Amirkabir University of Technology
(Tehran Polytechnic)

Operating Systems

Synchronization Tools-Part1

Seyyed Ahmad Javadi

sajavadi@aut.ac.ir

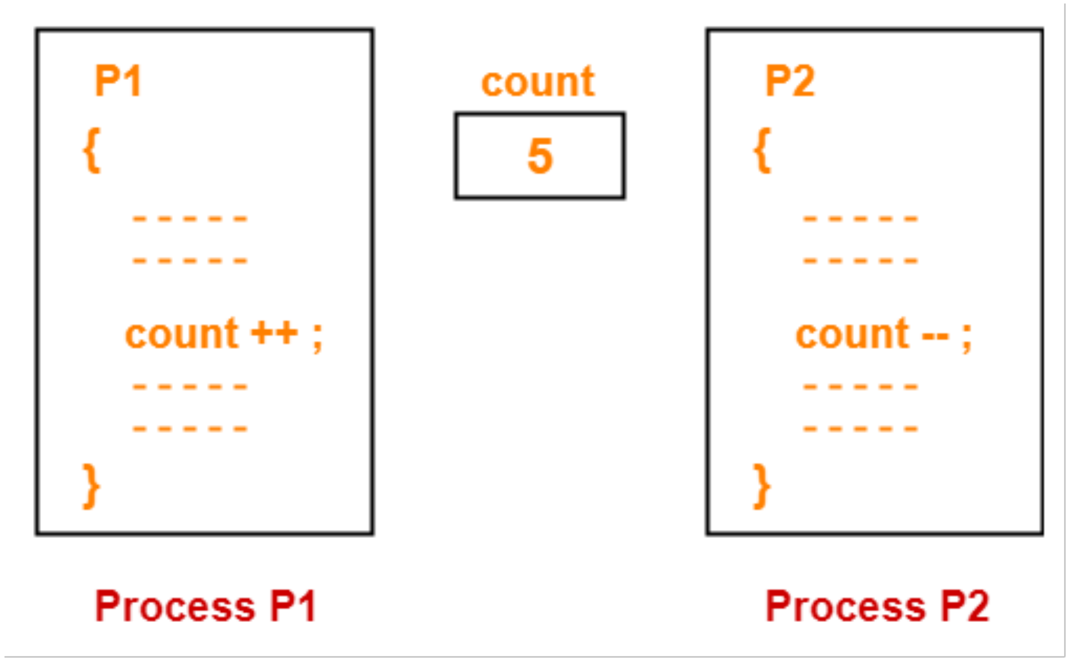
Fall 2021

Background

- **Processes can execute concurrently**
  - May be interrupted at any time, partially completing execution.
- Concurrent access to shared data may result in **data inconsistency**.
- Maintaining data consistency requires mechanisms to ensure the **orderly execution of cooperating processes**.

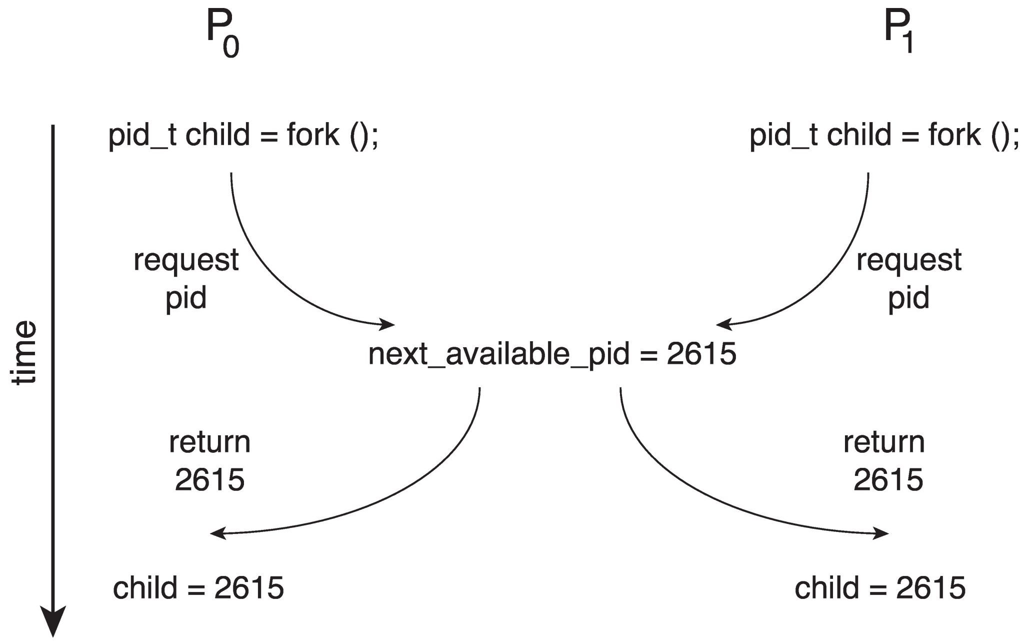
Background (cont.)

