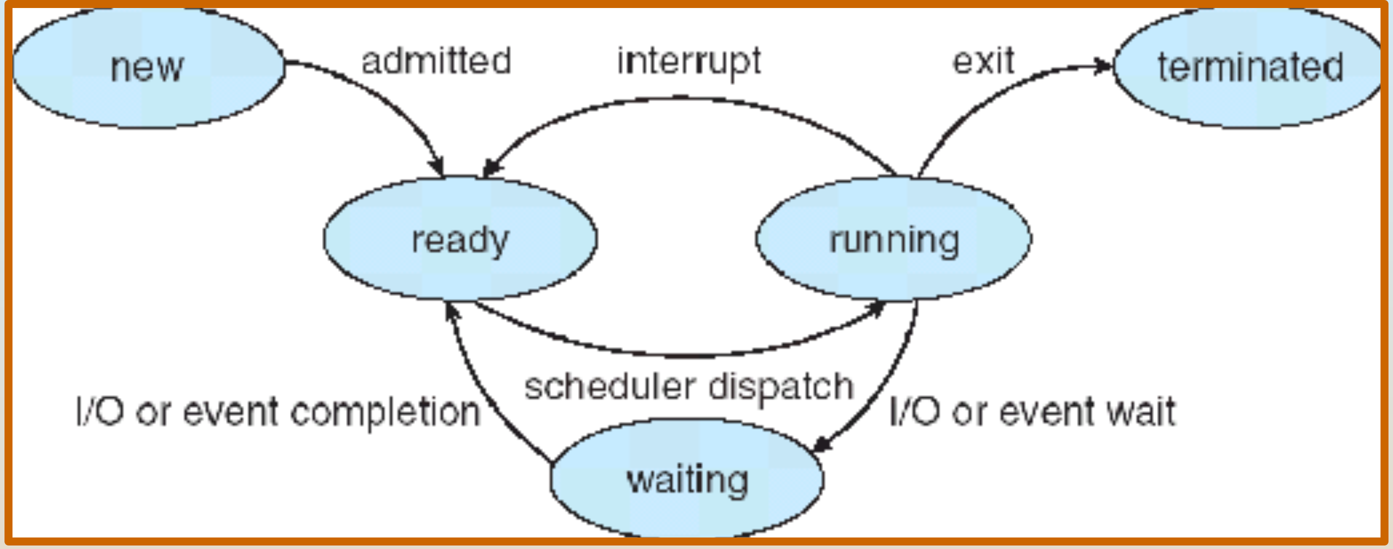
Processes

- Process – a program in execution; process execution must progress in sequential fashion

