Process Management

What is a Process?

- Process – an *instance* of a program in execution
  - Multiple instances of the same program are different processes
- A process has resources allocated to it by the OS during its execution
  - CPU time
  - Memory space for code, data, stack
  - Open files
  - Signals
  - Data structures to maintain different information about the process
  - …
- Each process identified by a unique, positive integer id (process id)

Process Control Block (PCB)

- The primary data structure maintained by the OS that contains information about a process
- One PCB per process
- OS maintains a list of PCB’s for all processes

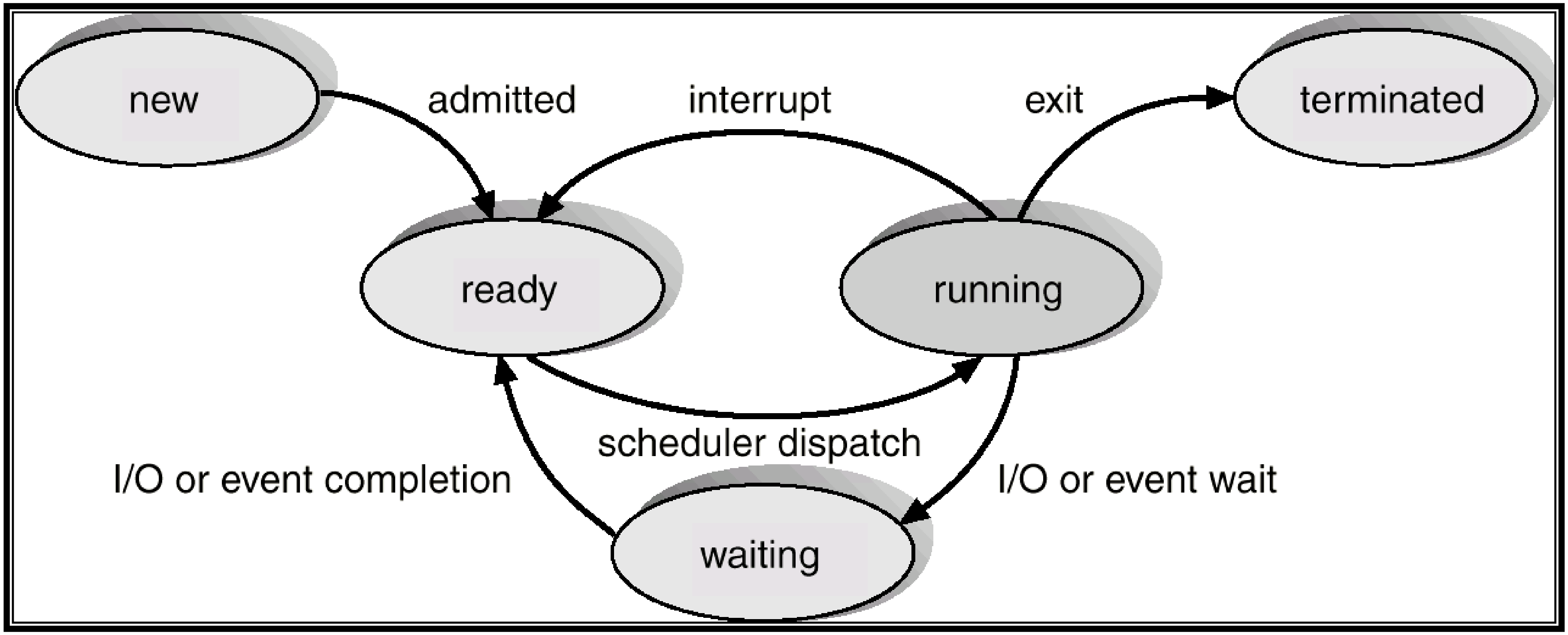
Typical Contents of PCB

- Process id, parent process id
- Process state
- CPU state: CPU register contents, PSW
- Priority and other scheduling info
- Pointers to different memory areas
- Open file information
- Signals and signal handler info
- Various accounting info like CPU time used etc.
- Many other OS-specific fields can be there
  - Linux PCB (*task_struct*) has 100+ fields

