IOWA STATE UNIVERSITY
Department of Electrical and Computer Engineering

Lecture 11:
Inter-Process Communication (IPC) I

Agenda

- Recap
- Inter-Process Communication I
  - Basic Concepts
  - Race Condition & Critical Region
  - Solutions of Mutual Exclusion

Recap

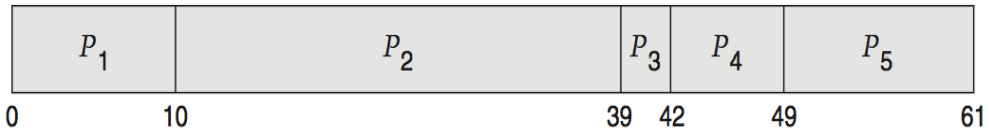
- Scheduling Algorithms
  - First-Come, First-Served (FCFS)
  - Shortest-Job-First (SJF)
  - Shortest Remaining Time Next
  - Round Robin (RR)
  - Priority Scheduling

Recap

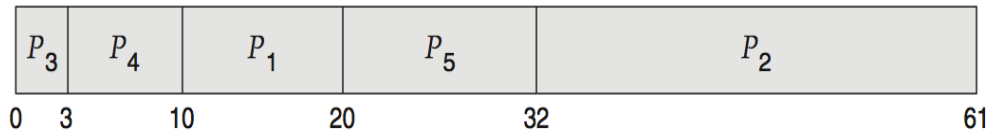
- Scheduling Algorithms
  - Consider 5 processes arriving in the order of P1-P5; all are ready at time 0
  - Draw Gantt Chart for each algorithm and calculate average waiting time

Process	Burst Time
P_1	10
P_2	29
P_3	3
P_4	7
P_5	12

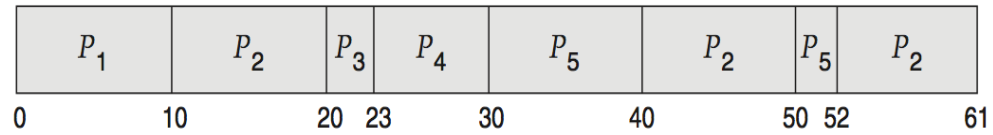
    - FCFS: 28



    - SJF: 13



    - RR (quantum = 10): 23



Agenda

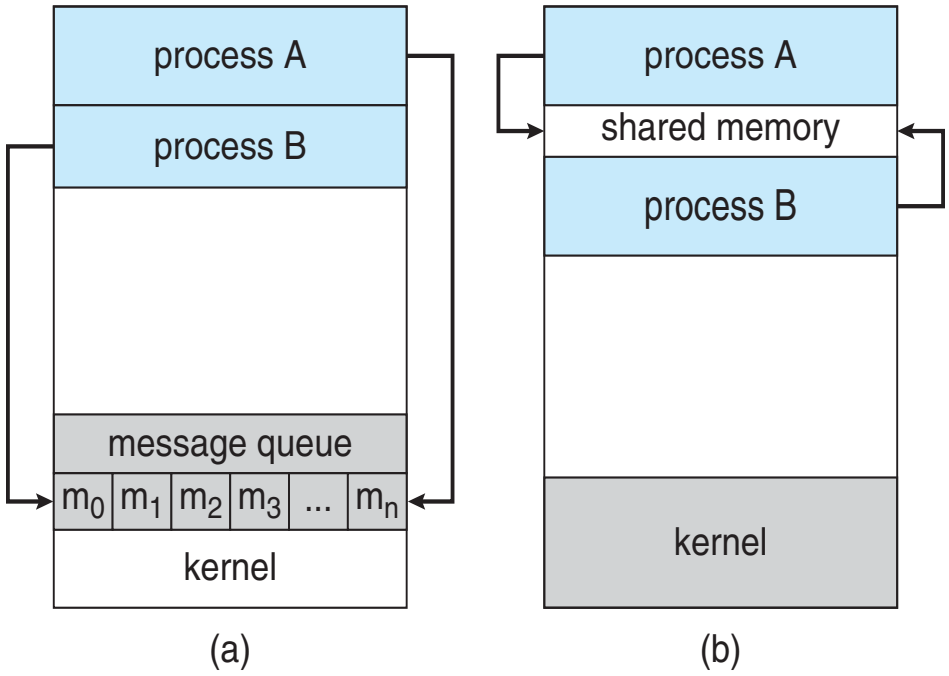
- ~~Recap~~
- Inter-Process Communication I
  - Basic Concepts
  - Race Condition & Critical Region
  - Solutions of Mutual Exclusion

