Chapter 5: Threads

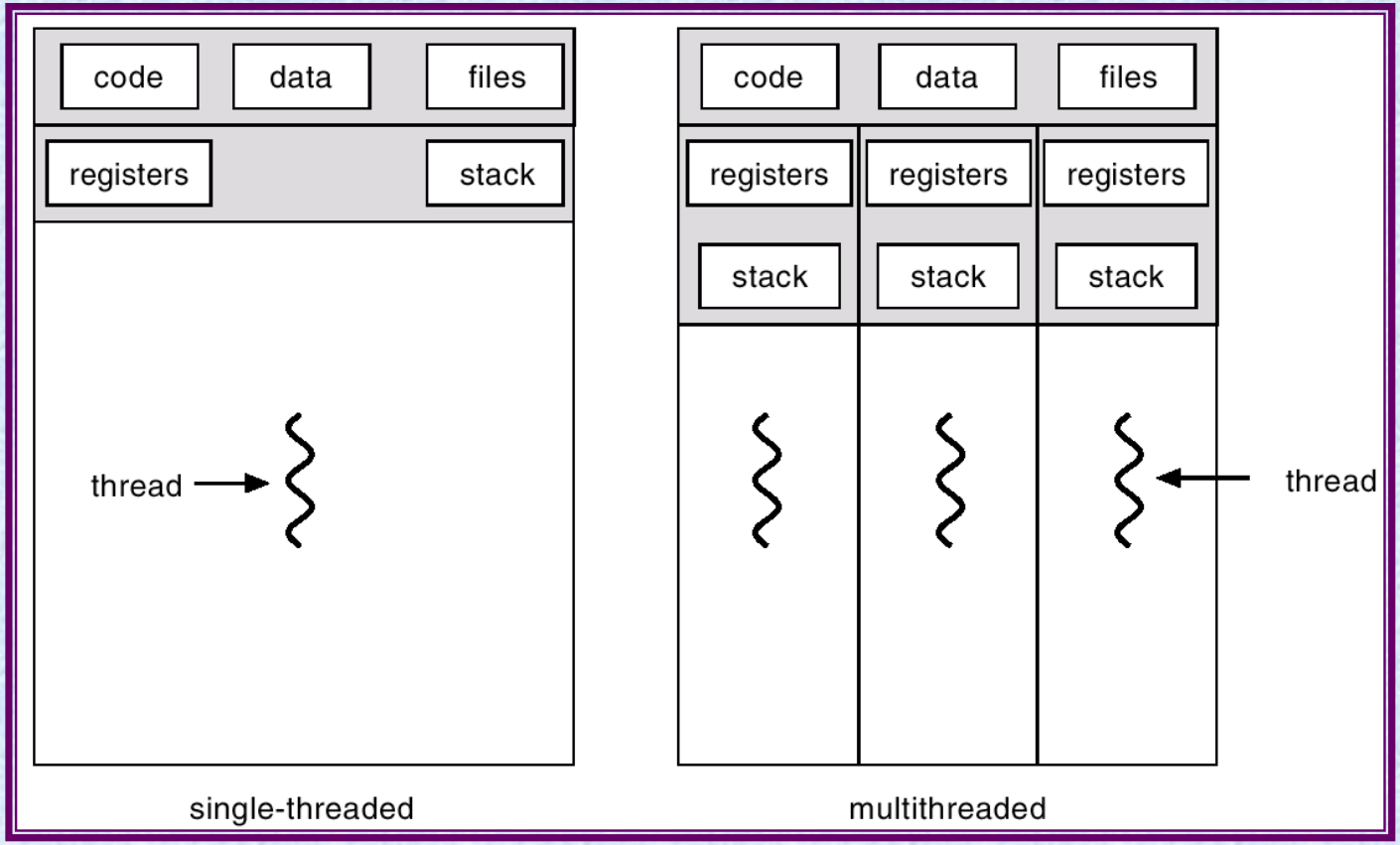
Definition

- **Processes with very little private memory are called *threads* or *light-weight processes*.**
- At a minimum, each thread needs:
  - a program counter and
  - a place to store a stack of return addresses;
  - all other values could be stored in memory shared by all threads.