Process States (5-state model)

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

Process State Transitions



Main Operations on a Process

- Process creation
  - Data structures like PCB set up and initialized
  - Initial resources allocated and iitialized if needed
  - Process added to ready queue (queue of processes ready to run)
- Process scheduling
  - CPU is allotted to the process, process runs
- Process termination
  - Process is removed
  - Resources are reclaimed
  - Some data may be passed to parent process (ex. exit status)
  - Parent process may be informed (ex. SIGCHLD signal in UNIX)

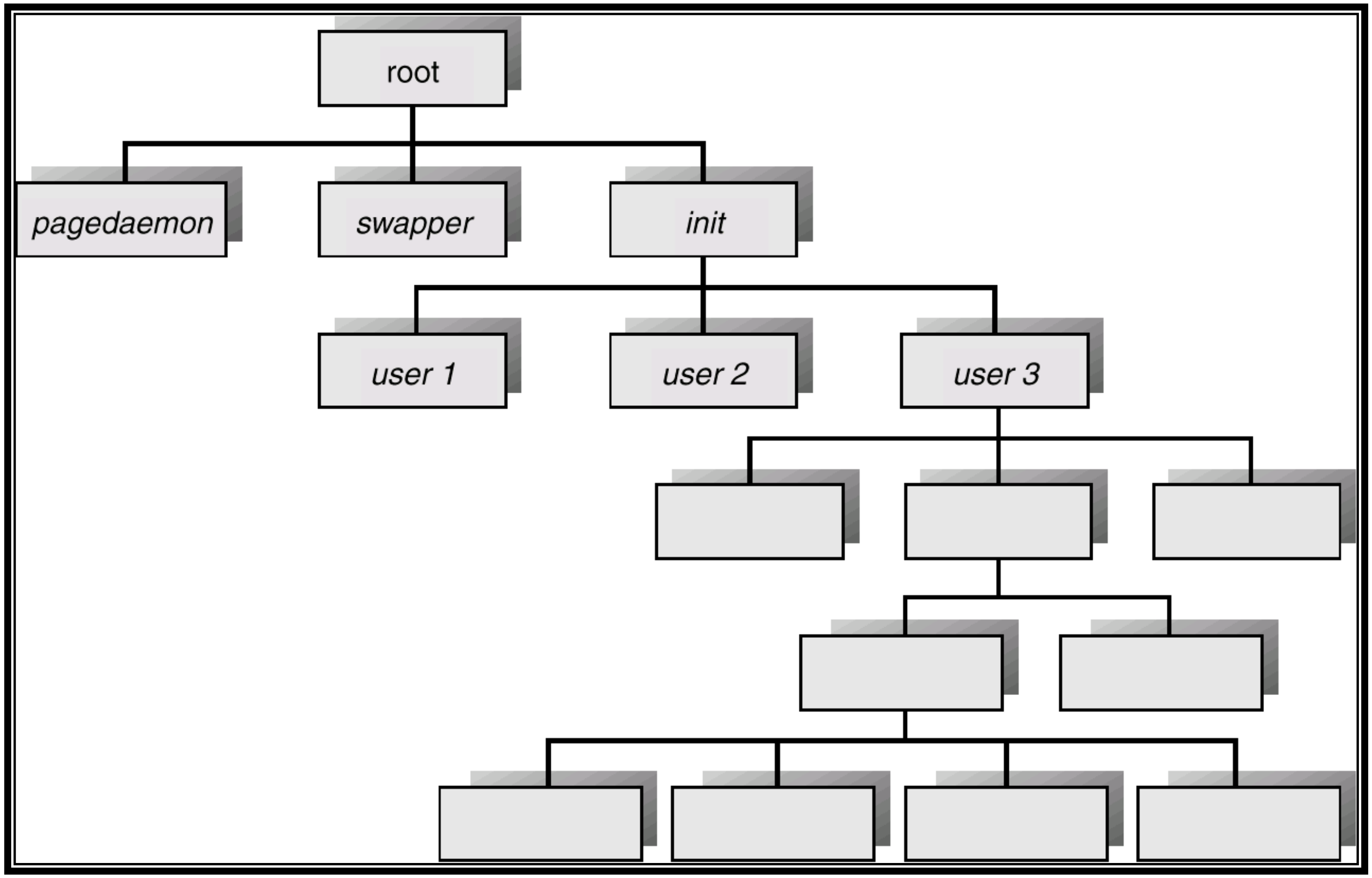
Process Creation

- A process can create another process
  - By making a system call (a function to invoke the service of the OS, ex. *fork( )*)
  - Parent process: the process that invokes the call
  - Child process: the new process created
- The new process can in turn create other processes, forming a tree of processes
- The first process in the system is handcrafted
  - No system call, because the OS is still not running fully (not open for service)

Process Creation (contd.)

- Resource sharing possibilities
  - Parent and children share all resources
  - Children share subset of parent’s resources
  - Parent and child share no resources
- Execution possibilities
  - Parent and children execute concurrently
  - Parent waits until children terminate
- Memory address space possibilities
  - Address space of child duplicate of parent
  - Child has a new program loaded into it

