Chapter 4: Multithreaded Programming

Prof. Li-Pin Chang

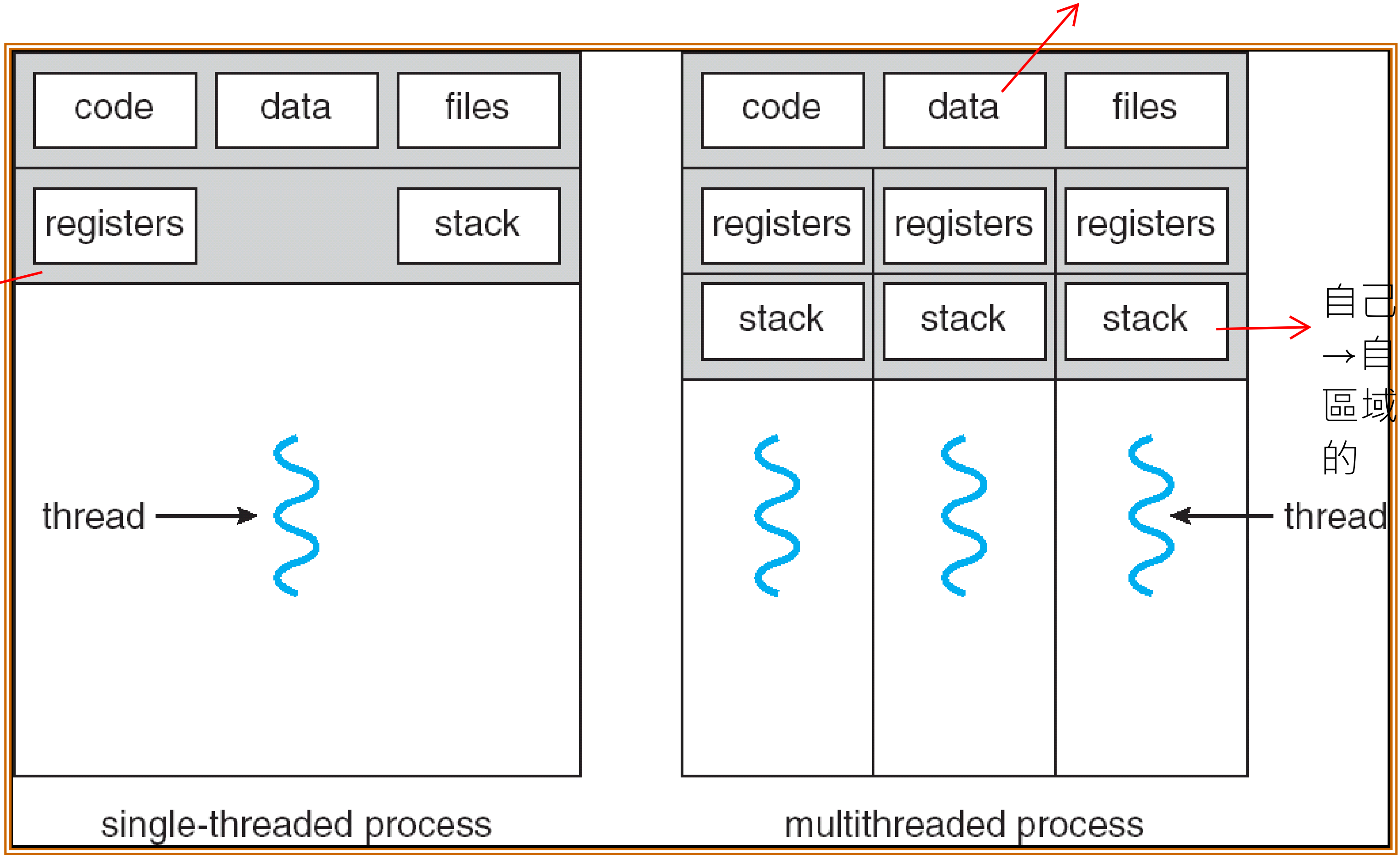
National Chiao Tung University

Chapter 4: Multithreaded Programming

- Overview
- Multithreading Models
- Thread Libraries
- Threading Issues
- Operating-System Examples