Threads

- A thread is an alternative model of program execution
- A process creates a thread through a system call
- Thread operates within process context
- Use of threads effectively splits the process state into two parts
  - Resource state remains with process
  - CPU state is associated with thread
- Switching between threads incurs less overhead than switching between processes

Single and Multithreaded Processes



Threads (continued)

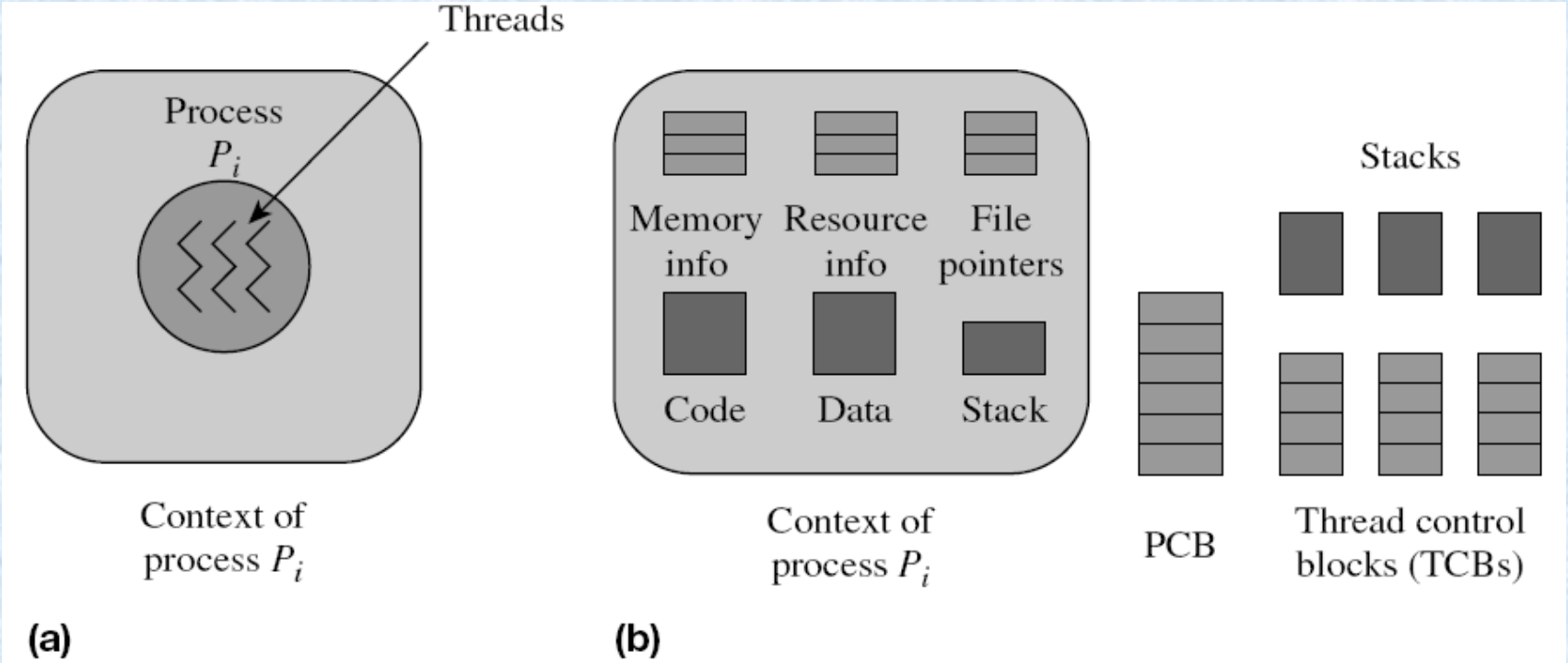


Figure 5.11 Threads in process P_i: (a) concept; (b) implementation.

