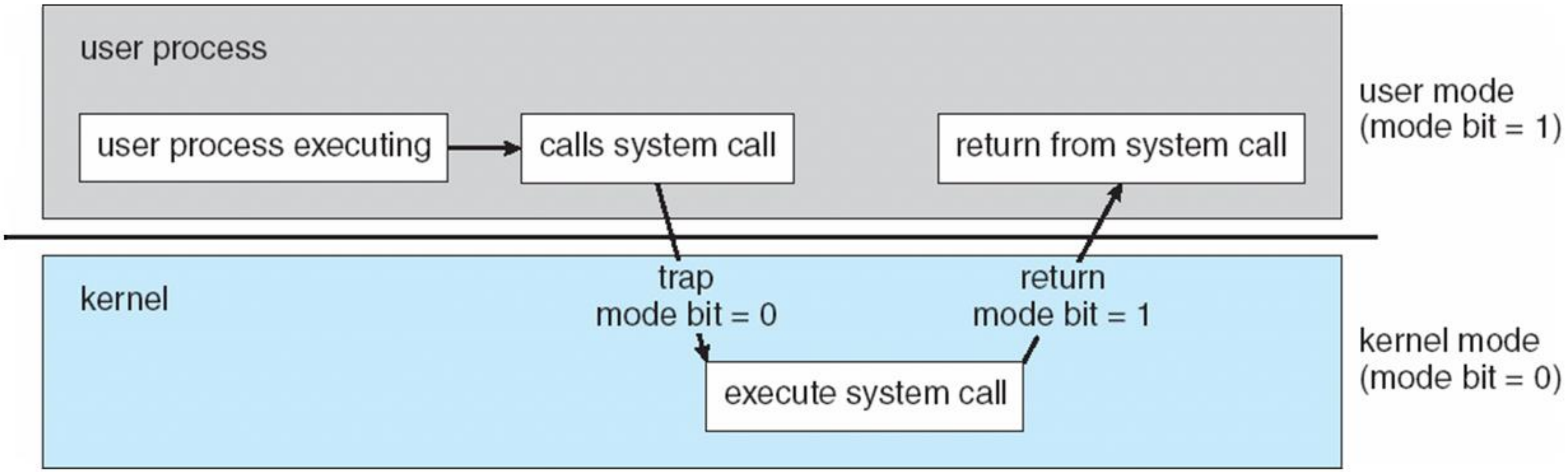
OS Organization

- OS is a *sleeping beauty*
  - User programs when scheduled run directly on the CPU
  - OS runs in response to “events”
- Privileged Instructions
  - OS must have exclusive access to hardware and critical data structures
- User mode & Kernel mode
  - “Mode” kept in a status bit in a protected control register
  - User programs execute in user mode
  - OS executes in kernel mode
  - CPU checks mode bit when protected instruction executes
  - Attempts to execute in user mode trap to OS

Mode Switching

- External Interrupt
  - Timer, I/O interrupts
- Internal Trap (faults)
  - Page fault, Invalid operation, ...
- System Call
  - I/O operations (file open), fork, …

mode switching



Events

- Interrupts, exceptions/faults, system calls, etc.

	Unexpected	Deliberate
Synchronous	fault	syscall trap
Asynchronous	interrupt	signal

- Hardware detects and reports “exceptional” conditions
  - Page fault
  - Memory access violation (unaligned, permission, not mapped, bounds…)
  - Illegal instruction
  - Divide by zero
- Some faults are handled by “fixing” the exceptional condition and returning to the faulting context
- The kernel may handle *unrecoverable* faults by killing the user process

Multiple choices

- Which of the following is true about the OS kernel?
  - A. It executes as a process.
  - B. It is always actively consuming CPU cycles in support of other processes.
  - C. It should execute as little as possible.
  - D. A & B
  - E. B & C

True/False

- Mode switching from kernel to user is triggered by interrupts, traps, and system calls.

True/False

- Segmentation fault is not the type of events handled by the OS

True/False

- Interrupts can be enabled or disabled by user-level processes for synchronization purpose.

