ESTGOH
Escola Superior de Tecnologia e Gestão de Oliveira do Hospital

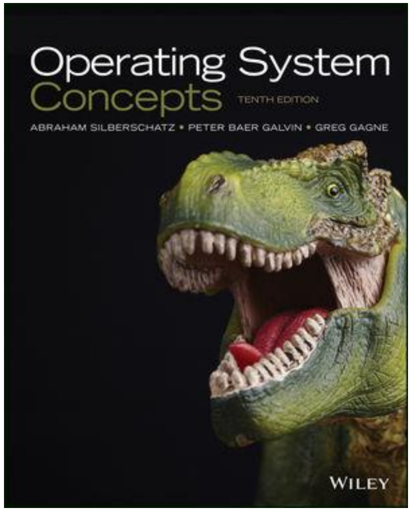
Instituto Politécnico de Coimbra

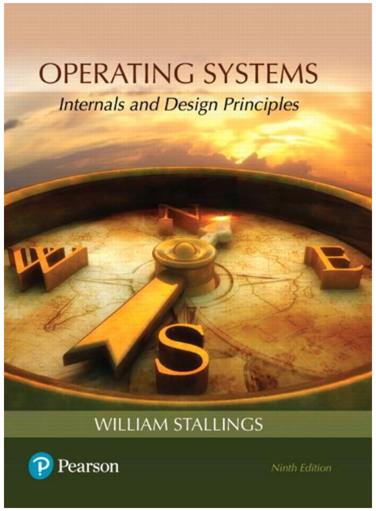
Sistemas Operativos

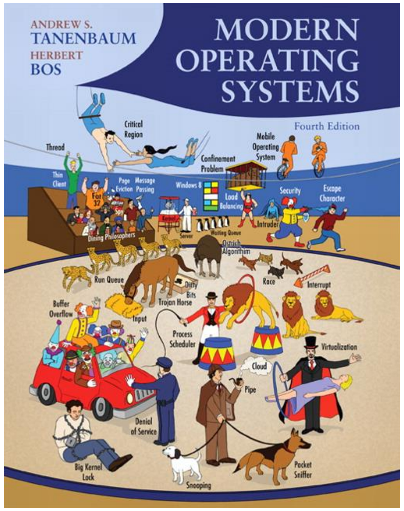
Threads

Disclaimer

- These slides and notes are based on the contents of the books:
  - Abraham Silberschatz, ” Operating System Concepts”, 10th Edition, Wiley, 2018;
  - William Stallings, “Operating Systems: Internals and Design Principles”, 9th Edition, Pearson, 2017;
  - Andrew S. Tanenbaum, “Modern Operating Systems”, 4th Edition, Pearson Education, 2014;
- The respective copyrights belong to their owners.







Outline

- **Motivation**
- **Thread Concept**
  - Examples
  - Benefits
- **Thread Fields**
  - Program counter
  - Registers
  - Stack
  - State
- **Multithreading Models**
  - Many-to-one Model (M-to-1)
  - One-to-one Model (1-to-1)
  - Many-to-many Model (M-to-N)
  - Two-level Model
- **Threading Libraries**
  - Pthreads, Win32, Java
- **Pthreads**
  - Management
    - *int pthread_create()*
    - *void pthread_exit()*
    - *int pthread_join()*
    - *int pthread_cancel()*
  - Compilation
    - *-D_REENTRANT*
    - *-lpthread*
- **References**

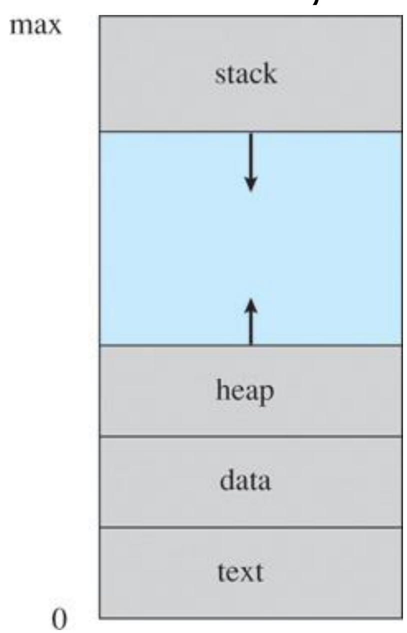
Threads

- **Motivation**
  - Process switching is an extremely heavy operation
    - necessary to remap address spaces
    - necessary to manage all information associated to processes
  - **Thread**
    - It's a flow of execution inside a program
    - The address space is the same (changing a global variable in a thread affects all others)
    - No context switching between processes
    - Commutation among threads is **very fast**
    - Communication among threads is easy to do and fast

Outline

- **Motivation**
- **Thread Concept**
  - Examples
  - Benefits
- **Thread Fields**
  - Program counter
  - Registers
  - Stack
  - State
- **Multithreading Models**
  - Many-to-one Model (M-to-1)
  - One-to-one Model (1-to-1)
  - Many-to-many Model (M-to-N)
  - Two-level Model
- **Threading Libraries**
  - Pthreads, Win32, Java
- **Pthreads**
  - Management
    - *int pthread_create()*
    - *void pthread_exit()*
    - *int pthread_join()*
    - *int pthread_cancel()*
  - Compilation
    - *-D_REENTRANT*
    - *-lpthread*
- **References**

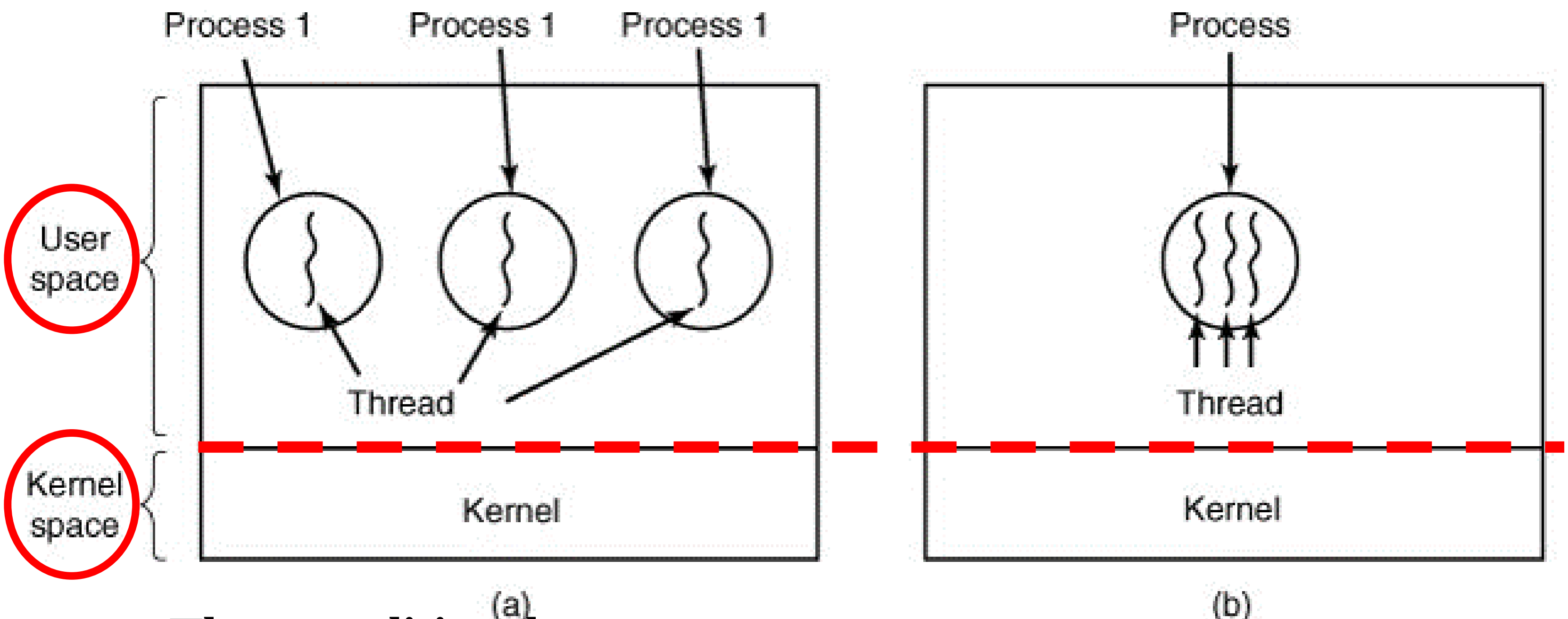
- **Thread Concept**
  - One way of looking at a **process** is that it is a <u>way to group related resources together</u>.
  - A process has an address space containing:
    - <u>program text and data</u>
    - other resources such as open files, child processes, pending alarms, signal handlers, accounting information, and more



process stack - which contains <u>temporary data</u> (such as function parameters, return addresses, and <u>local variables</u>)
data section - which contains <u>global variables</u>
heap - which is <u>memory that is dynamically</u> allocated during process run time (e.g. as a result of *malloc()*).
text - where the code of the program goes

- **Thread Concept**
  - The other concept a **process** has is a thread of execution which contains a:
    - a program counter (that keeps track of which instruction to execute next)
    - registers (which hold its current working variables).
    - a stack (which contains the execution history with one frame for each procedure called but not yet returned from).
  - Although a thread must execute in some process, the thread and its process are different concepts and can be treated separately.
  - **Processes are used to group resources together; threads are the entities scheduled for execution on the CPU**.