Table 5.8 Advantages of Threads over Processes

Advantage	Explanation
Lower overhead of creation and switching	Thread state consists only of the state of a computation. Resource allocation state and communication state are not a part of the thread state, so creation of threads and switching between them incurs a lower overhead.
More efficient communication	Threads of a process can communicate with one another through shared data, thus avoiding the overhead of system calls for communication.
Simplification of design	Use of threads can simplify design and coding of applications that service requests concurrently.

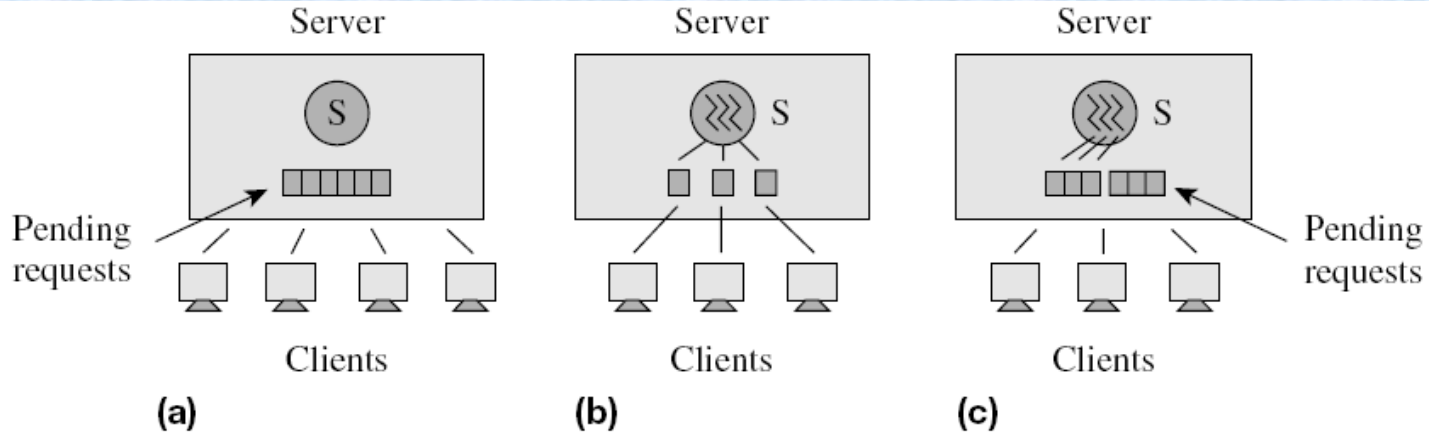


Figure 5.12 Use of threads in structuring a server: (a) server using sequential code; (b) multithreaded server; (c) server using a thread pool.

Benefits

- Responsiveness
- Resource Sharing
- Economy
- Utilization of MP Architectures

Kernel-Level, User-Level, and Hybrid Threads

- Kernel-Level Threads
  - Threads are managed by the kernel
- User-Level Threads
  - Threads are managed by thread library
- Hybrid Threads
  - Combination of kernel-level and user-level threads