True/False

- Page faults are unrecoverable faults

Short answer

- Faults and interrupts are both unexpected events, but faults are said to be synchronous. What is the meaning of “synchronous” here?

xv6

- xv6 is MIT's re-implementation of Unix v6
  - Written in ANSI C
  - Runs on RISC-V and x86
    - We will use the RISC-V version with the QEMU simulator
  - Smaller than v6
  - Preserve basic structure (processes, files, pipes. etc.)
  - Runs on multicores
  - Got paging support in 2011

Ken Thompson & Dennis Ritchie, 1975

Multiple choices

- Which of the following is true about xv6?
  A. xv6 is written in Modula 3
  B. xv6 is an emulator
  C. xv6 supports virtual memory
  D. xv6 does not support 64-bit architectures

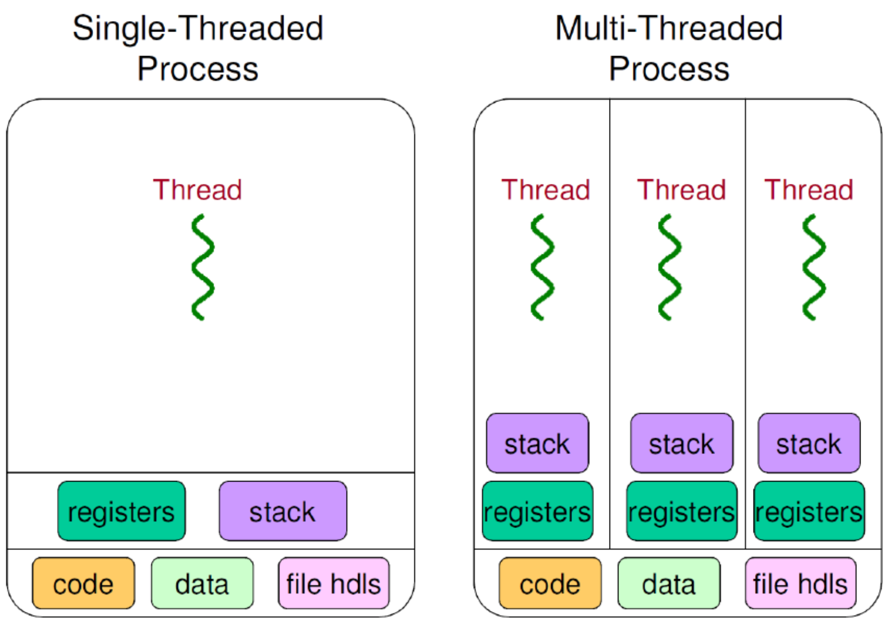
Short Answer

- Correct the wrong word in the following sentences:
  - System call functions of xv6 are implemented in assembly language
  - Xv6 follows the microkernel design.
  - RISV-V version of xv6 uses a 1MB page size by default.

Processes and Threads

- A process contains all the state for a program in execution
  - PCB is where OS keeps the execution state of each process
  - How to pause/resume a process? Context switch
  - Process management
    - fork(), exec(), wait(), exit()…

- Threads: Separate *execution* and *resource container* roles
  - The thread defines a *sequential execution stream* within a process
    The process defines the address space, resources, and general process attributes



User-level vs. Kernel-level threads

- Kernel-level threads
  - Integrated with OS (informed scheduling)
  - Slow to create, manipulate, synchronize

- User-level threads
  - Fast to create, manipulate, synchronize
  - Not integrated with OS (uninformed scheduling)

- Scheduler activations
  - Coordination between user and kernel schedulers

True/False

- Each thread has a separate address space

True/False

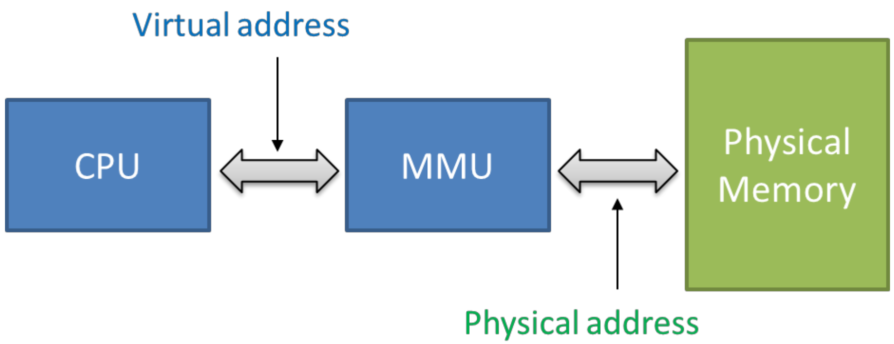
- In Unix-like systems, a process is created by another process except for the very first process

Short Answer

- User-level threads can provide higher performance than kernel-level threads. Why?

