Advanced Operating Systems
3.Processes/Threads.1

Chun-Han Lin (林均翰)

CSIE, NTNU

Key Points 3.1

- Chapter goals
- Concurrency
- Basic problems of concurrency

Goals

- **Processes and threads**
- **Fork and Exec**
- **Multithreading**
- **Inter process communication**
- **Scheduling (first take)**

Concurrency (1/2)

- Thread of execution
  - Independent fetch/decode/execute loop
  - Operate in some address spaces

- Uniprogramming: one thread at a time
  - MS/DOS, early Macintosh, batch processing
  - Easier for OS builders
  - Get rid of concurrency by defining it away
  - Does this make sense for personal computers?

Concurrency (2/2)

- **Multiprogramming: more than one thread at a time**
  - **Multics, UNIX/Linux, OS/2, Windows NT/2000/XP, Mac OS X**
  - **Often called multitasking, but multitasking has other meanings (talk about this later)**

- **Manycore $\Rightarrow$ multiprogramming, right?**



Basic Problems of Concurrency (1/2)

- **Basic problems of concurrency involves resources.**
  - **Hardware: single CPU, single DRAM, single I/O devices**
  - **Multiprogramming API: users think that they have exclusive access to shared resources.**

- **OS has to coordinate all activities.**
  - **Multiple users, I/O interrupts, …**
  - **How can it keep all these things straight?**



Basic Problems of Concurrency (2/2)

- Basic idea: virtual machine abstraction
  - Decompose a hard problem into simpler ones
  - Abstract the notion of an executing program
  - Then, worry about multiplexing these abstract machines

- Dijkstra did this for "THE system".
  - Few thousand lines vs 1 million lines in OS 360 (1K bugs)

Review 3.1

- Chapter goals
- Concurrency
- Basic problems of concurrency

Advanced Operating Systems
3.Processes/Threads.2

Chun-Han Lin (林均翰)
CSIE, NTNU

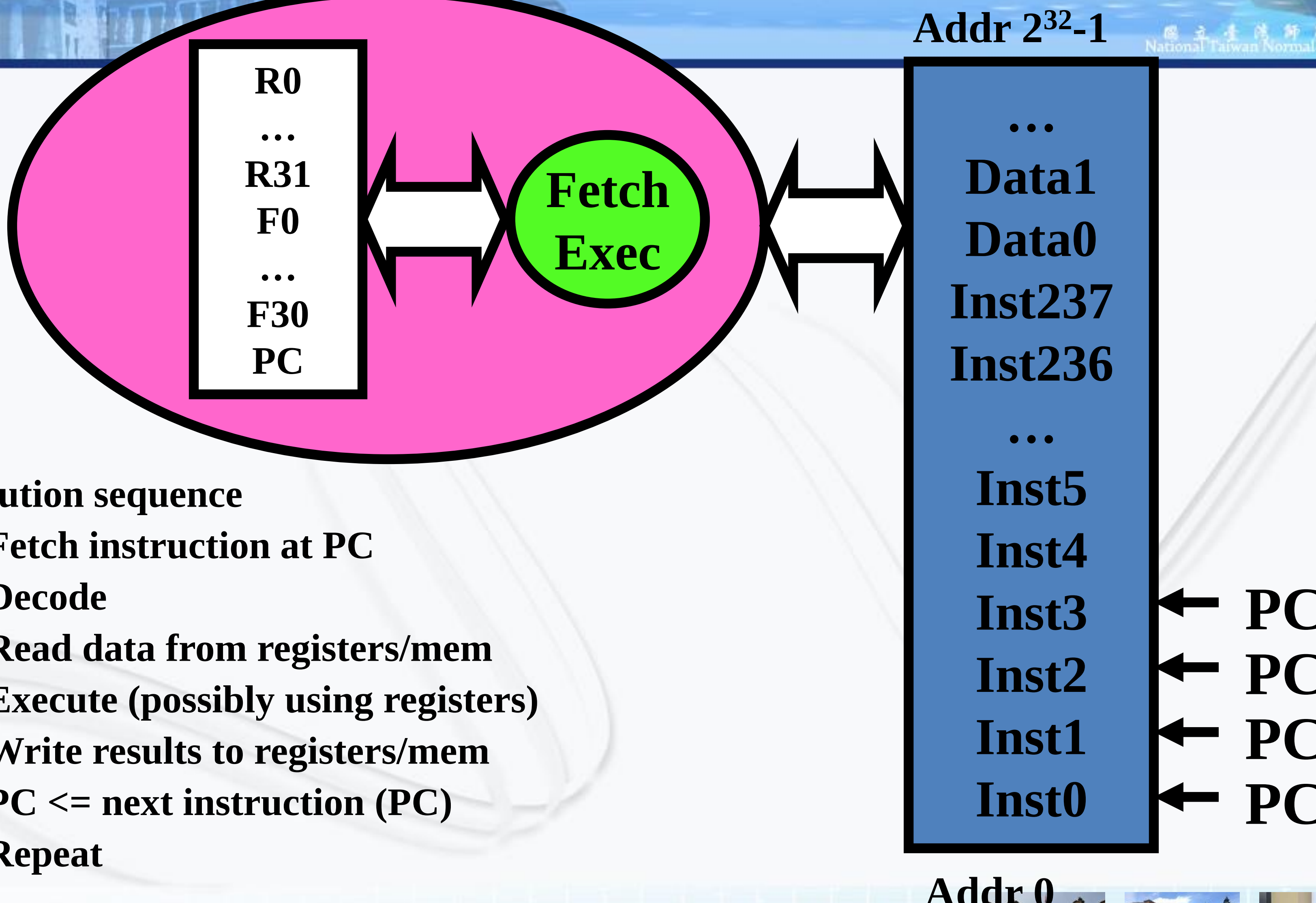


Key Points 3.2

- Recall: What happens during an execution?