OVERVIEW

Single and Multithreaded Processes



全域變數、動態記憶體共用

CPU contact

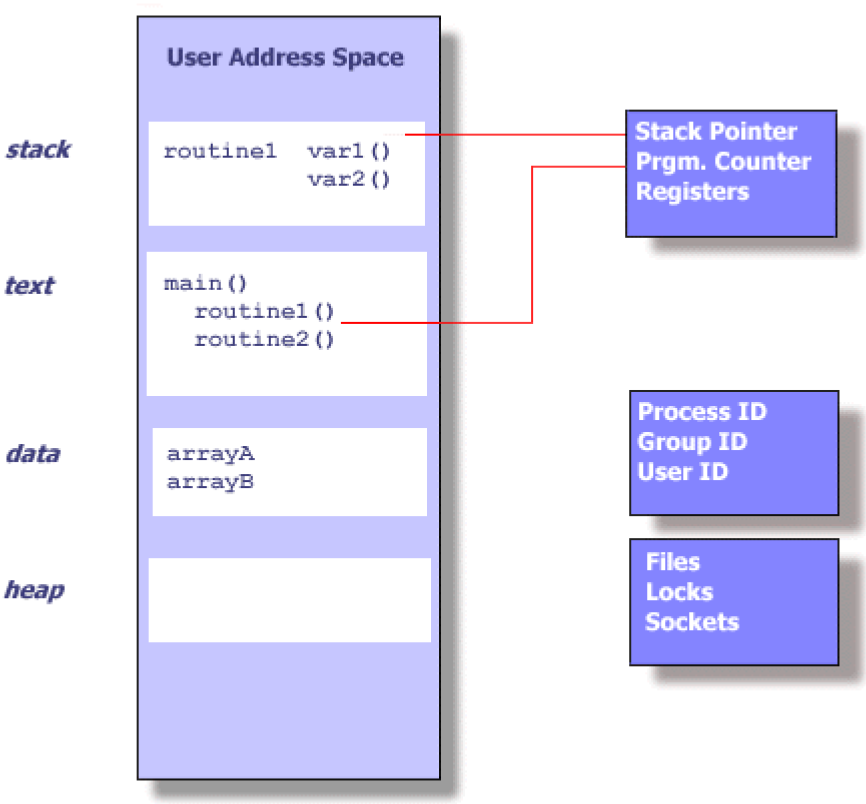
自己的stack
→自己的function call 區域變數也都是自己的

- A process is a “container” of all its threads

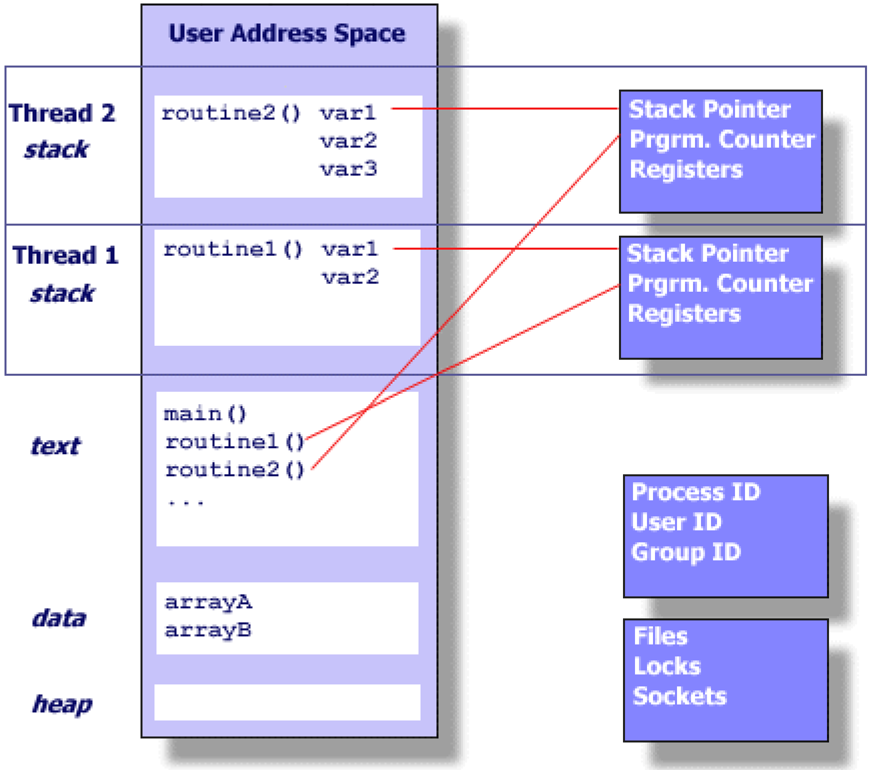
Benefits

- Responsiveness 一條thread負責UI，其他條去做會卡住的工作
  - A thread accepts UI inputs while another does computation
- Resource Sharing
  - To share code and most of the data structures
- Economy
  - A thread is a lightweight process
- Utilization of MP Architectures
  - To utilize multiple cores or to improve ILP

In Solaris it is 5 times slower to context switch a process than to context switch a thread, and 13 times slower for creation



A process



全域變數就能當sharing memory用了

Two threads in a process

- This independent flow of control is accomplished because a thread maintains its own:
  - Stack pointer
  - Registers
  - Scheduling properties (such as policy or priority)
  - Set of pending and blocked signals
  - Thread specific data.
- User threads are supported above the kernel and are managed without kernel support, while
- kernel threads are supported and managed directly by the operating system