- Textbook uses the terms job and process almost interchangeably
  可交换的

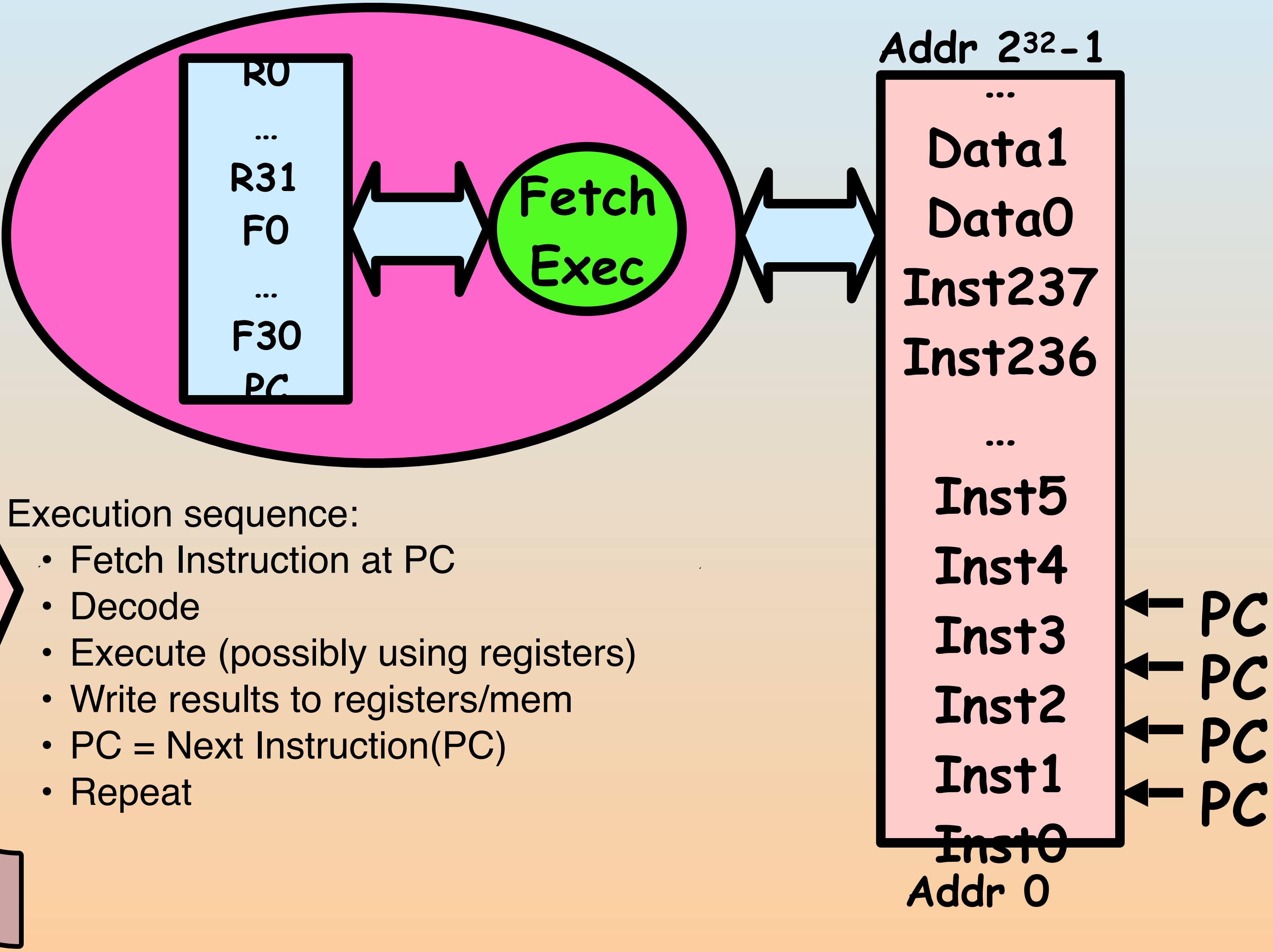
- As a process executes, its state changes:
  - new: The process is being created
  - running: Instructions are being executed
  - waiting: The process is waiting for some event to occur
  - ready: The process is waiting to be assigned to a process
  - terminated: The process has finished execution
    终止

Diagram of Process State



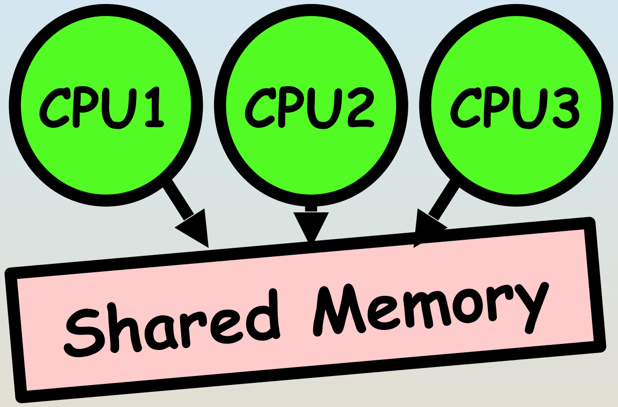
- As a process executes, it changes state
  - new: The process is being created
  - ready: The process is waiting to run
  - running: Instructions are being executed
  - waiting: Process waiting for some event to occur
  - terminated: The process has finished execution

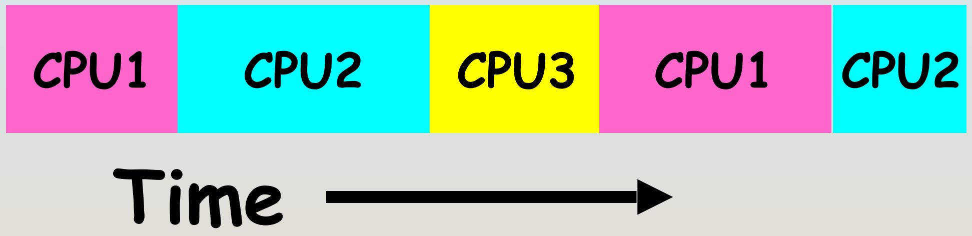
What happens during execution?