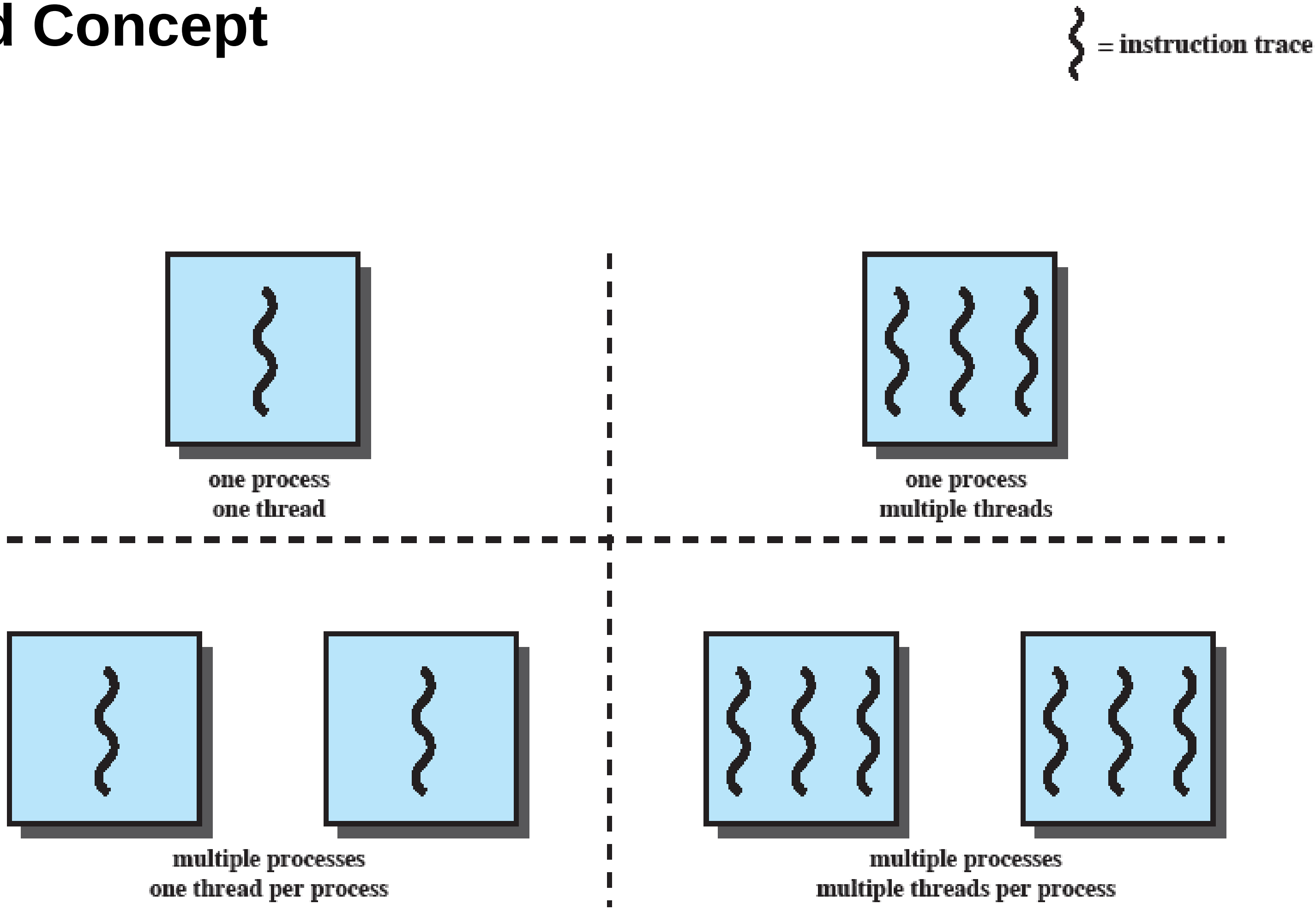
- **Thread Concept**
  - threads to allow multiple executions to take place in the same process environment, to a large degree independent of one another.



Three traditional processes. Each process has its own address space and a single thread of control. Each of the threads operates in a different address space.

Single process with three threads of control. All three threads share the same address space

- **Thread Concept**



William Stallings, "Operating Systems: Internals and Design Principles"

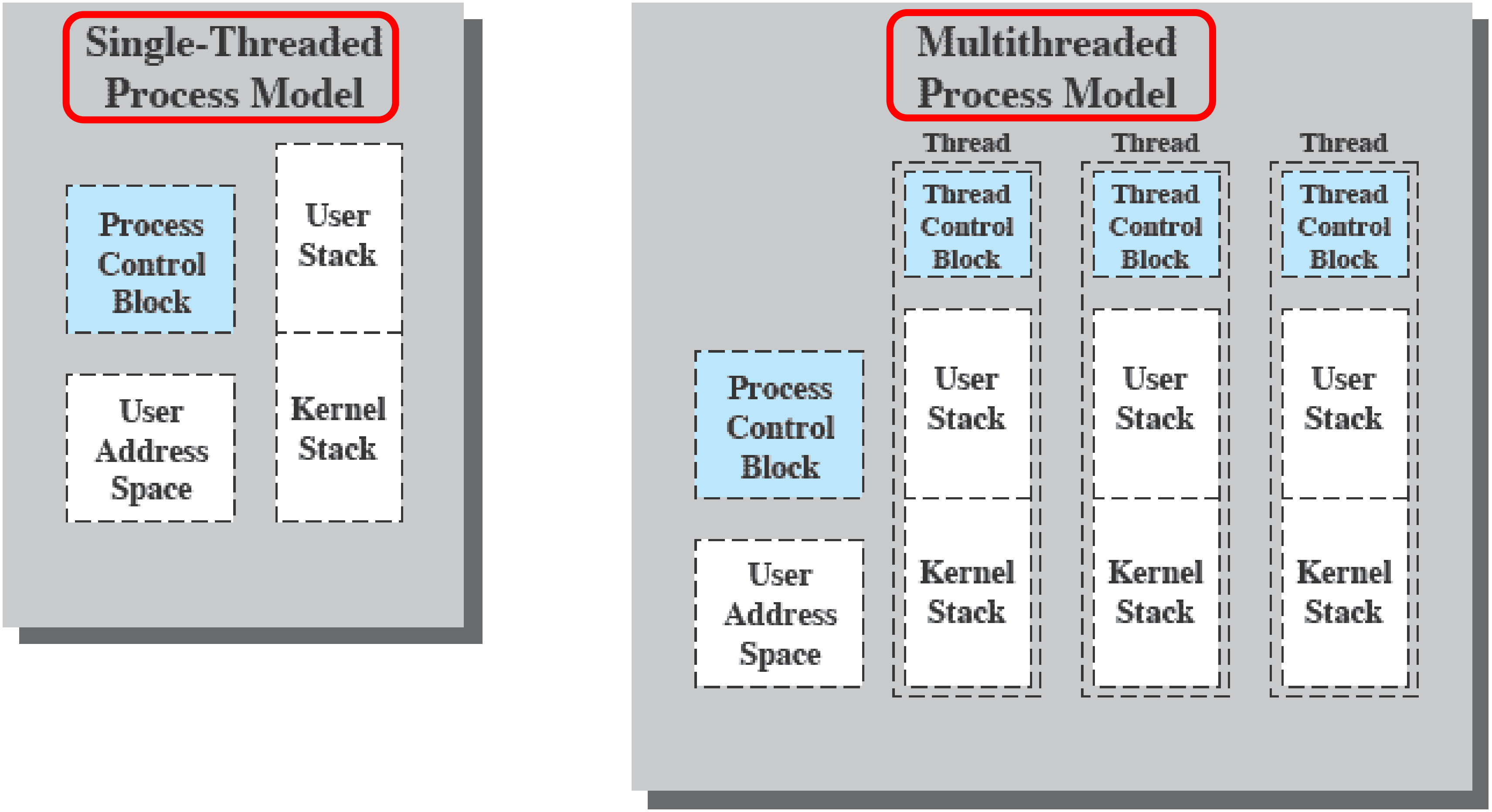
• Thread Concept

Threads:Processes	Description	Example Systems
1:1	Each thread of execution is a unique process with its own address space and resources.	Traditional UNIX implementations
M:1	A process defines an address space and dynamic resource ownership. Multiple threads may be created and executed within that process.	Windows NT, Solaris, Linux, OS/2, OS/390, MACH

William Stallings, “Operating Systems: Internals and Design Principles”

- **Thread Concept**

One way to view a **thread** is as an independent program counter operating within a process.