Virtual Memory

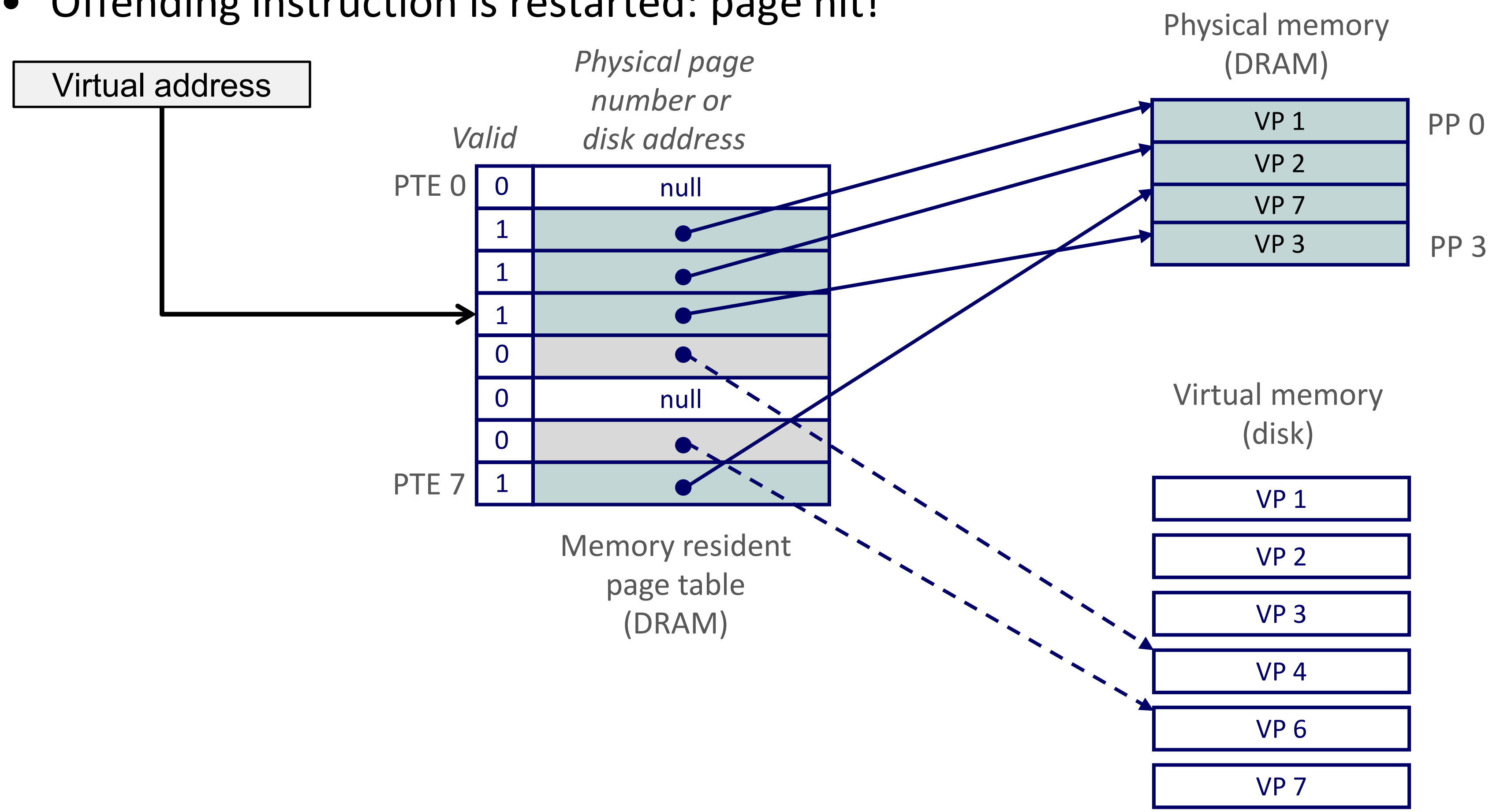
- Virtual Address
  - Independent of the actual physical location of data referenced
  - Instructions executed by the CPU issue virtual addresses



- Paging: split virtual address space into fixed-size pages
- *Page table:* an array of page table entries (PTEs) that maps virtual pages to physical pages
  - Per-process kernel data structure

Handling Page Fault

- Page miss causes page fault (an exception)
- Page fault handler selects a victim to be evicted (here VP 4)
- Offending instruction is restarted: page hit!



