Operating Systems

Vlastimil Babka
Lubomír Bulej
Vojtěch Horký
Petr Tůma
# Table of Contents

1. Introduction .................................................................................................................................1
   - Basic Concepts ..........................................................................................................................1
   - Hardware Building Blocks .........................................................................................................1
   - Basic Computer Architecture ....................................................................................................1
   - Advances In Bus Architecture ..................................................................................................4
   - Operating System Structure ......................................................................................................7

2. Process Management ..................................................................................................................9
   - Process Alone ..................................................................................................................................9
   - Starting A Process ..........................................................................................................................9
   - What Is The Interface ....................................................................................................................16
   - Achieving Parallelism ....................................................................................................................20
   - Multiprocessing On Uniprocessors ...............................................................................................20
   - How To Decide Who Runs ............................................................................................................23
   - What Is The Interface ....................................................................................................................26
   - Process Communication ................................................................................................................28
   - Shared Memory .............................................................................................................................28
   - Message Passing ..........................................................................................................................29
   - Process Synchronization ..............................................................................................................32
   - Means For Synchronization ..........................................................................................................32
   - Memory Models ............................................................................................................................34
   - What Is The Interface ....................................................................................................................36

3. Memory Management ...............................................................................................................47
   - Management Among Processes ....................................................................................................47
   - Separating Multiple Processes .....................................................................................................47
   - What Is The Interface ....................................................................................................................47
   - Allocation Within A Process .........................................................................................................49
   - Process Memory Layout ................................................................................................................49
   - Stack ..............................................................................................................................................49
   - Heap .............................................................................................................................................50

4. Device Management ..................................................................................................................55
   - Device Drivers .............................................................................................................................55
   - Asynchronous Requests ..................................................................................................................55
   - Synchronous Requests ..................................................................................................................57
   - Devices ..........................................................................................................................................58
   - Busses .........................................................................................................................................58
   - Disk Storage Devices ......................................................................................................................62

5. File Subsystem ...........................................................................................................................67
   - File Subsystem ..............................................................................................................................67
   - Abstractions And Operations .......................................................................................................67
   - Stream File Operations ...............................................................................................................67
   - Example: Windows Stream File Operations ...............................................................................69
   - Mapped File Operations ..............................................................................................................71
   - Whole File Operations ..................................................................................................................73
   - Directory Operations ....................................................................................................................74
   - Sharing Support ............................................................................................................................76
   - Consistency Support .....................................................................................................................77
   - File Subsystem Internals .............................................................................................................78
   - Disk Layout .................................................................................................................................78
   - Integration Of Multiple File Subsystems ....................................................................................84
### 6. Network Subsystem

- Abstractions And Operations ......................................................... 95
- Sockets ............................................................................................ 95
- Network Subsystem Internals ............................................................ 98
- Queuing Architecture ...................................................................... 99
- Packet Filtering ............................................................................. 99
- Example: Linux Packet Scheduling .............................................. 101
- Network Subsystem Applications ................................................. 102
- File Systems .................................................................................. 102

### 7. Security Subsystem

- Authentication .................................................................................. 107
- Linux PAM Example ....................................................................... 107
- Authorization .................................................................................. 107
- Example: Security Enhanced Linux .............................................. 108
Chapter 1. Introduction

Basic Concepts

Hardware Building Blocks

Composing NAND And NOR Gates

Composing RS Flip Flops
Chapter 1. Introduction

Basic Computer Architecture

Basic Computer Architecture Example
Chapter 1. Introduction

Processor Bus

Timing Diagram Example

Example: ISA Bus

ISA Bus Read Cycle
Chapter 1. Introduction

ISA Bus Write Cycle

![ISA Bus Write Cycle Diagram]
Advances In Bus Architecture

Burst Access

Example: PCI Bus

PCI Bus Read Cycle
Chapter 1. Introduction

PCI Bus Write Cycle

Multiple Buses

Multiple Buses Example
Operating System Structure

Monolithic Systems

Simple Monolithic Operating Systems Example
Chapter 1. Introduction
Chapter 2. Process Management

Process Alone

Starting A Process

Bootstrapping

Example: Booting MSIM

Simulator Bootstrap Example
Simulator configuration fragment.

add rom loadermem 0x1FC00000
loadmem generic 4K
loadmem load "kernel/loader.bin"

Loader entry code fragment.