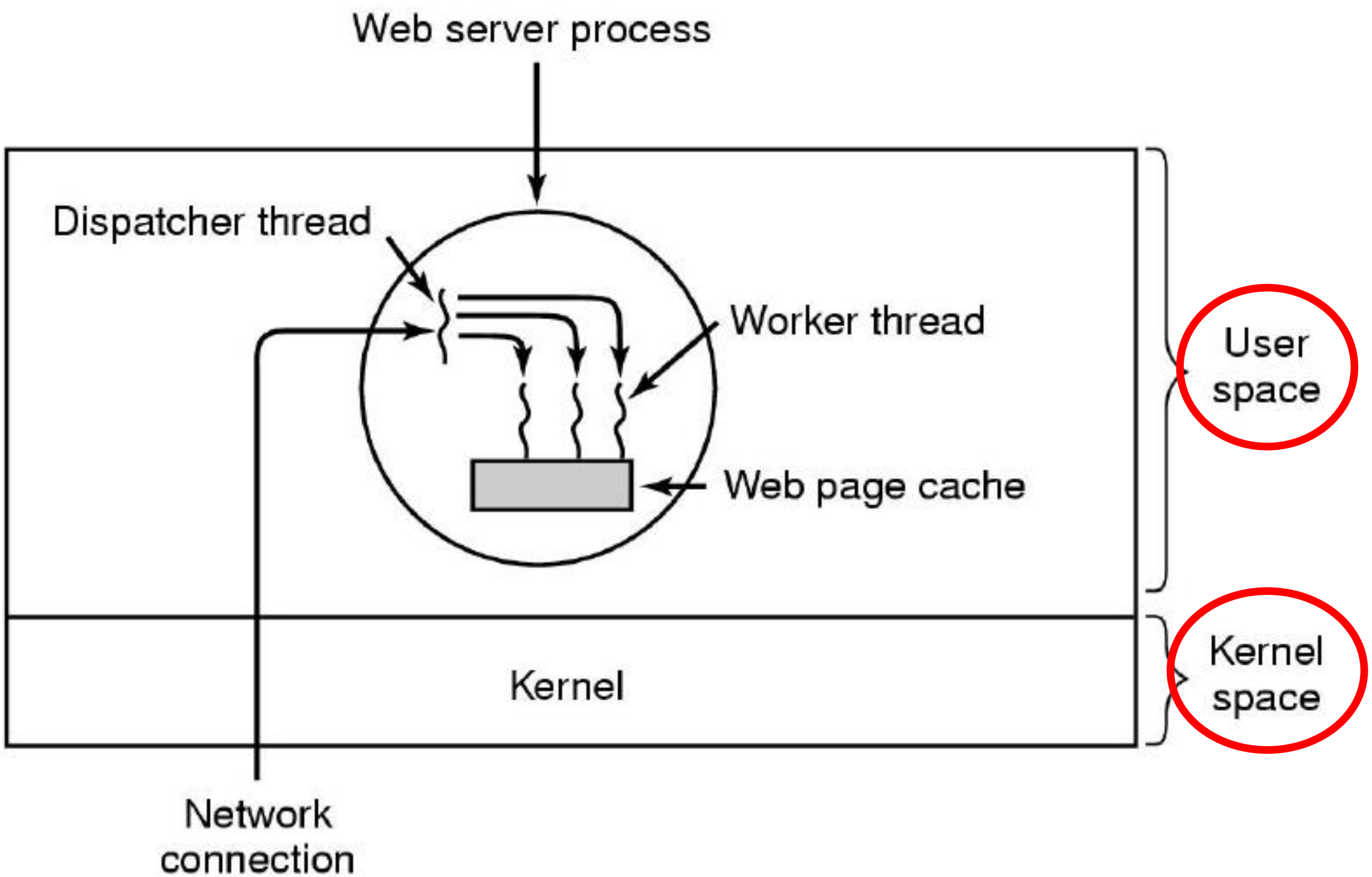


William Stallings, "Operating Systems: Internals and Design Principles"

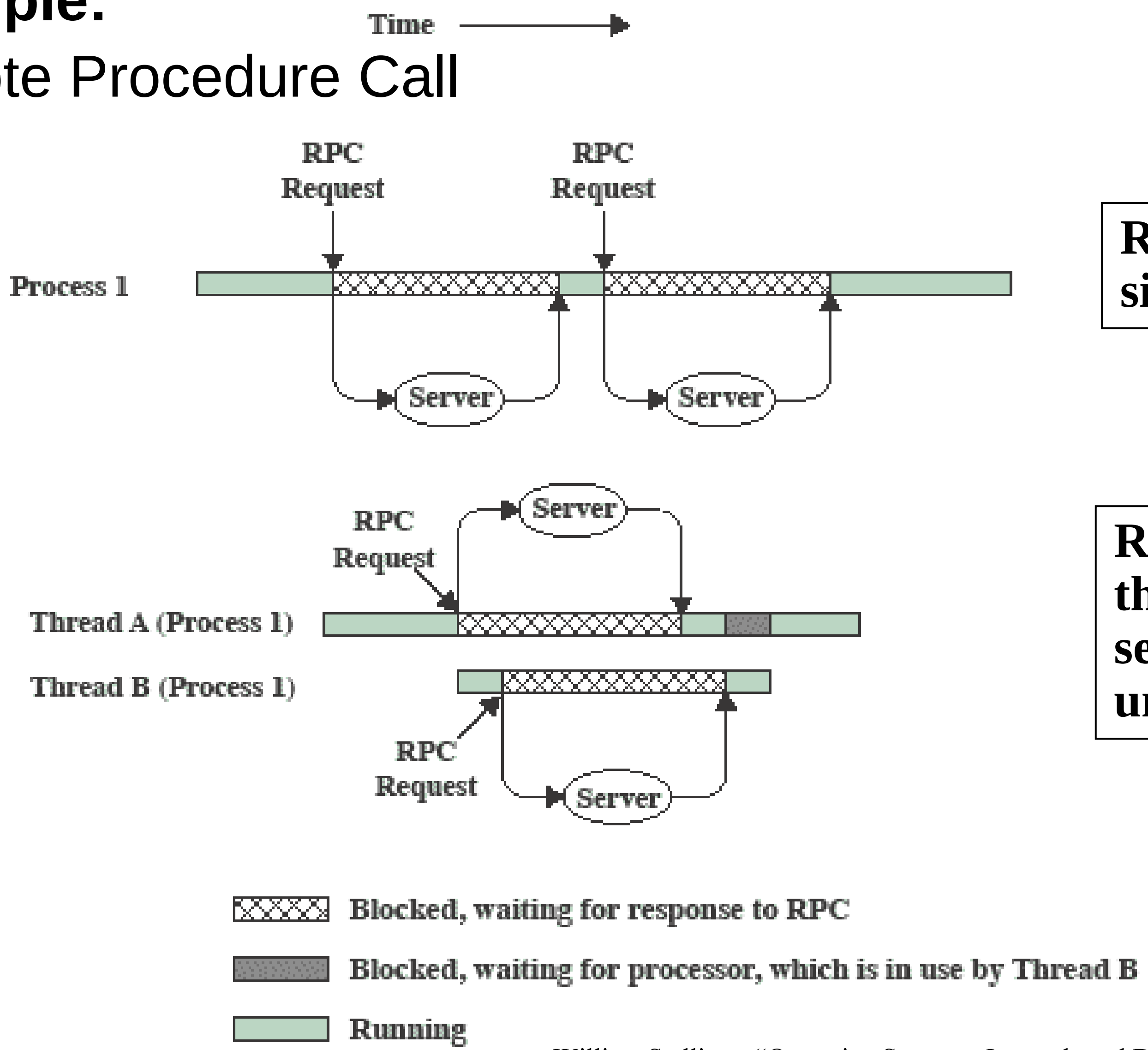
Threads

- **Example:**
- Web Browser:
  - web pages may contain multiple small images.
  - For each image on a web page, the browser must set up a separate connection to the page's home site and request the image. A great deal of time is spent establishing and releasing all these connections.
  - By having multiple threads within the browser, many images can be requested at the same time, greatly speeding up performance in most cases since with small images, the set-up time is the limiting factor, not the speed of the transmission line.

