
- The PID is assigned by the Linux kernel, and TID is assigned in the Pthread library.
- Represented by pthread_t.
- Obtaining a TID at runtime:

```
#include <pthread.h>

pthread_t pthread_self(void);
```

Creating Threads

- `pthread_create()` defines and launches a new thread.

```
#include <pthread.h>

int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
  void *(*start_routine)(void *), void *arg);
```

- `start_routine` has the following signature:

```
void *start_thread(void *arg);
```

Terminating Threads

- Terminating yourself by calling `pthread_exit()`.

```
#include <pthread.h>

void pthread_exit(void *retval);
```

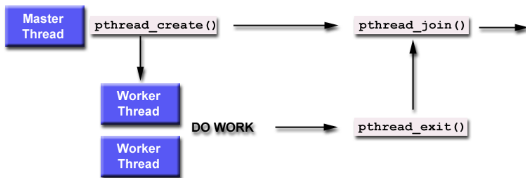
- Terminating others by calling `pthread_cancel()`.

```
#include <pthread.h>

int pthread_cancel(pthread_t thread);
```

Joining and Detaching Threads

- Joining allows one thread to block while waiting for the termination of another.

```
#include <pthread.h>

int pthread_join(pthread_t thread, void **retval);
int pthread_detach(pthread_t thread);
```

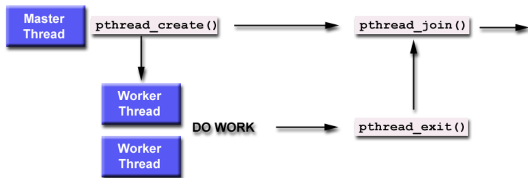


[https://computing.llnl.gov/tutorials/pthreads/#Joining]

Joining and Detaching Threads

- Joining allows one thread to block while waiting for the termination of another.
- You use join if you care about what value the thread returns when it is done, and use detach if you do not.

```
#include <pthread.h>

int pthread_join(pthread_t thread, void **retval);
int pthread_detach(pthread_t thread);
```



[https://computing.llnl.gov/tutorials/pthreads/#Joining]

A Threading Example

```
void *start_thread(void *message) {
  printf("%s\n", (const char *)message);
  return message;
}

int main(void) {
  pthread_t thread11, thread2;
  const char *message1 = "Thread 1";
  const char *message2 = "Thread 2";

  // Create two threads, each with a different message.
  pthread_create(&thread1, NULL, start_thread, (void *)message1);
  pthread_create(&thread2, NULL, start_thread, (void *)message2);

  // Wait for the threads to exit.
  pthread_join(thread1, NULL);
  pthread_join(thread2, NULL);

  return 0;
}
```

Implicit Threading

Implicit Threading

- Increasing the number of threads: program correctness more difficult with explicit threads.

Implicit Threading

- Increasing the number of threads: program correctness more difficult with explicit threads.

- Implicit threading: creation and management of threads done by compilers and run-time libraries rather than programmers.

Implicit Threading

- Increasing the number of threads: program correctness more difficult with explicit threads.

- Implicit threading: creation and management of threads done by compilers and run-time libraries rather than programmers.

- Three methods explored:
  - Thread Pools
  - OpenMP
  - Grand Central Dispatch

Thread Pools

- Create a number of threads in a pool where they await work.

Thread Pools

- Create a number of threads in a pool where they await work.
- Usually slightly faster to service a request with an existing thread than create a new thread.

Thread Pools

- Create a number of threads in a pool where they await work.
- 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.

OpenMP (1/2)

- Set of compiler directives and APIs for C, C++, FORTRAN.

OpenMP (1/2)

- Set of compiler directives and APIs for C, C++, FORTRAN.
- Identifies parallel regions: blocks of code that can run in parallel.

OpenMP (1/2)

- Set of compiler directives and APIs for C, C++, FORTRAN.
- 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`: run for loop in parallel.

OpenMP (2/2)

```c
#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;
}
```

Grand Central Dispatch (1/2)

- Apple technology for Mac OS X and iOS: extensions to C, C++ API, and run-time library.

Grand Central Dispatch (1/2)

- Apple technology for Mac OS X and iOS: extensions to C, C++ API, and run-time library.
- Allows identification of parallel sections.

Grand Central Dispatch (1/2)

- Apple technology for Mac OS X and iOS: extensions to C, C++ API, and run-time library.
- Allows identification of parallel sections.
- Block is in `^{ }`: `^{ printf("I am a block"); }`

Grand Central Dispatch (1/2)

- Apple technology for Mac OS X and iOS: extensions to C, C++ API, and run-time library.
- Allows identification of parallel sections.
- Block is in `^{ }`: `^{ printf("I am a block"); }`
- Blocks placed in dispatch queue.

Grand Central Dispatch (2/2)

- Two types of dispatch queues:
- Serial: blocks removed in FIFO order, queue is per process.
- Concurrent: removed in FIFO order but several may be removed at a time.

```
dispatch_queue_t queue = dispatch_get_global_queue
  (DISPATCH QUEUE PRIORITY DEFAULT, 0);

dispatch_async(queue, ^{ printf("I am a block."); });
```

Threading Issues

Semantics of fork() and exec()

- Does `fork()` duplicate only the calling thread or all threads?
  - Some UNIXes have two versions of fork.

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: replaces the running process including all threads.

Signal Handling

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

Signal Handling

- Where should a signal be delivered for multi-threaded?
  - Deliver the signal to the thread to which the signal applies.

Signal Handling

- 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.

Signal Handling

- 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.

Signal Handling

- 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.

Summary

Summary

- Single-thread vs. Multi-thread

Summary

- Single-thread vs. Multi-thread
- Interprocess vs. Intraprocess context switchings

Summary

- Single-thread vs. Multi-thread
- Interprocess vs. Intraprocess context switchings
- Concurrency vs. parallelism

Summary

- Single-thread vs. Multi-thread
- Interprocess vs. Intraprocess context switchings
- Concurrency vs. parallelism
- Multi-threading models: many-to-one, one-to-one, many-to-many

Summary

- Single-thread vs. Multi-thread
- Interprocess vs. Intraprocess context switchings
- Concurrency vs. parallelism
- Multi-threading models: many-to-one, one-to-one, many-to-many
- Multi-thread libraries: pthread

Summary

- Single-thread vs. Multi-thread
- Interprocess vs. Intraprocess context switchings
- Concurrency vs. parallelism
- Multi-threading models: many-to-one, one-to-one, many-to-many
- Multi-thread libraries: pthread
- Implicit threading

Questions?

Acknowledgements

Some slides were derived from Avi Silberschatz slides.