- In chapter 4, when we considered the Bounded Buffer problem with use of a counter that is updated concurrently by the producer and consumer, which **lead to race condition.**

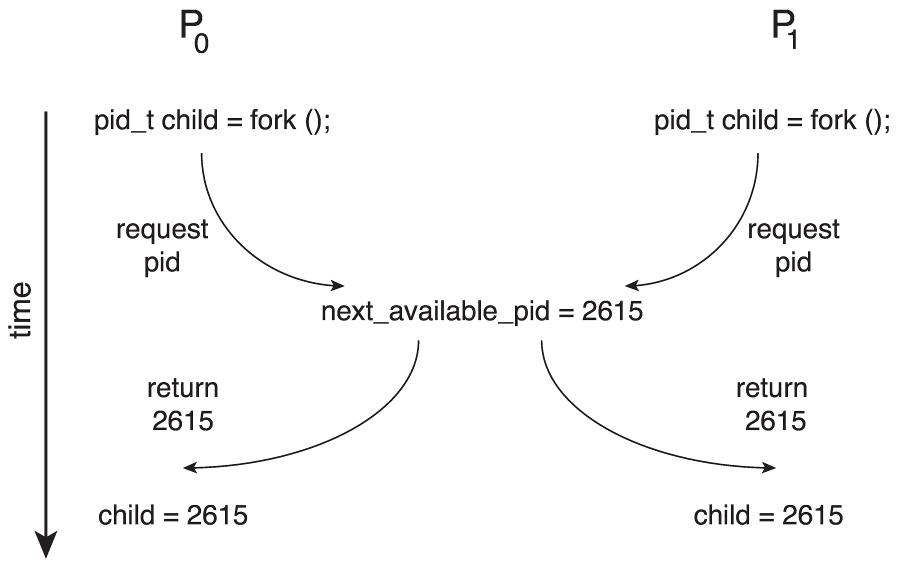


Race Condition

- Processes P_0 and P_1 are creating child processes using the **fork()** system call.
- Race condition on kernel variable next_available_pid which represents the next available process identifier (pid)



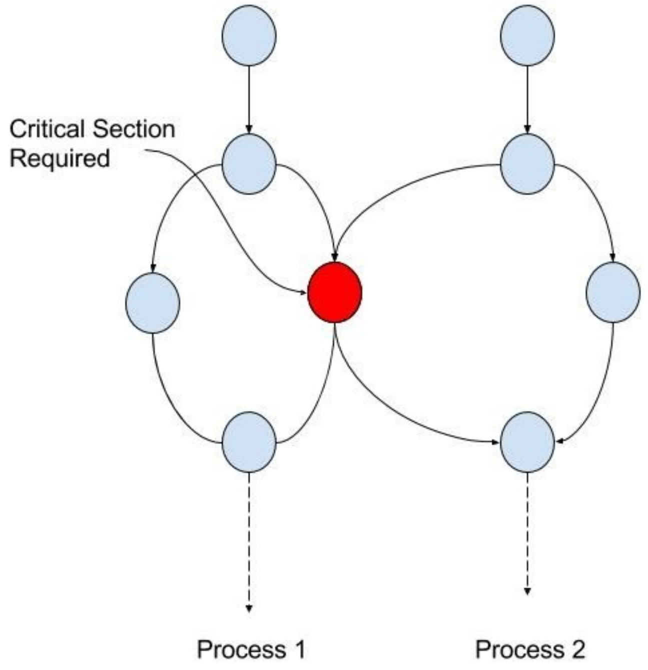
Race Condition (Cont.)



