ECE437/CS481

M02A: PROCESSES & THREADS PROCESSES CONCEPTS

Chapter 3.1-3.2

Xiang Sun

The University of New Mexico

Programs & Processes

- **What is a program?**
  - "A program contains code & static data stored in a file" (von Neumann architecture)
    - ✓ Source code – before compilation
    - ✓ Binary code – ready to be loaded

- **What is a process?**
  - "A process is a program that is being executed"
    - ✓ A program can execute many times, so corresponding to many processes
    - ✓ Different process have their own address spaces & process context

Process Context

❑Process context includes

- Execution information
  - ✓ CPU register set image (e.g., Program Counter (PC), Stack Pointer (SP), Instruction Register (IR), Program Status Word (PSW) and other general processor registers)
  - ✓ Process stack - containing temporary data (e.g. subroutine parameters, temporary variables, return addresses)

Register	Accumulator	Counter	Data	Base	Stack Pointer	Stack Base Pointer	Source	Destination
64-bit	RAX	RCX	RDX	RBX	RSP	RBP	RSI	RDI
32-bit	EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI
16-bit	AX	CX	DX	BX	SP	BP	SI	DI
8-bit	AH AL	CH CL	DH DL	BH BL				

General-Purpose Registers

Process Context

- Process context includes

  - Environment information

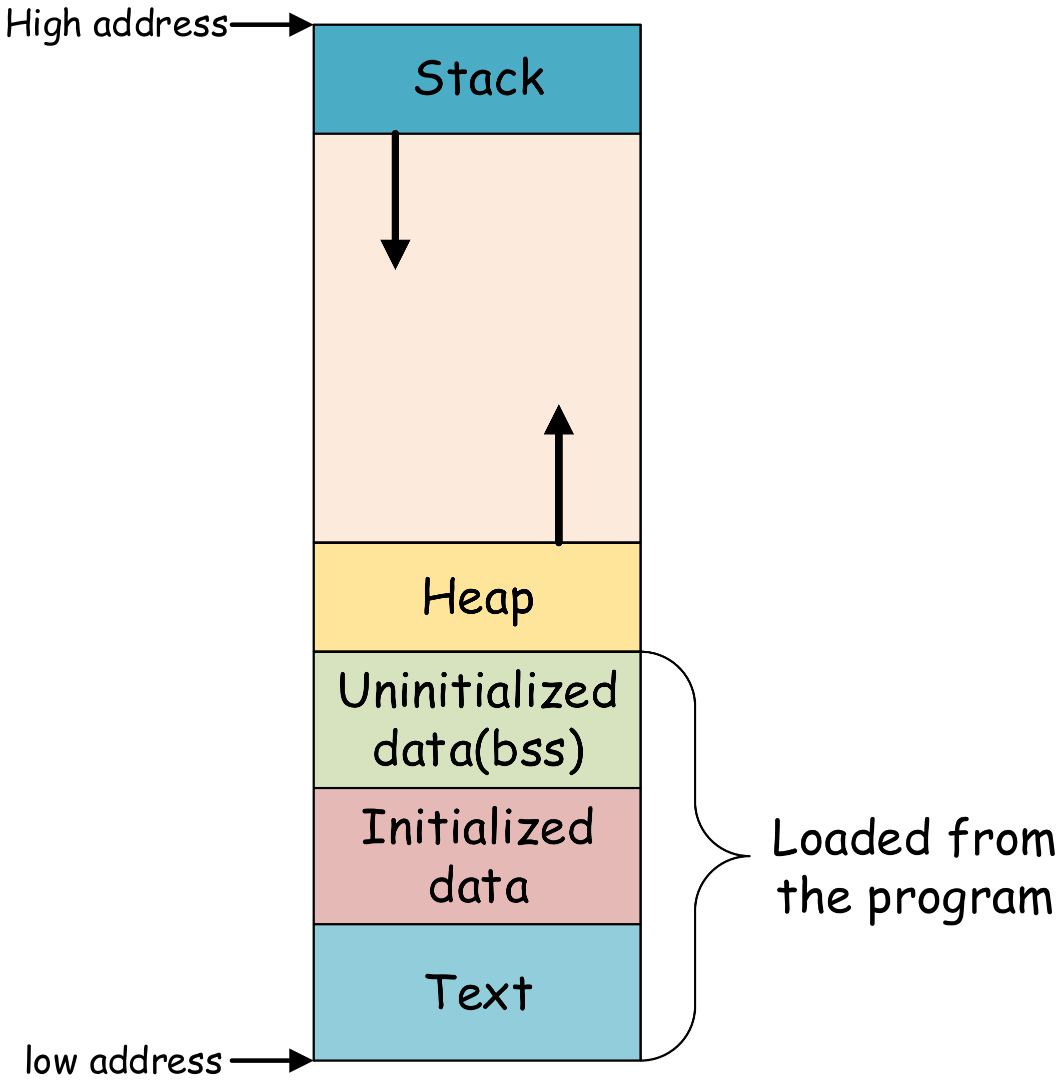
    - ✓ Memory address space or memory map, which is the various regions of memory that have been allocated to the process
    - ✓ Resources, e.g., Open file table and Communication channels & I/O devises

