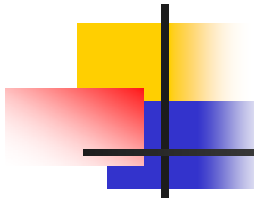
Operating Systems
(Threads & Concurrency)

Chapter 4

These lecture materials are modified from the source lecture notes written by A. Silberschatz, P. Galvin and G. Gagne.

Spring, 2020



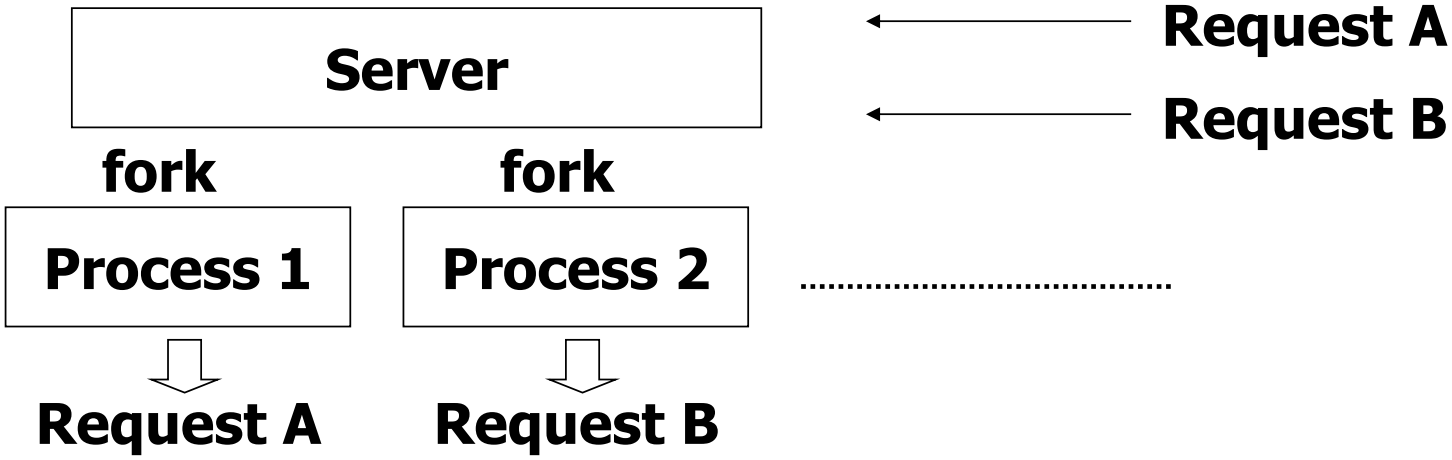
Outline

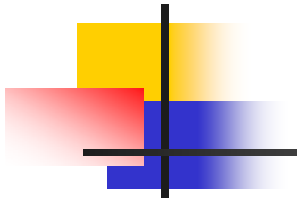
- Motivation
- Overview
- Multicore programming
- Multithreading models
- Thread libraries
- Threading issues
- Thread programming API

Motivation

Many programs must perform several tasks that do not need to be serialized.

Consider a web server;