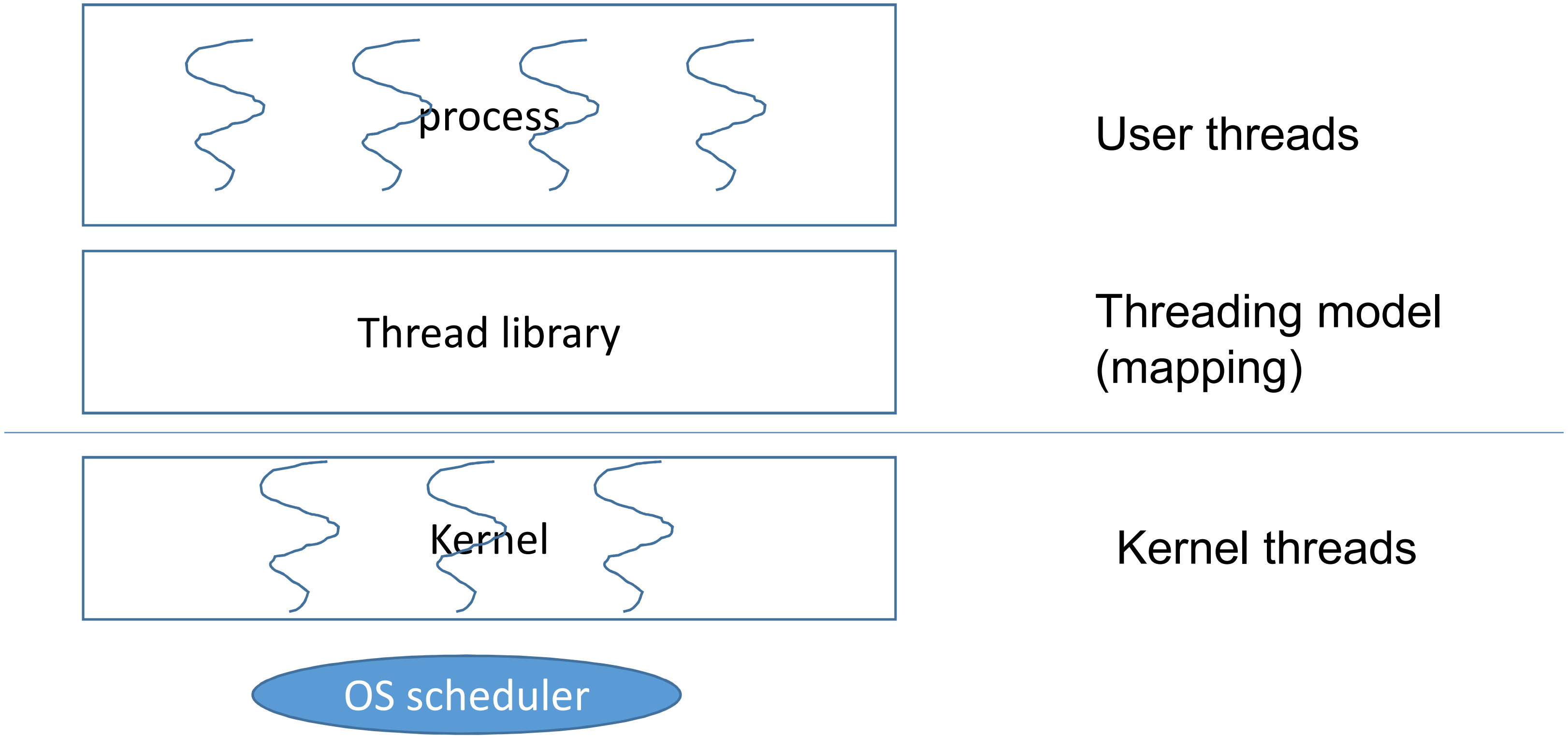
MULTITHREADING MODELS

Multithreading Models

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

Multithreading Models

User threads 和 Kernel threads 並不一定相等



process

User threads

Thread library

Threading model (mapping)

Kernel

Kernel threads

OS scheduler

Specification vs. Implementation

- Major thread libraries
  - Pthreads P for POSIX (UNIX家族的system call API)
  - Win32 threads
  - Java threads

Pthreads只是提供thread programming 的 API
至於User threads怎麼分配到真正執行的Kernel threads並沒有在標準定義

- Pthread is a specification (part of POSIX)
  - How Pthread is implemented (which threading model) is not part of the Pthread specification
  - Pthead can be implemented using 1-1, M-1, or M-M

Many-to-One

- Many user-level threads mapped to single kernel thread

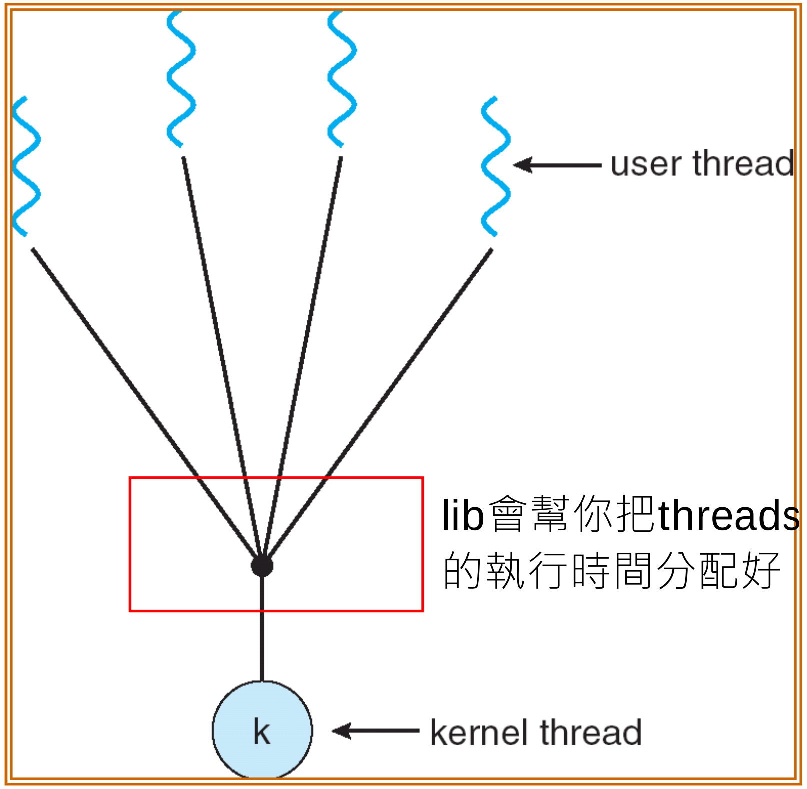
- Examples:
- Solaris Green Threads M-1
  - For JDK. Options: green or native

    1-1
- GNU Portable Threads
  - An implementation of the Pthread specification

OS看不到thread，只當成一個process

Many-to-One Model

通常只是為了邏輯上的清晰
(邏輯上的平行:大家都有在動)