  - System attributes

    - ✓ process identification or Process ID (PID)
    - ✓ process state information, e.g., waiting, ready, etc.
    - ✓ process control information

Process Memory Map

- Process's memory map in Unix/Linux
  - Text (code): the machine instructions to be executed. Usually, it stays unmodified over the lifecycle of the process.
  - Initialized data
  - Uninitialized data
  - Heap: dynamically allocated memory
    - ✓ grow via a system call to request more memory
  - Stack: memory space to manage function calls, returns, parameter passing, and local variables.
    - ✓ grow and shrink as the depth of function calls varies



Process Memory Map

❑ Initialized data vs Uninitialized data

```
#include <stdio.h>

int main(void)
{
    return 0;
}
```

text	data	bss	dec	hex
960	248	8	1216	4c0

```
#include <stdio.h>

int global; /* Uninitialized variable stored in bss*/

int main(void)
{
    return 0;
}
```

text	data	bss	dec	hex
960	248	12	1220	4c4

Process Memory Map

❑ Initialized data vs Uninitialized data

```
#include <stdio.h>

int global; /* Uninitialized variable stored in bss*/
int main(void)
{
 static int i; /* Uninitialized static variable stored in bss */
  return 0;
}
```

text	data	bss	dec	hex
960	248	16	1224	4c8

```
#include <stdio.h>

int global; /* Uninitialized variable stored in bss*/
int main(void)
{
    static int i = 100; /* Initialized static variable stored in DS*/
    return 0;
}
```