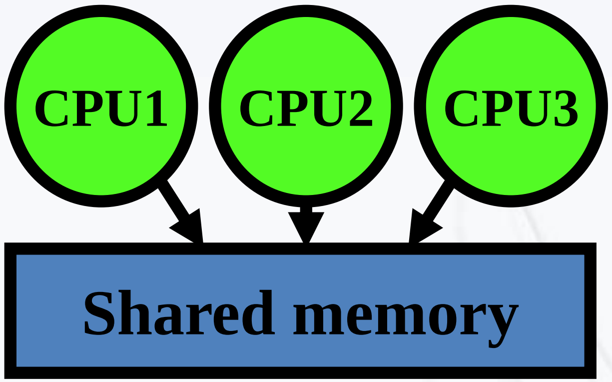
- How can we give the illusion of multiple processors?

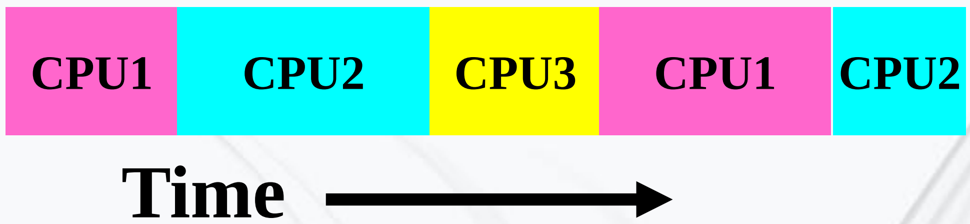
Recall: What happens during an execution?



- Execution sequence
  - Fetch instruction at PC
  - Decode
  - Read data from registers/mem
  - Execute (possibly using registers)
  - Write results to registers/mem
  - PC <= next instruction (PC)
  - Repeat

How can we give the illusion of multiple processors? (1/2)





- Assume a single processor
- How do we provide the illusion of multiple processors?
  - Multiplex in time!

How can we give the illusion of multiple processors? (2/2)

- Each virtual CPU needs a structure to hold …
  - Program Counter (PC), Stack Pointer (SP)
  - Registers (integer, floating point, others?)
  - Call the result a “thread” for now
- How to 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 a switch?
  - Timer, voluntary yield, I/O, other things

Review 3.2

- Recall: What happens during an execution?

- How can we give the illusion of multiple processors?

Advanced Operating Systems
3.Processes/Threads.3

Chun-Han Lin (林均翰)

CSIE, NTNU



Key Points 3.3

- **Properties of the simple multiprogramming technique**

- **What needs to be saved in modern x86?**

Properties of the Simple Multiprogramming Technique (1/2)

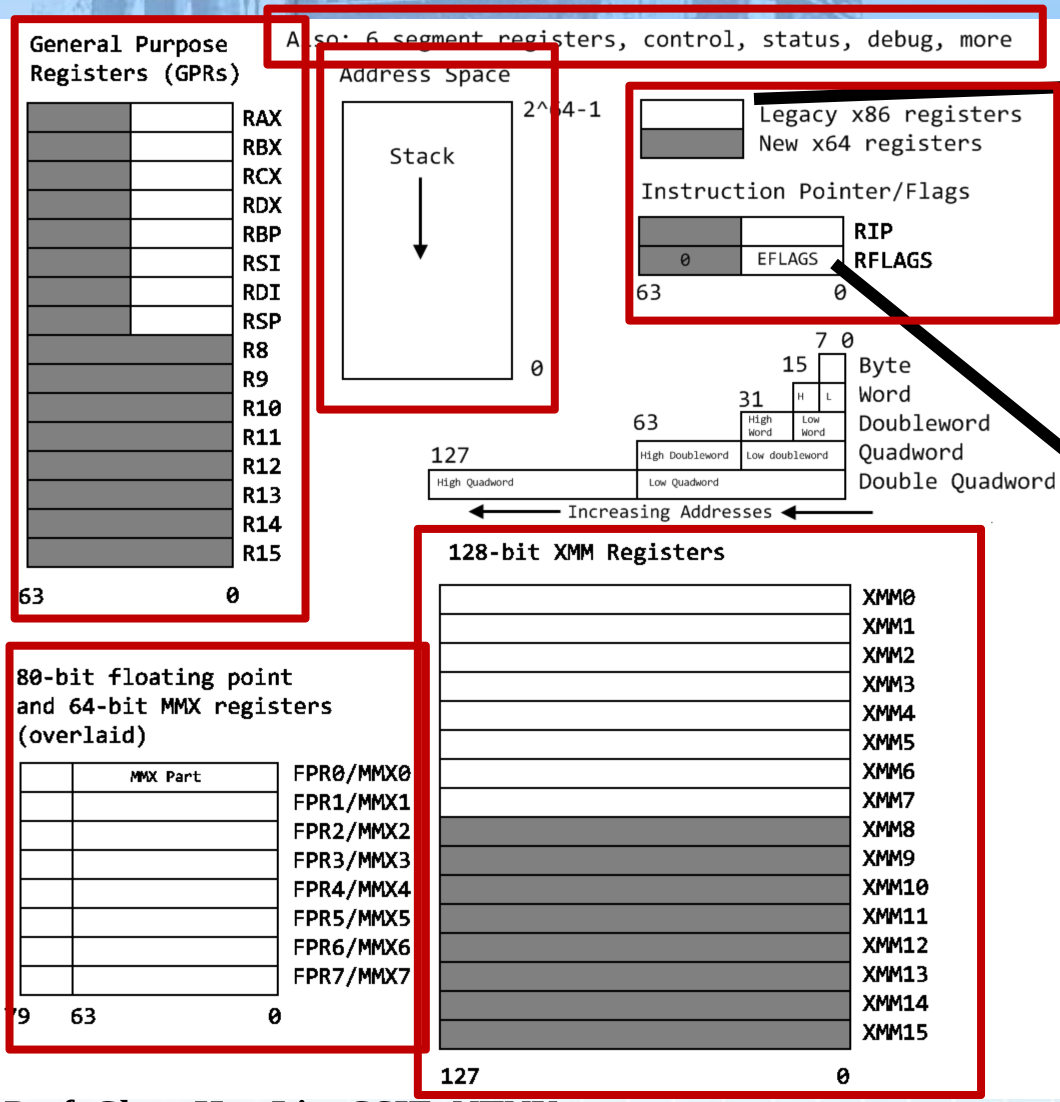
- All virtual CPU share same non-CPU resources.
  - I/O devices are the same.
  - Memory is the same.
- Consequence of the sharing
  - Each thread can access the data of every other thread. (good for sharing, bad for protection)
  - Threads can share instructions. (good for sharing, bad for protection)
  - Can threads overwrite OS functions?

