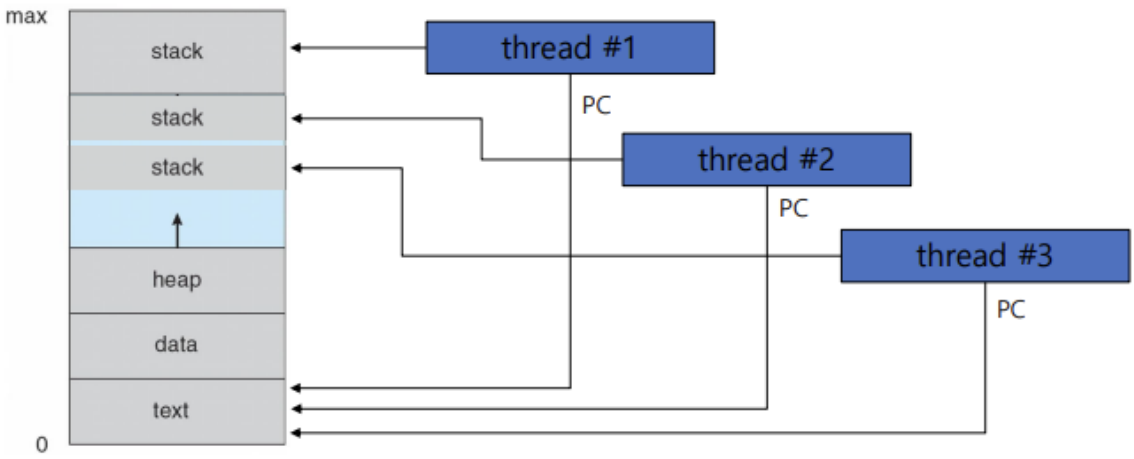
Thread

Insight

When a program is executed, it is stored in RAM in the form of a process and a process control block (PCB) for the process is generated in the Kernel. Even if it is similar or the same program, once it is executed, it becomes a different process.(fork()) But can't we create a small process(Thread) that shares memory and data in one process? It is Thread that started with this idea. Previously, the basic unit of CPU Scheduling is known as Process, but for modern OS, Thread is the basic unit of CPU Scheduling.



핵심: 비슷한 Process들은 따로 Fork로 생성해서 낭비하지 말고 하나의 Process를 Address Space로(Container)로 하고 거기에 많은 Thread를 만들어(그 안의 Thread들은 PC,Stack,Register 말고는 다 공유할 수 있다.) CPU 스케쥴링의 단위로 하자.

(1) Thread가 공유하는 것: Text,Data,Heap

(2) Thread가 공유하지 않는 Private 요소: Stack,PC(Process Counter),Register(자기가 사용중인 CPU)

Processes are not always ideal.....

(1) Processes are not very efficient

a. Each Process has its own PCB and OS resources

b. Creating a new process is often very expensive (Resource)

>>각각의 작업 공간을 만들어줘야 하기 때문에 낭비가 심하다.

(2) Processes don't directly share memory (Protection) >> through IPC

a. Each Process has its own address space (Process don't know PCB of the process know the address of process)

b. Parallel and concurrent programs often want to directly manipulate the same memory (shared memory is necessary)

>> I want to share it directly, but processes do not have direct access to memory and exchange information in IPC format. But it's time consuming and it's inefficient to keep data that can be used together from being exchanged separately

What can we do?