- **Example:**
- Web Server:



- **Example:**
- Remote Procedure Call



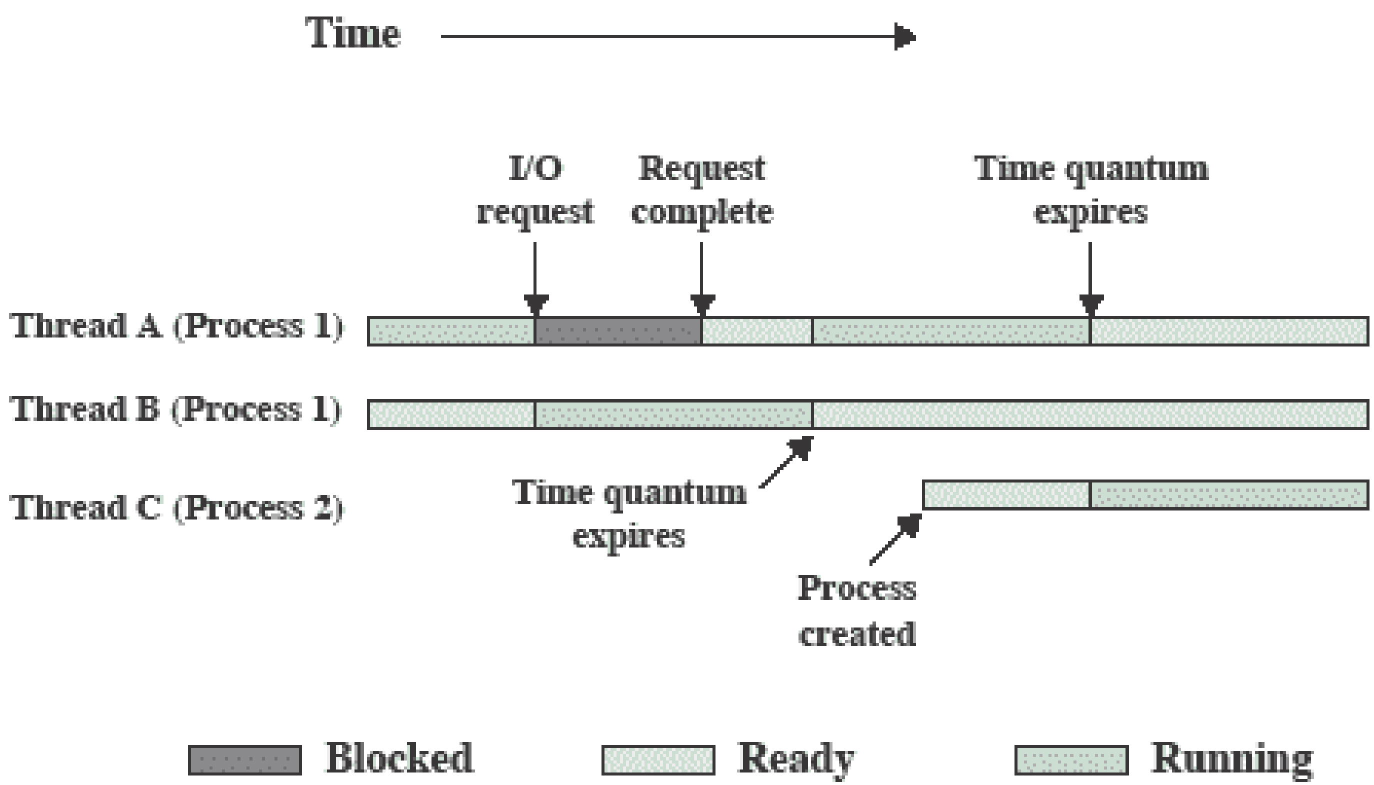
RPC using single thread

RPC using one thread per server (on a uniprocessor)

William Stallings, "Operating Systems: Internals and Design Principles"

Threads

- **Example:**



Multithreading on a uniprocessor

William Stallings, "Operating Systems: Internals and Design Principles"

Threads

- **Benefits**
  - **very light weight compared to processes:**
    - Light context switches
    - Fast to create and terminate
    - Fast to synchronize
    - Share resources
  - **Responsiveness:** multithreading may allow a <u>program to continue running even if part of it is blocked or is performing a lengthy operation</u>.
  - **Resource sharing/economy:** <u>threads share the memory and the resources of the process to which they belong</u>.
  - **Utilization of multiprocessor architectures**: threads may be running in parallel on different processors.

Outline

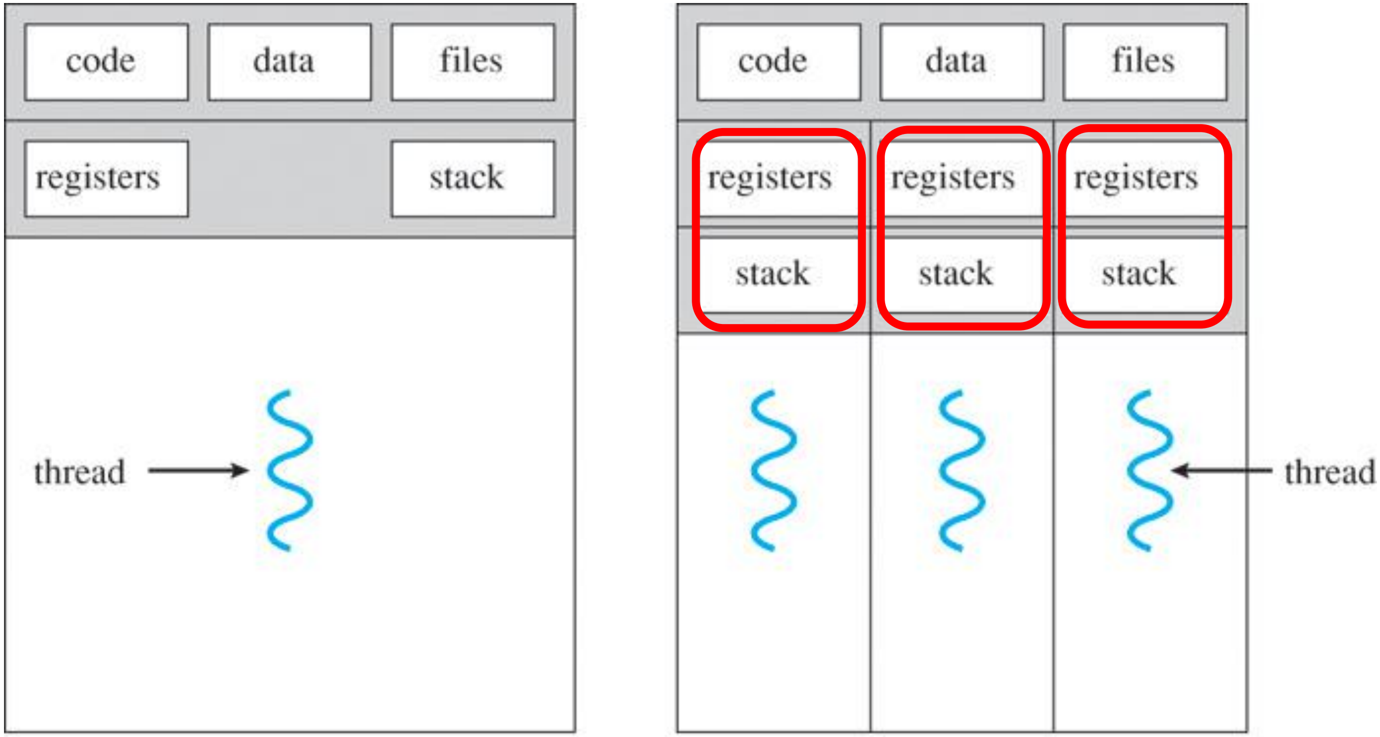
- **Motivation**
- **Thread Concept**
  - Examples
  - Benefits
- **Thread Fields**
  - Program counter
  - Registers
  - Stack
  - State
- **Multithreading Models**
  - Many-to-one Model (M-to-1)
  - One-to-one Model (1-to-1)
  - Many-to-many Model (M-to-N)
  - Two-level Model
- **Threading Libraries**
  - Pthreads, Win32, Java
- **Pthreads**
  - Management
    - *`int pthread_create()`*
    - *`void pthread_exit()`*
    - *`int pthread_join()`*
    - *`int pthread_cancel()`*
  - Compilation
    - *`-D_REENTRANT`*
    - *`-lpthread`*
- **References**

- **Thread Fields**
  - When multiple threads are present in the same address space, some fields are not per process, but per thread, so a separate thread table is needed, with one entry per thread.
  - Among the **per-thread items:**
    - program counter needed because threads, like processes, can be suspended and resumed
    - registers needed because when threads are suspended, their registers must be saved
    - stack to store local variables
    - state because threads, like processes, can be in running, ready, or blocked state.