- Execution sequence:
  - Fetch Instruction at PC
  - Decode
  - Execute (possibly using registers)
  - Write results to registers/mem
  - PC = Next Instruction(PC)
  - Repeat

What happens during execution?





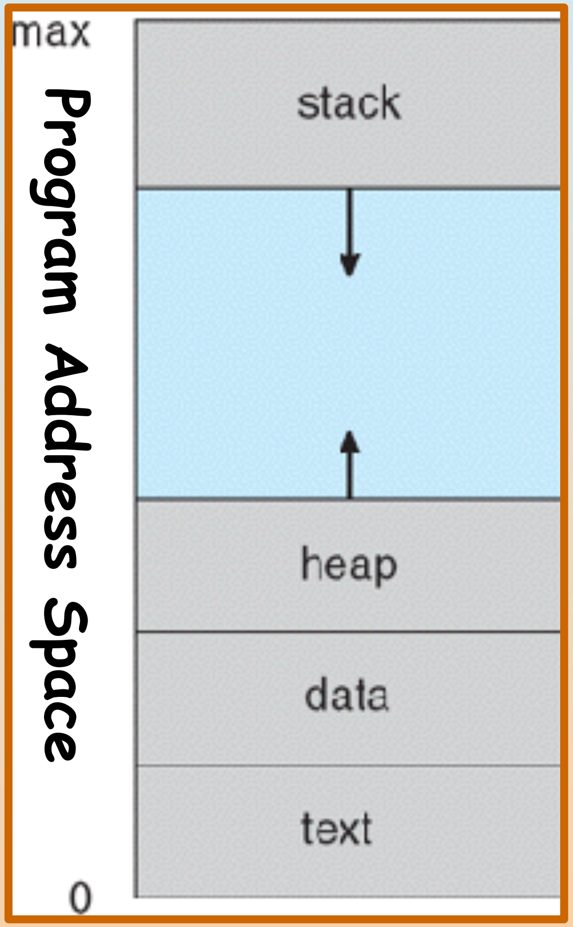
- Assume a single processor. How do we provide the illusion of multiple processors?
  - Multiplex in time!
- Each virtual "CPU" needs a structure to hold:
  - Program Counter (PC), Stack Pointer (SP)
  - Registers (Integer, Floating point, others…?)
- How switch from one CPU to the next?
  - Save PC, SP, and registers in current state block
  - Load PC, SP, and registers from new state block
- What triggers switch?
  - Timer, voluntary yield, I/O, other things