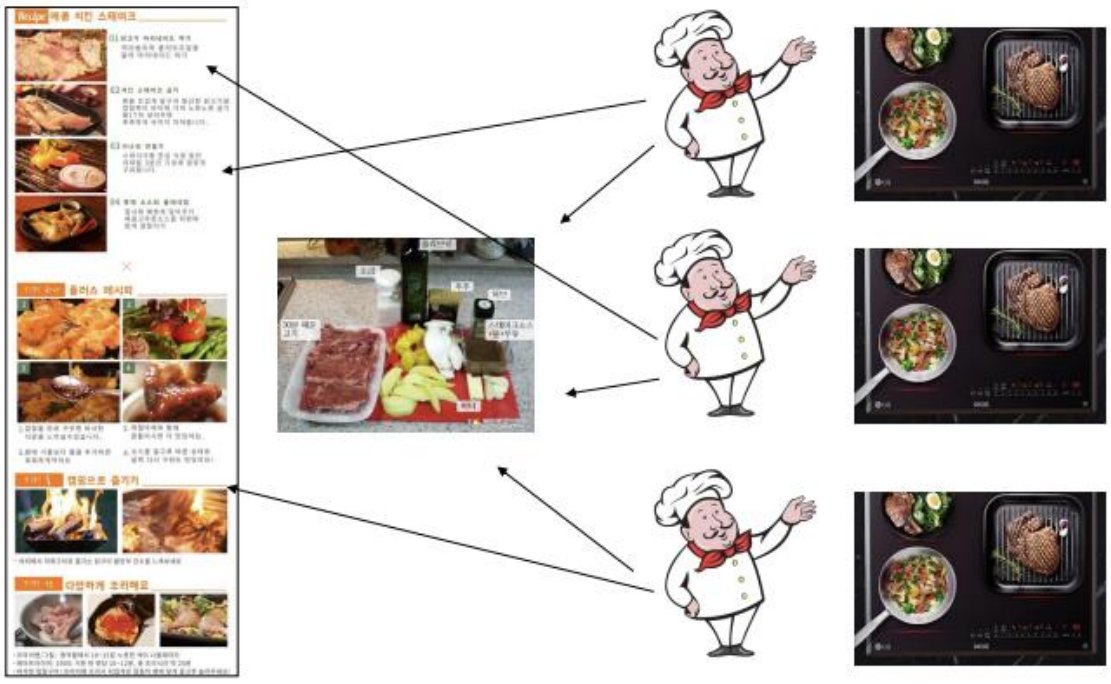
(1) Let's share same code and same data (Text Data Heap)

(2) What is private to each thread (Cpu Registers,Stack,and Program counter)

(3) The Process is the address space and OS resources (PCB)

(4) Each thread has its own CPU execution state (Thread is basic unit of CPU Scheduling)

(5) Process is shared office and Thread is Chef



Processes and Threads

(1) Process is just a container for its threads (The process is the address space and OS resources)

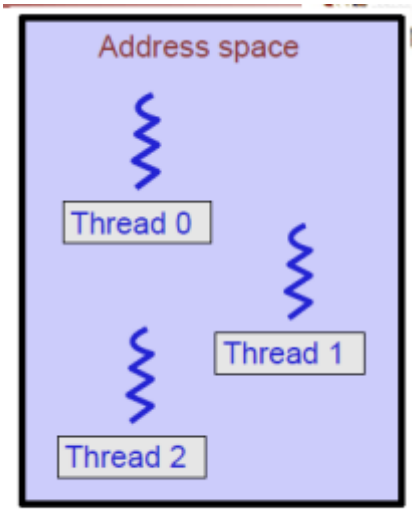
(2) Thread is a basic unit of CPU scheduling (Each thread has its own CPU execution state)

(3) Each thread is bound to its containing process

(4) Each thread has its own stack,CPU,Registers

(5) All threads within a process share the same address space and OS resources (Thread is chef in shared office)

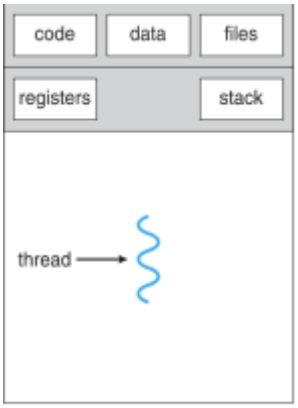
(6) Threads share memory,so they can communicate directly with other thread in same process



Process

Thread

(1) Simple Programs can have one thread per process (The process we have known is single thread process)