- Unless there is a mechanism to prevent P_0 and P_1 from accessing the variable `next_available_pid` **the same pid** could be assigned **to two different processes**!

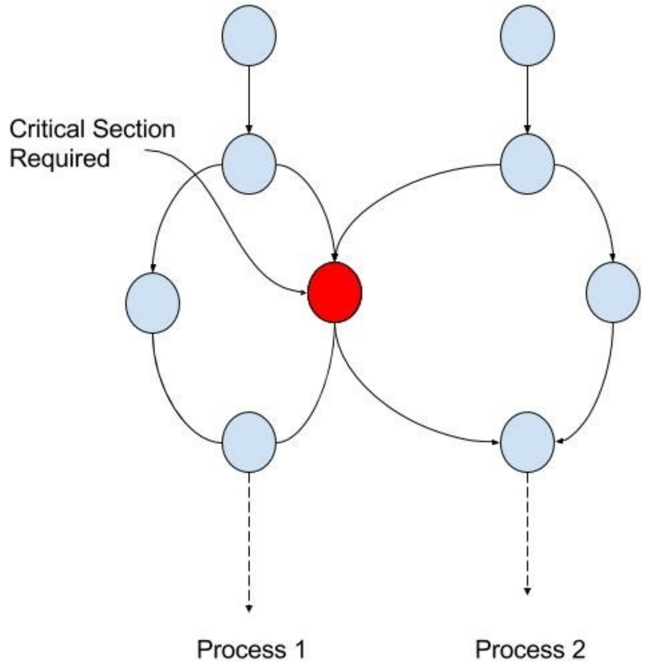
Critical Section Problem

- Consider system of n processes $\{p_0, p_1, \ldots p_{n-1}\}$
- Each process has **critical section** segment of code
  - Process may be changing common variables, updating table, writing file, etc.



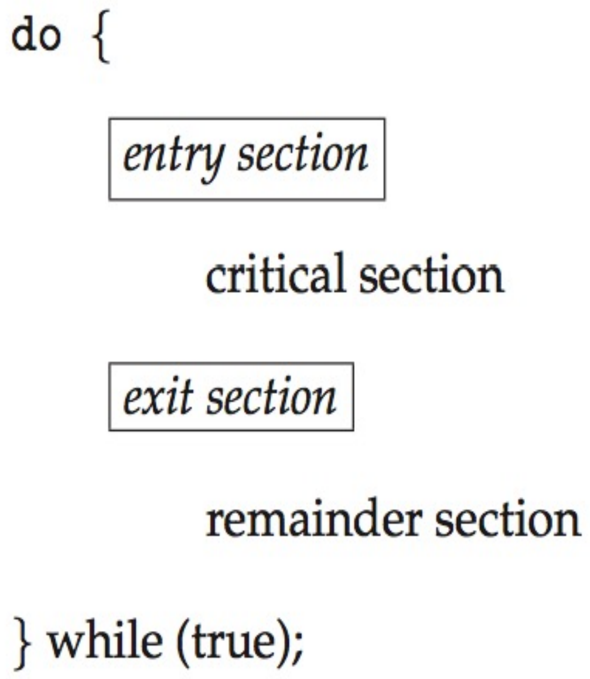
Critical Section Problem

- When one process in critical section, no other may be in its critical section.
- Critical section problem is to design protocol to solve this.



Critical Section

- Each process
  - must ask permission to enter critical section in **entry section**,
  - may follow critical section with **exit section**,
  - then **remainder section.**
- General structure of process ***P**$_i$*



```
do {
        entry section
            critical section
        exit section
            remainder section
} while (true);
```

Requirements for solution to critical-section problem

1. **Mutual Exclusion**

2. **Progress**

3. **Bounded Waiting**

1- Mutual Exclusion

- If process P_i is executing in its critical section, then no other processes can be executing in their critical sections.

- **No two processes simultaneously in critical region.**

2- Progress

- If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the process that will enter the critical section next cannot be postponed indefinitely.

- **No process running outside its critical region may block another process.**

3- Bounded Waiting

- A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.
  - Assume that each process executes at a nonzero speed
  - No assumption concerning **relative speed** of the n processes

- **No process must wait forever to enter its critical region.**