Properties of the Simple Multiprogramming Technique (2/2)

- This (unprotected) model is common in
  - Embedded applications,
  - Windows 3.1/Machintosh (switch only with yield), and
  - Windows 95—ME? (switch with both yield and timer).

What needs to be saved in modern x86?

64-bit register set

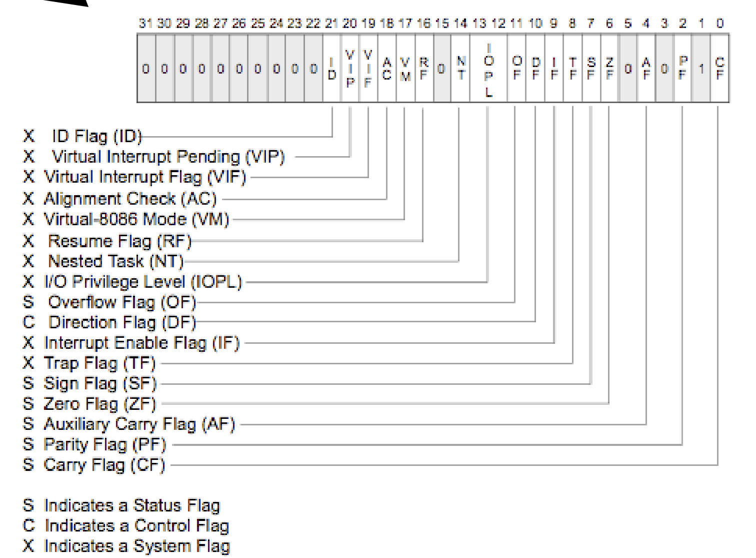


Traditional 32-bit subset

General-Purpose Registers

31 … 16	15 … 8	7 … 0	16-bit	32-bit
	AH	AL	AX	EAX
	BH	BL	BX	EBX
	CH	CL	CX	ECX
	DH	DL	DX	EDX
	BP			EBP
	SI			ESI
	DI			EDI
	SP			ESP

EFLAGS register



Review 3.3

- Properties of the simple multiprogramming technique
- What needs to be saved in modern x86?

Advanced Operating Systems
3.Processes/Threads.4

Chun-Han Lin (林均翰)

