Operating Systems - Processes

Mridul Sankar Barik

Jadavpur University

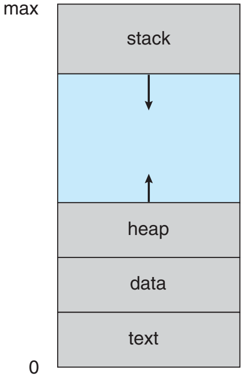
2024

Slide Credits

- Most of the slides are adapted from the companion lecture slides for the text book by Avi Silberschatz, Peter Baer Galvin, Greg Gagne

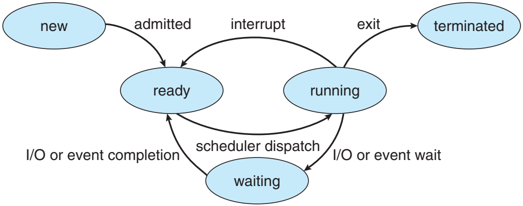
Process Concept

- Process is a program in execution
- A process includes
  - **Program code** (text section)
  - **Program counter and other processor registers**
    - Defines the current state of the process
  - **Stack**: contains temporary data
    - Function parameters, return addresses, local variables
  - **Data section**: contains global data
  - **Heap**: contains memory dynamically allocated during run time
- A Program is a **passive** entity stored on disk (executable file), whereas process is an **active** entity



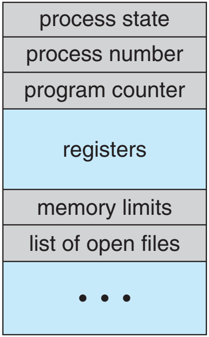
Process States

- **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 processor
- **Terminated**: The process has finished execution



Process Control Block (PCB)

- Each process is represented in the OS by a PCB
  - Process State
  - Program Counter
  - CPU Registers
  - CPU Scheduling Information
    - process priority
    - pointer to scheduling queues etc.
  - Memory Management Information
    - values of base, limit registers
    - page table or segment table etc.
  - Accounting Information
    - amount of CPU and real time used
    - time limits
    - job or process numbers etc.
  - I/O Status Information
    - list of I/O devices allocated to this process
    - list of open files etc.



process state

process number

program counter

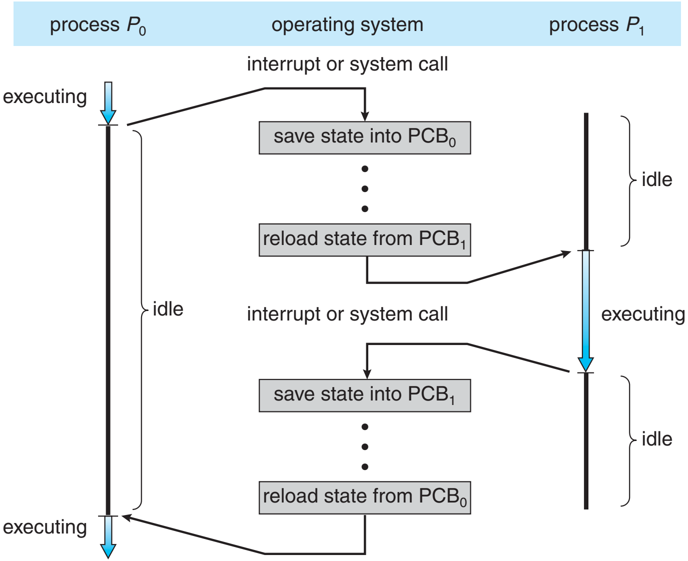
registers

memory limits

list of open files

• • •

CPU Switch From Process to Process

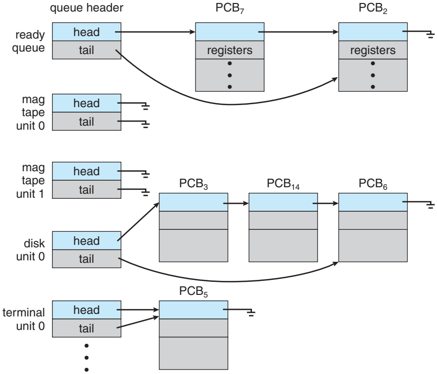


Threads

- So far, process has a single thread of execution
- Consider having multiple program counters per process
  - Multiple locations can execute at once
    - Multiple threads of control ⇒ threads
- Must then have storage for thread details, multiple program counters in PCB

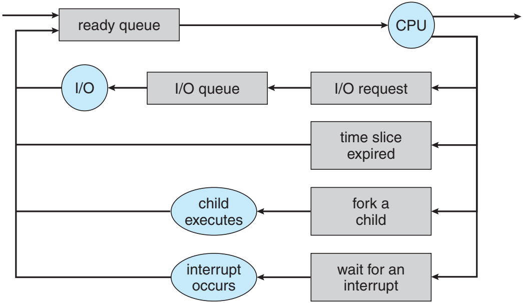
Process Scheduling I

- Maximize CPU use, quickly switch processes onto CPU for time sharing
- **Processes scheduler** selects among available processes for next execution on CPU
- Maintains scheduling queues of processes
    - **Job queue** – set of all processes in the system
    - **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
    - **Device queues** – set of processes waiting for an I/O device
- Processes migrate among the various queues



Process Scheduling II

- A new process is initially put in the ready queue, waits there until it is selected for execution
- Once a process executes it may
  - Issue an I/O request, and be placed in an I/O queue
  - Create a new process and wait for its termination
  - Be removed forcibly from the CPU as a result of an interrupt

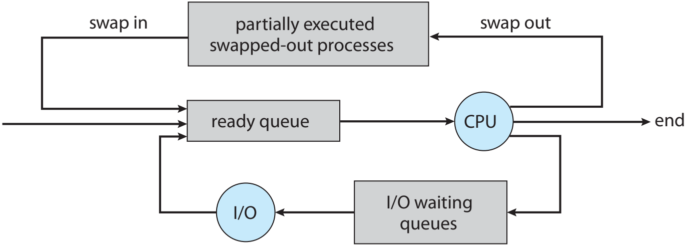


Schedulers I

- **Short-term scheduler** (or **CPU scheduler**) – selects which process should be executed next and allocates CPU
  - Sometimes the only scheduler in a system
  - Short-term scheduler is invoked frequently (milliseconds) (must be fast)
- **Long-term scheduler** (or **job scheduler**) – selects which processes should be brought into the ready queue
  - Long-term scheduler is invoked infrequently (seconds, minutes) (may be slow)
  - The long-term scheduler controls the **degree of multiprogramming**
- Processes can be described as either:
  - **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts
  - **CPU-bound process** – spends more time doing computations; few very long CPU bursts
- Long-term scheduler strives for good process mix

Schedulers II

- **Medium-term scheduler** can be added if degree of multiple programming needs to decrease
    - Remove process from memory, store on disk, bring back in from disk to continue execution: **swapping**



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 via a **context switch**
- **Context** of a process is represented in the PCB of a process
- **Context-switch time** is pure overhead
  - The system does no useful work while switching
  - The more complex the OS and the PCB ⇒ the longer the context switch
- Context switch time is dependent on hardware support
  - Sun UltraSPARC provides multiple sets of registers
  - Context switch simply requires changing the pointer to the current register set