Processes Tree on a UNIX System



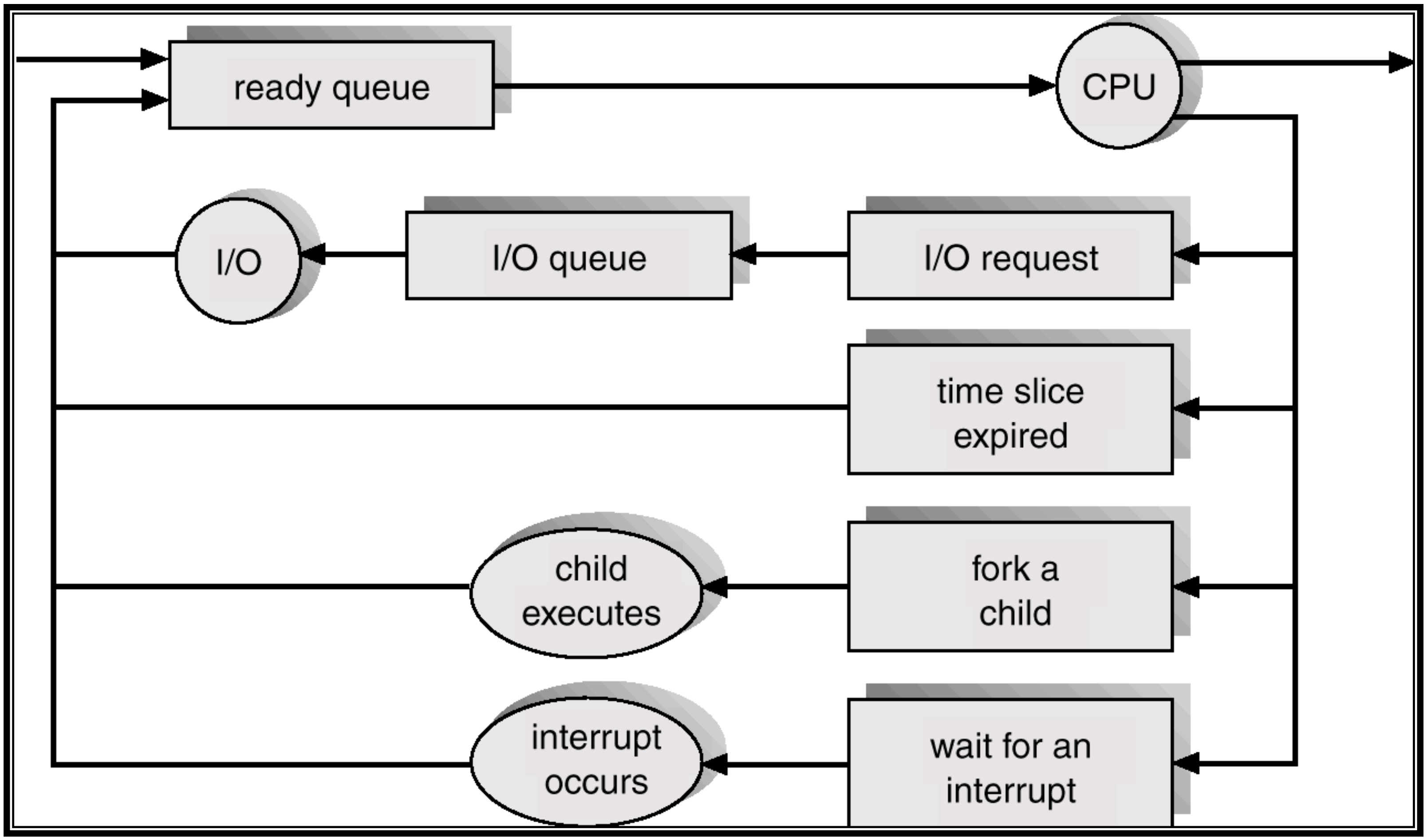
Process Termination

- Process executes last statement and asks the operating system to terminate it (ex. *exit*/*abort*)
- Process encounters a fatal error
  - Can be for many reasons like arithmetic exception etc.
- Parent may terminate execution of children processes (ex. *kill*). Some possible reasons
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
- Parent is exiting
  - Some operating systems may not allow child to continue if its parent terminates

Process Scheduling

- Ready queue – queue of all processes residing in main memory, ready and waiting to execute (links to PCBs)
- Scheduler/Dispatcher – picks up a process from ready queue according to some algorithm (CPU Scheduling Policy) and assigns it the CPU
- Selected process runs till
  - It needs to wait for some event to occur (ex. a disk read)
  - The CPU scheduling policy dictates that it be stopped
    - CPU time allotted to it expires (timesharing systems)
    - Arrival of a higher priority process
  - When it is ready to run again, it goes back to the ready queue
- Scheduler is invoked again to select the next process from the ready queue