- **Thread Fields**

Per process items	Per thread items
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

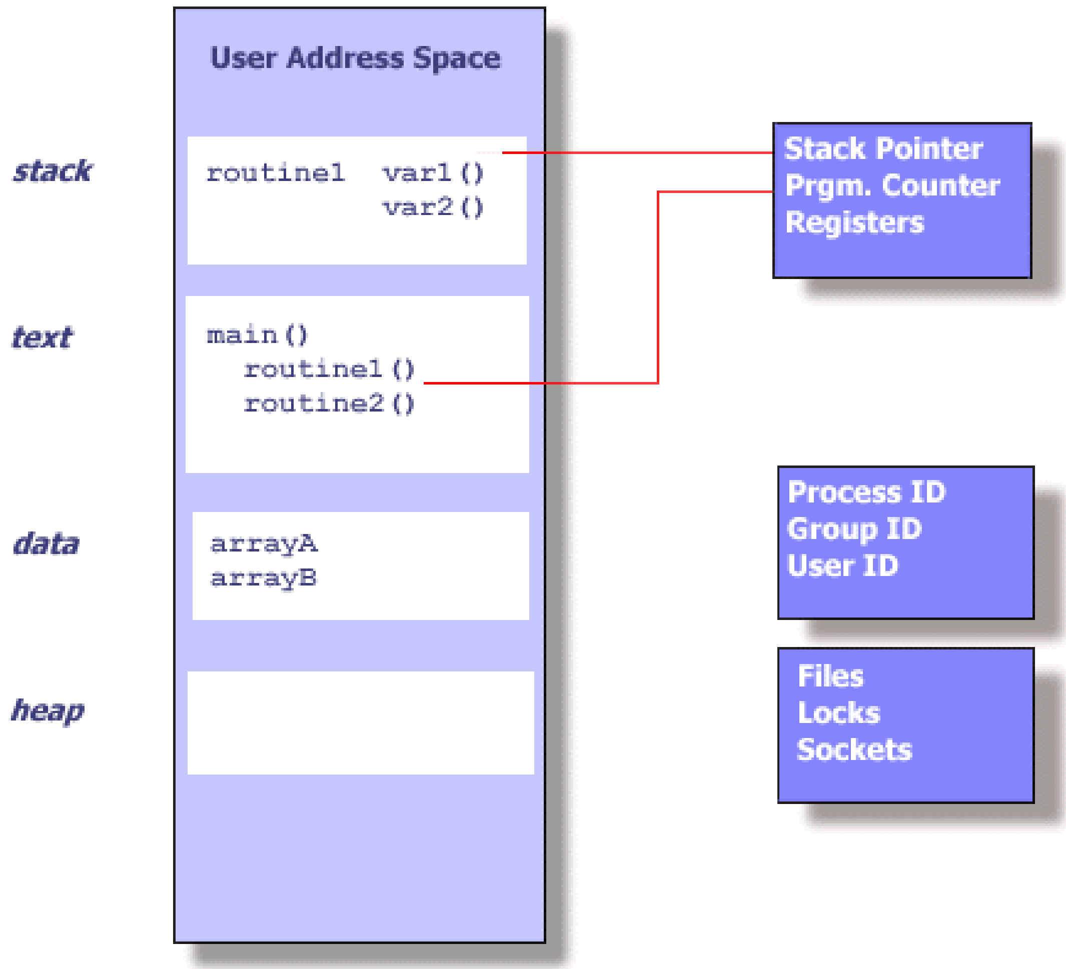
- **Thread Fields**
  - **Threads share the same address space and resources**! <u>Changing one thing in one thread affects all others!</u>
  - It is the responsibility of the programmer to assure the correctness in the concurrent access to data and resources!



single-threaded process
multithreaded process

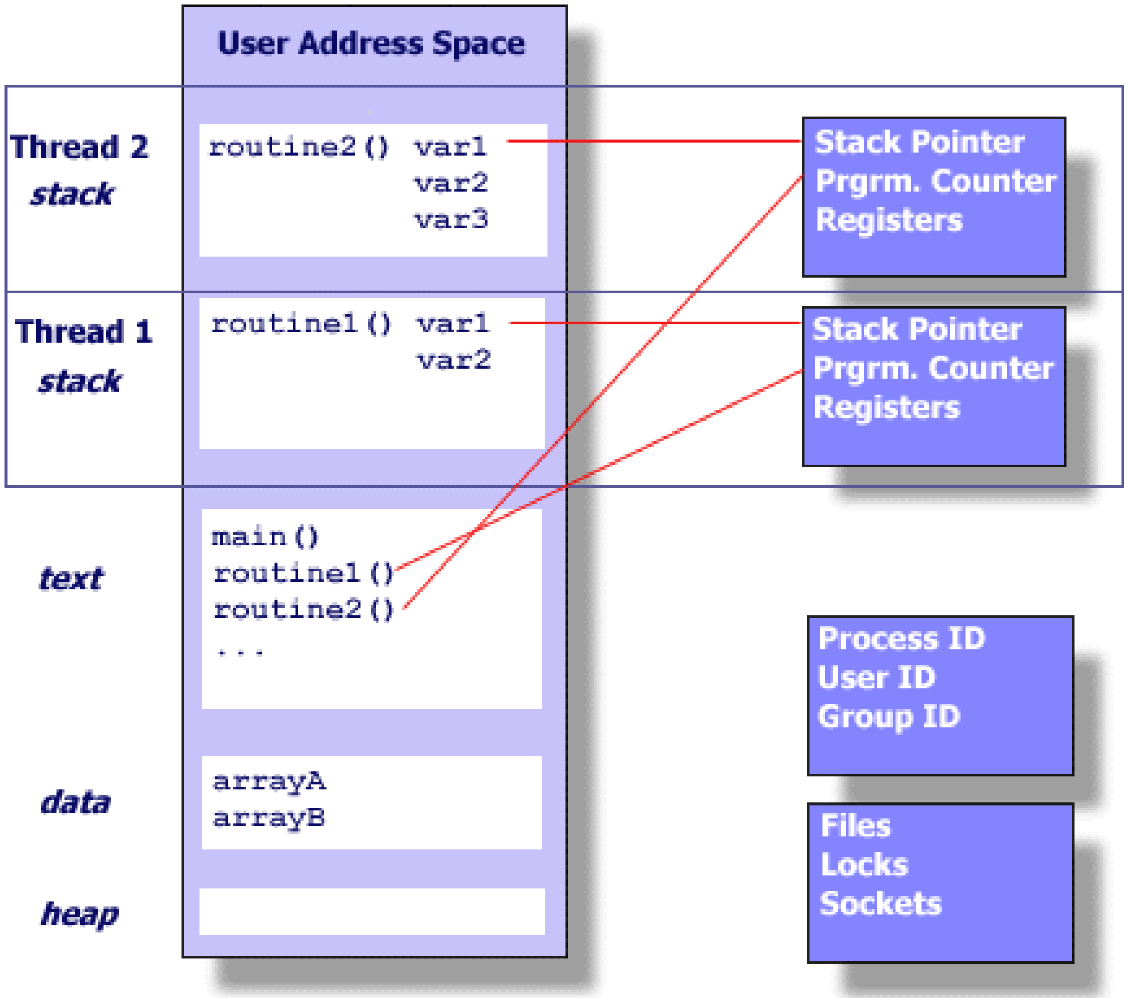
Abraham Silberschatz, " Operating System Concepts"