Basic Concepts

- Processes within a system may be independent or cooperating
  - Example reasons for cooperating:
    - Information sharing
    - Computation speedup
    - ...
  - Cooperating processes need **inter-process communication** (**IPC**) mechanism:
    - Communicate with each other
    - Ensure
      - (1) do not get in each other's way
      - (2) proper ordering when dependencies are present
    - Violating the two may lead to "concurrency bugs"

Basic Concepts

- Two basic methods of IPC
  - Message passing
  - Shared memory

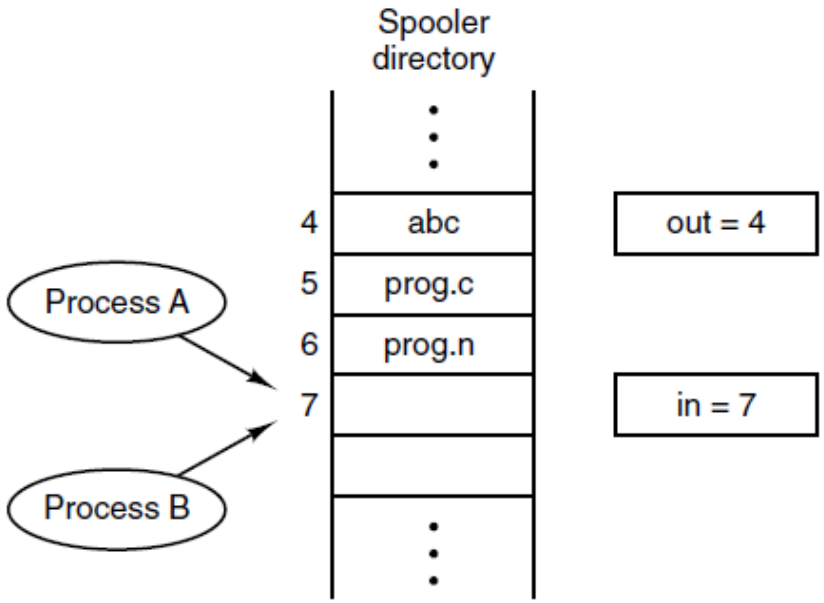


Agenda

- ~~Recap~~
- Inter-Process Communication I
  - ~~Basic Concepts~~
  - Race Condition & Critical Region
  - Solutions of Mutual Exclusion

Race Condition

- two or more processes are reading or writing some shared data and the final result depends on who runs precisely when
- E.g., a printer with two processes
  - Two shared variables
    - out: next file to be printed
    - in: next free slot



Race Condition

- E.g., two threads perform “counter = counter +1”
  - “counter” is a shared variable
  - Initially counter = 50
  - Possible result?

