Advanced Operating Systems

Dr. Gurdip Singh

Sequential Program/Systems

- A system is sequential if no more than one control point can be serviced at a time.
- A program is sequential if no more than one control point can be enabled at a time.

Concurrent program/system

- A program is concurrent if multiple control points can be enabled at the same time.
- A system is concurrent if multiple control points can be serviced at the same time.
- Since multiple control points can be enabled/serviced at one time, the underlying scheduling system (called the scheduler or dispatcher) may select different schedules each time it runs the parallel program. Thus, **concurrent software is non-deterministic**.

Why concurrent programming

- Exploiting parallelism in an application
  - Matrix multiplication
- Availability of concurrent platforms
  - Multiple cores on one chip
  - Fast networks
- Inherently concurrent applications
  - Real-time control

Example system

- Consider a control system:
  1. Brake Sensor: When it detects a brake action, it must actuate the brakes within *20ms*. The execution time is *5ms*.
  2. AC Button: It must start the AC within *4s* after it detects the button press. The execution time is *500ms*.
  3. Cruise Control button. It must activate the cruise control within *2s* . The execution time is *100ms.*

Control-Loop System

```
main( ) {

   ….initialize….

   while (1)
{
        if (flagBrake)     {activateBrakes( );    flagBrake = false;}
        if (flagAC)              {activateAC( );        flagAC = false;}
        if (flagCruise) {activateCruise( );         flagCruise = false;}
  }
}
```

Each interrupt handler simply
sets the corresponding flag

Control-Loop System

- Pros:
  - It’s a simple sequential code
  - Debugging is easy; Developed systems are reliable.
- Cons:
  - It is difficult to satisfy real-time requirements.
    - For example, what will happen if a brake action is detected while the control loop is executing ActivateCruise( ), which takes the maximum of 100ms？

Event-Triggered System

- The system assigns a thread to each task
- For each sensor interrupt, the system dispatches the thread assigned to the corresponding task
  - Brake controller: create a thread that executes BrakeFunction( )
- The thread priorities should be (1) the Brake control task (highest), (2) the Cruise control task, and (3) the AC control task (lowest)

```
BrakeFunction( ) {
  init( );
  while (…) {
     :
     activateBrake( );
     :
  }}
```

```
AC_Control( ) {
  init( );
  while (…) {
     :
     activateAC( );
     :
  }}
```

```
Cruise_Control( ) {
  init( );
  while (…) {
     :
     activateCruise( );
     :
  }}
```

Concurrent Programs

- Advantages:
  - can increase CPU utilization
  - can provide fair service to many clients
  - can improve response time
  - allow programmers to focus only on sequential execution of each thread
- Disadvantages:
  - synchronization code is required to coordinate the execution of the multiple threads
  - Scheduling

Operating System

- Process/thread creation
- Program execution: Scheduling
- Access to I/O devices
- Memory management
- Accounting

  :