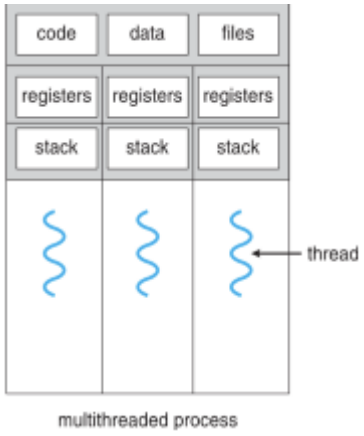
single-threaded process

(2) Complex Programs can have multiple threads

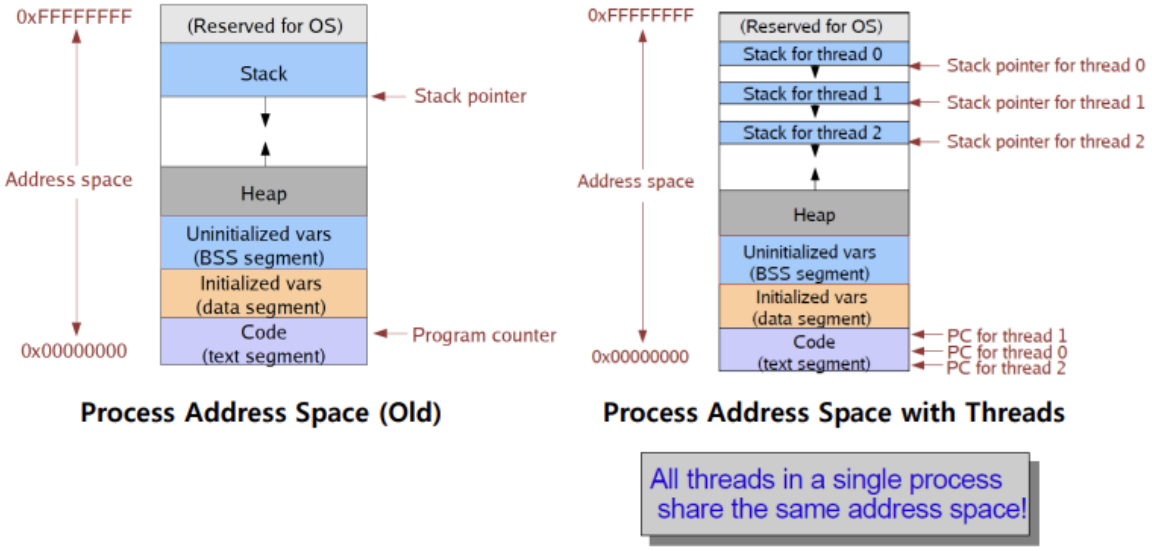
a. Multi-Threaded process

b. Multiple threads running in same process's address space(address space == container == process)

c. Each Thread can run on different CPUs(cores) while sharing memory resource (In Multi Processors, the concept of Multi-Threaded process is necessary for efficient resource consuming)



Process Address Space with Threads



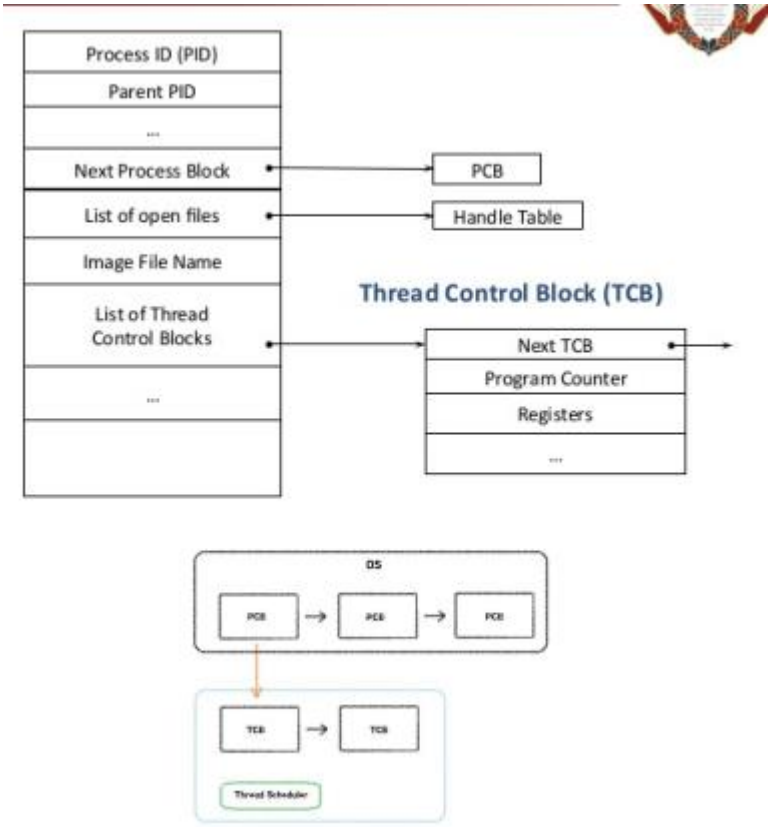
(1) Stack and Process counter and Registers is Private resources to each Thread.

