CS323 Operating Systems
Process Synchronization

Yuanyuan Zhou
Lecture 8
2/7/2003

Content of this lecture

- Administrative announcements
- Data Races
  - A simple game
- Critical region and mutual exclusion
- Mutual exclusion using busy waiting
  - Disabling Interrupts
  - Lock Variables
  - Strict Alternation
  - Peterson's solution
  - TSL
  - Sleep and Wakeup
- Summary

Administrative

- Quiz1 due Today 5pm
- MP1(thread scheduling)
- 1 unit project proposal due 2/24
- Hidden slides
  - Print out another version 1 day after the class

Review

- Process: a program in execution
- Threads: a light weight process
- Scheduling:
  - Which process/thread should run?

Inter-Process Communication (IPC)

- Communication
  - Pass information to each other
- Mutual exclusion & Synchronization
  - Keep each other's hair
  - Proper sequencing
- The last one also applies to threads

A simple game

- Two volunteers
  - Producer: produce 1 card per iteration
    - Step1: increment the counter
    - Step2: put the card on the table
  - Consumer:
    - Step1: check the counter to see if it is zero
    - Step2a: if the counter is zero, go back to step1
    - Step2b: if the counter is nonzero, take a card from the table
    - Step3: decrement counter
- I am the OS
  - I decide who should go, who should stop

First Round

- Stop Producer before step2 and let Consumer go.
- What happens?
- Two volunteers
  - Producer: produce 1 card per iteration
    - Step1: increment the counter
    - Step2: put the card on the table
  - Consumer:
    - Step1: check the counter to see if it is zero
    - Step2a: if the counter is zero, go back to step1
    - Step2b: if the counter is nonzero, take a card from the table
    - Step3: decrement the counter

switch

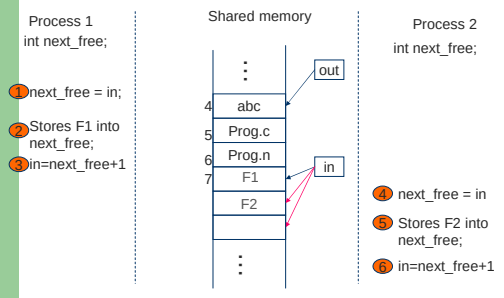
Second Round

- Stop Producer before step2 and let Consumer go.
- What happens?
- Two volunteers
  - Producer: produce 1 card per iteration
    - Step1: put the card on the table
    - Step2: increment the counter
  - Consumer:
    - Step1: check the counter to see if it is zero
    - Step2a: if the counter is zero, go back to step1
    - Step2b: if the counter is nonzero, take a card from the table
    - Step3: decrement the counter

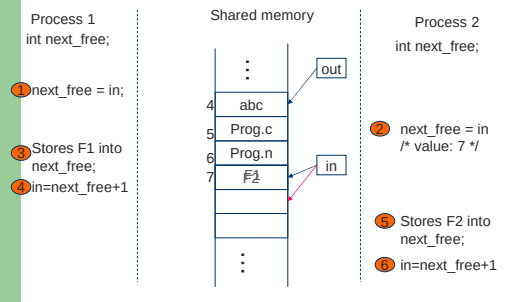
Data Races

- Reason: data sharing
- Previous game: producer and consumer
  - Share the counter
  - Share the cards
- Examples
  - Thread:
  - Process:

Spooling Example: Correct



Spooling Example: Races



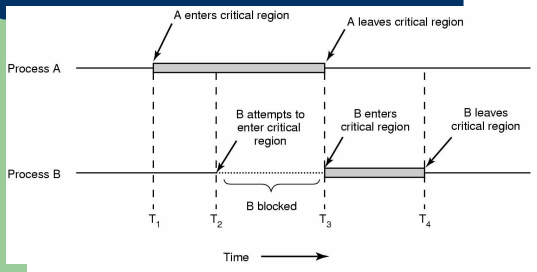
Critical Region (Critical Section)

```
Process {
  while (true) {
      ENTER CRITICAL SECTION
      Access shared variables; // Critical Section;
      LEAVE CRITICAL SECTION
      Do other work
  }
}
```