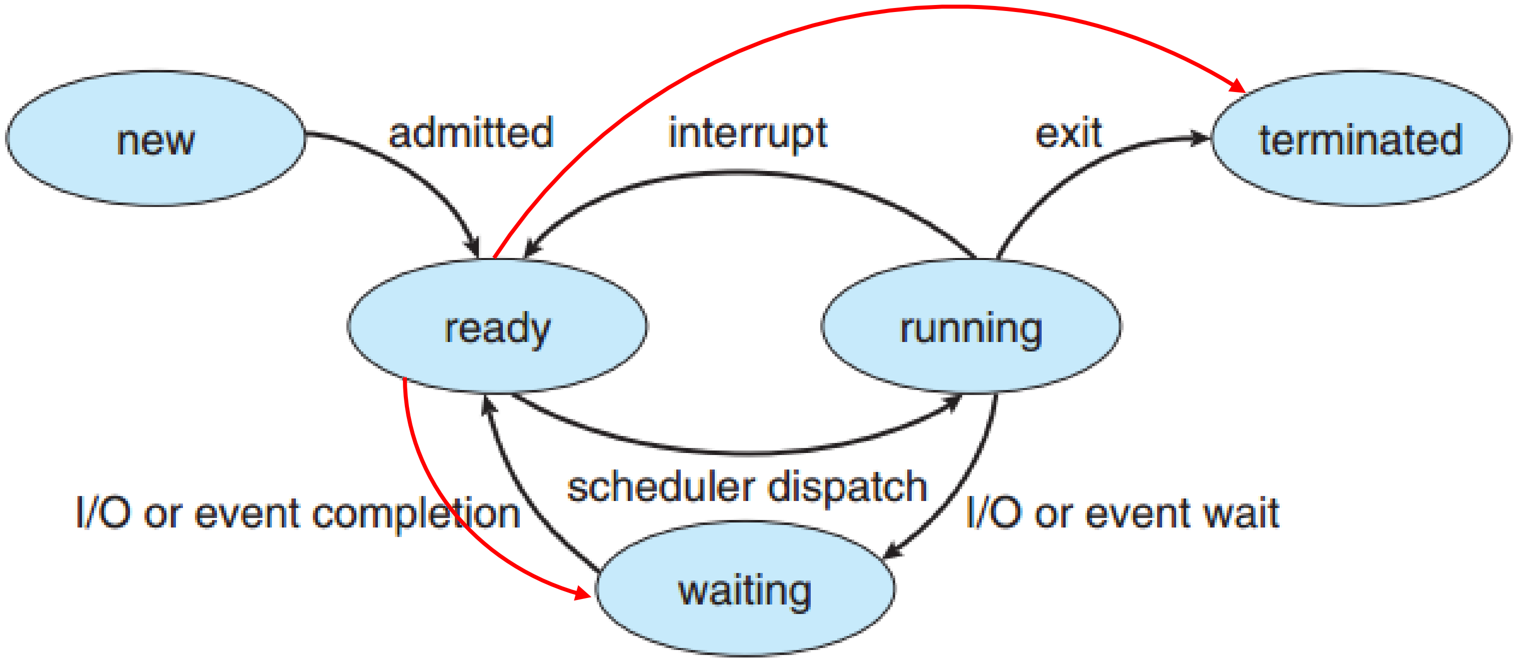
text	data	bss	dec	hex
960	252	12	1224	4c8

Process State

- **New** (Start)
  - The process is being created but has **not yet admitted** to the pool of executable processes
    - ✓ Admit --- go to Ready state: The OS will move a process to the Ready state when it can take an additional process.
- **Terminated** (Exit):
  - The process has finished execution.
- **Ready**
  - The process is waiting to be assigned to a processor.
- **Running**
  - The process is currently being executed.
  - If we assume a single processor computer, at most one process can be in this state at a time. But not true for today's multicore system。
- **Waiting** (Blocked)
  - The process is waiting for some event to occur.

Process State

- Ready→Running/Waiting/Terminated
  - Scheduler dispatch --- go to the Running state
  - Suspend --- go to the Waiting state
  - Kill --- go to the Terminated state

