Synchronization Mechanisms

CS 518

Marc Beck

11/19/2004

Outline

- What is synchronization
- Why are methods of synchronization needed
- Synchronization mechanisms in single processor computers
- Synchronization mechanisms in multiprocessor computers
- How kernel 2.6 complicates synchronization

Not Being Covered

Message passing systems for multiple processors

Deadlock – what it is, how to detect it, how to prevent it

Any architecture besides Intel

not related to the synchronization on computer systems

What is Synchronization

The act of bringing two or more processes to known points in their execution at the same clock time

Coordinating behavior of multithreaded processes, or multiple processes

Controlling access to shared resources or critical sections of code

Why is Synchronization needed

Synchronization is used to eliminate race conditions.

Process A:

N++;

Process B:

N++;

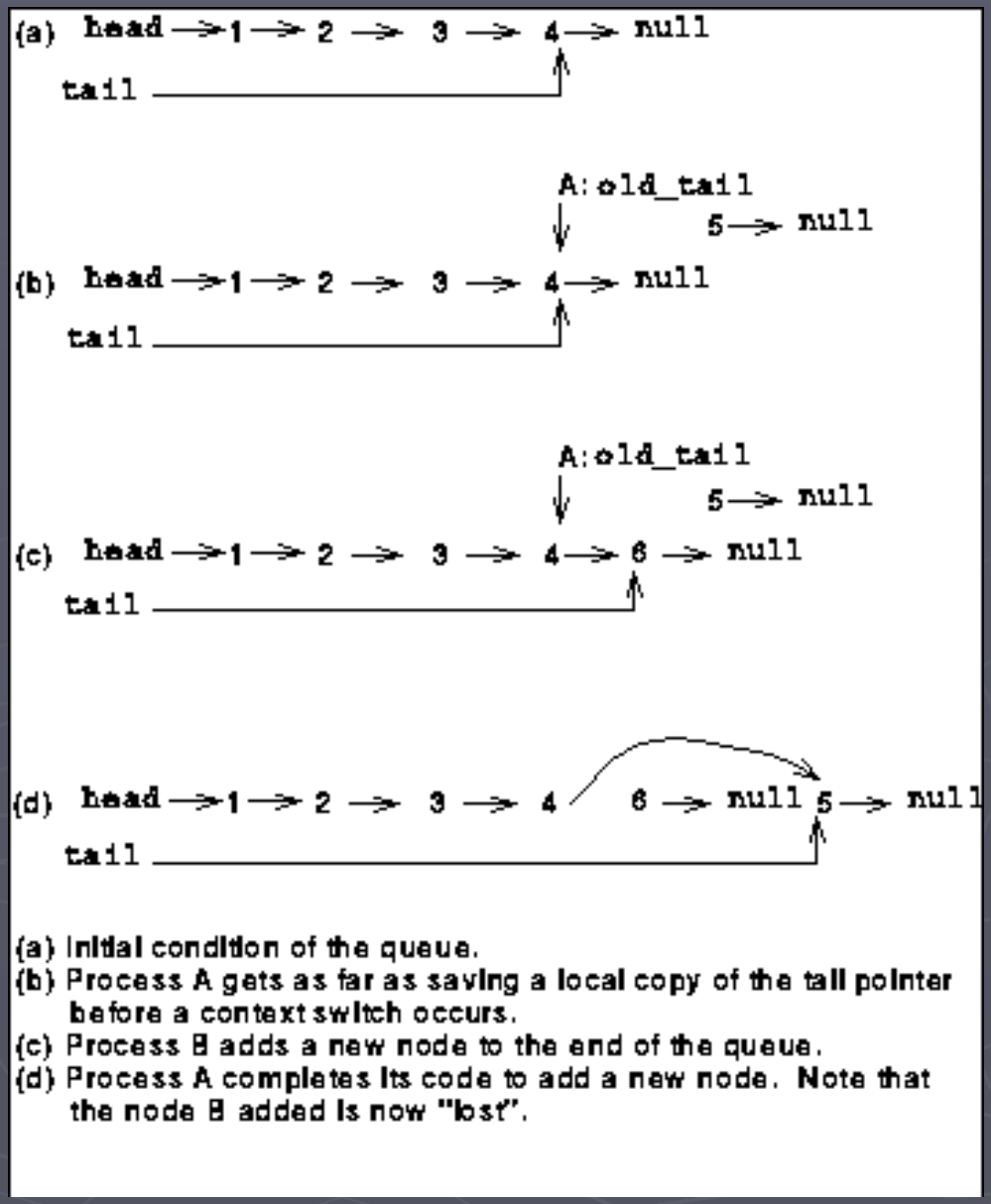
Load N to reg

reg = reg +1

Store reg to N

A: load N to acc (context switch)
B: load N to acc (context switch)
A: acc := acc + 1 (context switch)
B: acc := acc + 1 (context switch)
A: store acc to N (context switch)
B: store acc to N (context switch)

Another Race Condition Example



(a) Initial condition of the queue.
(b) Process A gets as far as saving a local copy of the tail pointer before a context switch occurs.
(c) Process B adds a new node to the end of the queue.
(d) Process A completes its code to add a new node. Note that the node B added is now "lost".

Synchronization methods for single processor computers

- Task Blocking
- Wait Queues
- Semaphore
- Mutex
- Monitor

Task Blocking

#define Task: a process running in kernel space

Tasks may have to wait for a certain condition before continuing execution (I/O, critical section, etc)

Task can block itself by changing state:

TASK_RUNNING to TASK_UNINTERRUPTABLE or TASK_INNTERUPTABLE

Task Blocking Cont.

In TASK_INTERRUPTABLE the task can be reawakened by a signal or interrupt

TASK_UNINTERRUPTABLE tasks can only be reawakened by an interrupt that changes the machine state such that the task is able to run again.

ie. I/O interrupt is thrown because I/O operation has completed.

Wait Queues