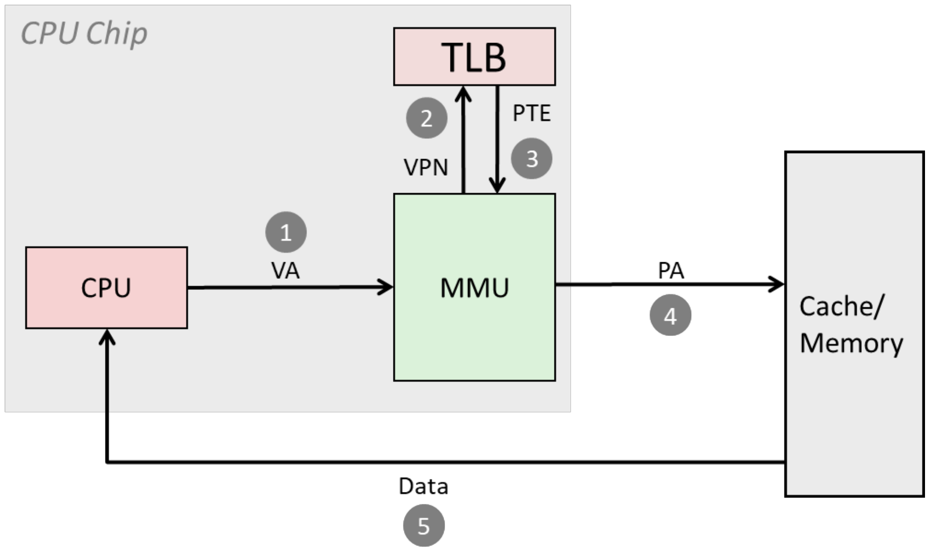
Copy on Write

- Defer large copies as long as possible, hoping to avoid them altogether
- Instead of copying pages, create shared mappings of parent pages in child virtual address space
- Shared pages are protected as read-only in parent and child
  - Reads happen as usual
  - Writes generate a protection fault, trap to OS, copy page, change page mapping in client page table, restart write instruction

Speeding up Translation with a TLB

- Small hardware cache in MMU
- Maps virtual page numbers to physical page numbers
- Contains complete page table entries for small number of pages