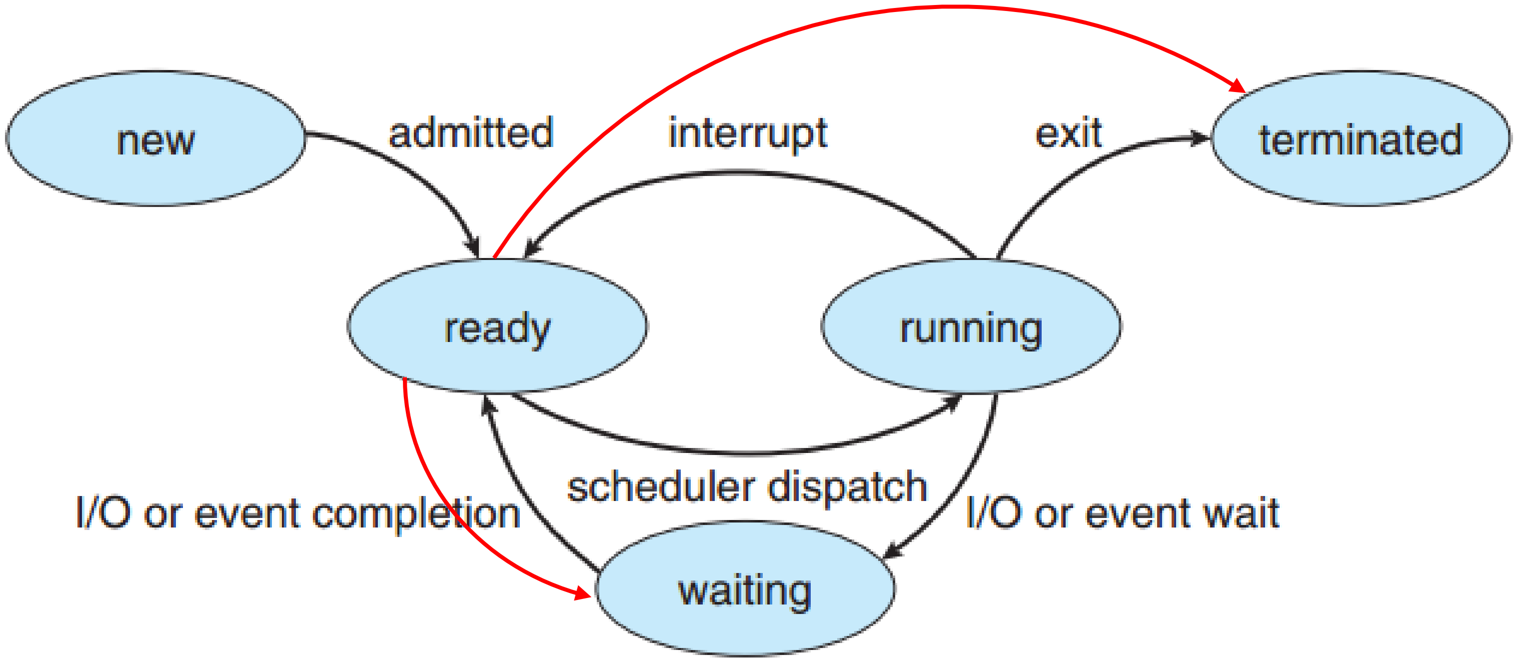


Process State

- Running→Read/Waiting/Terminated
  - Yield (i.e., the running process voluntarily release control of the processor) --- go to the Ready state
  - Time slice is up (i.e., the running process has reached the maximum allowable time) --- go to the Ready state
  - Arrival of high priority process (i.e., the running process will be preempted due to its lower priority) --- go to the Ready state
  - I/O request --- go to the Waiting state
  - Suspend --- go to the Waiting state
  - Terminate --- go to the Terminated state

Process State

- Waiting→Ready/Terminated
  - I/O complete, wakeup --- go to the Ready state
  - Kill --- go to the Exit /terminated state



Process State

❑ Process Control Block (PCB)

- Each process is represented in the OS by a PCB
- An important data structure to keep the context of a process
  - ✓ PID: a unique integer as process ID
  - ✓ UID: user ID who is executing this process
  - ✓ Process state: the current process state
  - ✓ Event: any event for which the process may be waiting
  - ✓ Memory management info: information of page table, memory limits, segment table, etc.
  - ✓ CPU registers: CPU registers where process need to be stored for execution.
  - ✓ Scheduling priority : parameters for the process scheduling
  - ✓ Program counter: a pointer to the address of the next instruction to be executed for this process.
  - ✓ Accounting into : timing, usage, ...
  - ✓ IO status information: a list of I/O devices allocated to the process.
- In linux:
  - ✓ defined in task_struct (include/linux/sched.h)---over 95 fields!!!