.globl __start
.ent __start
__start:
    la $ra, 0x80000400 ;hardcoded kernel entry address
    j $ra             ;jump
    nop              ;branch delay slot
.end __start

Kernel entry code fragment.

.section .excvec, "ax" ;emit code into .execvec section
    ;flags say allocatable and executable
.org 0x400 ;hardcoded kernel entry address
.globl start
.ent start
start:
    la $sp, 0x80000400 ;hardcoded stack pointer address
    jal main          ;jump and link
    nop              ;branch delay slot
    halt             ;a macro to stop the simulator
.end start

Linker script fragment.

SECTIONS {
    .kernel 0x80000000 : { /* output section kernel with address */
        *(.execvec) /* input section .execvec goes first */
        *(.text .text.*) /* .text sections come next */
        *(.rodata .rodata.*) /* .rodata sections next */
    }
Chapter 2. Process Management

Relocating

Absolute Addressing Example
Declaring and accessing a global variable in C.

```c
static int i; // declare a global variable
...
i = 0x12345678; // access the global variable
```

The C code compiled into Intel 80x86 assembler.

```assembly
.comm i,4,4 ; declare i as 4 bytes aligned at 4 bytes boundary
...
movl $0x12345678,i ; write value 12345678h into target address i
```

The assembler code compiled into Intel 80x86 machine code.

```
C705 ; movl
C0950408 ; target address 080495C0h
78563412 ; value 12345678h
```

Relative Addressing Example
Declaring and accessing a global variable in C.

```c
static int i; // declare a global variable
...
i = 0; // access the global variable
```

The C code compiled into position independent Intel 80x86 assembler.

```assembly
.comm i,4,4 ; declare i as 4 bytes aligned at 4 bytes boundary
... 
call __get_thunk ; get program starting address in ECX
addl $_GOT_,%ecx ; calculate address of global table of addresses in ECX
movl $0,i@GOT(%ecx) ; write value 0 into target address i relative from ECX
```

The assembler code compiled into position independent Intel 80x86 machine code.

```
E8 ; call
1C000000 ; target address 0000001Ch bytes away from here
81C1 ; addl target ECX
D9110000 ; value 000011D9h
C781 ; movl target address relative from ECX
20000000 ; target address 00000020h bytes away from ECX
00000000 ; value 00000000h
```

Example: Program Image In Intel HEX

Intel HEX Format

```
:LLAAAATTxxxxxCC
```

- LL - length of the data
- AAAA - address of the data in memory
- TT - indication of last line
Chapter 2. Process Management

- xxxx - data
- CC - checksum of the data

Example: Program Image In DOS

DOS EXE Format

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>2</td>
<td>Magic (0AA55h)</td>
</tr>
<tr>
<td>02h</td>
<td>2</td>
<td>Length of last block</td>
</tr>
<tr>
<td>04h</td>
<td>2</td>
<td>Length of file in 512B blocks (L)</td>
</tr>
<tr>
<td>06h</td>
<td>2</td>
<td>Number of relocation table entries (R)</td>
</tr>
<tr>
<td>08h</td>
<td>2</td>
<td>Length of header in 16B blocks (H)</td>
</tr>
<tr>
<td>0Ah</td>
<td>2</td>
<td>Minimum memory beyond program image in 16B blocks</td>
</tr>
<tr>
<td>0Ch</td>
<td>2</td>
<td>Maximum memory beyond program image in 16B blocks</td>
</tr>
<tr>
<td>0Eh</td>
<td>4</td>
<td>Initial stack pointer setting (SS:SP)</td>
</tr>
<tr>
<td>12h</td>
<td>2</td>
<td>File checksum</td>
</tr>
<tr>
<td>14h</td>
<td>4</td>
<td>Initial program counter setting (CS:IP)</td>
</tr>
<tr>
<td>18h</td>
<td>2</td>
<td>Offset of relocation table (1Ch)</td>
</tr>
<tr>
<td>1Ah</td>
<td>2</td>
<td>Overlay number</td>
</tr>
<tr>
<td>1Ch</td>
<td>R*4h</td>
<td>Relocation table entries</td>
</tr>
<tr>
<td>H*10h</td>
<td>L*200h</td>
<td>Program image</td>
</tr>
</tbody>
</table>