But process creation is resource-intensive and time-consuming

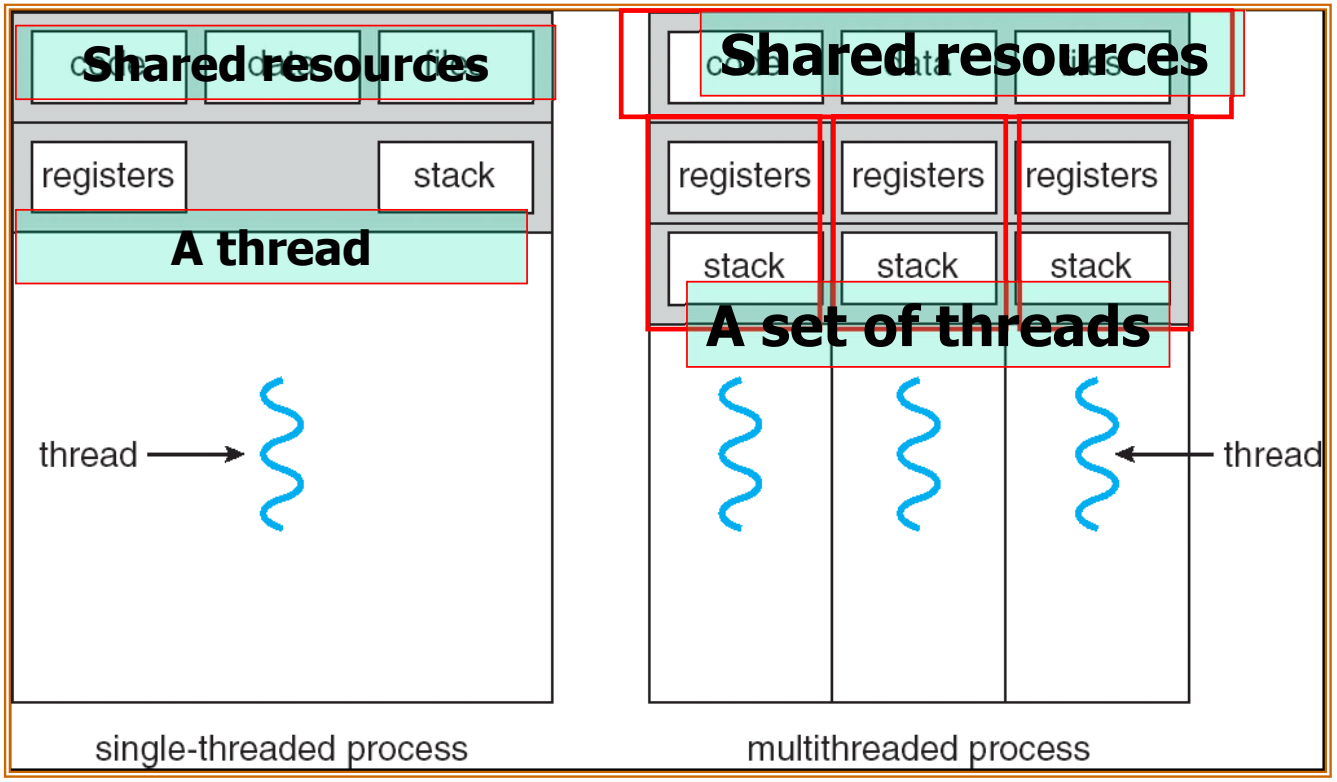
1. **Allocate a new PID, PCB and fill them in**
2. **Duplicate the address space**
3. **Make the child runnable**
4. **…**

In addition, an IPC mechanism is needed

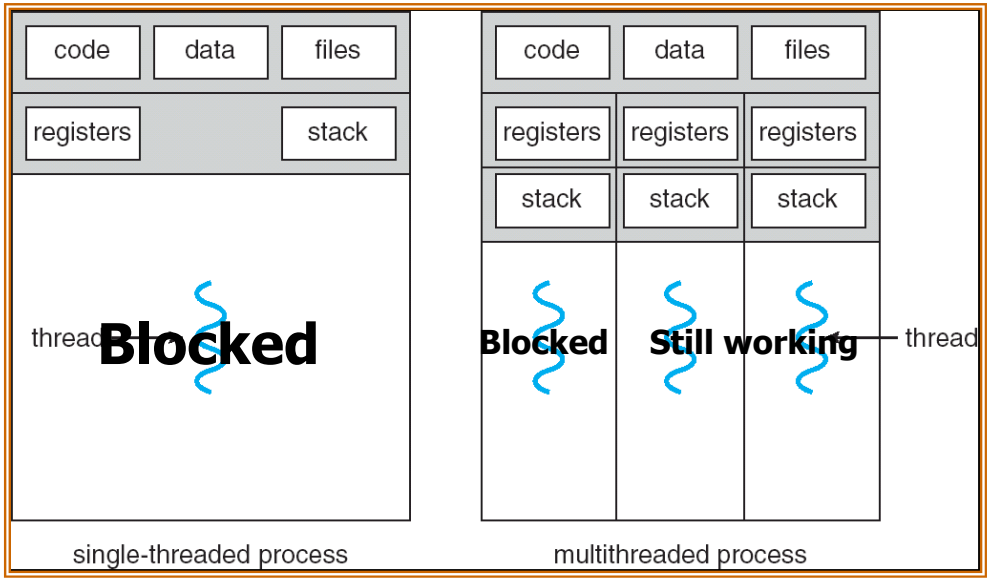
Therefore, a multithreading mechanism that reduces such overhead is required