- **Thread Fields**



https://computing.llnl.gov/tutorials/pthreads/

- **Thread Fields**

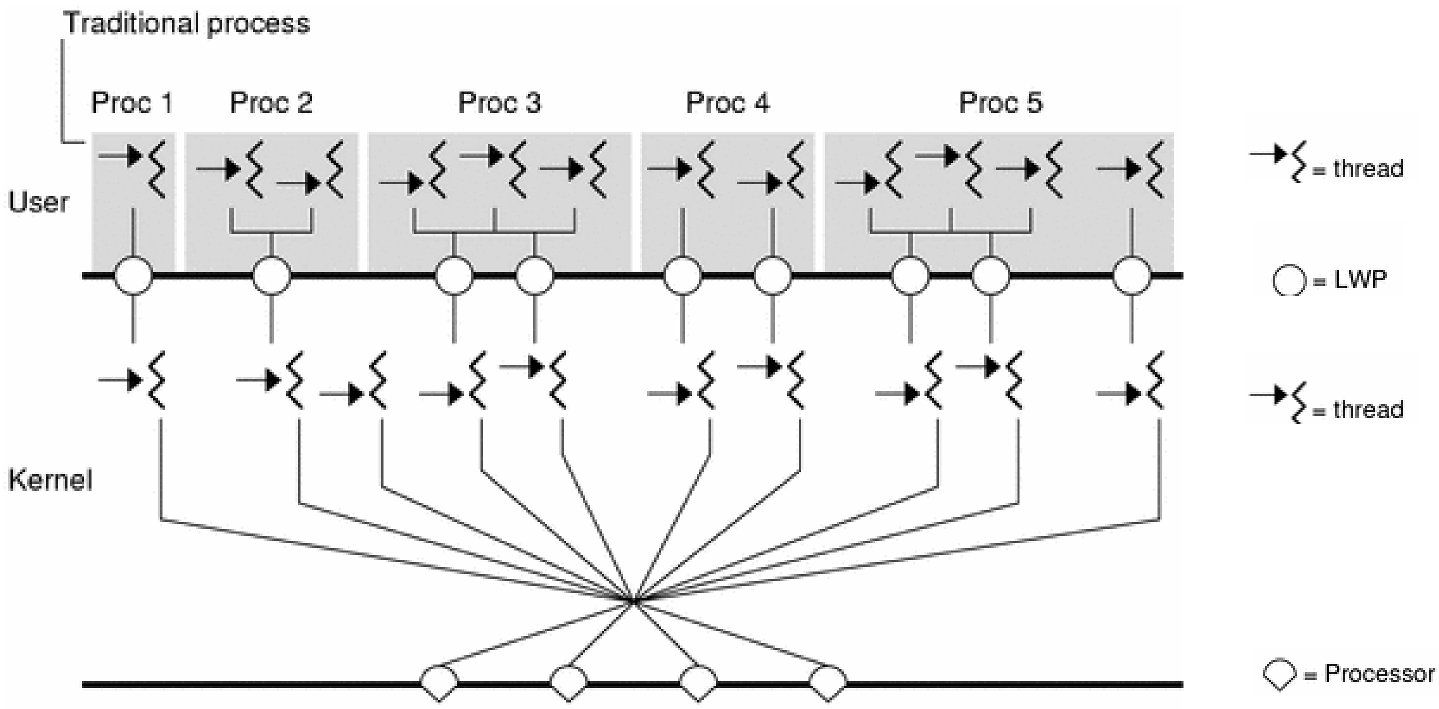


https://computing.llnl.gov/tutorials/pthreads/

Outline

- **Motivation**
- **Thread Concept**
  - Examples
  - Benefits
- **Thread Fields**
  - Program counter
  - Registers
  - Stack
  - State
- **Multithreading Models**
  - Many-to-one Model (M-to-1)
  - One-to-one Model (1-to-1)
  - Many-to-many Model (M-to-N)
  - Two-level Model
- **Threading Libraries**
  - Pthreads, Win32, Java
- **Pthreads**
  - Management
    - *`int pthread_create()`*
    - *`void pthread_exit()`*
    - *`int pthread_join()`*
    - *`int pthread_cancel()`*
  - Compilation
    - *`-D_REENTRANT`*
    - *`-lpthread`*
- **References**

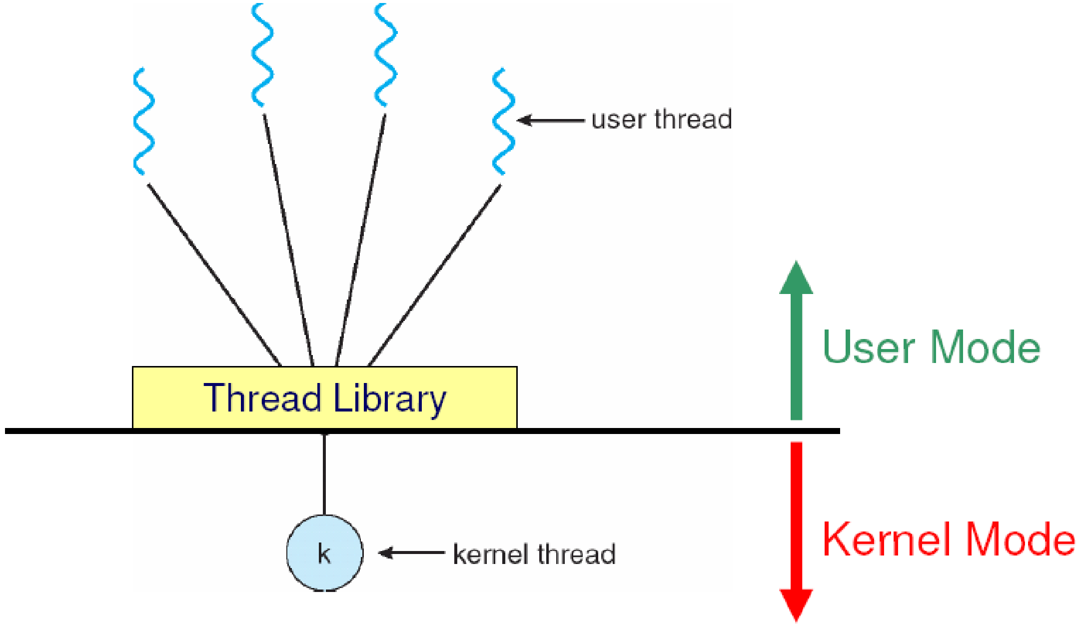
- **Multithreading Models**
  - Support for threads may be provided either at the user level, for user threads, or by the kernel, for kernel threads.
    - **User threads** are supported above the kernel and are managed without kernel support
    - **Kernel threads** are supported and managed directly by the operating system.



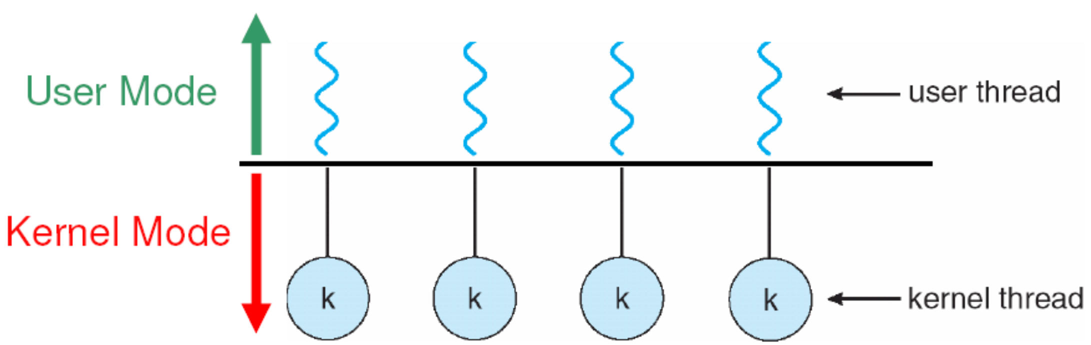
Threads

- **Multithreading Models**
  - There must exist a relationship between user threads and kernel threads
    - **Many-to-one Model (M-to-1)**:
      - maps many user-level threads to one kernel thread.
    - **One-to-one Model (1-to-1)**:
      - maps each user thread to a kernel thread.
    - **Many-to-many Model (M-to-N)**:
      - multiplexes many user-level threads to a smaller or equal number of kernel threads.
    - **Two-level Model:**
      - multiplexes many user-level threads to a smaller or equal number of kernel threads but also allows a user-level thread to be bound to a kernel thread.