Linking

Example: Executable And Linking Format

Headers

ELF Executable Header Example

> readelf --file-header /bin/bash

ELF Header:

- Magic: 7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
- Class: ELF64
- Data: 2’s complement, little endian
- Version: 1 (current)
- OS/ABI: UNIX - System V
- ABI Version: 0
- Type: EXEC (Executable file)
- Machine: Advanced Micro Devices X86-64
- Version: 0x1
- Entry point address: 0x41d238
- Start of program headers: 64 (bytes into file)
- Start of section headers: 964960 (bytes into file)
- Flags: 0x0
- Size of this header: 64 (bytes)
- Size of program headers: 56 (bytes)
- Number of program headers: 10
- Size of section headers: 64 (bytes)
- Number of section headers: 32
- Section header string table index: 31
Chapter 2. Process Management

**ELF Library Header Example**

> readelf --file-header /lib/libc.so.6

ELF Header:
- Magic: 7f 45 4c 46 01 01 01 03 00 00 00 00 00 00 00 00
- Class: ELF32
- Data: 2’s complement, little endian
- Version: 1 (current)
- OS/ABI: UNIX - GNU
- ABI Version: 0
- Type: DYN (Shared object file)
- Machine: Intel 80386
- Version: 0x1
- Entry point address: 0x44564790
- Start of program headers: 52 (bytes into file)
- Start of section headers: 2009952 (bytes into file)
- Flags: 0x0
- Size of this header: 52 (bytes)
- Size of program headers: 32 (bytes)
- Number of program headers: 10
- Size of section headers: 40 (bytes)
- Number of section headers: 43
- Section header string table index: 42

**Sections**

**ELF Sections Example**

> readelf --sections /lib/libc.so.6

There are 43 section headers, starting at offset 0x1eab60:

Section Headers:

<table>
<thead>
<tr>
<th>[Nr]</th>
<th>Name</th>
<th>Type</th>
<th>Addr</th>
<th>Off</th>
<th>Size</th>
<th>ES Flg Lk Inf Al</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>.rel.dyn</td>
<td>REL</td>
<td>4455f3e4</td>
<td>0143e4</td>
<td>002a18</td>
<td>A 4 0 4</td>
</tr>
<tr>
<td>[10]</td>
<td>.rel.plt</td>
<td>REL</td>
<td>44561dfc</td>
<td>016dfc</td>
<td>000058</td>
<td>A 4 11 4</td>
</tr>
<tr>
<td>[11]</td>
<td>.plt</td>
<td>PROGBITS</td>
<td>44561e60</td>
<td>016e60</td>
<td>0000c0</td>
<td>AX 0 0 16</td>
</tr>
<tr>
<td>[12]</td>
<td>.text</td>
<td>PROGBITS</td>
<td>44561f20</td>
<td>016f20</td>
<td>14010c</td>
<td>AX 0 0 16</td>
</tr>
<tr>
<td>...</td>
<td>.data</td>
<td>PROGBITS</td>
<td>446f9040</td>
<td>1ad040</td>
<td>000e7c</td>
<td>WA 0 0 32</td>
</tr>
<tr>
<td>[33]</td>
<td>.bss</td>
<td>NOBITS</td>
<td>446f9ec0</td>
<td>1adebc</td>
<td>002bfc</td>
<td>WA 0 0 32</td>
</tr>
<tr>
<td>[34]</td>
<td>.comment</td>
<td>PROGBITS</td>
<td>00000000</td>
<td>1adebc</td>
<td>00002c</td>
<td>01 MS 0 0 1</td>
</tr>
<tr>
<td>[35]</td>
<td>.note.stapsdt</td>
<td>NOTE</td>
<td>00000000</td>
<td>ladee8</td>
<td>002c4</td>
<td>0 0 4</td>
</tr>
<tr>
<td>[36]</td>
<td>.symtab</td>
<td>SYMTAB</td>
<td>00000000</td>
<td>laelac</td>
<td>021880</td>
<td>10 37 6229 4</td>
</tr>
<tr>
<td>[37]</td>
<td>.strtab</td>
<td>STRTAB</td>
<td>00000000</td>
<td>lcfa2c</td>
<td>01a786</td>
<td>00 0 0 1</td>
</tr>
</tbody>
</table>