Representation of Process Scheduling



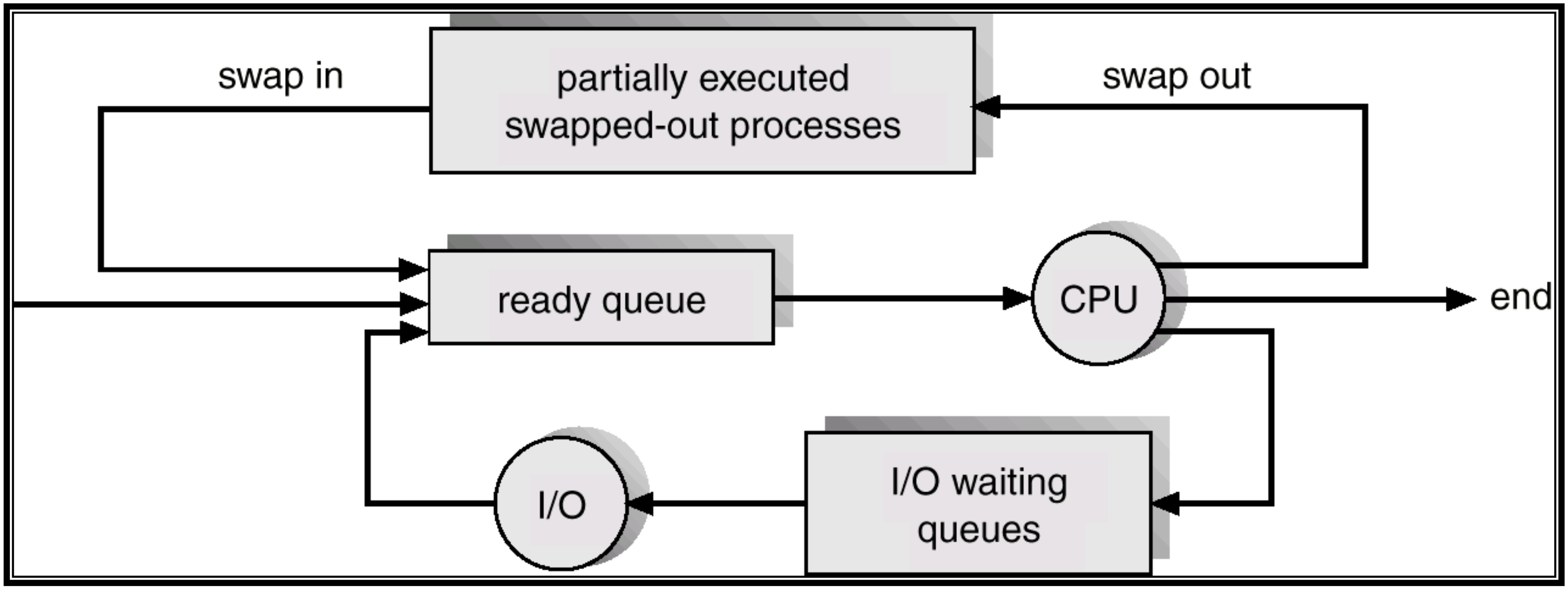
Schedulers

- Long-term scheduler (or job scheduler)
  - Selects which processes should be brought into the ready queue
  - Controls the *degree of multiprogramming* (no. of jobs in memory)
  - Invoked infrequently (seconds, minutes)
  - May not be present in an OS (ex. linux/windows does not have one)
- Short-term scheduler (or CPU scheduler)
  - Selects which process should be executed next and allocates CPU
  - Invoked very frequently (milliseconds), must be fast

What if all processes do not fit in memory?

- Partially executed jobs in secondary memory (swapped out)
  - Copy the process image to some pre-designated area in the disk (swap out)
  - Bring in again later and add to ready queue later

Addition of Medium Term Scheduling



Other Questions

- How does the scheduler gets scheduled? (Suppose we have only one CPU)
  - As part of execution of an ISR (ex. timer interrupt in a time-sharing system)
  - Called directly by an I/O routine/event handler after blocking the process making the I/O or event request
- What does it do with the running process?
  - Save its context
- How does it start the new process?
  - Load the saved context of the new process chosen to be run
  - Start the new process

Context of a Process

- Information that is required to be saved to be able to restart the process later from the same point
- Includes:
  - CPU state – all register contents, PSW
  - Program counter
  - Memory state – code, data
  - Stack
  - Open file information
  - Pending I/O and other event information