Kernel-Level Threads

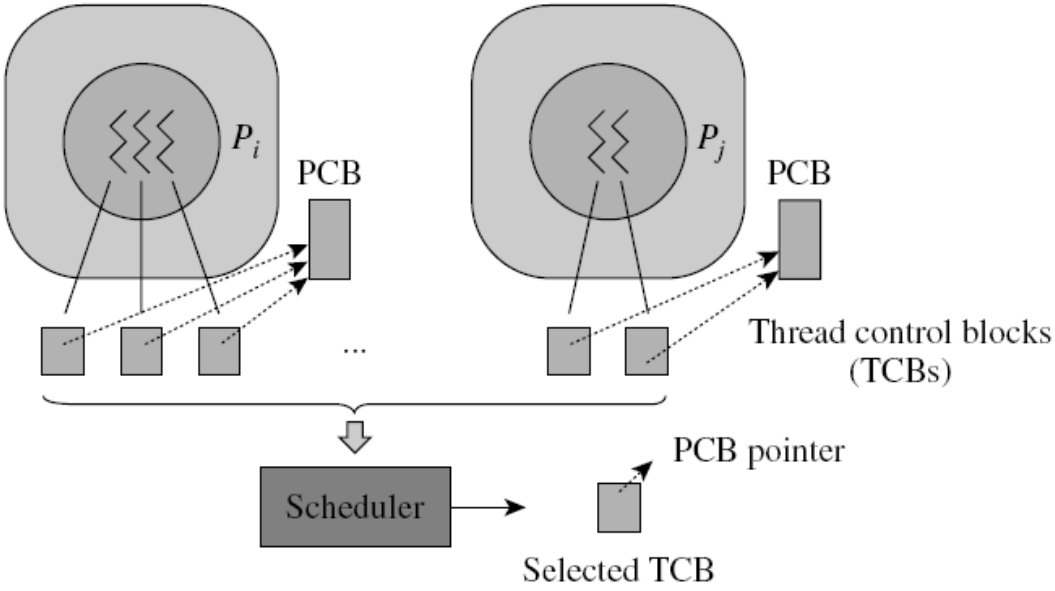


Figure 5.14 Scheduling of kernel-level threads.

- A kernel-level thread is like a process except that it has a smaller amount of state information
- Switching between threads of same process incurs the overhead of event handling

User-Level Threads

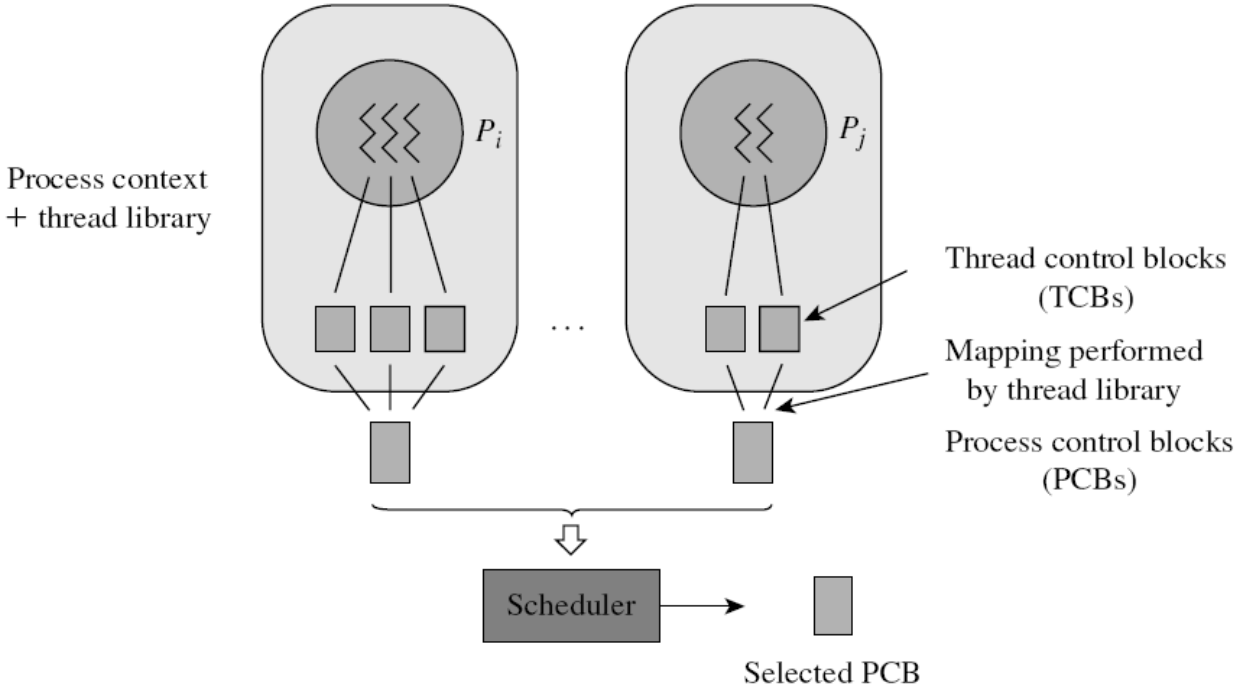


Figure 5.15 Scheduling of user-level threads.