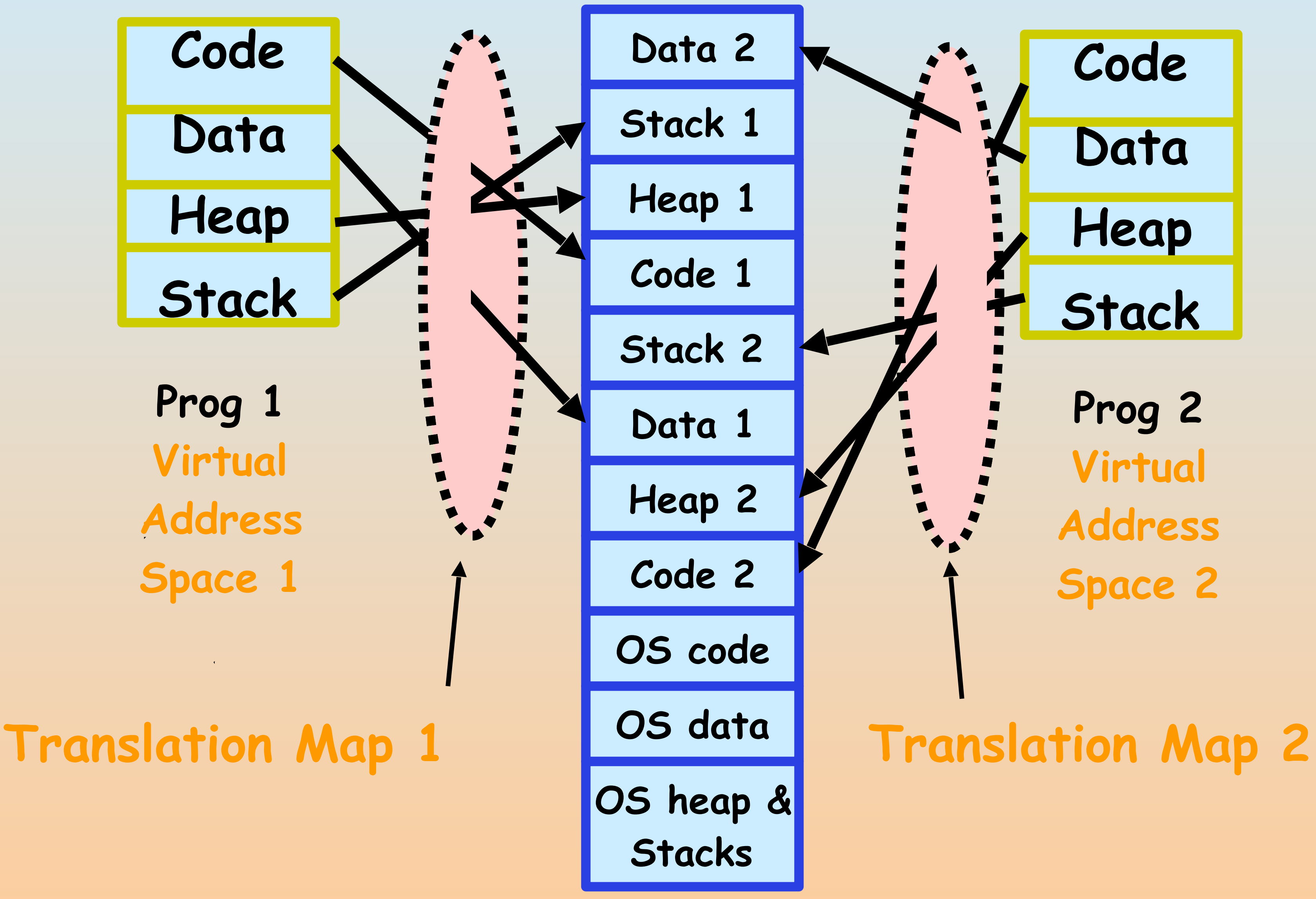
Simple multiprogramming

- All virtual CPUs share same non-CPU resources
  - I/O devices the same
  - Memory the same
- Consequence of sharing:
  - Each thread can access the data of every other thread (good for sharing, bad for protection)
  - Threads can share instructions (good for sharing, bad for protection)
  - Can threads overwrite OS functions?
- This (unprotected) model common in:
  - 嵌入的 Embedded applications
  - Windows 3.1/Machintosh (switch only with yield)
  - Windows 95—ME? (switch with both yield and timer)

Program's Address Space

- Address space ⇒ the set of accessible addresses + state associated with them:
  - For a 32-bit processor there are 232 = 4 billion addresses
- What happens when you read or write to an address?
  - Perhaps Nothing
  - Perhaps acts like regular memory
  - Perhaps ignores writes
  - Perhaps causes I/O operation
    - (Memory-mapped I/O)
  - Perhaps causes exception (fault)