Critical Region Requirement

- **Mutual Exclusion:** No other process must execute within the critical section while a process is in it.
- **Progress:** If no process is waiting in its critical section and several processes are trying to get into their critical section, then entry to the critical section cannot be postponed indefinitely.
- **Bounded Wait:** A process requesting entry to a critical section should only have to wait for a bounded number of other processes to enter and leave the critical section.
- **Speed and Number of CPUs:** No assumption may be made about speeds or number of CPUs.

Critical Regions (2)



Mutual exclusion using critical regions

Mutual Exclusion With Busy Waiting

- Possible Solutions
  - Disabling Interrupts
  - Lock Variables
  - Strict Alternation
  - Peterson's solution
  - TSL
  - Sleep and Wakeup

Disabling Interrupts

- How does it work?
  - Disable all interrupts just after entering a critical section and re-enable them just before leaving it.
- Why does it work?
  - With interrupts disabled, no clock interrupts can occur. (The CPU is only switched from one process to another as a result of clock or other interrupts, and with interrupts disabled, no switching can occur.)
- Problems:
  - What if the process forgets to enable the interrupts?
  - Multiprocessor? (disabling interrupts only affects one CPU)
- Only used inside OS

Lock Variables

```
While (lock);
lock = 1;
EnterCriticalSection;
  access shared variable;
LeaveCriticalSection;
Lock = 0;
```

Does the above code work?

Strict Alternation

```
Thread Me; /* For two threads */
{
  while (true)
  {  while ( turn != my_thread_id) { };
     Access shared variables; // Critical Section;
     turn = other_thread_id;
     Do other work
  }
}
```

Satisfies mutual exclusion but not progress.
Why?

Using Flags

```
int flag[2]= {false, false};
Thread Me;
{
  while (true)
  {  flag[my_thread_id] = true;
     while (flag[other_thread_id] ) { };
     Access shared variables; // Critical Section;
     flag[my_thread_id] = false;
     Do other work
  }
}
```

Can block indefinitely

Why?

Peterson's Solution

```
 int flag[2]={false, false};
 int turn;
Thread Me;
{
while (true)
   {  flag[my_thread_id] = true;
      turn = other_thread_id;
      while (flag[other_thread_id]
            and turn == other_thread_id ) { };
      Access shared variables; // Critical Section;
      flag[my_thread_id] = false;
      Do other work
   }
}
```

It works!!!

Why?

Test & Set (TSL)

- Requires hardware support
- Does test and set atomically

```
char Test_and_Set ( char* target);
\\ All done atomicall
{    char temp = *target;
     *target = true;
     return(temp)
}
```


TSL instruction

```
enter_region:
     TSL REGISTER,LOCK          | copy lock to register and set lock to 1
     CMP REGISTER,#0            | was lock zero?
     JNE enter_region           | if it was non zero, lock was set, so loop
     RET | return to caller; critical region entered


leave_region:
     MOVE LOCK,#0               | store a 0 in lock
     RET | return to caller
```


Other Similar Hardware Instruction

- Swap = TSL

```
void Swap (char* x,* y);
\\ All done atomically

{    char temp = *x;
     *x = *y;
     *y = temp
}
```


Sleep and Wakeup

- Problem with previous solutions
  - Busy waiting
  - Wasting CPU
  - Priority Inversion:
    - a high priority waits for a low priority to leave the critical section
    - the low priority can never execute since the high priority is not blocked.
- Solution: sleep and wakeup
  - When blocked, go to sleep
  - Wakeup when it is OK to retry entering the critical section

Reminder

- Next lecture: Synchronization (chapter 2.3 & 2.4)
- Quiz1 due today at 5pm(only 1 try)
- MP1