lib會幫你把threads
的執行時間分配好

M-1 Model的API都會特別包裝過。
不然一個system call就整個process
wait，所有threads卡死。

並沒有實際上的平行效果
一樣是1個CPU核心來跑

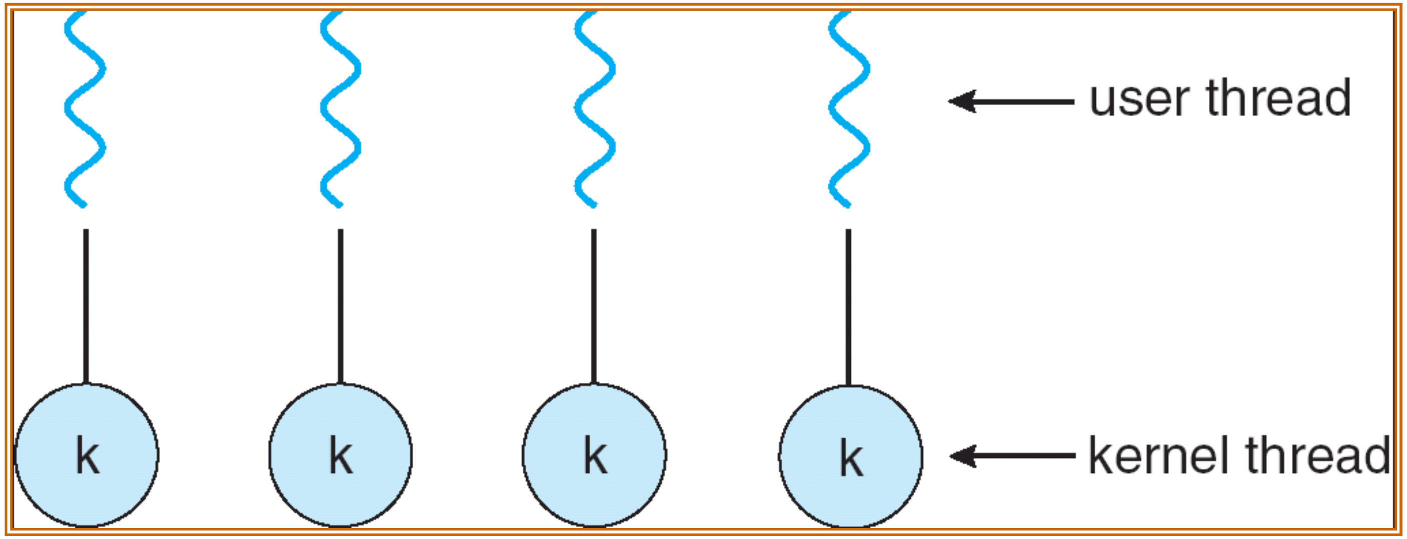
- If one single thread issues a blocking system call, then the entire collection of threads become blocked
- Not MP-aware
- Not truly concurrent

One-to-One

- Each user-level thread maps to kernel thread

- Examples
  - Windows NT/XP/2000
  - Linux
  - Solaris 9 and later

One-to-one Model



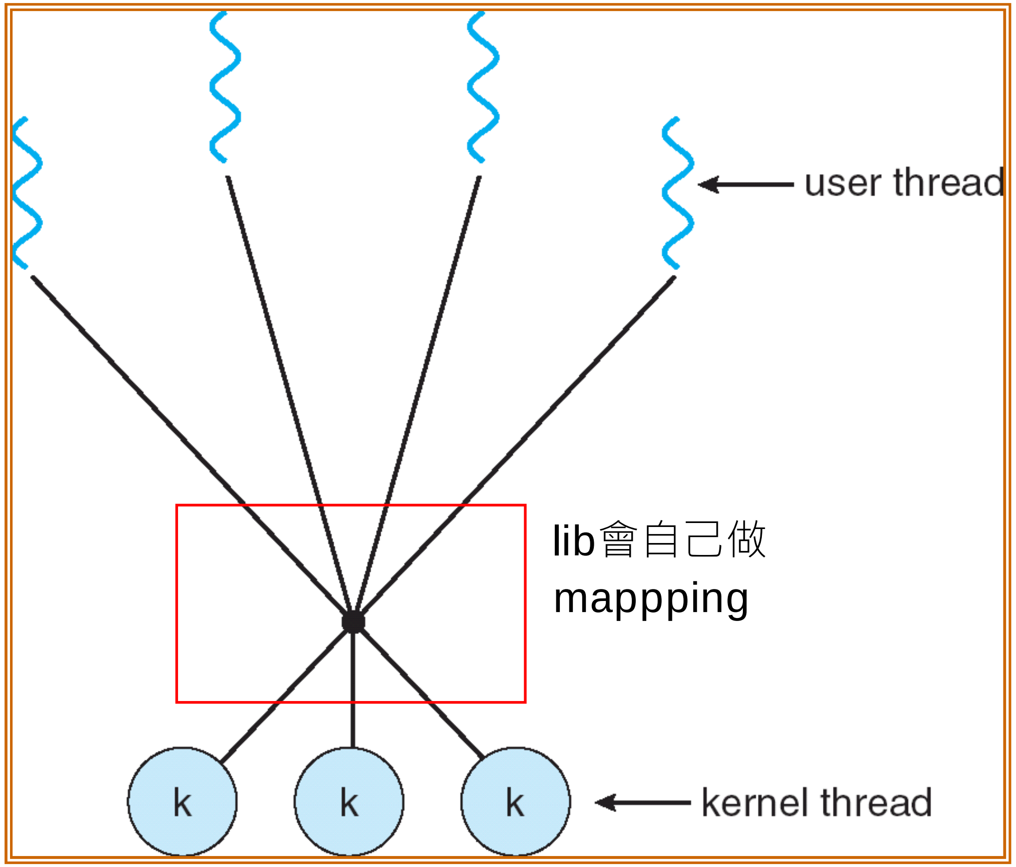
- Higher management overheads for threads
- MP-aware
- A blocking call will not block the whole thread set

管理的消耗會多一點，因為每個thread都要再kernel都有對應
能顯著加快速度(多核CPU)

Many-to-Many Model

- Allows many user level threads to be mapped to many kernel threads
  - One thread won’t block the entire process
- Allows the operating system to create a sufficient number of kernel threads
  - More economic than 1-1 model
- Solaris prior to version 9
- Windows NT/2000 with the ThreadFiber package