Linux provides a single fundamental abstract data structure used to maintain lists of tasks waiting on a condition: wait queues

Wait queues are circular list of elements in which each element contains a pointer to a task_struct process descriptor

The wait queue structure is defined in include/linux/wait.h

Wait Queues Cont.

- Tasks are inserted into wait queues by using the __add_wait_queue() function, and removed with __remove_wait_queue()

- These functions can also be found in include/linux/wait.h

Semaphores

Protected variable that restrics access to a shared resource

Value of the semaphore is the number of units of the resource that is free

for example: if a file can support a limited number of readers simultaneously and the semaphore controlling access = 4, then 4 more readers may access the file.

Mutex

MUTual EXclusion

Simply a binary semaphore

Used to prevent more than one process from accessing a recourse or critical section

Monitors

Monitors are software modules

Local data variables are accessible only by the monitor

Process enters monitor by invoking one of its procedures

Only one process may be executing in the monitor at a time

Invented by British computer scientist C. A. R. Hoare, most well known for the development of quicksort

Hardware Support needed to guarantee mutual exclusion

Processors must have functionality to support mutual exclusion

Test and Set Function

```
if (i == 0)
 {
       i = 1;
       return true;
 }
else
 {
       return false;
}
```

This operation is completely atomic

Hardware Support needed to guarantee mutual exclusion cont.

Atomic instructions:

On this implementation of a semaphore

```
P(Semaphore s)
 {   while (s <= 0) ; /* wait until s>0 */
      s = s-1; /* must be atomic operation */
 }

 V(Semaphore s)
 { s = s+1; /* must be atomic operation */
 }
```

The increment and decrement operations MUST be atomic on the processor, if they are not we may have a race condition and could lose mutual exclusion

Atomic Instruction Example

Asm/atomic.h

```
#ifdef CONFIG_SMP
12 #define LOCK "lock ; "
13 #else
14 #define LOCK ""
15 #endif

static __inline__ void atomic_inc(atomic_t *v)
106 {
107 __asm__ __volatile__(
108 LOCK "incl %0"
109 :"=m" (v->counter)
110 :"m" (v->counter));
111 }
```

Hardware Support needed to guarantee mutual exclusion cont.