- TLB Hit



Multi-Level Page Tables

- Example: 2-level paging

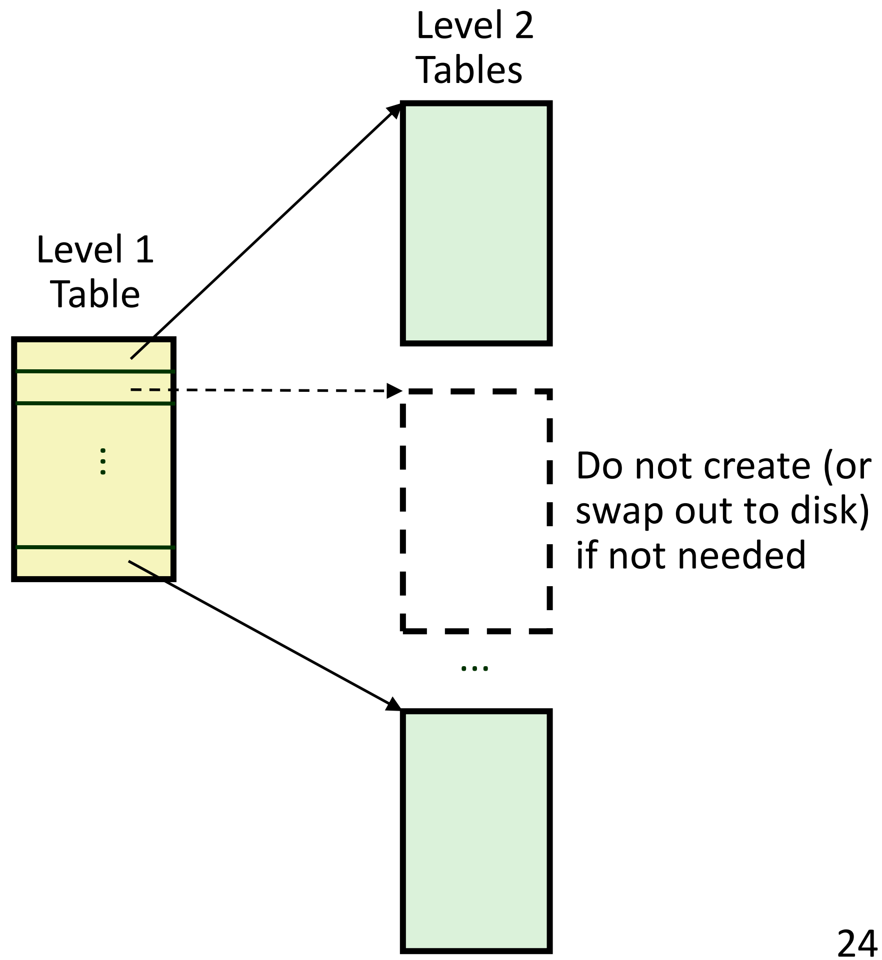
page number		page offset
p_1	p_2	d

Level 1 table

- Each PTE points to a L2 page table
- Always memory resident

Level 2 table

- Each PTE points to a page
- Paged in and out like any other data



True/False

- With virtual memory, a user-level process can choose arbitrary page sizes for spatial efficiency.

Multiple choices

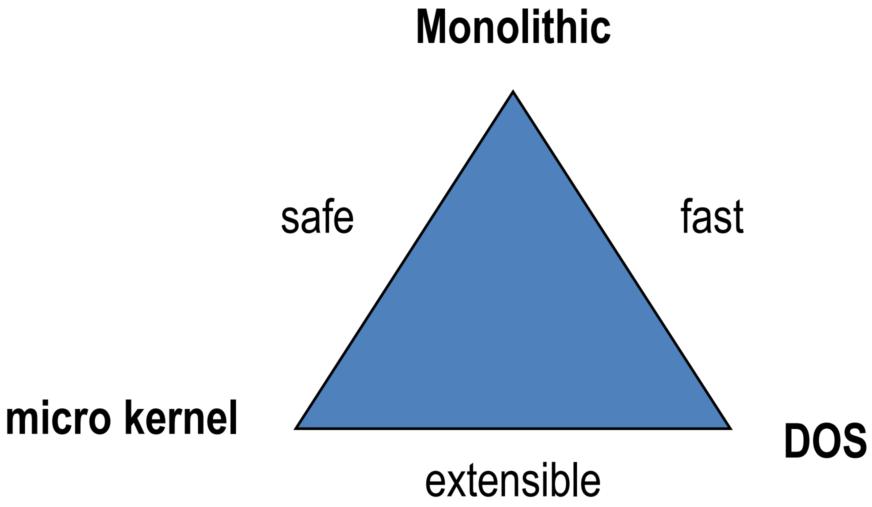
- How many of the following statements is correct about multi-level paging?
  - A. Reduces average memory access time
  - B. Reduces runtime overhead of page fault handling
  - C. Reduces spatial overhead of page tables
  - D. With multi-level paging, threads belonging to the same process can use different page tables

Multiple choices

- Choose all that are correct: Copy-on-Write (CoW)
  - A. CoW reduces the cost of fork().
  - B. CoW defers allocation of physical memory until a write attempt happens.
  - C. Shared pages among parent and child processes are read & write protected.
  - D. CoW does not require MMU.

Extensibility: OS structure

- DOS-like structure:
  - Good performance and extensibility
  - Bad protection
- Monolithic kernels:
  - Good performance and protection
  - Bad extensibility
- Microkernels
  - Very good protection
  - Good extensibility
  - Bad performance!



Exokernel

- Safely expose machine resources
- Higher-level abstractions are implemented in applications
  - Exposes hardware to Library OS
  - Not even mechanisms are implemented by exokernel
  - Every process would need a Library OS
- Safety ensured by secure bindings

True/False

- Exokernel follows the monolithic architecture design.

Multiple choices

- This figure shows the round-trip latency of network messages in a libOS (ExOS) with and without ASH (application-specific handler). Which of the following best explains this result?