Overview

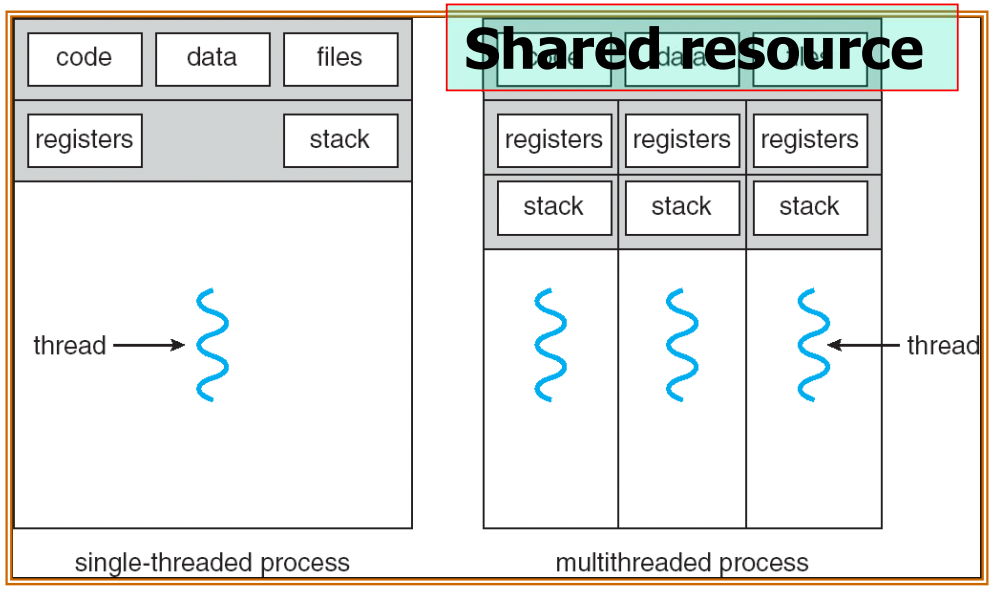
- Single vs. multi-threaded processes

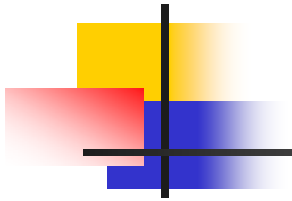


- Benefits
  - 1. Responsiveness
    - Multithreading allows a program to continue running if part of it is blocked or is performing a lengthy operation



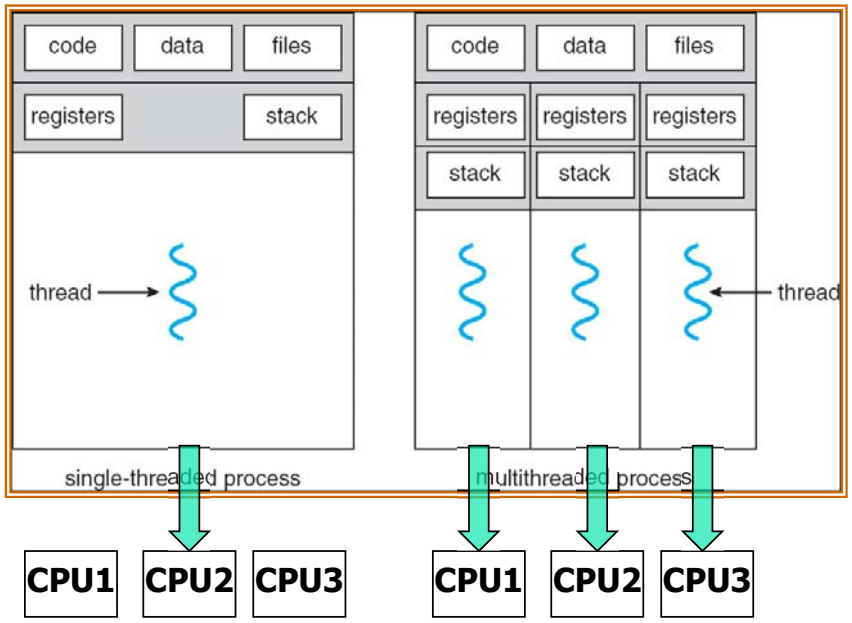
- Benefits
  - 2. Resource Sharing
    - Threads share binary code, data, resource of the process