Interrupt disabling

Because the linux kernel (2.4) functions cannot be preempted while they are executing, the ability to disable interrupts can give mutual exclusion (on single processor machines only)

Side note, I don't know how Intel's Hyperthreading technology affects this

Synchronization Mechanisms for multiprocessor systems

- Standard mutual exclusion techniques are not enough to guarantee mutual exclusion on a multiprocessor machine.

Spinlocks

- Because other processors can throw interrupts while another cpu is in a kernel function, disabling interrupts is not enough to guarantee mutual exclusion

- Spinlocks are a method of locking other processors when one is in a critical section

Structure of a Spinlock

Structure of a Spinlock

The structure of a spinlock is very simple, it contains a single shared variable

Spinlock Functions

The functions performed on spinlocks are simple and involve trying to obtain the variable in the spinlock or releasing it when the processor is done with the critical section. The source code for these functions can be found here:
http://www.tamacom.com/tour/kernel/linux/S/10389.html

Because this locks all processors from all critical sections they should be very short (ten's of lines) and execute quickly.

Another mechanism to reduce the overhead of spinlocks are reading and writing spinlocks

Other multiprocessor additions

In addition to spinlocks the Intel architecture provides commands to provide mutual exclusion: lock

LOCK - Lock Bus Usage: LOCK

LOCK: (386+ prefix)

Modifies flags: None This instruction is a prefix that causes the CPU assert bus lock signal during the execution of the next instruction. Used to avoid two processors from updating the same data location. The 286 always asserts lock during an XCHG with memory operands. This should only be used to lock the bus prior to XCHG, MOV, IN and OUT instructions.

Atomic Instruction Example (copy)

Asm/atomic.h

```
#ifdef CONFIG_SMP
12 #define LOCK "lock ; "
13 #else
14 #define LOCK ""
15 #endif

static __inline__ void atomic_inc(atomic_t *v)
106 {
107 __asm__ __volatile__(
108 LOCK "incl %0"
109 :"=m" (v->counter)
110 :"m" (v->counter));
111 }
```

Notice the #define LOCK as "lock ;" if CONFIG_SMP is defined. This means that if you are running on a multiprocessor machine atomic instructions get the special lock assembly instruction inserted.

Additional Multiprocessor Synchronization Mechanisms

Ticket lock

- Every process trying to receive the lock takes a "ticket" which is numbered and then waits on a global number
- Advantage: only one processor tries to obtain the lock when the lock is given up by the processor that was holding it
- Problem: all processors still spin on the same variable

Array-based lock

- Idea is to use fetch and increment (if that is supported by the cpu) to obtain a *unique location* in the array spin in which to spin (all processors then spin on different parts of an array)
- Acquire method uses a fetch and increment operation to obtain the next available location in the array.
- Release method writes "unlocked" to the next location in the array
- Each lock requires space proportional to the number of processors

Kernel 2.6

Kernel 2.6 is preemptable

This means a much finer grain of locking mechanism

- **preempt_enable()** -- decrements the preempt counter
- **preempt_disable()** -- increments the preempt counter
- **get_cpu()** -- calls preempt_disable() followed by a call to smp_processor_id()
- **put_cpu()** -- re-enables preemption()

These are defined in /include/linux/preempt.h which can be viewed here: http://www.tamacom.com/tour/kernel/linux/S/8567.html

Off Topic

"Computer science is no more about computers than astronomy is about telescopes."

Edsger Dijkstra

Sources

Kernel Projects for Linux
Gary Nutt

Cornell Theory Center
http://www2.sscc.ru/Litera/gloss/Glossary.html

http://www.mcs.drexel.edu/~shartley/OSusingSR/races.html\

Linux Kernel Source Tour
http://tamacom.com/tour/kernel/linux/

Operating Systems Internals and Design Principles
William Stallings

Wikipedia online dictionary
http://www.wikipedia.com

Leif Enblom: Mälardalen University
http://www.idt.mdh.se/kurser/ct3210/ht03/syllabus.php