Many-to-Many Model



Process contention scope

System contention scope

能避免被block住

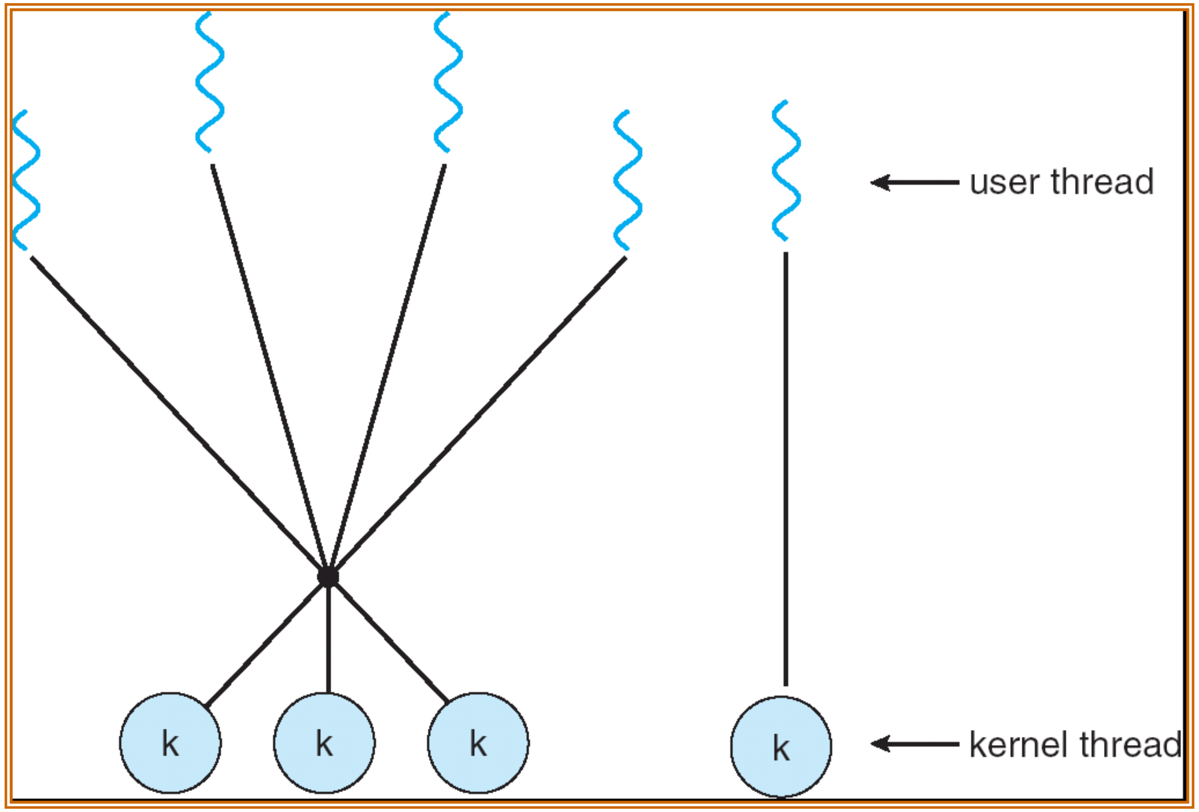
- A mixture of 1-1 and M-1

Two-level Model

- Similar to M:M, except that static binding between user threads and kernel threads is permitted

- Examples
  - IRIX
  - HP-UX
  - Tru64 UNIX
  - Solaris 8 and earlier

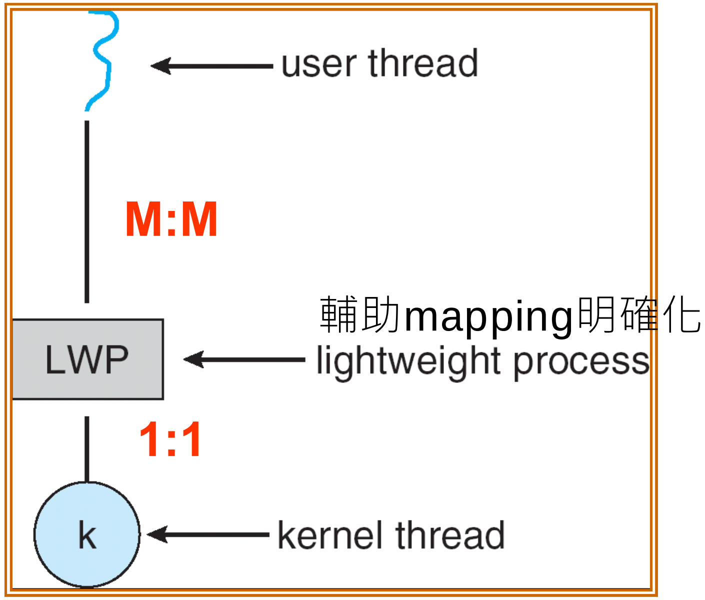
Two-level Model



- Different thread library may adopt different threading models 同一款lib也會因為實作不同而會有不同的model
  - Pthread and Java threads do not explicitly define their threading models
  - GNU Portable Thread explicitly specifies the many-to-one model
- You should not assume the threading model of a thread library
  - Check the programmers’ manual first

Light-Weight Processes

- LWP is an optional abstraction of scheduling units
- An LWP is like a virtual processor on which user threads are scheduled
- Basically the mapping of LWPs to kernel threads is 1-1
- The mapping of user threads to LWP is 1-1, M-1, or M-M



大部分都在做I/O

- Consider a multithreaded process consisting of 2 IO-bound threads and 4 CPU-bound threads. Let the threading model be M:M. Let there be 5 kernel threads.

Ans：3顆
因為5個kernel thread中有2個都在等I/O，只有3個threads能跑CPU

At most how many CPU cores that the multithreaded process can fully utilize?