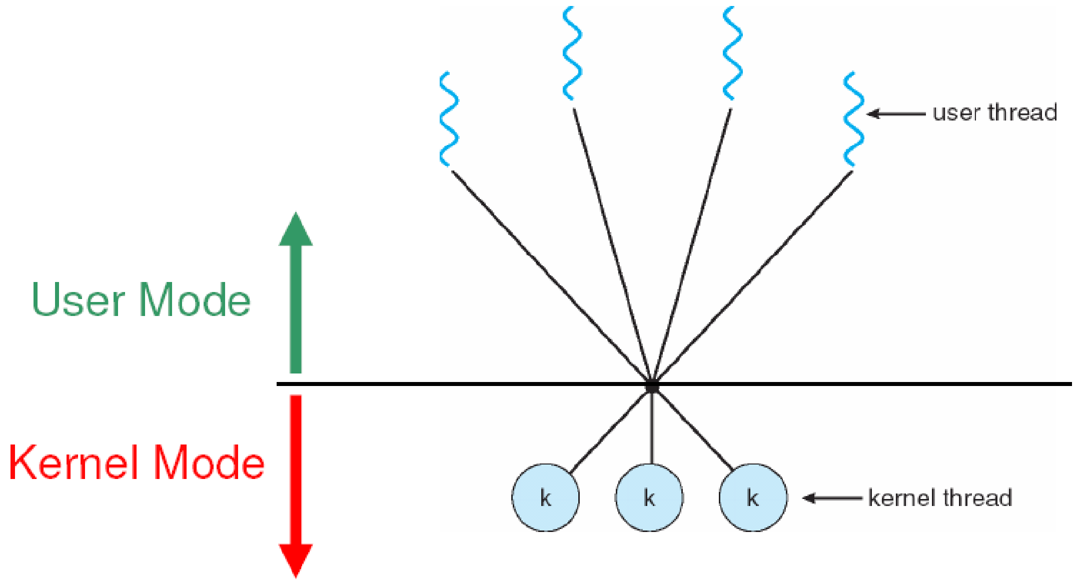
- **Multithreading Models**
  - **Many-to-one Model (M-to-1 model)**
    - Thread management is done by the thread library in user space, so it is efficient
    - The kernel is completely unaware of the existence of threads
    - The entire process will block if a thread makes a blocking system call.
    - Because only one thread can access the kernel at a time, multiple threads are unable to run in parallel on multiprocessors.



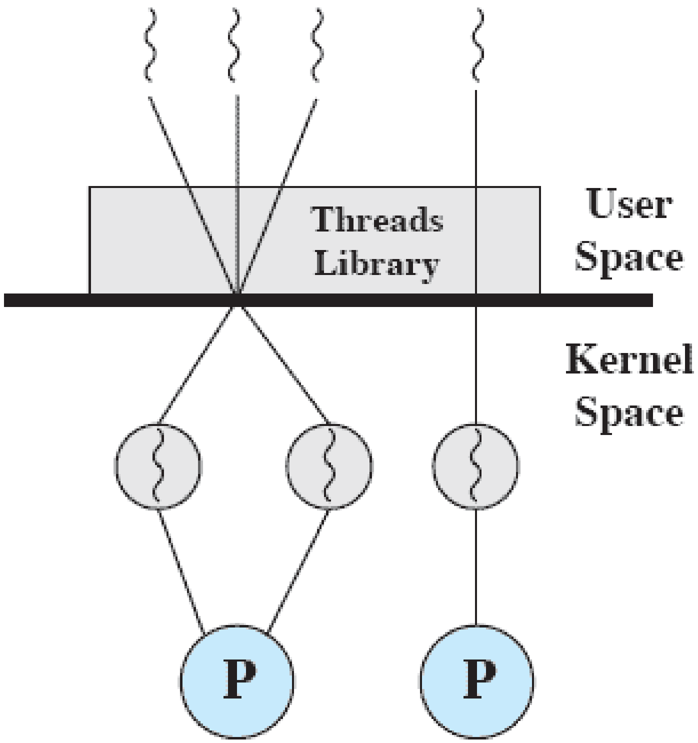
- **Multithreading Models**
  - **One-to-one Model (1-to-1 model)**
    - Threads are implemented exclusively in kernel space
    - The kernel does all the scheduling of threads
    - It provides more concurrency than the many-to-one model by allowing another thread to run when a thread makes a blocking system call
    - Allows multiple threads to run in parallel on multiprocessors.
    - The only drawback to this model is that creating a user thread requires creating the corresponding kernel thread.



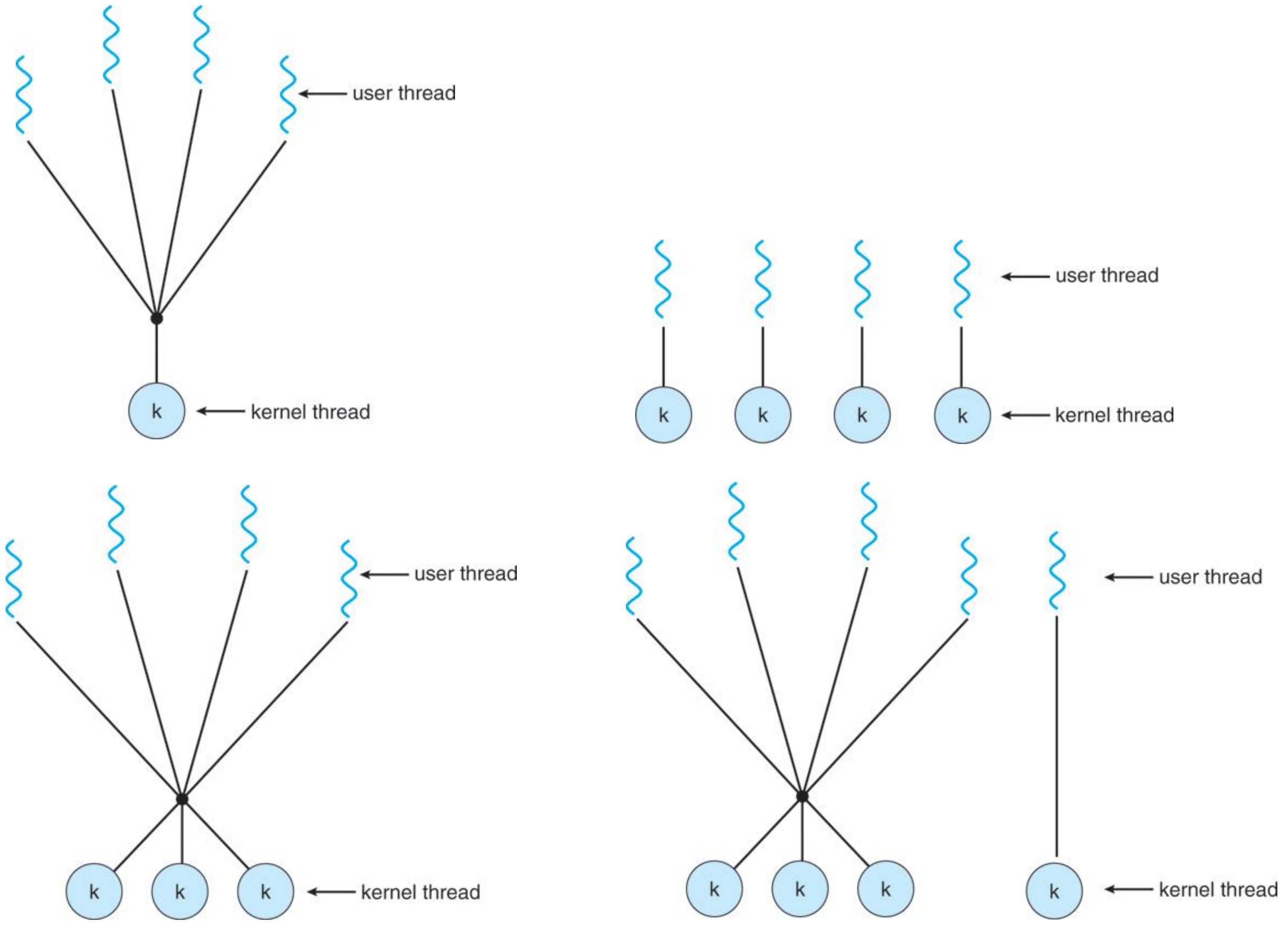
- **Multithreading Models**
  - **Many-to-many Model (M-to-N model)**
    - A number of kernel threads can map to a different number of user threads.
    - The number of kernel threads may be specific to either a particular application or a particular machine (an application may be allocated more kernel threads on a multiprocessor than on a uniprocessor).
    - When a thread performs a blocking system call, the kernel can schedule another thread for execution.



- **Multithreading Models**
  - **Two level model**
    - multiplexes many user-level threads to a smaller or equal number of kernel threads, but also allows a user-level thread to be bound to a kernel thread



- **Multithreading Models**



Outline

- **Motivation**
- **Thread Concept**
  - Examples
  - Benefits
- **Thread Fields**
  - Program counter
  - Registers
  - Stack
  - State
- **Multithreading Models**
  - Many-to-one Model (M-to-1)
  - One-to-one Model (1-to-1)
  - Many-to-many Model (M-to-N)
  - Two-level Model
- **Threading Libraries**
  - Pthreads, Win32, Java
- **Pthreads**
  - Management
    - *int pthread_create()*
    - *void pthread_exit()*
    - *int pthread_join()*
    - *int pthread_cancel()*
  - Compilation
    - *-D_REENTRANT*
    - *-lpthread*
- **References**

- **Threading Libraries**
  - A thread library provides the programmer an API for creating and managing threads. There are two primary ways of implementing a thread library:
    - to provide <u>a library entirely in user space</u> with no kernel support. All code and data structures for the library exist in user space. This means that invoking a function in the library results in a local function call in user space and not a system call.
    - to implement <u>a kernel-level library</u> supported directly by the operating system. In this case, code and data structures for the library exist in kernel space. Invoking a function in the API for the library typically results in a system call to the kernel.