- Fast thread switching because kernel is not involved
- Blocking of a thread blocks all threads of the process
- Threads of a process: No concurrency or parallelism

Hybrid Thread Models

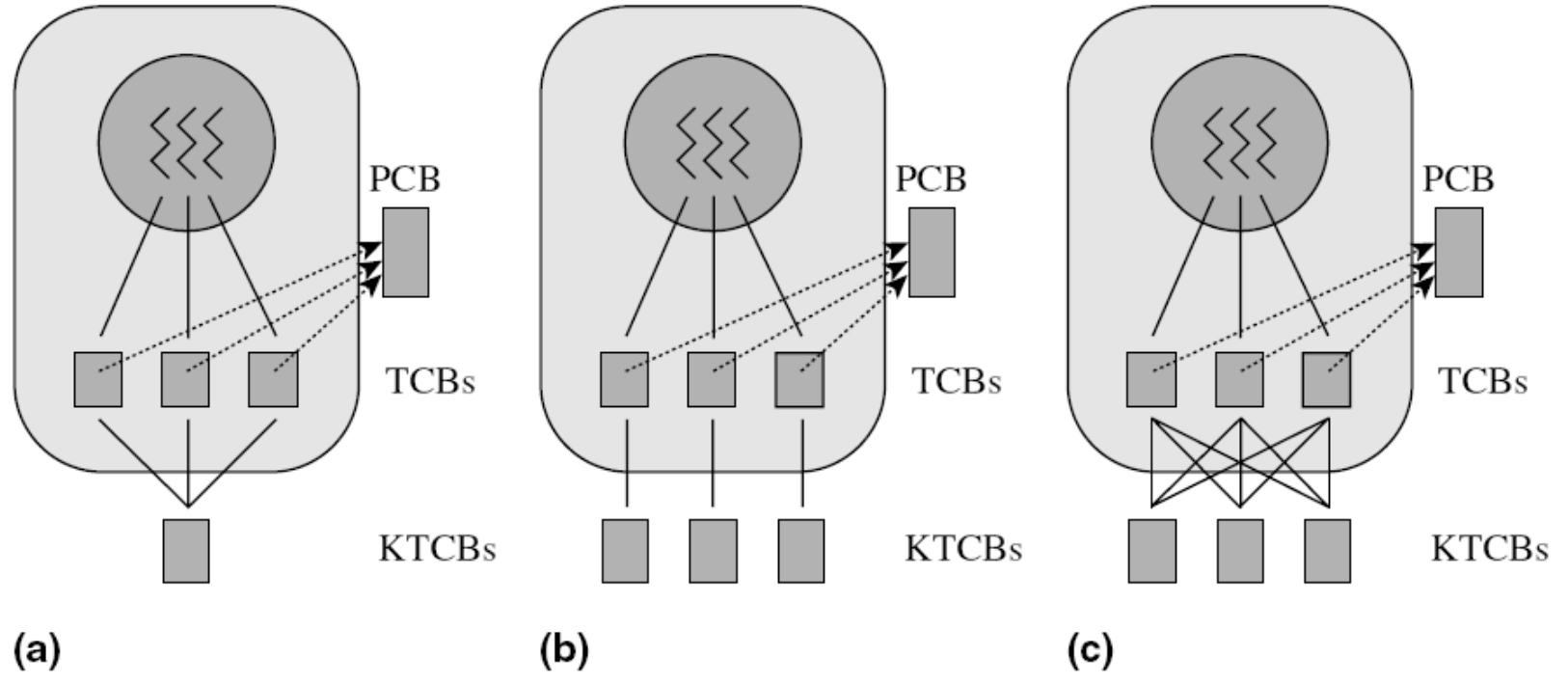


Figure 5.17 (a) Many-to-one; (b) one-to-one; (c) many-to-many associations in hybrid threads.

- Can provide a combination of parallelism and low overhead