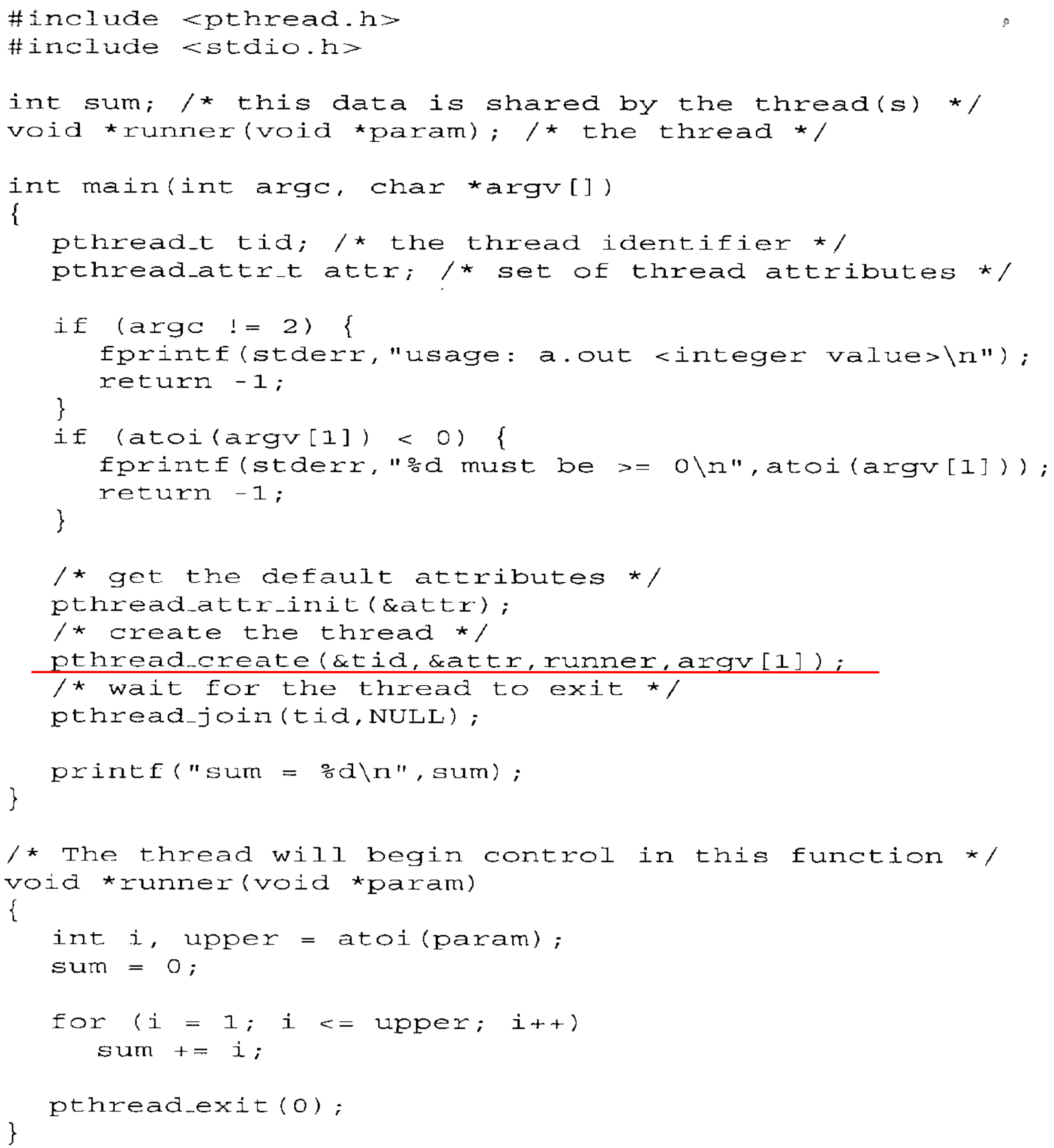
- What if there were 2 I/O-bound threads and ~~4~~ 2 CPU-bound threads?

Ans：2顆
因為只有2個CPU threads→只有2個threads要跑CPU
kernel threads再多都沒用(只需要跑兩個)

THREAD LIBRARIES

Pthread

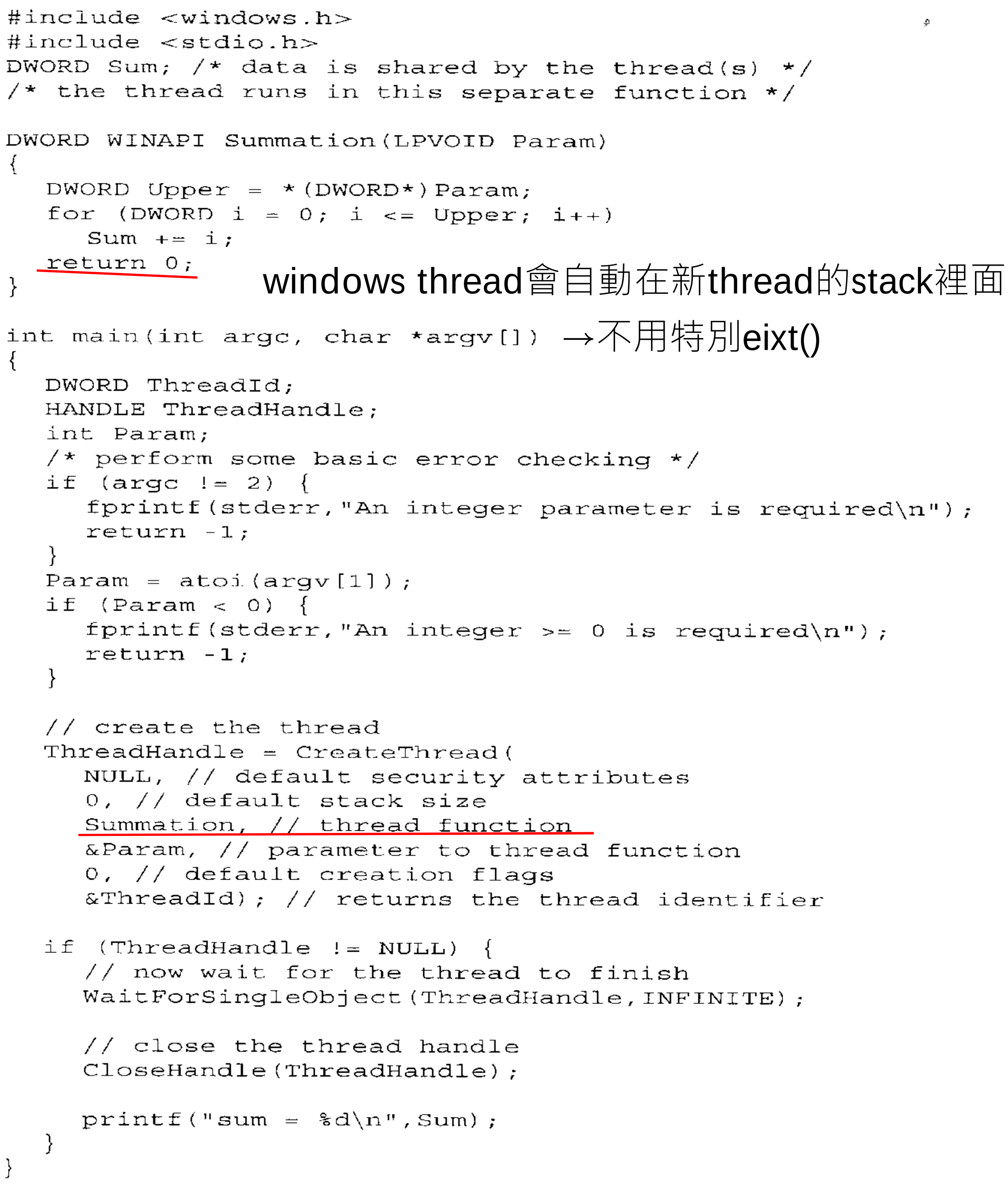
- =POSIX thread
- Pthread is a “specification”, not an implementation
- Implementations:
  - GNU portable thread
  - Linux Pthread
  - Mac OS X Pthread



