## Chapter 2

**HZ**

how many times the timer interrupts fires in a second;

relates to how often the OS does process context changes;

larger the HZ value, more often the OS does context switches

Usually define in:

/linux/include/asm-generic/param.h

/linux/arch/arm/include/asm/param.h

#define HZ 100

**Jiffy**

Holds the number of times the system timer has fired its interrupt since the system booted.

Every second, the Jiffy increments by HZ.

**Long Delays**

Busy wait

The following code causes the CPU to wait 3 second:

Unsigned long timeout = jiffies + 3\*HZ;

While (time\_before(jiffies, timeout)) continue;

**Long Delay with Yield**

Yield CPU while wait

Unsigned long timeout = 3\*HZ;

set\_current\_state(TASK\_INTERRUPTABLE);

schedule\_timeout(timeout);

**Timer**

One-shot timer

#include <linux/timer.h>

Struct timer\_list my\_timer;

init\_timer(&my\_timer); // also see setup\_timer()

my\_timer.expire = jiffies +n\*HZ;

my\_timer.function = timer\_func;

my\_timer.data = func\_parameter;

add\_timer(&my\_timer);

static void timer\_func(unsigned long func\_parameter)

{

// do something here

// timer periodic firing

my\_timer.expire = jiffies +n\*HZ;

my\_timer.function = timer\_func;

my\_timer.data = func\_parameter;

add\_timer(&my\_timer);

}

**Short Delays**

mdelay, udelay, ndelay

these short delays cannot use jiffy, and thus they are busy-wait;

implementation typically runs a busy-loop internally, loops\_per\_jiffy

**Critical Section**

A code section that accesses shared resources

**Spinlocks**

Protects the critical section; ensure a single thread enters a critical section at a time; any other thread desires to enter the critical section has to remain spinning until the first thread exits.

spin\_lock, spin\_unlock

**Mutex**

Puts the waiting thread to sleep

mutex\_lock, mutex\_unlock

In an interrupt handler, uses spin-lock, otherwise use mutex.

Local\_irq\_disable(): disable interrupts in local CPU

Local\_irq\_enable(): enable interrupts in local CPU

Local\_irq\_save(): disable interrupts in local CPU

Local\_irq\_restore(): restore interrupts in local CPU

// save interrupt state; disable interrupt; disable preemption

spin\_lock\_irqsave; spin\_unlock\_irqrestore

On SMP systems, only interrupts on the local CPU are disabled when a spinlock is acquired.

A process context thread may enter a critical section, disable the local interrupt, while an interrupt handler may try to enter the critical section. In this case, the interrupt handler needs to busy wait by calling **spin\_lock**/**spin\_unlock**.

**Atomic operations**

Perform light-weight one-shot operations such as increment counters, set bit, etc.

Implementation is architecture dependent.

Check include/asm-your-arch/atomic.h for atomic operations support.

**Reader-Writer Locks**

A thread does read, the other does write, but don’t do both.

Multiple readers are allowed inside a critical section simultaneously.

If a writer enters a critical section, no reader and writer can enter inside.

read\_lock/unlock, write\_lock/unlock

read/write\_lock\_irqsave, read/write\_unlock\_irqrestore

**Sequence lock**: useful if write operations far outnumber read access

**Read-Copy Update**: useful if read operations far outnumber write access

**Debugging**

Enable **CONFIG\_SMP** and **CONFIG\_PREEMPT, CONFIG\_DEBUG\_SPINLOCK**

**Process File System**

Process file system is a virtual file system; gives data to the innards data of the kernel.

Data are generated by the kernel on the fly; gives information on the kernel config parameter kernel structure statistics, etc.

It’s a pseudo file system, and thus the file size are 0.

Examine the content of these files: cat /proc/cpuinfo, /proc/meminfo, /proc/stat, etc.

**Physical Address Space**

ZONE\_DMA: 0 – 16MB

ZONE\_NORMAL (LOW memory): 16 MB – 896 MB

ZONE\_HIGH (HIGH memory): 896 MB – 4 GB

**Virtual Address Space**

Default kernel configuration splits the 4GB address space into:

3GB virtual memory space for user processes

1GB space for kernel

Kmalloc is a memory-allocation function returns a contiguous memory from ZONE\_NORMAL.

Void\* kmalloc (int byte\_count, int flags);

Flags:

**GFP\_KERNAL**: use by process context code; kmalloc is allowed to go to sleep and wait for pages to get freed up

**GFP\_ATOMIC**: use by interrupt handler code; kmalloc is not allowed to go to sleep and wait for pages to get freed up

**GFP\_DMA**: allocates memory from the ZONE\_DMA