❑ Process list

- ➢ Each existing process has one entry, the entry itself is PCB
- ➢ Given a process ID to find its PCB through the process table, needed by kill, wakeup, suspend...

```
TASKS 227 (578 thr), 1 run, 225 slp, 1 oth sorted automatically by cpu_percent

 CPU%  MEM%  VIRT   RES   PID USER        NI S    TIME+ IOR/s IOW/s Command
  4.4   0.2 79.0M 15.2M 27889 nicolargo    0 R  0:03.24     0     0 /home/nicolargo/virtualenvs/glances-develop/bin/
  4.1   3.6  676M  284M  1107 root         0 S  5:09.56     0     0 /usr/bin/X :0 -background none -verbose -auth /v
  1.6   0.9  717M 70.8M 22440 nicolargo    0 S  1:16.40     0     0 /usr/lib/firefox/plugin-container /usr/lib/flash
  1.6   4.7 2.10G  371M 22870 nicolargo    0 S 35:31.50     0     0 /usr/bin/gnome-shell
  1.3   0.6 1.02G 47.2M  4089 nicolargo    0 S  0:49.50     0     0 /usr/bin/python /usr/bin/terminator
  0.3   0.3  386M 27.5M  1982 nicolargo    0 S  8:37.84     0     0 /usr/bin/ibus-daemon --daemonize --xim
  0.3   2.2 1.64G  176M  7042 nicolargo    0 S 23:44.30     0     0 vlc
  0.3   6.5 1.98G  515M  8411 nicolargo    0 S  2:30.55     0     0 /usr/bin/perl /usr/bin/shutter
  0.3   0.0     0     0 19741 root         0 S  0:01.75     0     0 kworker/0:0
  0.3   0.0     0     0 26267 root         0 S  0:00.47     0     0 kworker/2:1
  0.3   0.0     0     0 27127 root         0 S  0:00.11     0     0 kworker/u16:0
  0.0   0.1 36.5M 6.45M     1 root         0 S  0:07.73     0     0 /sbin/init
  0.0   0.0     0     0     2 root         0 S  0:00.80     0     0 kthreadd
  0.0   0.0     0     0     3 root         0 S  0:01.32     0     0 ksoftirqd/0
  0.0   0.0     0     0     5 root       -20 S  0:00.00     0     0 kworker/0:0H
  0.0   0.0     0     0     7 root         0 S  1:05.30     0     0 rcu_sched
```