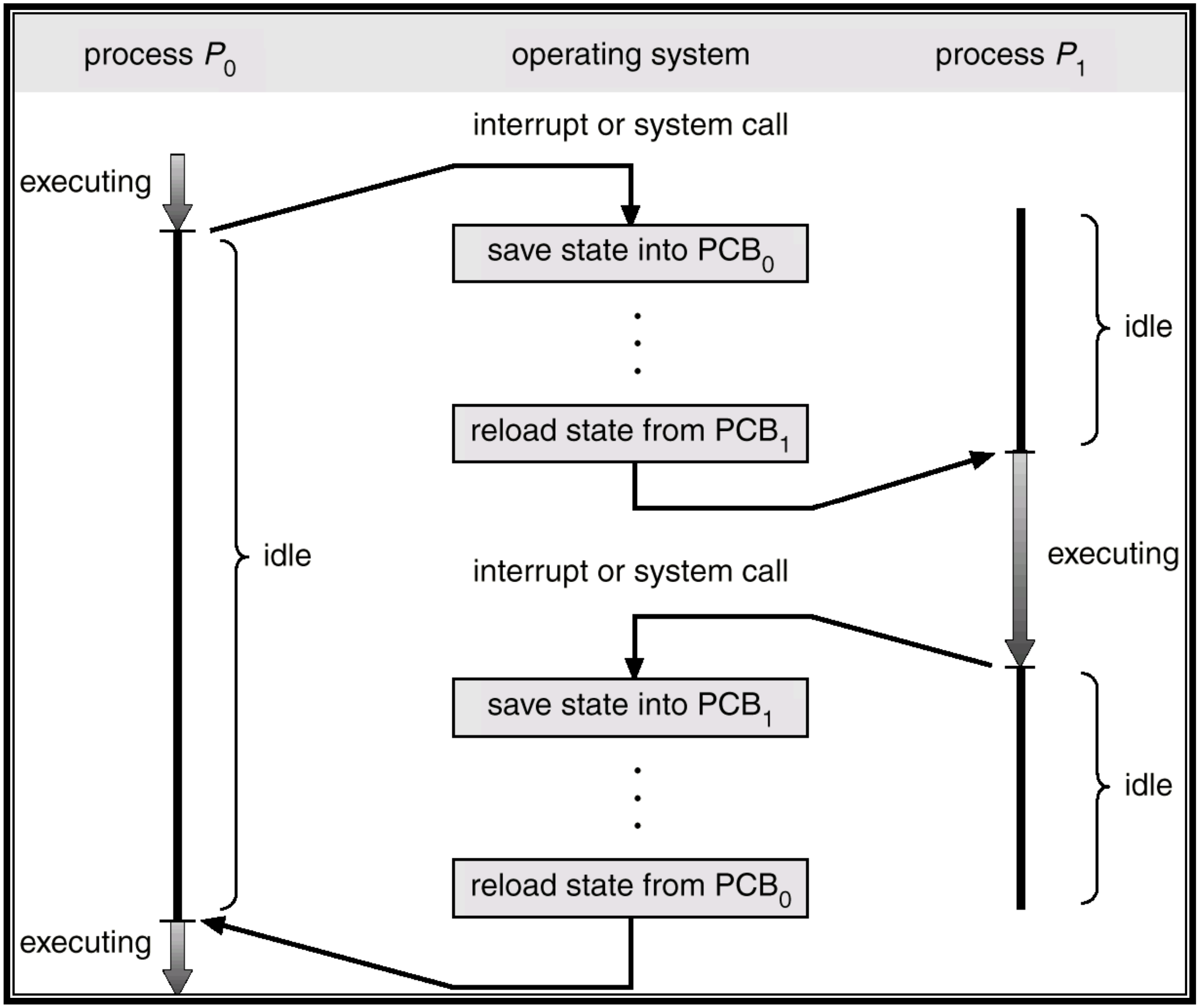
Context Switch

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process
- Context-switch time is overhead; the system does no useful work while switching
- Time dependent on hardware support

Handling Interrupts

- H/w saves PC, PSW
- Jump to ISR
- ISR should first save the context of the process
- Execute the ISR
- Before leaving, ISR should restore the context of the process being executed
- Return from ISR restores the PC
- ISR may invoke the dispatcher, which may load the context of a new process, which runs when the interrupt returns instead of the original process interrupted

CPU Switch From Process to Process



Example: Timesharing Systems

- Each process has a time quantum *T* allotted to it
- Dispatcher starts process P_0, loads a external counter (timer) with counts to count down from *T* to 0
- When the timer expires, the CPU is interrupted
- The ISR invokes the dispatcher
- The dispatcher saves the context of P_0
  - PCB of P_0 tells where to save
- The dispatcher selects P_1 from ready queue
  - The PCB of P_1 tells where the old state, if any, is saved
- The dispatcher loads the context of P_1
- The dispatcher reloads the counter (timer) with *T*
- The ISR returns, restarting P_1 (since P_1's PC is now loaded as part of the new context loaded)
- P_1 starts running