Control Blocks (Thread Control Block)

(1) Ready Queue is now a list of TCBs waiting for CPU resource

(2) Context switching is done for TCBs.

(3) Remind that the unit of CPU scheduling is Thread



Benefits of Multithread

(1) Resource Sharing

a. Threads share memory resources of process – (CPU run-time resources such as register,stack,PC are not shared) (process 안에 독립적으로 존재하여 각기 자기 일을 하지만 memory resources를 공유(text,data,heap,files)함으로써 기존 fork로 process를 만드는 비효율을 극복하였다.)

b. Easier to share than IPC because threads share code,data,files resources directly in same Address Space

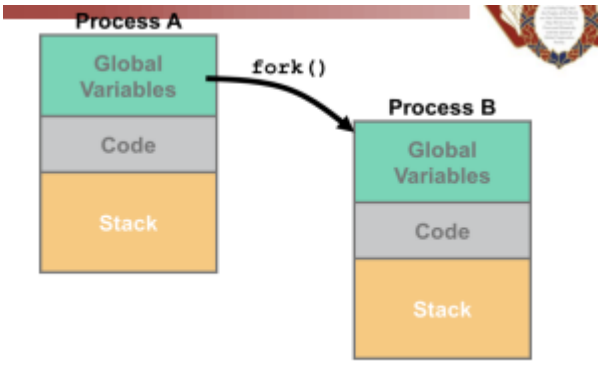
(2) Lighter weight

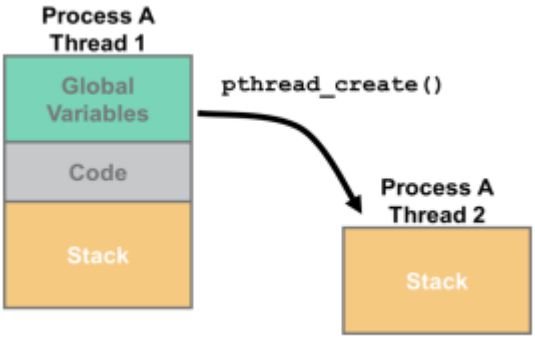
a. Creation,Deletion faster

The things OS have to do for creating of Thread is assigning Stack Pointer in Stack and Program Counter in Code

b. Context-switching faster

CPU에 정보를 빼고 다시 넣는 시간은 Thread든 기존 Process든 모두 같다. 하지만 Thread의 경우 같은 Thread라면 Context-switching 과정에서 Cache를 비우지 않아도 되기 때문에 Process 간 Context-switching보다 훨씬 빠르다





.

(3) Allows one process to use multiple CPUs or cores (Thread가 하나였다면 core가 많아도 하나밖에 동작시킬 수 없지만 많은 threads가 있다면 하나의 prcoess가 많은 cpu cores를 동시에 사용할 수 있다. Thread는 Multi-core와 만났을 때 시너지 효과가 극대화 된다.)

a. A multithreaded process can take advantage multiprocessor architectures

b. A process can run many threads in parallel on different processor cores

(정말 simultaneously하게 여러 threads를 돌릴 수 있게 됨)

(4) Non-blocking system call (기존 single threaded process 였다면 wait를 만나면 끝난다 하지만 thread가 많다면 하나가 wait에 걸려도 나머지는 독립적이기 때문에 자기 할일을

한다.)