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

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

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);
}
```

Win32 Thread

- Again Win32 thread is a specification, implementation varies among WinXP, Win7, etc.
- Win32 thread APIs are very similar to Pthread APIs
- Win32 threads are referred to as objects/handle
    - WaitForSingleObject
    - CloseHandle



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

DWORD WINAPI Summation(LPVOID Param)
{
   DWORD Upper = *(DWORD*)Param;
   for (DWORD i = 0; i <= Upper; i++)
      Sum += i;
   return 0;
}

int main(int argc, char *argv[])
{
   DWORD ThreadId;
   HANDLE ThreadHandle;
   int Param;
   /* perform some basic error checking */
   if (argc != 2) {
      fprintf(stderr,"An integer parameter is required\n");
      return -1;
   }
   Param = atoi(argv[1]);
   if (Param < 0) {
      fprintf(stderr,"An integer >= 0 is required\n");
      return -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

   if (ThreadHandle != NULL) {
      // now wait for the thread to finish
      WaitForSingleObject(ThreadHandle,INFINITE);

      // close the thread handle
      CloseHandle(ThreadHandle);

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

windows thread會自動在新thread的stack裡面加一個中斷點

→不用特別eixt()

OPERATING-SYSTEM THREAD SUPPORT

Windows XP Threads

- Implements the one-to-one mapping
- Each thread contains
  - A thread id
  - Register set
  - Separate user and kernel stacks
  - Private data storage area
- The register set, stacks, and private storage area are known as the context of the threads
- The primary data structures of a thread include:
  - ETHREAD (executive thread block)
  - KTHREAD (kernel thread block)
  - TEB (thread environment block)

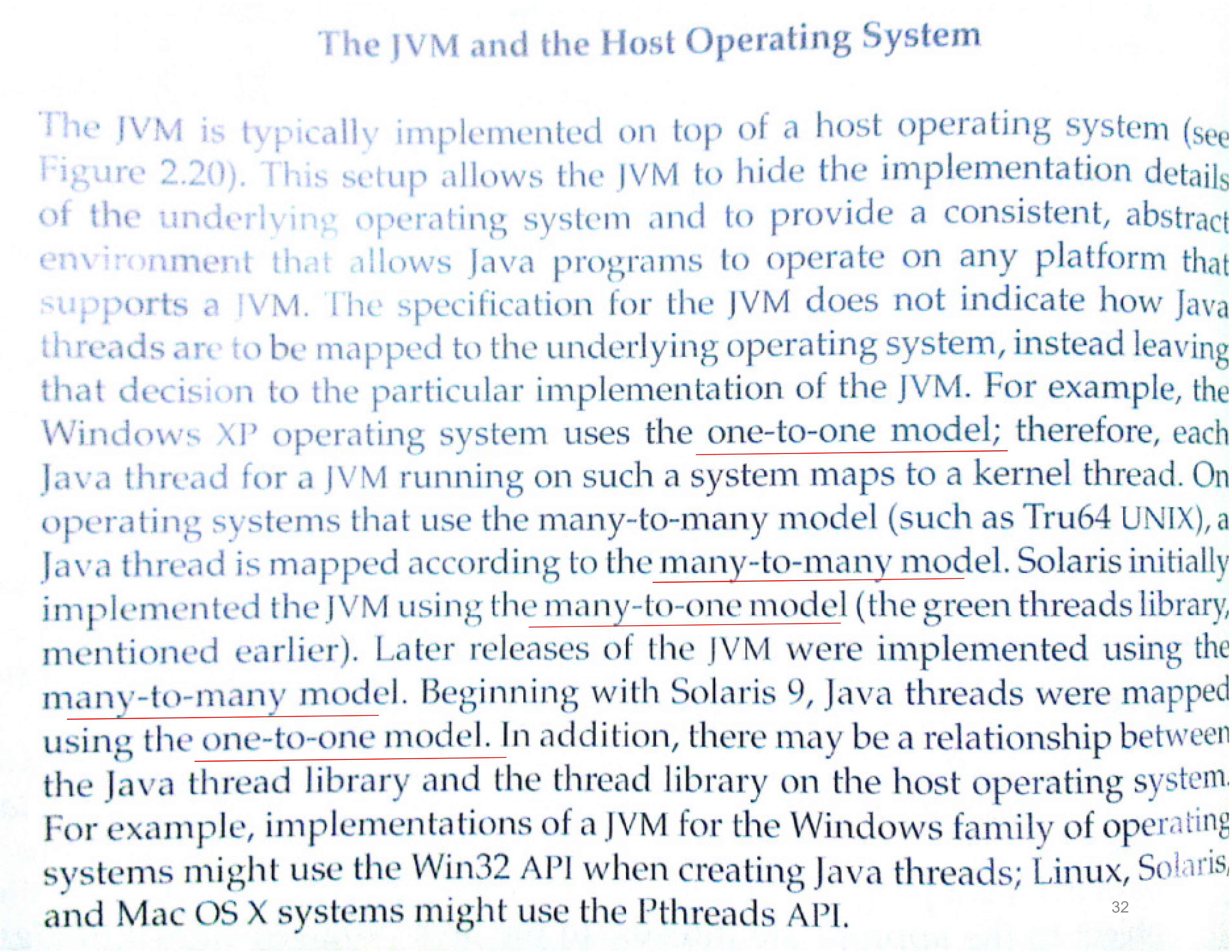
Linux Threads

- 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)
    - But the stacks are separate
    - Different form vfork()

Java Threads

- Java threads are managed by the JVM

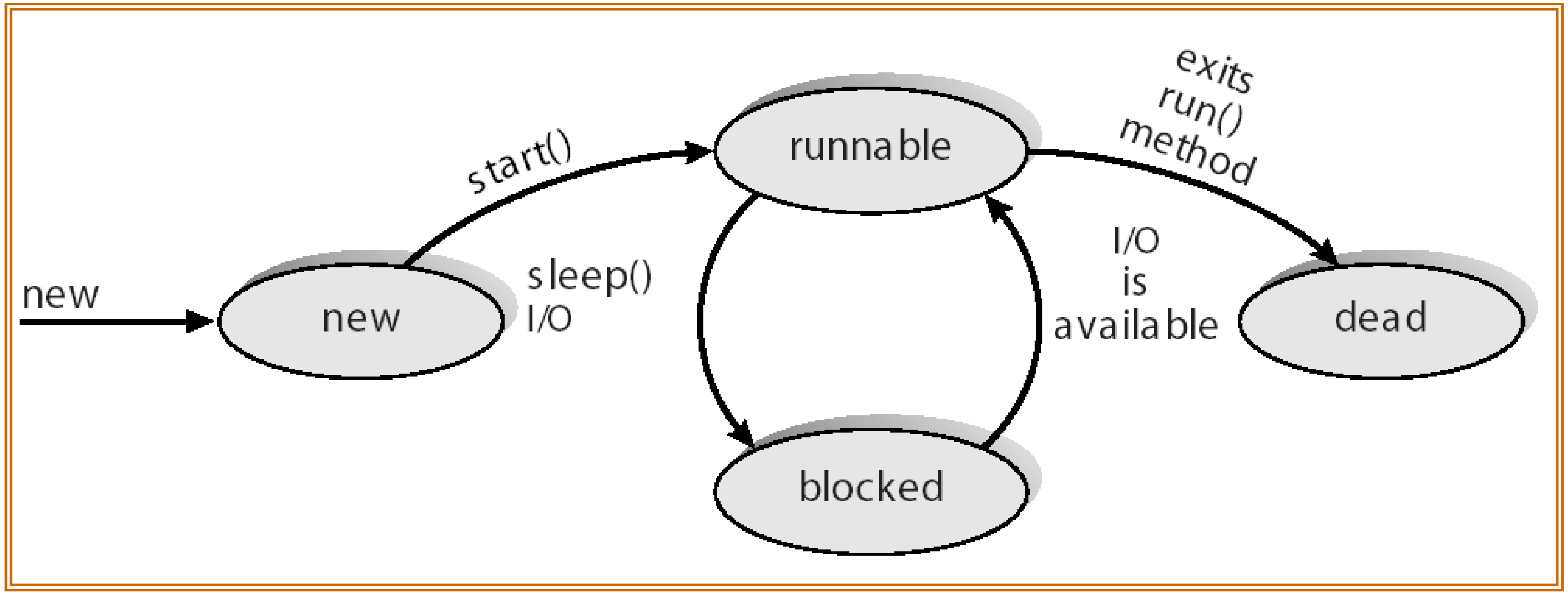
- Java threads may be created by:
  - Extending Thread class
  - Implementing the Runnable interface



The JVM and the Host Operating System