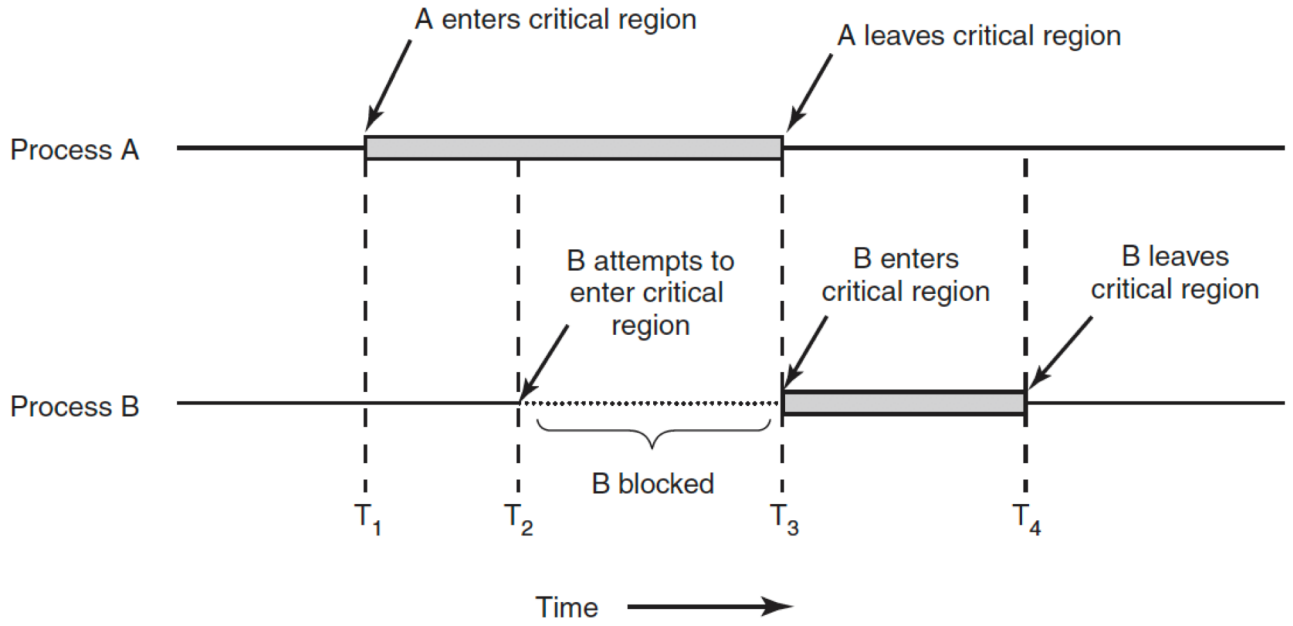
Race Condition

- E.g., two threads perform “counter = counter +1”
  - “counter” is a shared variable
  - Initially counter = 50

OS	Thread1	Thread2	(after instruction) PC	%eax	counter
			100	0	50
	mov 0x8049a1c, %eax		105	50	50
	add $0x1, %eax		108	51	50
interrupt					
save T1’s state					
restore T2’s state			100	0	50
		mov 0x8049a1c, %eax	105	50	50
		add $0x1, %eax	108	51	50
		mov %eax, 0x8049a1c	113	51	51
interrupt					
save T2’s state					
restore T1’s state			108	51	50
	mov %eax, 0x8049a1c		113	51	**51**

Critical Region

- A piece of code that **accesses a shared variable** and must not be concurrently executed by more than one thread
  - Multiple threads executing critical section can result in a race condition.
  - Need to support **mutual exclusion** or critical sections



Critical Region

- Four conditions for a good solution
  - No two processes may be simultaneously inside their critical regions
  - No assumptions may be made about speeds or the number of CPUs
  - No process running outside its critical region may block any process
  - No process should have to wait forever to enter its critical region

Agenda

- ~~Recap~~
- Inter-Process Communication I
  - ~~Basic Concepts~~
  - ~~Race Condition & Critical Region~~
  - Solutions of Mutual Exclusion

Solutions of Mutual Exclusion

- Disabling Interrupts
  - Disable interrupts when a process is in critical region
  - Cons
    - Not safe
      - What if a process does not turn them on?
      - More suitable for kernel instead of user processes
    - Not applicable for multiple processors
      - processes on other processors may still access shared variables
  - Less used today

Solutions of Mutual Exclusion

- Strict Alternation
  - Use an variable to keep track of whose turn it is to enter the critical region

```
while (TRUE) {
    while (turn != 0)        /* loop */ ;
    critical_region( );
    turn = 1;
    noncritical_region( );
}
```

(a)

```
while (TRUE) {
    while (turn != 1)        /* loop */ ;
    critical_region( );
    turn = 0;
    noncritical_region( );
}
```

(b)

Solutions of Mutual Exclusion

- Strict Alternation
  - Use an variable to keep track of whose turn it is to enter the critical region

```
while (TRUE) {
    while (turn != 0)        /* loop */ ;
    critical_region();
    turn = 1;
    noncritical_region();
}
```

(a)

```
while (TRUE) {
    while (turn != 1)        /* loop */ ;
    critical_region();
    turn = 0;
    noncritical_region();
}
```

(b)

  - Cons
    - **Busy waiting** (continuously testing a variable until some value appears) wastes CPU cycles
      - A lock that uses busy waiting is called a **spin lock**

Agenda

- ~~Recap~~
- Inter-Process Communication I
  - ~~Basic Concepts~~
  - ~~Race Condition & Critical Region~~
  - ~~Solutions of Mutual Exclusion~~

Questions?

*acknowledgement: slides include content from “Modern Operating Systems” by A. Tanenbaum, “Operating Systems Concepts” by A. Silberschatz etc., “Operating Systems: Three Easy Pieces” by R. Arpaci-Dusseau etc., and anonymous pictures from internet.