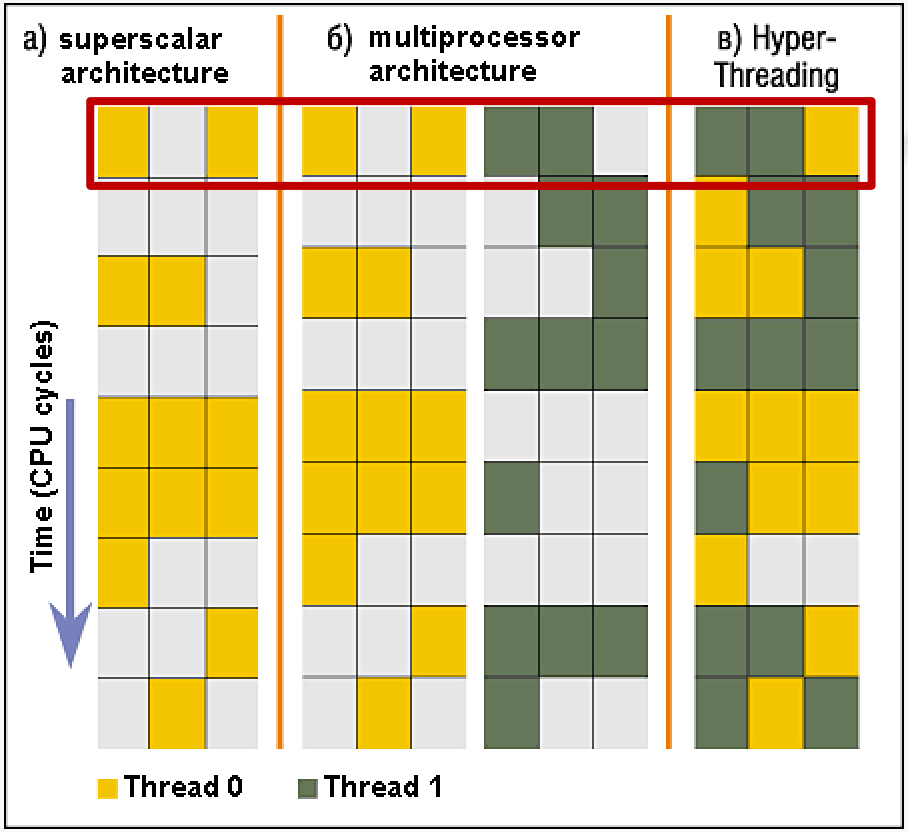
CSIE, NTNU

Key Points 3.4

- **Modern technique: SMT/Hyperthreading**

Modern Technique: SMT/Hyperthreading (1/2)

- **Hardware technique**
  - Exploit natural properties of superscalar processors to provide illusion of multiple processors
  - Higher utilization of processor resources



Modern Technique: SMT/Hyperthreading (2/2)

- Schedule each thread as if there were separate CPU
  - However, not linear speedup!
  - If we have multiprocessor, we should schedule each processor first.

- Original technique is called Simultaneous MultiThreading (SMT).
  - Alpha, SPARC, Pentium 4 (Hyperthreading), Power 5



Review 3.4

- **Modern technique: SMT/Hyperthreading**

Advanced Operating Systems
3.Processes/Threads.5

Chun-Han Lin (林均翰)

CSIE, NTNU



Key Points 3.5

- **How to protect threads from one another?**
- **Program's address space**

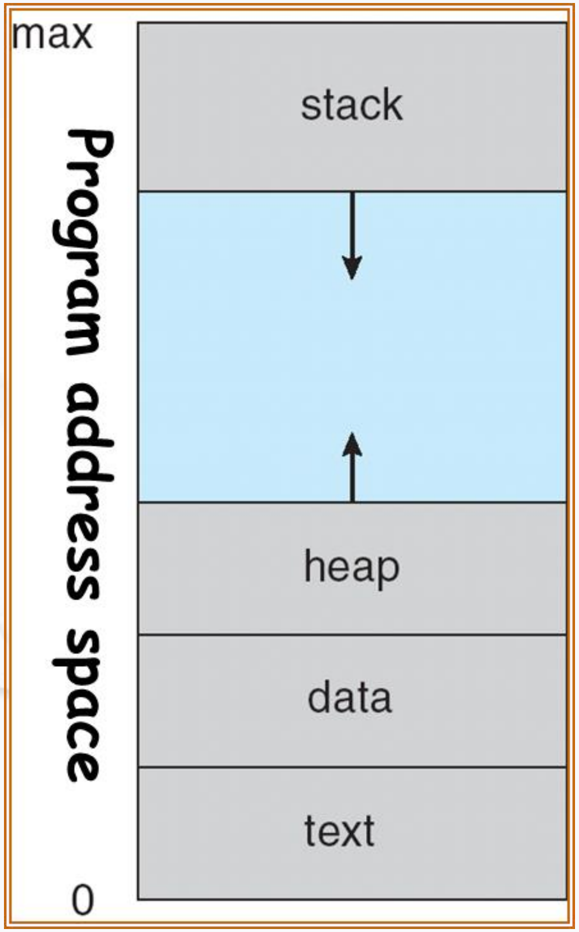
How to protect threads from one another?

- Need three important things

1. Protection of memory
   - Every task does not have access to all memory.
2. Protection of I/O devices
   - Every task does not have access to every device.
3. Protection of access to processors (preemptive switching from task to task).
   - Use timers
   - Must not be possible to disable a timer from user code

Program's Address Space

- **Address space = the set of accessible addresses + the state associated with them**
  - **For a 32-bit processor, there are 2^{32} = 4 billion addresses.**
- **What happens when you read or write to an address?**
  - **Perhaps nothing**
  - **Perhaps acts like a regular memory**
  - **Perhaps ignores the write**
  - **Perhaps causes an I/O operation (memory-mapped I/O)**
  - **Perhaps causes an exception (fault)**



Review 3.5

- How to protect threads from one another?