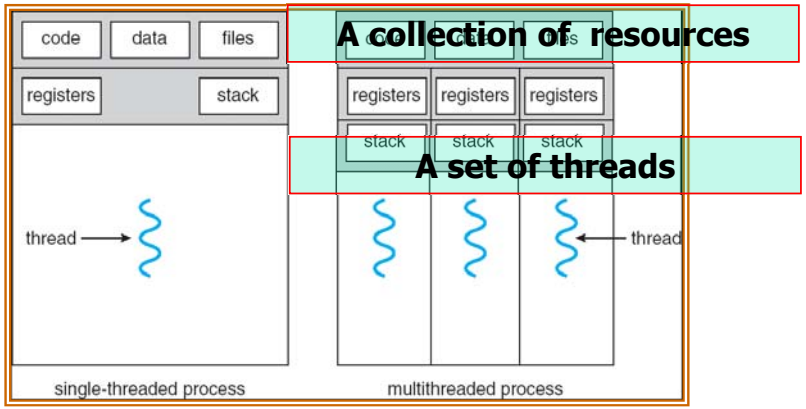


- Benefits
  - 3. Economy
    - Allocating memory and resources for process creation is costly
    - But it is more economical to create or context-switch threads
    - For example, in Solaris
      - Process creation: 30 times slower than thread creation
      - Process context switch: 5 times slower than thread switch

- Benefits
  - 4. Utilization of MP (multi-processor) Architectures
    - Threads may be running in parallel on different processors



- Fundamental concepts
  - A process is a compound entity that can be divided into:
    - A set of threads
      - A dynamic object that represents a control point in the process
    - A collection of resources
      - An address space, open files and so on



Multicore programming

- Types of parallelism
  - **Data parallelism** – distributes subsets of the same data across multiple cores, same operation on each
  - **Task parallelism** – distributing threads across cores, each thread performing unique operation

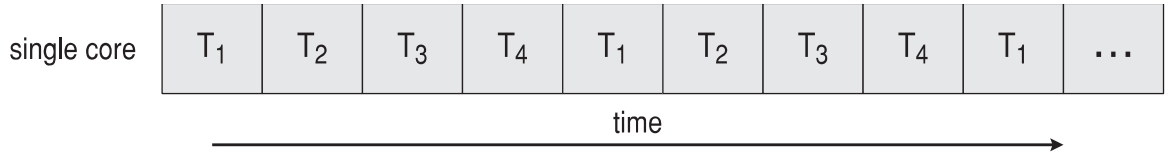
```
for(i=1;i<10000000;i++)
    a+=i;
```

```
for(i=1;i<5000000;i++)
    a+=i;
```

```
for(i=50000000;i<10000000;i++)
    a+=i;
```

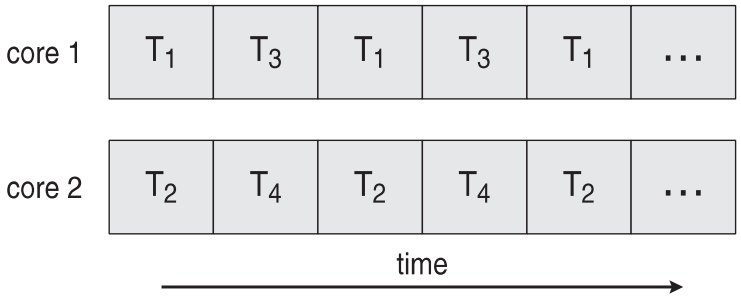
Concurrent execution on single-core system:

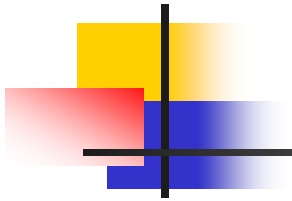
(동시성, 병행성)



Parallelism on a multi-core system:

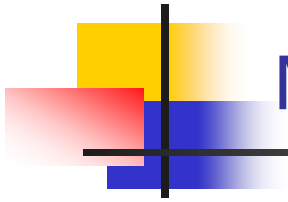
(병렬성)





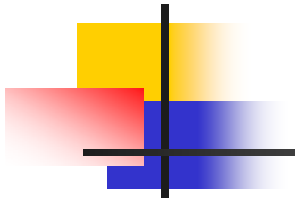
- Why multicore programming is difficult ?
  - 1. Identifying task
  - 2. Balance
  - 3. Data splitting
  - 4. Data dependency
  - 5. Testing and debugging

New approach to software design is needed !