State Queues

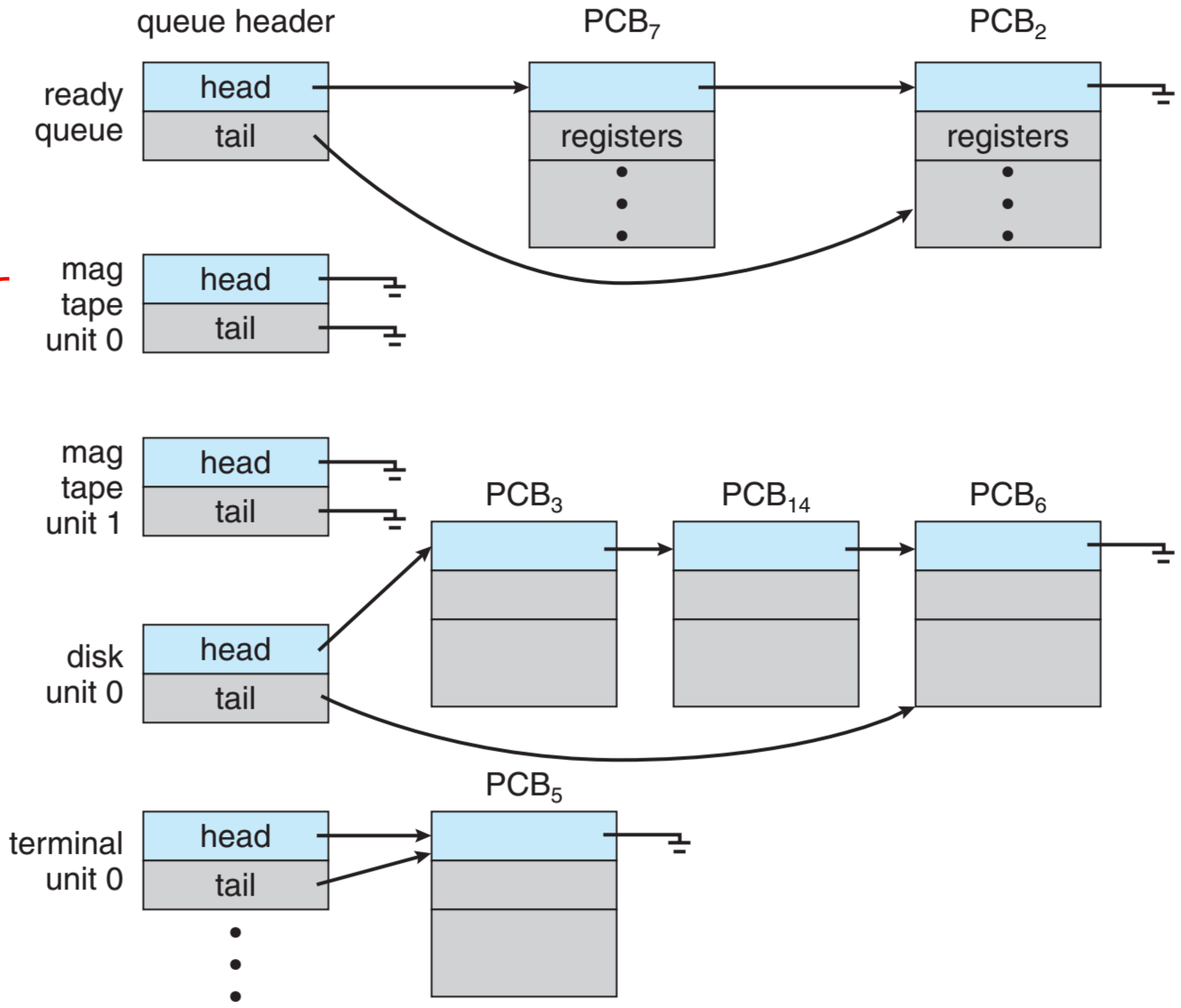
- The OS maintains a collection of queues that represent the state of all processes in the system
  - Typically, one queue stands for a state, e.g., ready, waiting, …
  - Each PCB is queued onto a specific state queue based on its status.
  - Once a process changes state, its PCB is unlinked from one queue, and linked onto another.

State Queues

- There may be many waiting queues, one for each type of wait (particular device, timer, message, …)
- Running queue? Terminated queue?

Ready queue

Waiting queue



Process Context Switch

❑ Process context switch

---Switching of the CPU from process A to another, requiring storing the state of process A such that it can be restored and execution resumed from the same point later.