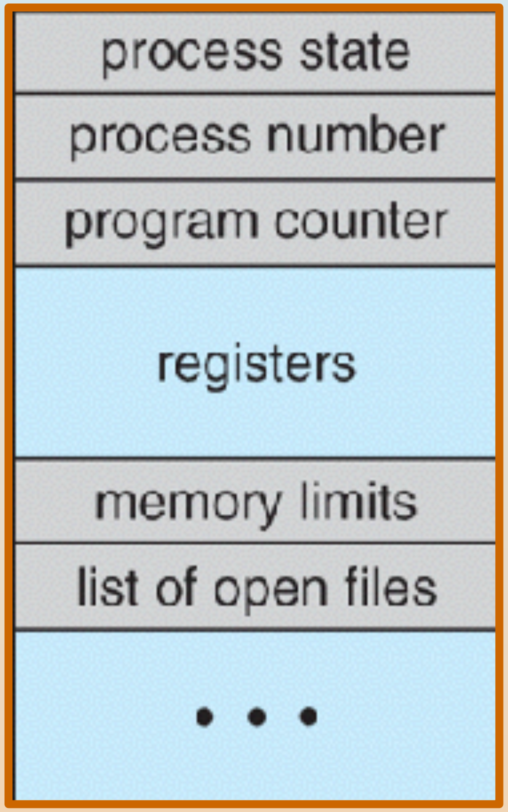




Physical Address Space

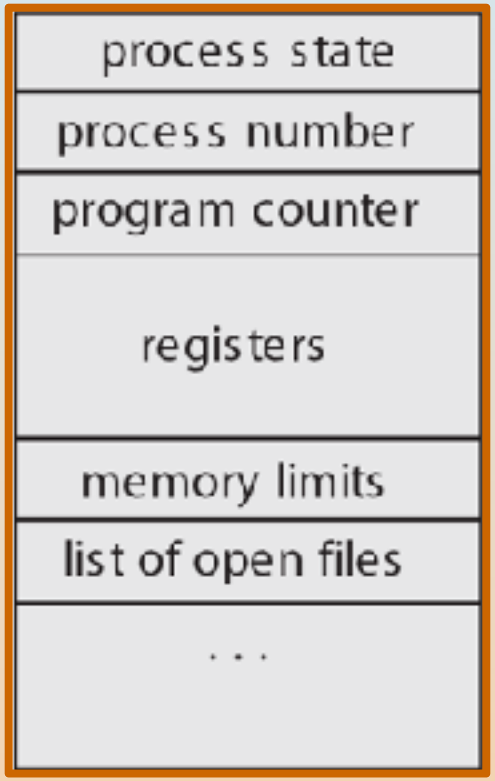
How do we multiplex processes?

- The current state of process held in a process control block (PCB):
    - This is a "snapshot" of the execution and protection environment
    - Only one PCB active at a time
- Give out CPU time to different processes (Scheduling):
    - Only one process "running" at a time
    - Give more time to important processes
- Give pieces of resources to different processes (Protection):
    - Controlled access to non-CPU resources
    - Sample mechanisms:
        - Memory Mapping: Give each process their own address space
        - Kernel/User duality: Arbitrary multiplexing of I/O through system calls