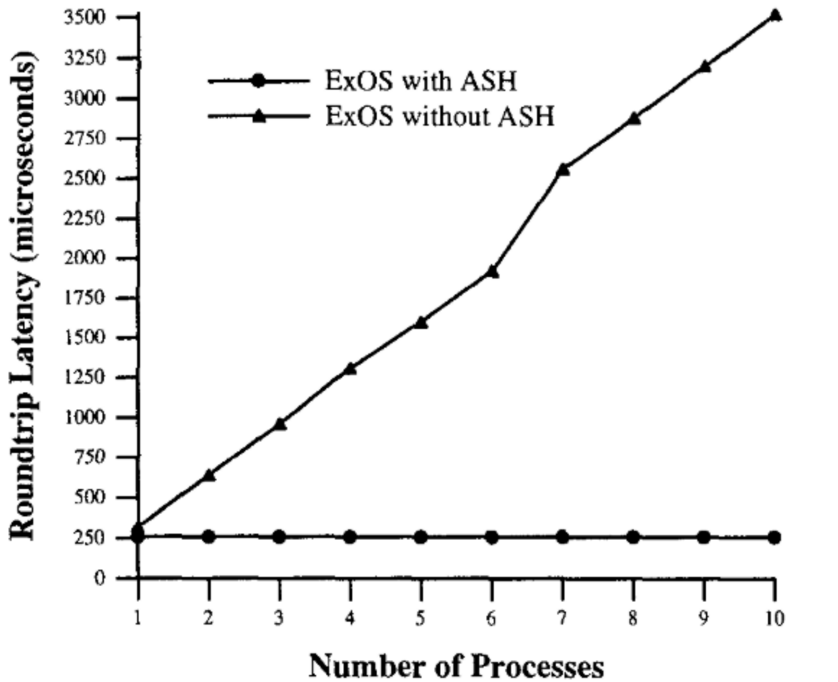


Figure 2: Average roundtrip latency with increasing number of active processes on receiver.

A. Performance benefit comes from Modula-3
B. Without ASH, ExOS suffers from garbage collection
C. ASH reduces the overhead of guard conditions.
D. ASH allows network responses to be sent before ExOS is scheduled.

Scheduling

- Classic algorithms
  - First Come, First Served (FCFS)
    - Convoy effect (a.k.a. head-of-line blocking)
  - Shortest Job First (SJF) / Preemptive SJF (PSJF)
  - Round Robin (RR)
  - Priority-based Scheduler
    - Starvation, priority inversion
  - Earliest Deadline First (EDF)
  - Multiple-level feedback queue (MLFQ)
- Multiprocessors:
  - Global scheduling (a.k.a. Single Queue Multiprocessor Scheduling)
  - Partitioned Scheduling (a.k.a. Multiple Queue Multiprocessor Scheduling)

Lottery scheduling

- Key idea: give each process a bunch of tickets
  - Each time slice, scheduler holds a lottery
  - Process holding the winning ticket gets to run
- Chance to get scheduled is determined by # of tickers
  - Elegant way to implement fair-share scheduling
- Tickets can be used for a variety of resources
- Ticket transfer, Ticket inflation, Compensation tickets

Stride scheduling

- Deterministic version of lottery scheduling
  - Randomness does not guarantee fairness
- Stride scheduling:
  - Each process is given some tickets
  - Each process has a stride = a big # / # of tickets
    - Stride = inversely proportional to # of tickets
  - Each time a process runs, its pass += stride
  - Scheduler chooses process with the lowest pass to run next
- Can use compensation tickets

True/False

- Non-preemptive scheduling cannot be used with priority-based scheduling

True/False

- Preemptive scheduling has less overhead than non-preemptive scheduling

Short Answer

- What is the difference between CPU-bound and I/O-bound processes?

Multiple choices

- Find the wrong statement:
  - A. FCFS may lead to long response time
  - B. Starvation does not occur in priority-based scheduling
  - C. Global scheduling has only one ready queue for all processors
  - D. Global scheduling may suffer from task migration overhead

Multiple choices

- Choose the correct statement:
  - A. Lottery scheduling achieves deterministic fairness.
  - B. Compensation tickets in lottery scheduling are to solve the priority inversion problem.
  - C. In stride scheduling, error is independent of allocation time.
  - D. Unlike lottery scheduling, stride scheduling cannot use compensation tickets