Process Synchronization (2)

Dr. Jun Zheng

CSE325 Principles of Operating Systems

9/30/2019

NEW MEXICO TECH
SCIENCE · ENGINEERING · RESEARCH · UNIVERSITY

Critical Section Problem

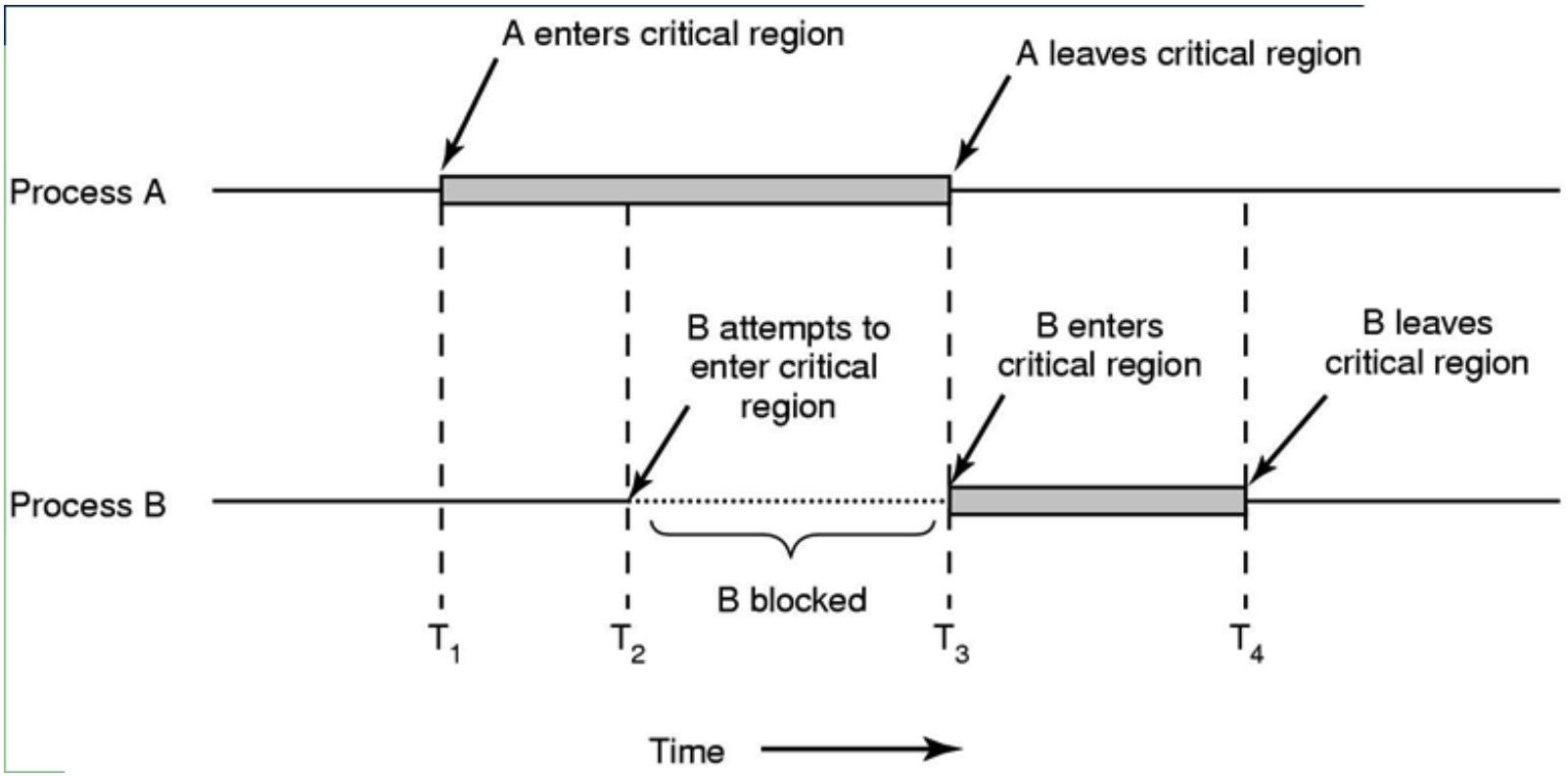
- Consider system of ***n*** processes {$p_0, p_1, \ldots p_{n-1}$}
- Each process has **critical section** segment of code
  - Process may be changing common variables, updating table, writing file, etc
  - When one process in critical section, no other may be in its critical section
- ***Critical section problem*** is to design protocol to solve this
- Each process must ask permission to enter critical section in **entry section**, may follow critical section with **exit section**, then **remainder section**

Critical Section

General structure of process P_i

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

Critical Region Illustrated



Solution to Critical-Section Problem

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

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 processes that will enter the critical section next cannot be postponed indefinitely

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

Peterson's Solution

- Good algorithmic description of solving the critical section problem
- Two process solution
- Assume that the **`load`** and **`store`** machine-language instructions are **atomic**; that is, cannot be interrupted
- The two processes share two variables:
  - **`int turn;`**
  - **`Boolean flag[2]`**
- The variable **`turn`** indicates whose turn it is to enter the critical section
- The **`flag`** array is used to indicate if a process is ready to enter the critical section. **`flag[i] = true`** implies that process P_i is ready!

Algorithm for Process P_i

```
do {
    flag[i] = true;
    turn = j;
    while (flag[j] && turn = = j); //busy wait
        critical section
    flag[i] = false;
        remainder section
} while (true);
```

Peterson's Solution (Cont.)

- Provable that the three CS requirement are met:
  1. Mutual exclusion is preserved

     P_i enters CS only if:

     either `flag[j] = false` or `turn = i`
  2. Progress requirement is satisfied
  3. Bounded-waiting requirement is met

Synchronization Hardware

- More solutions using techniques ranging from hardware to software-based APIs available to both kernel developer and application programmers.
- All solutions based on idea of **locking**
  - Protecting critical regions via locks
- Modern machines provide special atomic hardware instructions
  - **Atomic** = non-interruptible
  - Either test memory word and set value
  - Or swap contents of two memory words

Solution to Critical-section Problem Using Locks

```
do {
   acquire lock
      critical section
   release lock
      remainder section
} while (TRUE);
```

test_and_set Instruction

Definition:

```
boolean test_and_set (boolean *target)
{
        boolean rv = *target;
        *target = TRUE;
        return rv:
 }
```

1. Executed atomically
2. Returns the original value of passed parameter
3. Set the new value of passed parameter to "TRUE".

NEW MEXICO TECH
SCIENCE · ENGINEERING · RESEARCH · UNIVERSITY

Solution using test_and_set()

- Shared Boolean variable lock, initialized to FALSE
- Solution:

```
do {
   while (test_and_set(&lock))
      ; /* do nothing */
   /* critical section */
   lock = false;
   /* remainder section */
} while (true);
```