- Program's address space

Advanced Operating Systems
3.Processes/Threads.6

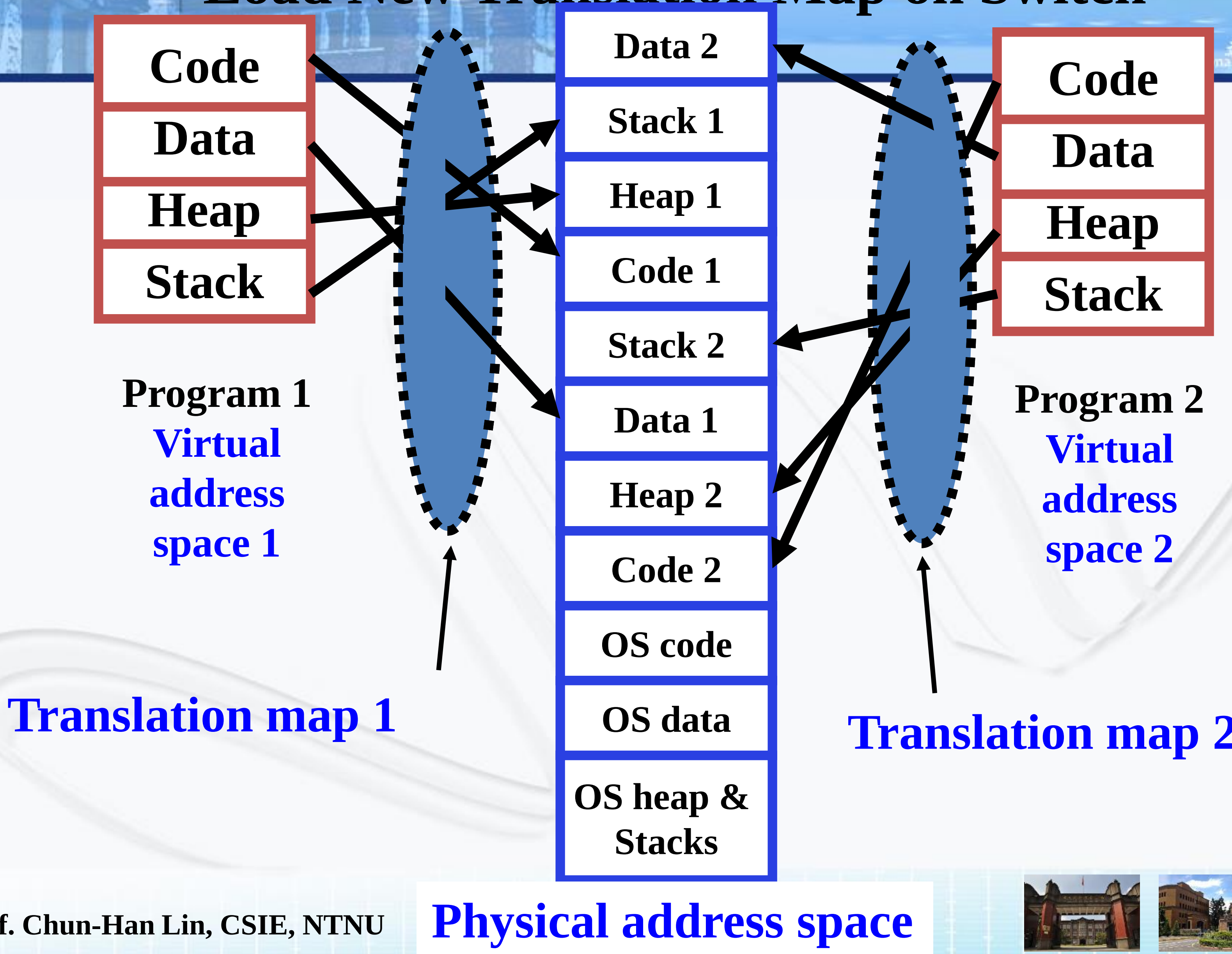
Chun-Han Lin (林均翰)
CSIE, NTNU

Key Points 3.6

- Providing illusion of separate address space: Load new translation map on switch