Threads

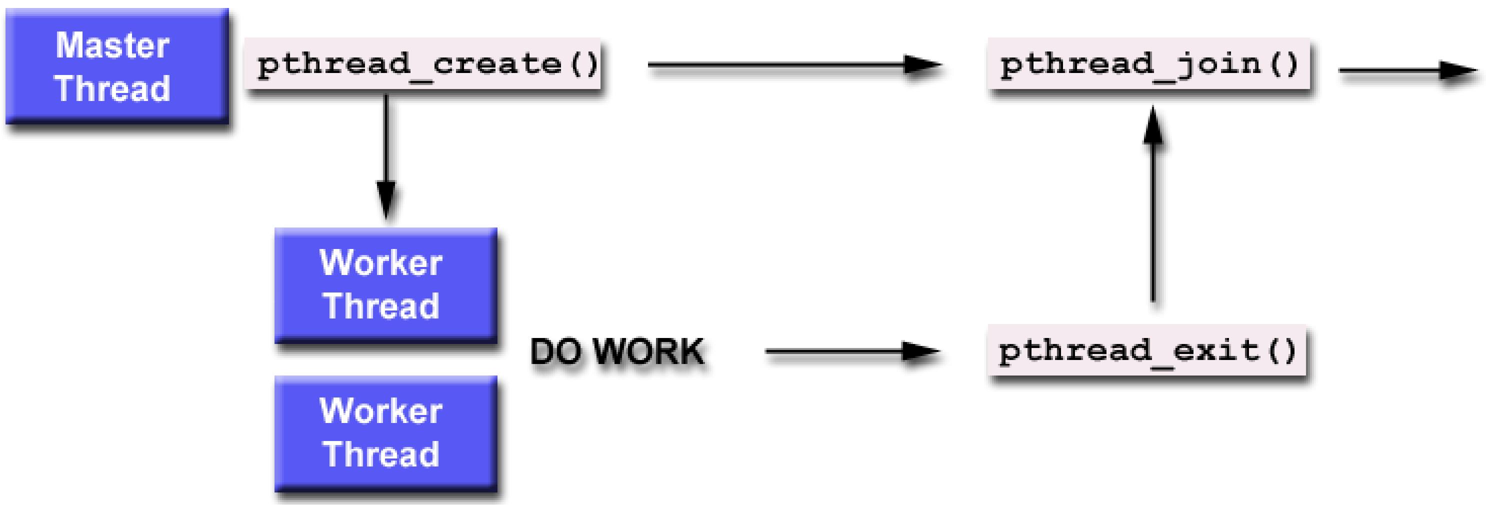
- **Threading Libraries**
  - Three main thread libraries are in use:
    - **POSIX Threads (Pthreads)**: the threads extension of the POSIX standard, may be provided as either a user- or kernel-level library.
    - **Win32 Threads**: kernel-level library available on Windows systems.
    - **Java Threads**: because in most instances the JVM is running on top of a host operating system, the Java thread API is typically implemented using a thread library available on the host system.
      - **Windows systems:**
        - typically implemented using the Win32 API;
      - **UNIX and Linux systems:**
        - often use Pthreads

Outline

- **Motivation**
- **Thread Concept**
  - Examples
  - Benefits
- **Thread Fields**
  - Program counter
  - Registers
  - Stack
  - State
- **Multithreading Models**
  - Many-to-one Model (M-to-1)
  - One-to-one Model (1-to-1)
  - Many-to-many Model (M-to-N)
  - Two-level Model
- **Threading Libraries**
  - Pthreads, Win32, Java
- **Pthreads**
  - Management
    - *`int pthread_create()`*
    - *`void pthread_exit()`*
    - *`int pthread_join()`*
    - *`int pthread_cancel()`*
  - Compilation
    - *`-D_REENTRANT`*
    - *`-lpthread`*
- **References**

Threads

- **Pthreads**
- **Management**
  - Pthreads refers to the **POSIX standard (IEEE 1003.1c)** defining an API for thread creation and synchronization:
    - *int pthread_create()*
    - *void pthread_exit()*
    - *int pthread_join()*
    - *int pthread_cancel()*



https://computing.llnl.gov/tutorials/pthreads/

- **Pthreads**
  - **Example 1**

```
/Temp/so$ ./thread_demo
Hello, I'm a Slooowww THREAD!
Hello, I'm a FAST Thread!
Hello, I'm a FAST Thread!
Hello, I'm a FAST Thread!
Hello, I'm a FAST Thread!
Hello, I'm a Slooowww THREAD!
Hello, I'm a FAST Thread!
Hello, I'm a FAST Thread!
Hello, I'm a FAST Thread!
Hello, I'm a Slooowww THREAD!
Hello, I'm a FAST Thread!
Hello, I'm a FAST Thread!
Hello, I'm a Slooowww THREAD!
Hello, I'm a FAST Thread!
Hello, I'm a FAST Thread!

/Temp/so$ _
```

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

void* slow_thread() {
        while (1) {
                printf("Hello, I'm a Slooowww THREAD!\n");
                sleep(3);
        }
}

void* fast_thread() {
        while (1) {
                printf("Hello, I'm a FAST Thread!\n");
                sleep(1);
        }
}

int main() {
        pthread_t thr_slow, thr_fast;

        pthread_create(&thr_slow, NULL, slow_thread, NULL);
        pthread_create(&thr_fast, NULL, fast_thread, NULL);

        pthread_exit(NULL);
        return 0;
}
```

summation of a nonnegative integer in a separate thread.

- **Pthreads**
  - **Example 2**

Pthreads programs must include the pthread. h header file.

Separate threads begin execution in a specified function which is in this case *runner ()*

The statement pthread_t tid declares the identifier for the thread we will create.

The pthread_attr_t attr declaration represents the attributes for the thread. Namely, the stack size and scheduling information.

```
#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;
  }
```

- **Pthreads**
  - **Example 2**

At this point, the program has two threads: the initial (or parent) thread in *main ()* and the summation (or child) thread performing the summation operation in the *runner ()* function.

set the attributes in the function call pthread_attr_init (&attr). Because we did not explicitly
set any attributes, we use the default attributes provided.

```
/* 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 parent thread will wait for it to complete by calling the *pthread_join ()* function

```
/* 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);
}
```

Both threads share the global data sum.

The summation thread will complete when it calls the function pthread_exit ()

A separate thread is created with the *pthread_create ()* function call, passing the thread identifier, the attributes for the thread, the name of the function where the new thread will begin execution (in this case, the *runner ()* function) and the integer parameter that was provided on the command line, argv [l]

- **Pthreads**
  - **Example 3**

```c
// Ids used by the threads
pthread_t my_thread[N];
int         id[N];
```

```c
// Worker thread
void* worker(void* idp) {
  int my_id = *((int*) idp);

  printf("Hello, I'm thread %d\n", my_id);
  sleep(rand()%3);
  printf("Hello, I'm thread %d, going away!\n", my_id);

  pthread_exit(NULL);
  return NULL;
}
```

```c
int main()
{
  // Creates N threads
  for (int i=0; i<N; i++) {
    id[i] = i;
    pthread_create(&my_thread[i], NULL, worker, &id[i]);
  }

  // Waits for them to die
  for (int i=0; i<N; i++)
    pthread_join(my_thread[i], NULL);

  return 0;
}
```

Threads

- **Pthreads**
- **Compilation**
  - use these library calls we must include the file pthread.h and link with the threads library using `-lpthread`
  - Re-entrant code can be called more than once, whether by different threads or by nested invocations in some way, and still work correctly (e.g. `errno` variable). The macro `REENTRANT` must be defined in the compilation
  - Example:
  - `$ gcc -D_REENTRANT thread1.c -o thread1 -lpthread`

References

- Abraham Silberschatz, ” Operating System Concepts”, 10th Edition, Wiley, 2018
- William Stallings, “Operating Systems: Internals and Design Principles”, 9th Edition, Pearson, 2017
- Andrew S. Tanenbaum, “Modern Operating Systems”, 4th Edition, Pearson Education, 2014