Multi-Threading Model

- There are three types of threads;
  - Kernel thread
  - User thread



- Kernel thread
  - It is created and destroyed as needed internally by the kernel
  - It is managed by the operating system
- User thread
  - It is supported above the kernel and managed by the thread library
  - It does not involve kernel

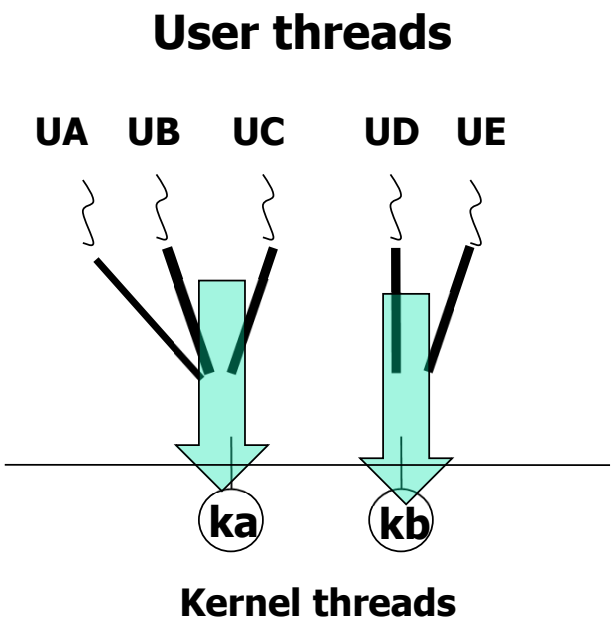
ka,kb : kernel threads

UA,UB,UC,UD,UE : User threads

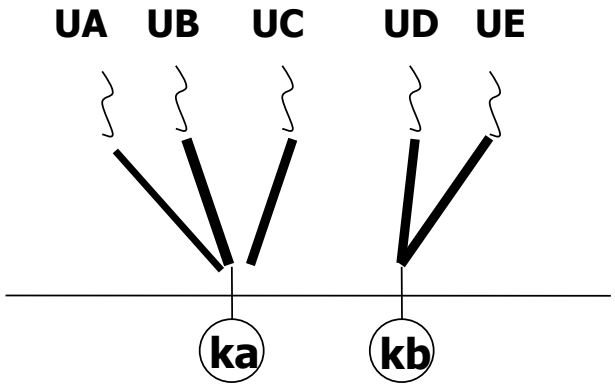
A scheduler is not aware of the existence of user threads

If a kernel thread "ka" is blocked, then user threads UA,UB and UC are blocked as well