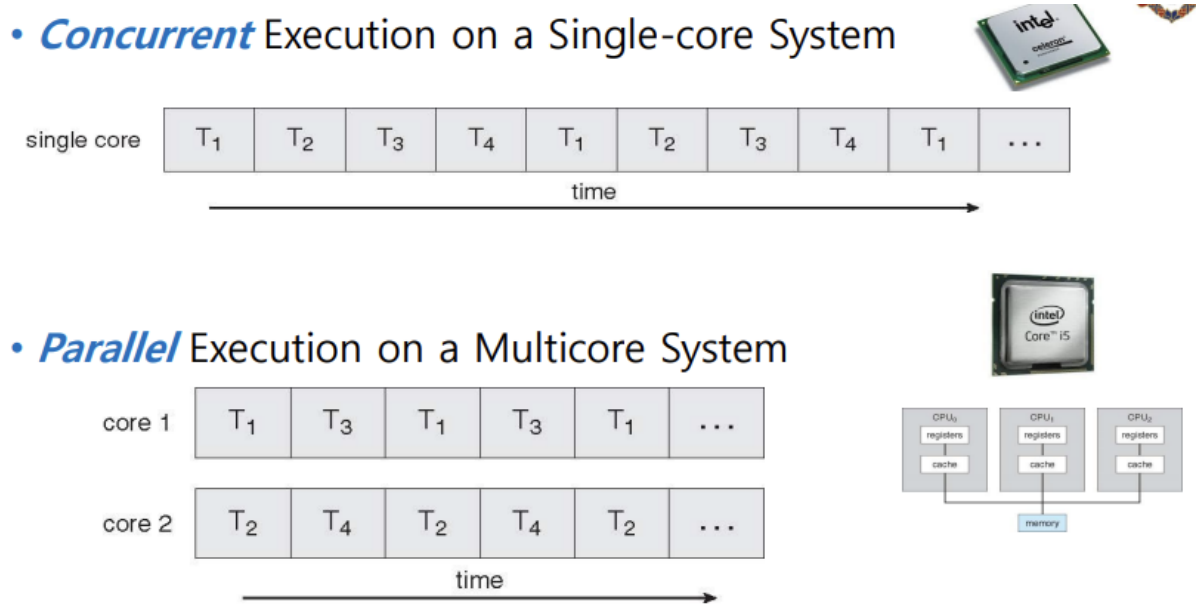
a. Allow continued execution if part of process is blocked

b. Multithreaded process: another thread can continue

(기존 single threaded process의 경우 I/O operation을 위해 wait에 대기하면 끝이지만 Multi-threaded process의 경우 process 안에 하나의 thread가 wait에 걸려도 다른 thread들은 계속 execution한다)

c. So better responsiveness to the user

MultiCore Programming

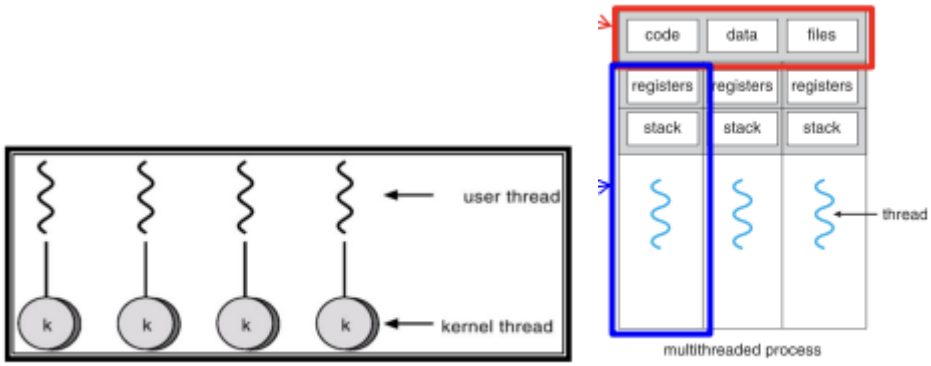


(1) Concurrency supports more than one task making progress – Single Core Processor can provide Concurrency (CPU Scheduling으로 여러 task를 처리할 수 있지만 simultaneously는 아니다 너무 빨라서 인간 눈에 Simultaneously하게 보여 그렇지)

(2) Parallelism implies a system can perform more than one task simultaneously – Need Multi-core processor

(3) By using multithreading, One process (Adress Space) can use multiple processors (cores)

Multithreading Models



(1) Kernel Threads (우리가 지금까지 배운 Threads가 kernel Threads OS가 제작 관리한다.)

A. Threads supported by the kernel (Thread creation and management requires system calls)

Created by system call,each thread needs thread control block (TCB) – kernel에 올라가야 하므로 OS가 관리해야 한다.