- x86 memory model with segmentation

Providing Illusion of Separate Address Space: Load New Translation Map on Switch



x86 Memory Model with Segmentation

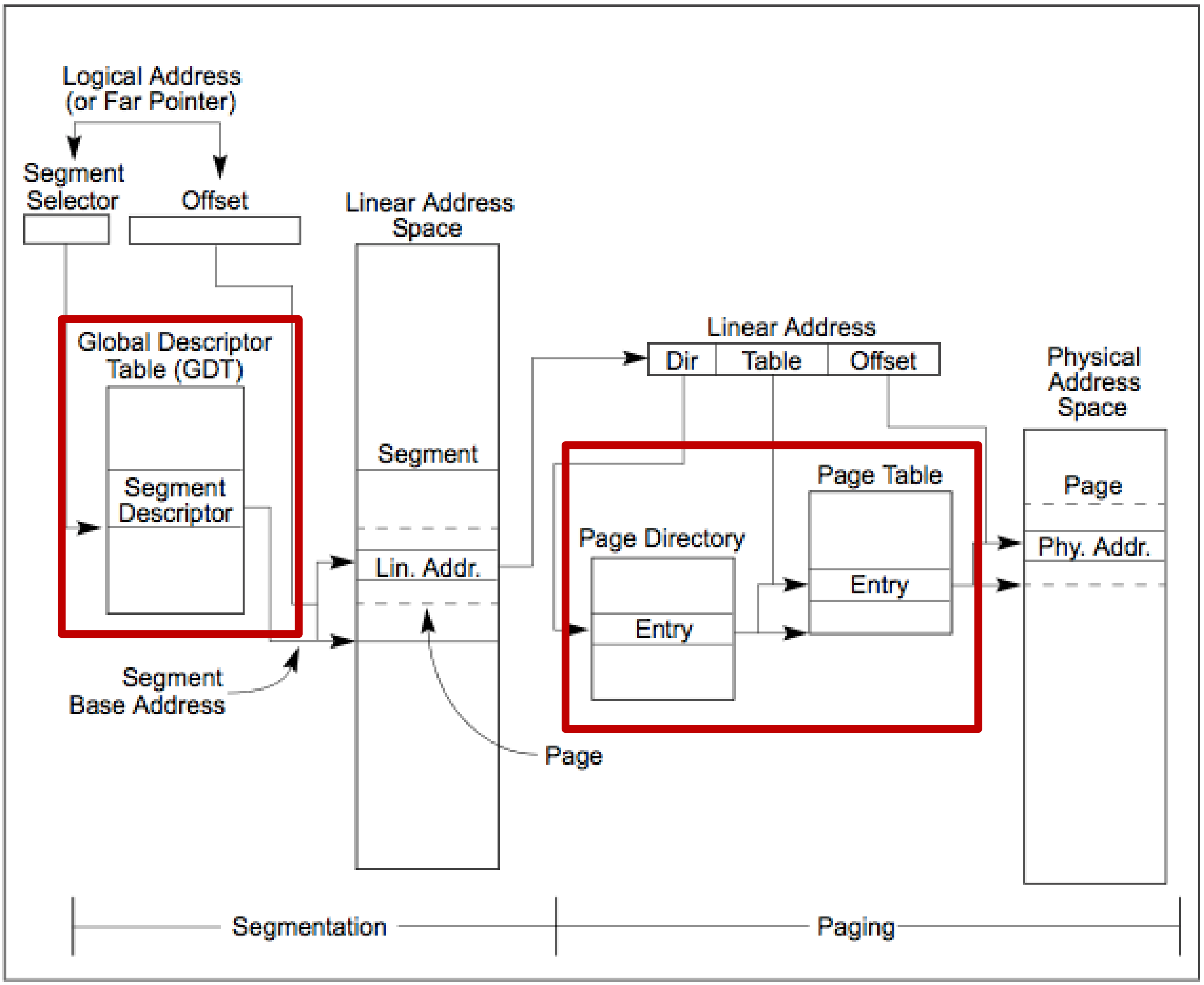


Figure 3-1. Segmentation and Paging

Six x86 Segment Registers

- CS: Code Segment
- SS: Stack Segment
  - Stack segments are data segments which must be read/write segments. Loading the SS register with a segment selector for a nonwritable data segment generates a general-protection exception (#GP).
- DS: Data Segment
- ES/FS/GS: Extra (usually data) Segment registers
  - FS and GS are used for thread-local storage/by glibc.
- The hidden part is like a cache so that segment descriptor information doesn’t have to be looked up each time.

Visible Part	Hidden Part	
Segment Selector	Base Address, Limit, Access Information	CS
		SS
		DS
		ES
		FS
		GS

Review 3.6

- Providing illusion of separate address space: Load new translation map on switch

- x86 memory model with segmentation

Advanced Operating Systems
3.Processes/Threads.7

Chun-Han Lin (林均翰)

CSIE, NTNU

Key Points 3.7

- UNIX process
- Modern "lightweight" process with threads

UNIX Process (1/2)

- **A process is an OS abstraction to represent what is needed to run a single program.**
  - **Originally: a single, sequential stream of execution in its own address space**
  - **A modern process has multiple threads in same address space!**
- **Two parts**

1. **Sequential program execution streams**
   - **Code is executed as one or more sequential stream of execution (threads).**
   - **Each thread includes its own state of CPU registers.**
   - **Threads are either multiplexed in software (OS) or hardware (SMT/hyperthreading).**

UNIX Process (2/2)

2. **Protected resources**
    - Main memory state (contents of address space)
    - I/O state (i.e. file descriptors)

- This is a virtual machine abstraction.
    - Someone might say that the only point of an OS is to support a clean process abstraction.

Modern "Lightweight" Process with Threads (1/2)

- **A thread is a sequential execution stream within a process.**
  - Sometimes called a lightweight process
  - A process still contains a single address space.
  - No protection between threads

- **Multithreading is a single program made up of a number of different concurrent activities.**
  - Sometimes called multitasking, as in Ada

Modern "Lightweight" Process with Threads (2/2)

- Why separate the concept of a thread from that of a process?
  - Discuss the thread part of a process (concurrency)
  - Separate from the address space (protection)
  - Heavyweight process ≡ process with one thread
- Linux confuses this model a bit.
  - Processes and threads are the same.
  - Really means that threads are managed separately and can share a variety of resources (such as address spaces)
  - Threads are related to one another in fashion similar to processes with threads within.

Review 3.7

- **UNIX process**
- **Modern “lightweight” process with threads**

Advanced Operating Systems
3.Processes/Threads.8

Chun-Han Lin (林均翰)