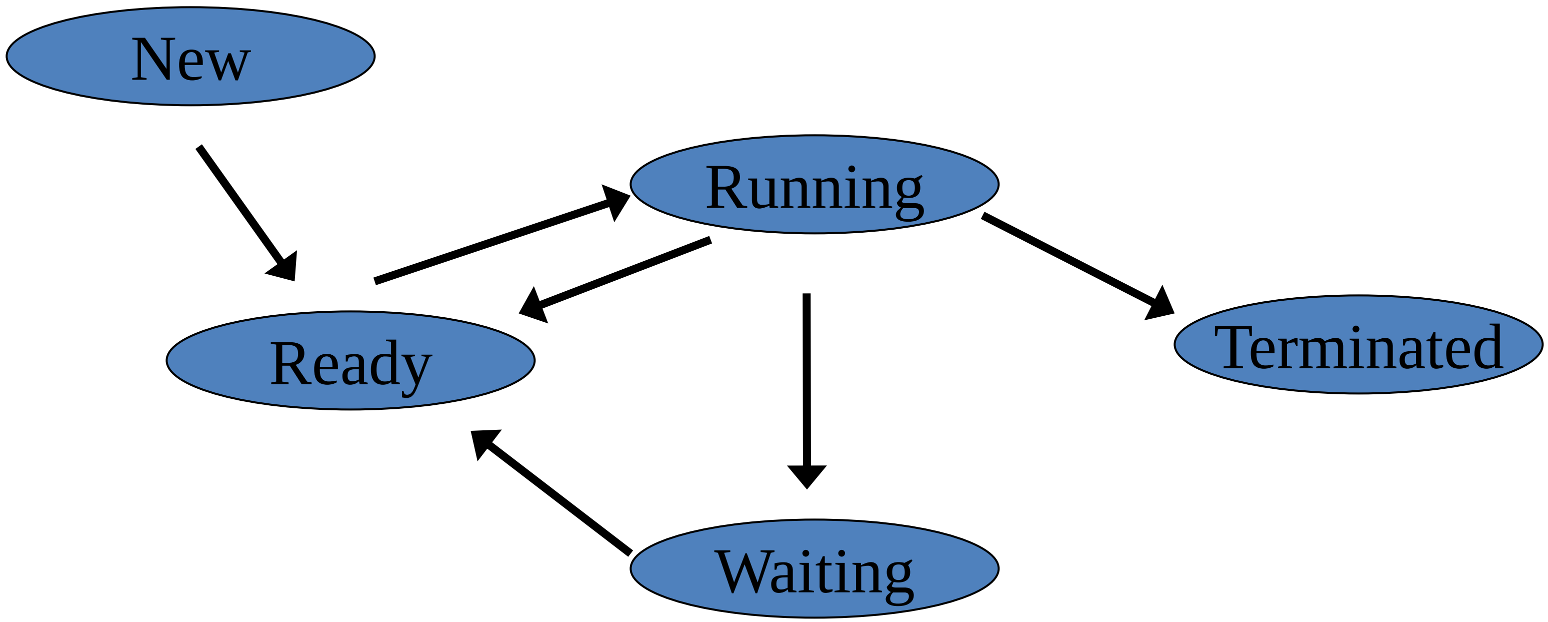
Processes

- All the runnable software on a computer, including the OS, is organized into a number of sequential processes (or processes in short)
- Process: a unit of work scheduled by OS, consisting of
  1. program's data and stack (and the memory areas)
  2. program counter, stack pointer, and other CPU registers
  3. all other information necessary to run the program: process id, priority, accounting information, memory information, open files, process state, etc.
  4. executable code (and the memory area)

stored in *process control blocks (PCBs)*

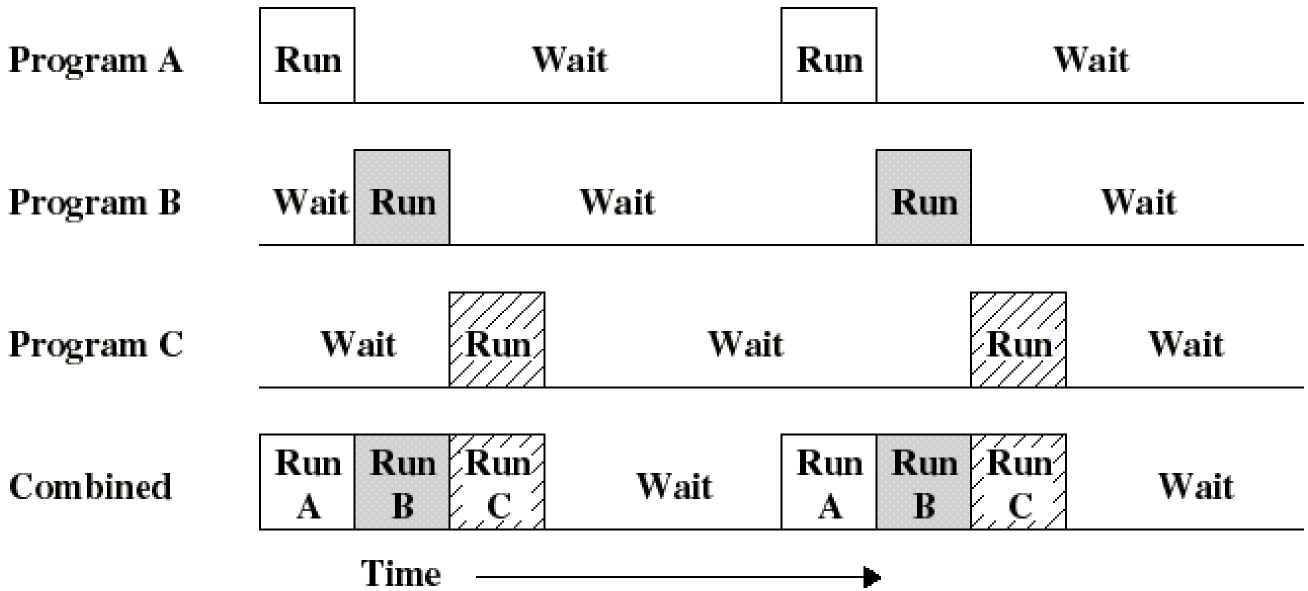
Process States

- **New** – process is being created.
- **Running** – instructions are being executed.
- **Waiting** – blocked waiting for an external event to occur; e.g., message arrival.
- **Ready** – ready to run on a processor.
- **Terminated** – finished, awaiting garbage collection.