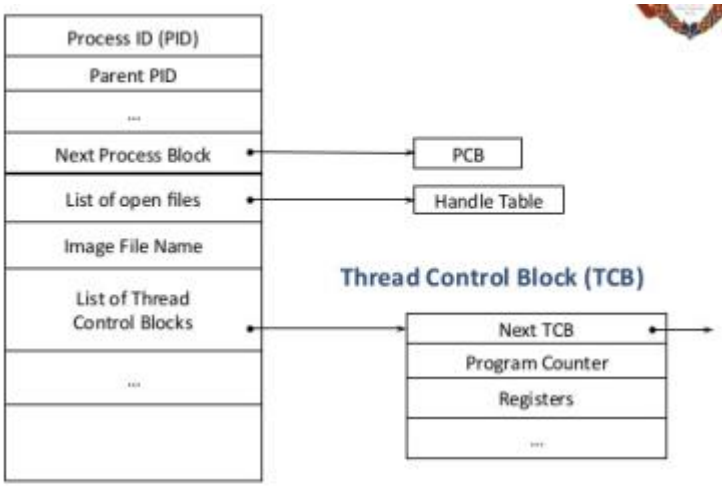
B. Pros

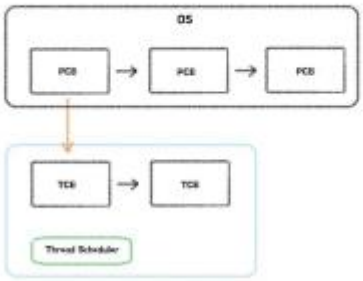
a. Can run multiple threads on multi-core

b. Concurrency: another thread can run when one thread makes blocking system call (주의해야하는 점 한 Process안에 있는 Thread가 System call로 wait을 가더라도 다른 thread는 자기 할 일을 할 수 있다.)

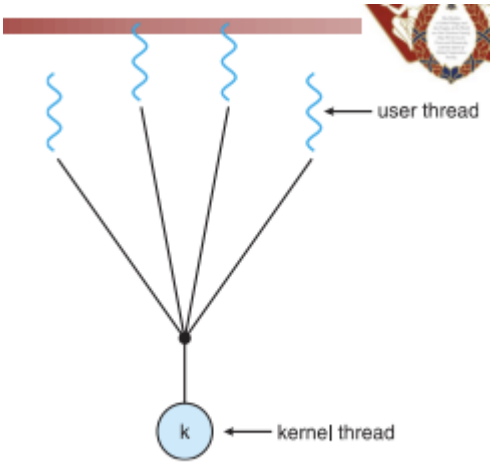
C. Cons

Every thread operation must go through kernel (여전히 fork로 생성해야 하는 process operation보다 훨씬 빠르고 효율적이다.)


Process ID (PID)
Parent PID
...
Next Process Block
List of open files
Image File Name
List of Thread Control Blocks
...
PCB
Handle Table
Thread Control Block (TCB)
Next TCB
Program Counter
Registers
...


OS
PCB
PCB
PCB
TCB
TCB
Thread Scheduler

(2) User thread



A. Created,managed without kernel support (no need for system call so fast) – therefore, Kernel do not know user thread exists in kernel thread

B. Implement thread in user library

C. Many user threads mapped to single kernel thread

D. Pros

a. Fast and efficient (no need for system calls)

operation	User thread	Kernel thread	Process
Null fork	34	948	11300
Signal-wait	37	441	1840

b.

E. Cons

a. One thread blocking causes all to block (Kernel don't know user thread exists in kernel thread)

b. Multiple user-level threads may not run in parallel on multicore system (User level thread cannot be executed by multiple cpu cores)

>> Because Kernel processes the unit of kernel thread. Kernel don't know what user thread is

Thread Libraries

(1) Pthread_create() to create new thread with 'pthread' library

Tid,Function,argv is needed for creating Thread

(2) Pthread_join() function to suspend parent thread until child thread terminates. Similar to wait system call in process

Thread vs Process Creation

(1) Fork()

a. Two separate processes

b. Child process start from same position as parent (clone)

c. Independent memory space for each process (Address Space)

(2) Pthread_create()

a. Two separate threads

b. Child thread starts from a function

c. Share memroy

Conclusion

(1) Thread Concept

a. Basic unit of CPU shceduling

b. Shares with other threads belonging to the same process its code,data and other resources (heap files) such as open files and signals (not register,stack PC)

(2) Multi-threading models (kernel thread,User Thread)

(3) Thread Libraries : Pthread,Java Thread

Future work

- Who gets to go next when a thread blocks/yields?
  - Scheduling! (Ch. 5)

- What happens when multiple threads are sharing the same resource?
  - Synchronization! (Ch. 6)