The JVM is typically implemented on top of a host operating system (see Figure 2.20). This setup allows the JVM to hide the implementation details of the underlying operating system and to provide a consistent, abstract environment that allows Java programs to operate on any platform that supports a JVM. The specification for the JVM does not indicate how Java threads are to be mapped to the underlying operating system, instead leaving that decision to the particular implementation of the JVM. For example, the Windows XP operating system uses the one-to-one model; therefore, each Java thread for a JVM running on such a system maps to a kernel thread. On operating systems that use the many-to-many model (such as Tru64 UNIX), a Java thread is mapped according to the many-to-many model. Solaris initially implemented the JVM using the many-to-one model (the green threads library, mentioned earlier). Later releases of the JVM were implemented using the many-to-many model. Beginning with Solaris 9, Java threads were mapped using the one-to-one model. In addition, there may be a relationship between the Java thread library and the thread library on the host operating system. For example, implementations of a JVM for the Windows family of operating systems might use the Win32 API when creating Java threads; Linux, Solaris, and Mac OS X systems might use the Pthreads API.

Java Thread States



THREADING ISSUES

Semantics of fork() and exec()

- Does fork() duplicate only the calling thread or all threads?
  - Undefined, but in many UNIX variants the entire process (including all its threads) is duplicated
- How about exec()?
  - Again undefined, but in many UNIX variants the entire process (including all its threads) is terminated

Signal Handling

- Synchronous signal SIGNAL只會送給造成SIG的thread
  - Always delivered to the thread that causes the signal
  - E.g., access violation
- Asynchronous signal
  - Typically delivered to the first thread that dose not block the signal
  - E.g., process termination
- Varies from implementation to implementation

會根據實作變化

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
    - It is costly to repeatedly create and delete threads
  - Allows the number of threads in the application(s) to be bound to the size of the pool
    - Multiplexing tasks over threads, similar to the concept of the many-to-many model

End of Chapter 4