Multiprogramming

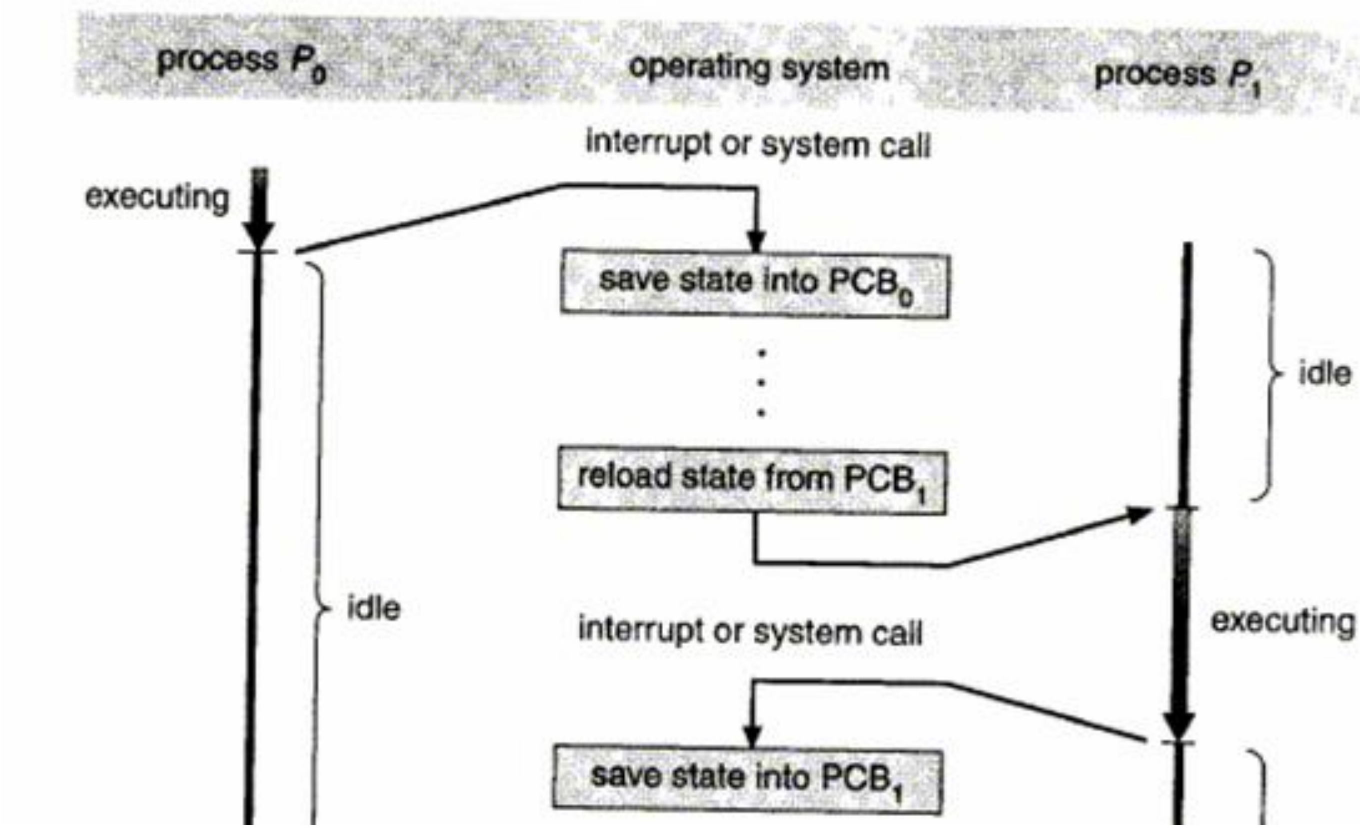
- In a multiprogramming system, the CPU switches from program to program, giving the users the illusion of parallelism (pseudo parallelism)



Multiiprogramming with three programs

Implementing Processes

- When a context switch occurs between processes P0 and P1, the current state of **running** process P0 is saved in the PCB for P0, and the state of **ready** process P1 is restored from the PCB for P1.