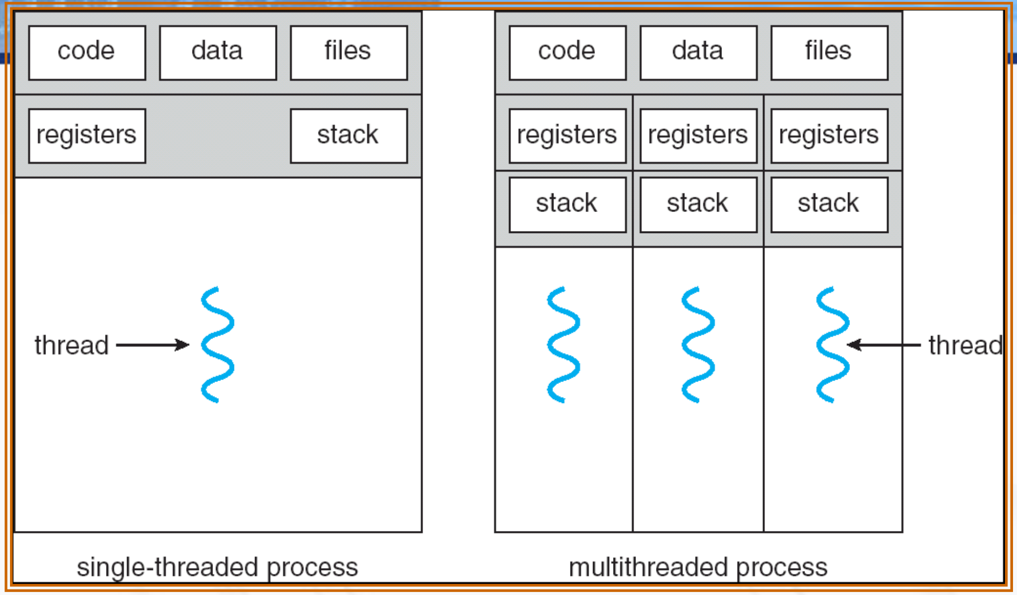
CSIE, NTNU



Key Points 3.8

- **Single and multithreaded processes**
- **Process comprises**
- **Processes – Address space**

Single and Multithreaded Processes

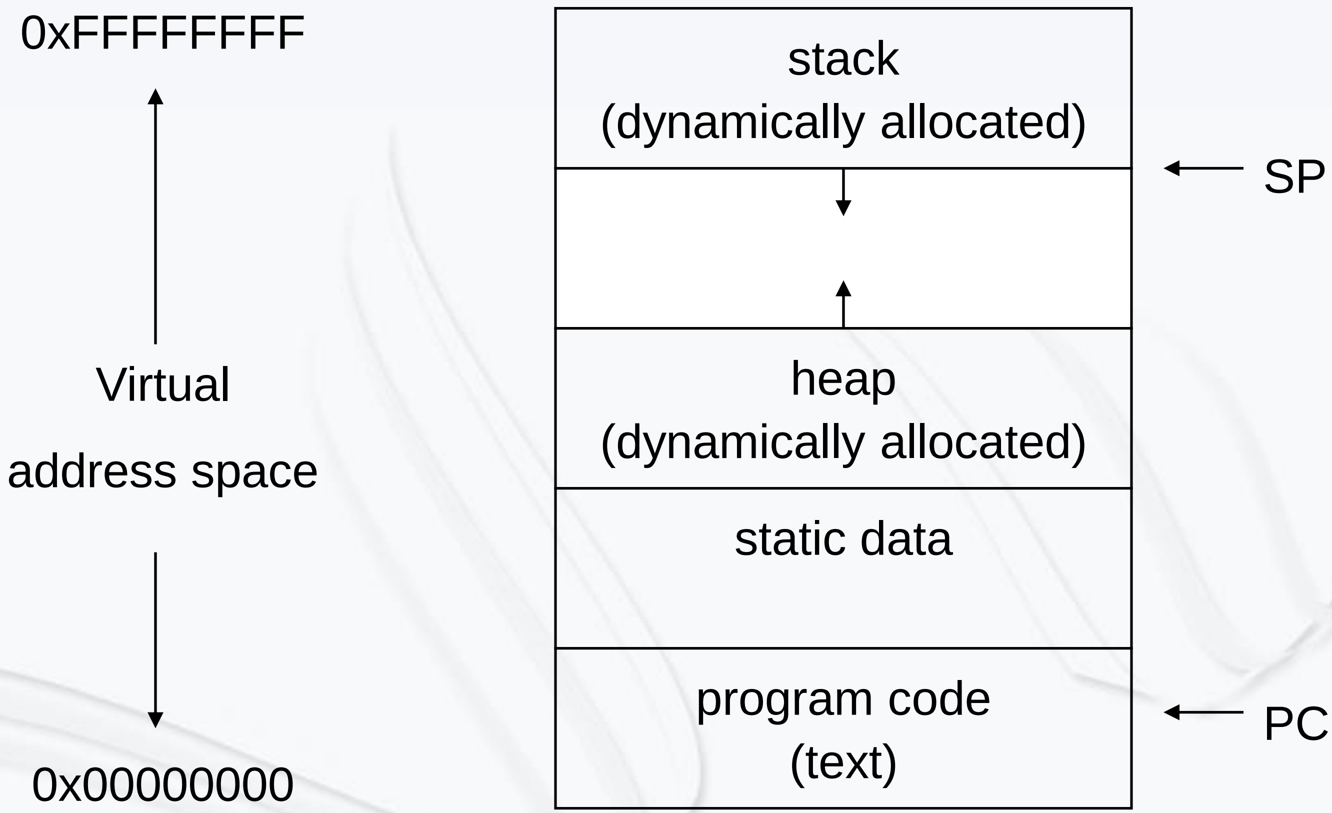


- **Threads encapsulate concurrency: active component**
  - **How do thread stacks stay separate from one another? They may not!**
- **Address spaces encapsulate protection: passive part**
  - **Keep buggy program from trashing the system**
- **Why have multiple threads per address space?**