➢ Why need context switch

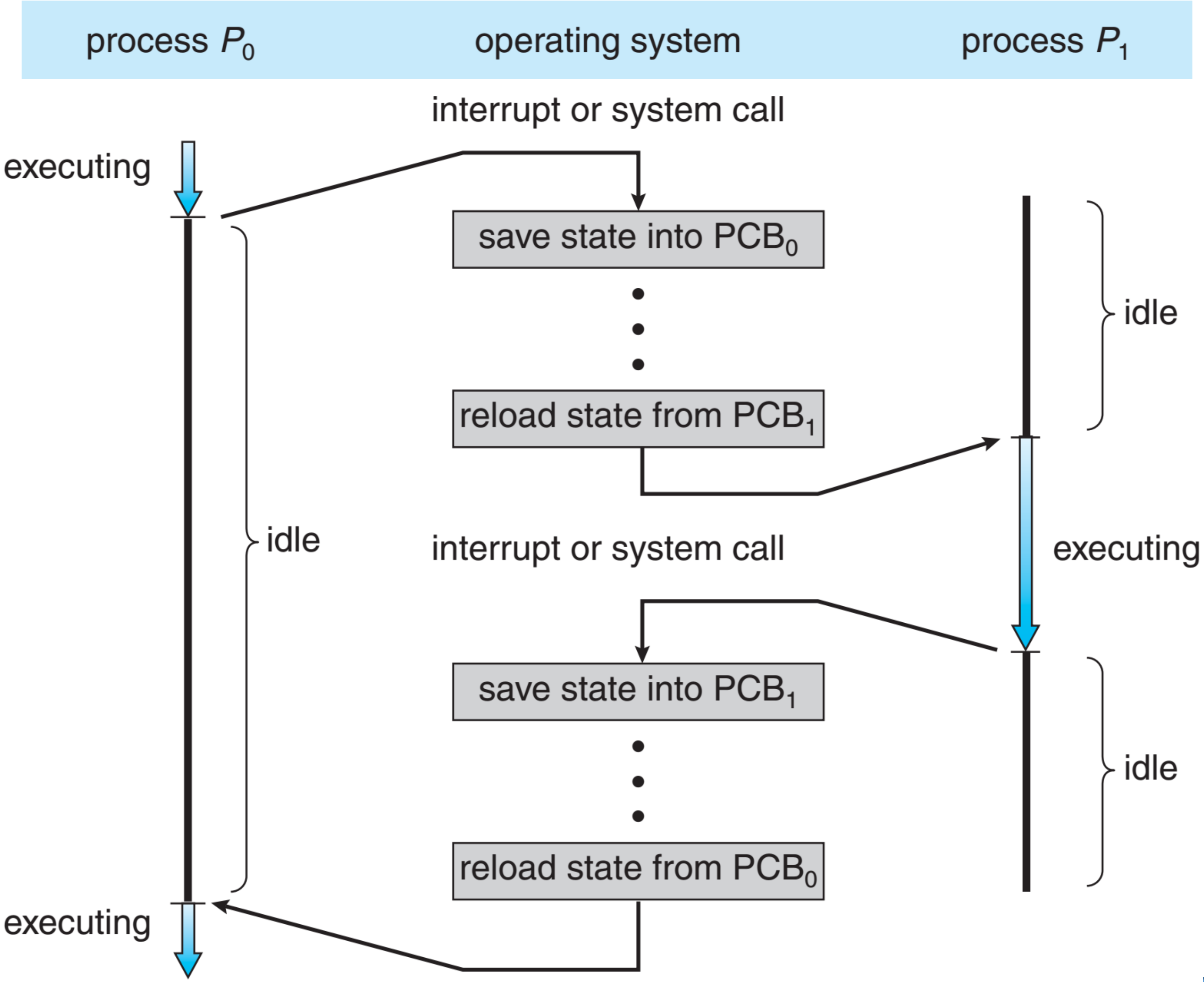
--- improve CPU utilization

➢ How to switch the process context --- Dispatch

1. stopping the current process's computation
2. saving enough information about its execution context
3. changing process state and putting it back to corresponding queues
4. selecting another process from the ready queue
5. changing the newly selected process' state
6. restarting computation of the new process

Process Context Switch

- Switching between processes
- Context of a process represented in the process's PCB
- Context-switch time is the overhead; the system does no useful work while switching
- Tricky: context switch can improve the CPU utilization; however, context switch may incur overheads which may reduce CPU utilization.



Process Context Switch

- When to do context switch
  - I/O request
  - CPU time slice is up
  - Arrival of higher priority process
  - Yield by process itself
  - Termination of process
    - ✓ processes may terminate itself by using an exit() system call
    - ✓ one process may terminate another, if given appropriate permissions

- Scheduler specifies the policy used to select a process **from the ready queue to run next**