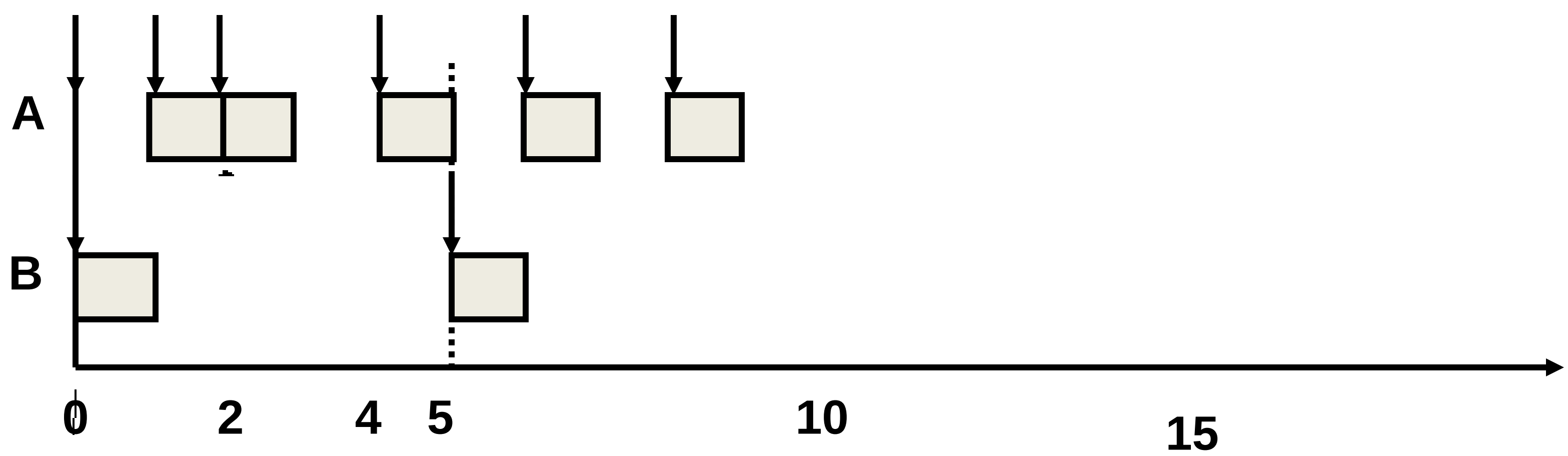


Process Scheduling

- In a general operating system, a context switch occurs after each **time quantum** has expired; e.g., every 20 msec.
- Non-preemptive scheduling

Example

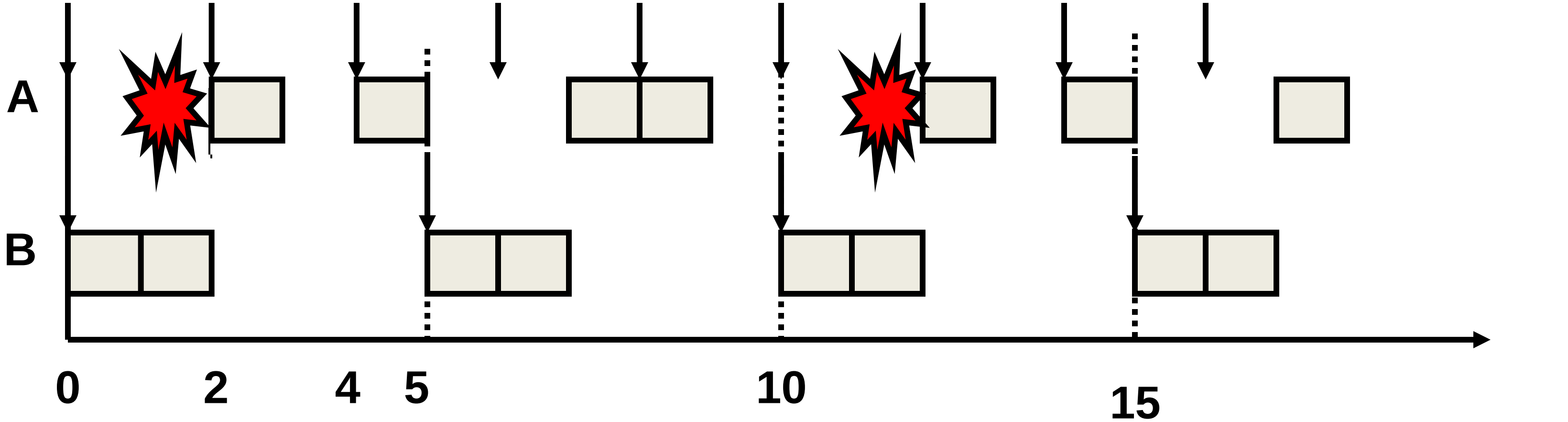
Task	Period	Deadline	Run-Time
T_i	D_i	C_i	
A	2	2	1
B	5	5	1



(thanks to M. Neilsen)

Example

Task T_i	Period D_i	Deadline C_i	Run-Time
A	2	2	1
B	5	5	2



Process Scheduler

- The **process scheduler** determines which process to schedule (switch to) next.
- Types of schedulers:
  - **preemptive**
  - **non-preemptive**
- Scheduling Criteria:
  - CPU Utilization
  - Throughput - completions/time period
  - Turnaround time - total execution time
  - Waiting time - time spent in ready queue
  - Response time - time until first response
  - No missed deadlines - in a hard real-time system this is the most important criteria!

Example

Task τ_i	Period T_i	Deadline D_i	Run-Time C_i
A (High Priority)	2	2	1
B (Low Priority)	5	5	2