One kernel thread can be allocated to one processor

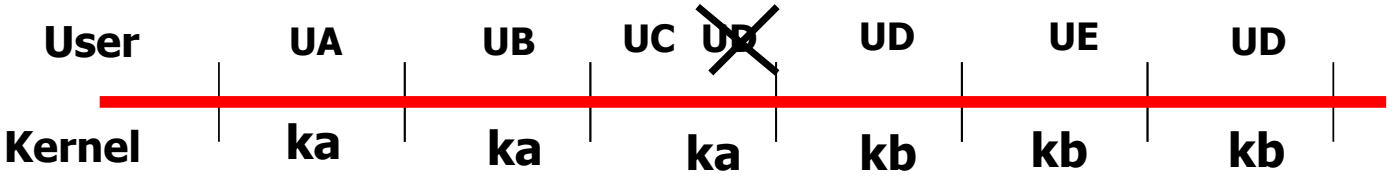


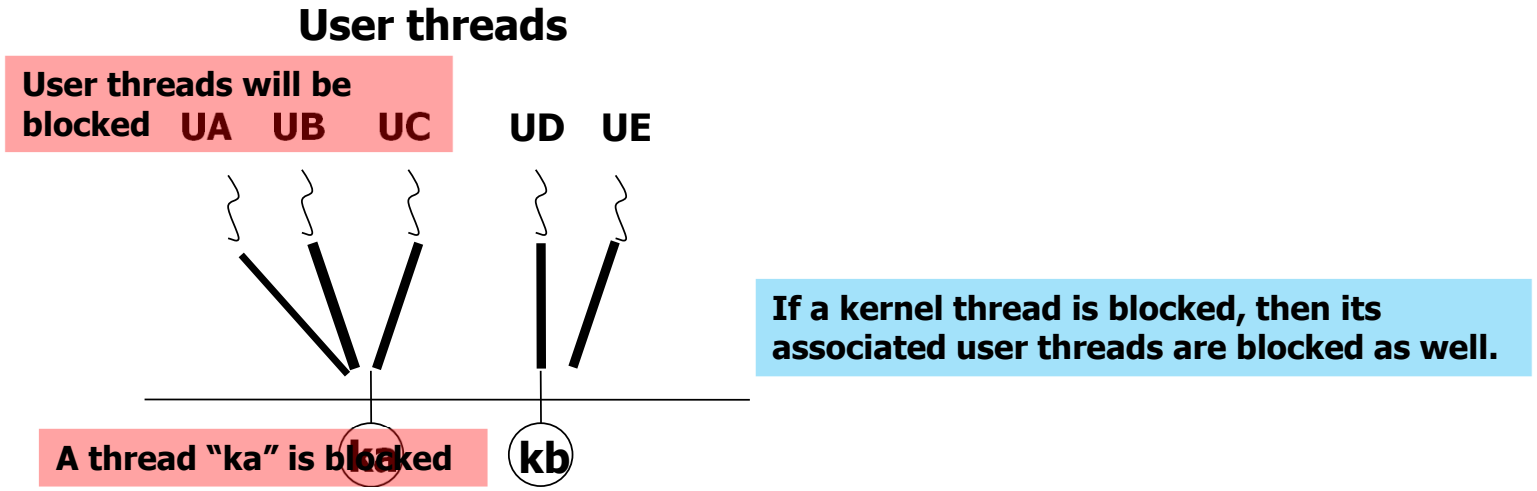
User threads

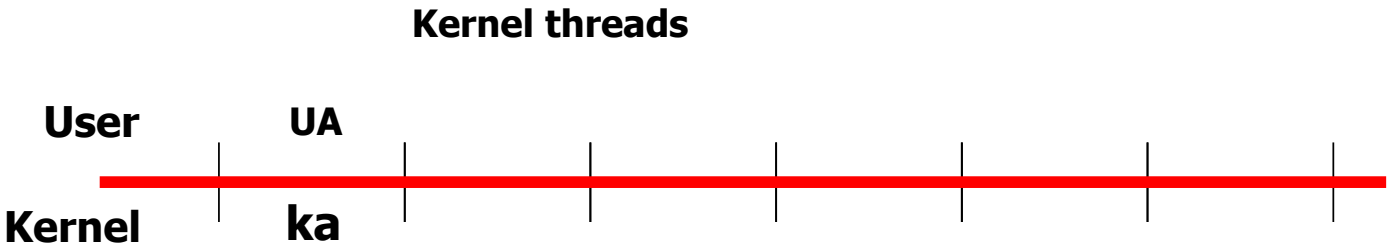


A scheduler is not aware of the existence of user threads

Kernel threads

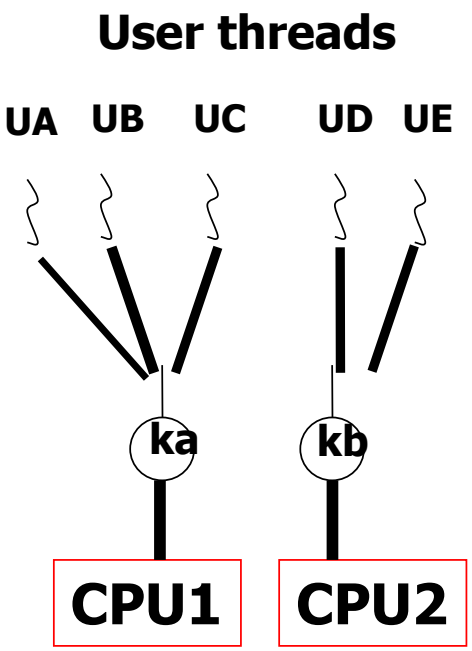


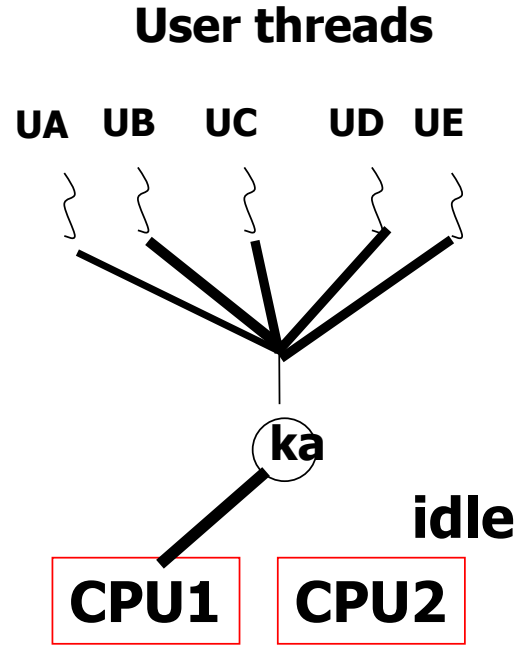




1. A thread "UA" calls the disk I/O

2. The state of the thread "ka" is changed into the "waiting" state





One kernel thread can be allocated to one processor