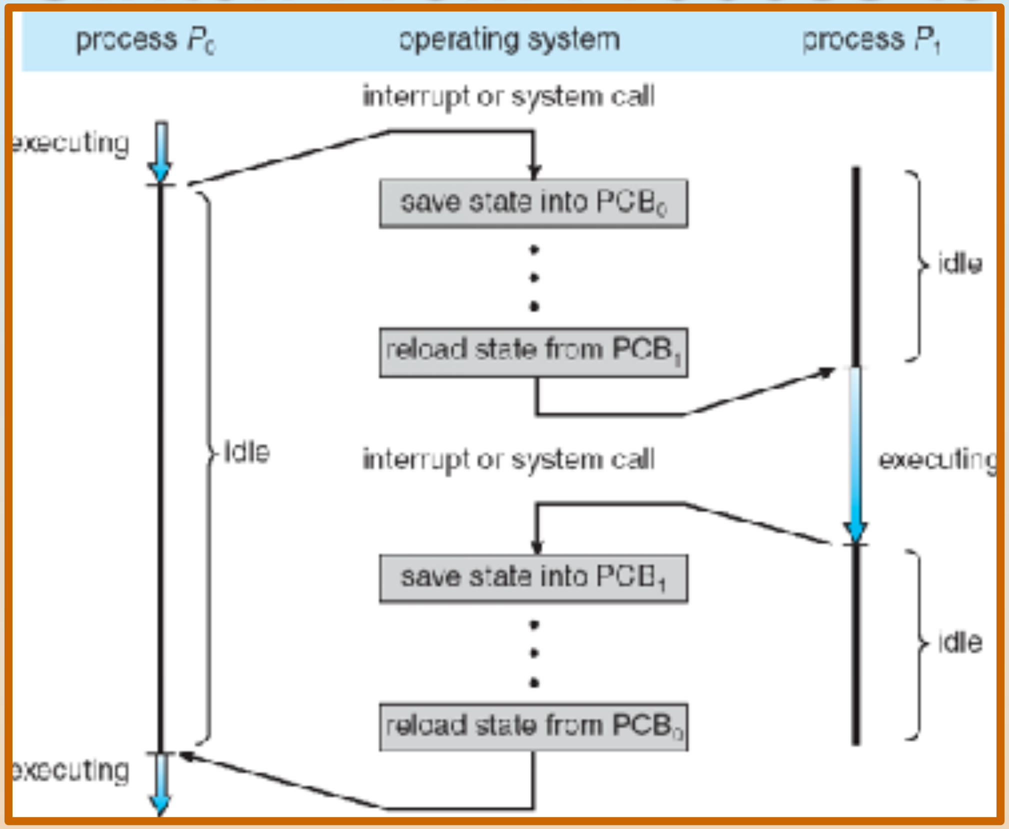


Process Control Block

Process Control Block (PCB)



CPU Switch From Process to Process



- This is also called a “context switch”
- Code executed in kernel above is overhead
  - Overhead sets minimum practical switching time
  - Less overhead with SMT/hyperthreading, but… contention for resources instead

中断

- 程序执行过程中，当发生某个事件时，中止CPU上现行程序的运行，引出处理该事件的程序执行的过程。
  - 能充分发挥处理机的使用效率
  - 提高系统的实时处理能力

中断的分类

- 从中断事件的性质和激活的手段分类：
  - 强迫中断：强迫性中断事件不是正在运行的程序所期待的，而是由于某种事故或外部请求信息所引起的，分为：机器故障中断事件；程序性中断事件；外部中断事件；输入输出中断事件。
  - 自愿中断：自愿性中断事件是正在运行的程序所期待的事件。正在运行的程序对操作系统有某种需求，一旦机器执行到一条访管指令时，便自愿停止现行程序的执行而转入访管中断处理程序处理。

- 按照中断信号的来源分类：
  - 外中断(又称中断)：指来自处理器和主存之外的中断。外中断包括电源故障中断、时钟中断、控制台中断、它机中断和I/O中断等。不同的中断具有不同的中断优先级，处理高一级中断时，往往会屏蔽部分或全部低级中断。
  - 内中断(又称异常)：指来自处理器和主存内部的中断。内中断包括：通路校验错、主存奇偶错、非法操作码、地址越界、页面失效、调试指令、访管中断、算术操作溢出等各种程序性中断。异常是不能被屏蔽的，一旦出现应立即响应并加以处理。

中断

- 中断是由与现行指令无关的中断信号触发的(异步的)，且中断的发生与CPU处在用户模式或内核模式无关，在两条机器指令之间才可响应中断，一般来说，中断处理程序提供的服务不是为当前进程所需的，如时钟中断、硬盘读写服务请求中断；
- 异常是由处理器正在执行现行指令而引起的，一条指令执行期间允许响应异常，异常处理程序提供的服务是为当前进程所用的。异常包括很多方面，有出错(fault)，也有陷入(trap)。

- 硬中断和软中断：
  - 中断和异常要通过硬件设施来产生中断请求，可看作硬中断。
  - 不必由硬件发信号而能引发的中断称软中断，软中断是利用硬件中断的概念，用软件方式进行模拟，实现宏观上的异步执行效果。
    - 软中断是由内核或进程对某个进程发出的中断信号，可看作内核与进程或进程与进程之间用来模拟硬中断的一种信号通信方式。

中断处理

- 机器故障中断事件的处理:事件是由硬件故障(例如，电源故障、主存储器故障)产生。中断处理能做的工作是：保护现场，防止故障蔓延，报告给操作员并提供故障信息以便维修和校正，对程序中所造成的破坏进行估价和恢复。
- 程序性中断事件的处理:采用中断续元处理来进行程序性中断事件的处理
- 外部中断事件的处理:时钟是操作系统进行调度工作的重要工具，例如让分时进程作时间片轮转、让实时进程
- 控制台中断事件的处理：操作员可以利用控制台开关请求操作系统工作，当使用控制台开关后，就产生一个控制台中断事件通知操作系统。操作系统处理这种中断就如同接受一条操作命令一样，转向处理操作命令的程序执行。
- I/O中断的处理