Process Comprises

- What is in a process?
    - An address space, usually protected and virtual, is mapped into memory.
    - The code for the running program
    - The data for the running program
    - An execution stack and stack pointer (SP); also heap
    - The program counter (PC)
    - A set of processor registers: general purpose and status
    - A set of system resources
        - Files, network connections, pipes, …
        - Privileges, (human) user association, …
        - Personalities (Linux)
    - …

Processes – Address Space



See also Silbershatz, figure 3.1

Review 3.8

- **Single and multithreaded processes**

- **Process comprises**

- **Processes – Address space**

Advanced Operating Systems
3.Processes/Threads.9

Chun-Han Lin (林均翰)

CSIE, NTNU

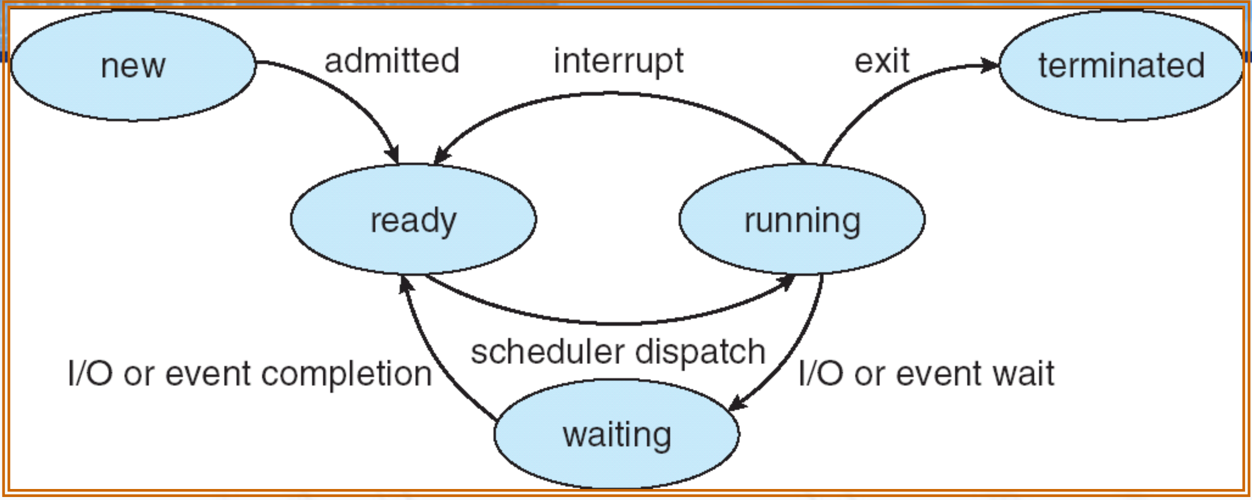
Key Points 3.9

- **Process – Starting and ending**
- **Lifecycle of process**
- **How do we multiplex processes?**

Process – Starting and Ending

- **Processes are created …**
  - **When the system boots**
  - **By the actions of another process (more later)**
  - **By the actions of a user**
  - **By the actions of a batch manager**
- **Processes terminate …**
  - **Normally – exit**
  - **Voluntarily on an error**
  - **Involuntarily on an error**
  - **Terminated (killed) by actions of a user or a process**

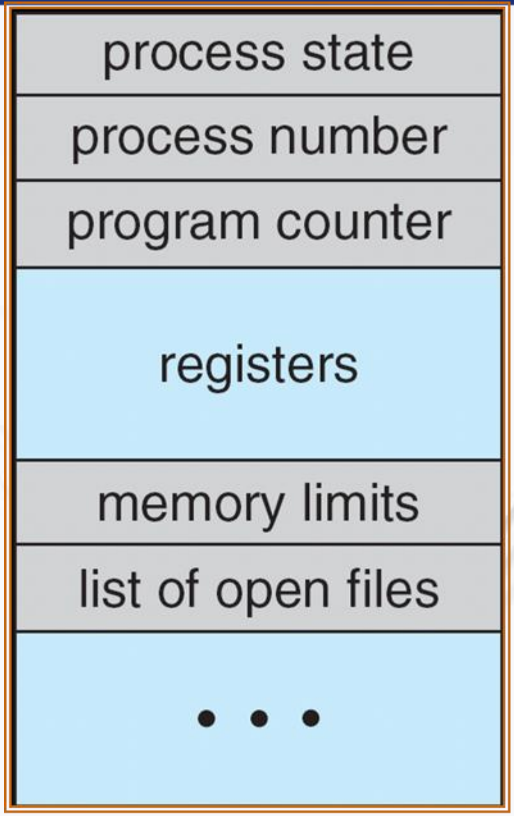
Lifecycle of Process (or Thread)



- As a process or thread executes, it changes state.
  - New: the process is being created.
  - Ready: the process is waiting to run.
  - Running: instructions are being executed.
  - Waiting: the process is waiting for some events to occur.
    - Can be interruptible or non-interruptible.
  - Terminated: the process has finished execution stays as zombie until relays results to parent.

How do we multiplex processes? (1/2)

- **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 is active at a time.**



Process Control Block

How do we multiplex processes? (2/2)

- **Give out CPU time to different processes (scheduling)**
  - Only one process is running at a time.
  - Give more time to important processes
- **Give pieces of resources to different processes (protection)**
  - Control 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

Review 3.9

- **Process – Starting and ending**
- **Lifecycle of process**
- **How do we multiplex processes?**