中断优先级和多重中断

- 中断的屏蔽：主机可允许或禁止某类中断的响应，如允许或禁止所有的I/O中断、外部中断、及某些程序性中断。有些中断是不能被禁止的，例如，计算机中的自愿性访管中断就不能被禁止。

- 多重中断事件的处理：中断正在进行处理期间，这时CPU又响应了新的中断事件，于是暂时停止正在运行的中断处理程序，转去执行新的中断处理程序，这就叫多重中断（又称中断嵌套）
  - 禁止再发生中断
  - 定义中断优先级
  - 响应并进行中断处理:

Threads

- A thread (or lightweight process) is a basic unit of CPU utilization; it consists of:
  - program counter
  - register set
  - stack space
- A thread shares with its peer threads its:
  - code section
  - data section
  - operating-system resources
  - collectively known as a job.
- A traditional or heavyweight process is equal to a job with one thread

Threads (Cont.)

- In a multiple threaded job, while one server thread is blocked and waiting, a second thread in the same job can run.
  - Cooperation of multiple threads in same job confers higher throughput and improved performance. 赋予

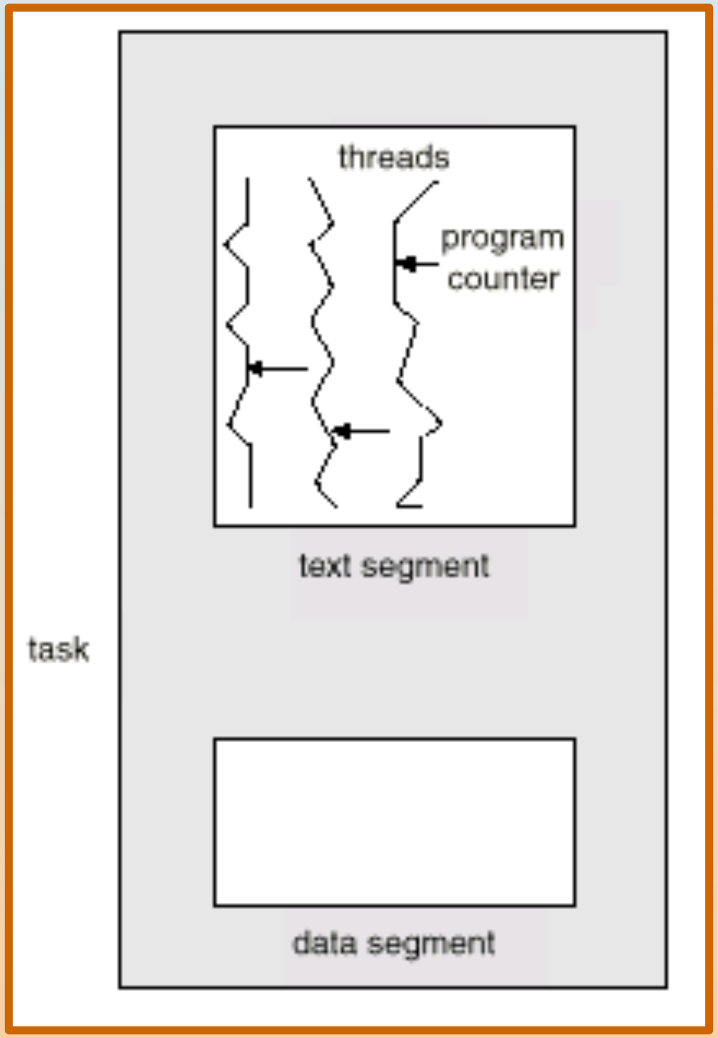
  吞吐量
  - Applications that require sharing a common buffer (i.e., producer-consumer) benefit from thread utilization.
- Threads provide a mechanism that allows sequential processes to make blocking system calls while also achieving parallelism.

  线程提供了一种机制，允许顺序进程阻止系统调用，同时实现并行性
- Kernel-supported threads (Mach and OS/2).

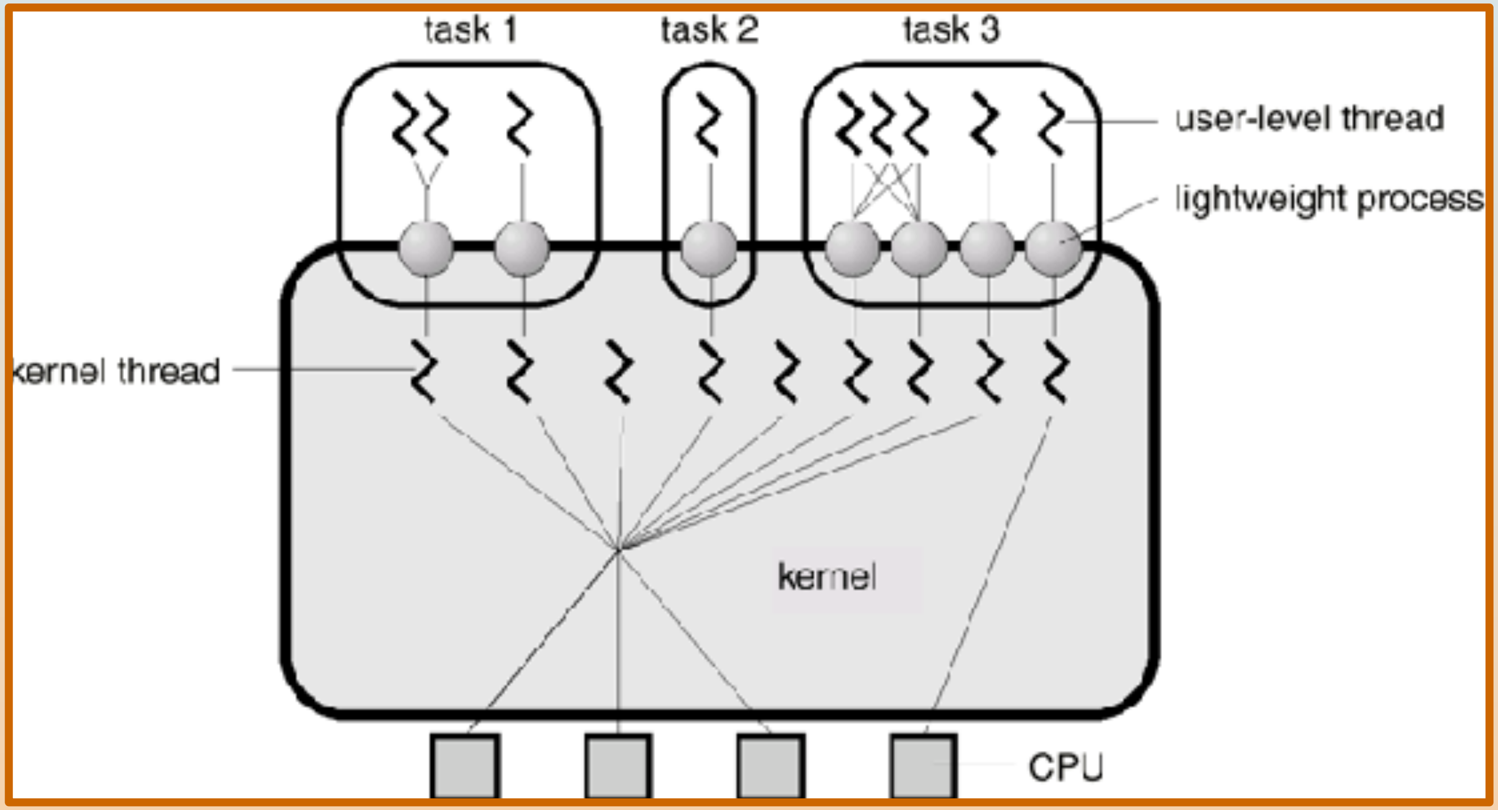
- User-level threads; supported above the kernel, via a set of library calls at the user level (Project Andrew from CMU).

- Hybrid approach implements both user-level and kernel-supported threads (Solaris 2).

Multiple Threads within a job



Solaris 2 Threads


task 1
task 2
task 3
user-level thread
lightweight process
kernel thread
kernel
CPU