... Key to Flags:
- W (write), A (alloc), X (execute), M (merge), S (strings)
- I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
- O (extra OS processing required) o (OS specific), p (processor specific)
Chapter 2. Process Management

**ELF Relocations Example**

```
> readelf --relocs /lib/libc.so.6
```

Relocation section '.rel.dyn' at offset 0x134e4 contains 1457 entries:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Info</th>
<th>Type</th>
<th>Sym.Value</th>
<th>Sym. Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>436f71b0</td>
<td>00000008</td>
<td>R_386_RELATIVE</td>
<td>436f8e74</td>
<td>stderr</td>
</tr>
<tr>
<td>436f8e74</td>
<td>0000000e</td>
<td>R_386_TLS_TPOFF</td>
<td>436f9d7c</td>
<td>malloc</td>
</tr>
<tr>
<td>436f9008</td>
<td>004de07</td>
<td>R_386_JUMP_SLOT</td>
<td>435c3b70</td>
<td>malloc</td>
</tr>
</tbody>
</table>

```
...
```

**Segments**

**ELF Segments Example**

```
> readelf --segments /bin/bash
```

Elf file type is EXEC (Executable file)
Entry point 0x41d238
There are 10 program headers, starting at offset 64

Program Headers:

<table>
<thead>
<tr>
<th>Type</th>
<th>Offset</th>
<th>VirtAddr</th>
<th>PhysAddr</th>
<th>FileSiz</th>
<th>MemSiz</th>
<th>Flags</th>
<th>Align</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHDR</td>
<td>0x0000000000000040</td>
<td>0x0000000000400040</td>
<td>0x0000000000000230</td>
<td>R E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTERP</td>
<td>0x0000000000000270</td>
<td>0x0000000000400270</td>
<td>0x0000000000000270</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOAD</td>
<td>0x0000000000000000</td>
<td>0x0000000000400000</td>
<td>0x00000000000d60000</td>
<td>R E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DYNAMIC</td>
<td>0x00000000000001f0</td>
<td>0x00000000006000f0</td>
<td>0x00000000000001f0</td>
<td>RW</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STACK</td>
<td>0x0000000000000000</td>
<td>0x0000000000000000</td>
<td>0x0000000000000000</td>
<td>RW</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Section to Segment mapping:
Segment Sections...
00 .interp
02 .interp .note .dynsym .rela .init .fini .plt .text...

**ELF Dynamic Information Example**

```
> readelf --dynamic /bin/bash
```

Dynamic section at offset 0xd9df0 contains 30 entries:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Type</th>
<th>Name/Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000000000000001</td>
<td>(NEEDED)</td>
<td>Shared library: [libtinfo.so.5]</td>
</tr>
<tr>
<td>0x0000000000000001</td>
<td>(NEEDED)</td>
<td>Shared library: [libdl.so.2]</td>
</tr>
<tr>
<td>0x0000000000000001</td>
<td>(NEEDED)</td>
<td>Shared library: [libc.so.6]</td>
</tr>
<tr>
<td>0x000000000000000c</td>
<td>(INIT)</td>
<td>0x41adb0</td>
</tr>
<tr>
<td>0x000000000000000d</td>
<td>(FINI)</td>
<td>0x4a6374</td>
</tr>
<tr>
<td>0x0000000000000019</td>
<td>(INIT_ARRAY)</td>
<td>0x6d9dd8</td>
</tr>
<tr>
<td>0x0000000000000005</td>
<td>(STRTAB)</td>
<td>0x8e2b30</td>
</tr>
<tr>
<td>0x0000000000000006</td>
<td>(SYMTAB)</td>
<td>0x403988</td>
</tr>
</tbody>
</table>

...
Chapter 2. Process Management

Calling Operating System

Example: CP/M System Call Interface

CP/M BDOS System Call Example

ReadKey:       mvi c,1 ; keyboard read service
call 5 ; call BDOS entry point
cpi a,0Dh ; is returned key code ENTER ?
jnz ReadKey ; repeat keyboard read until it is

CP/M BIOS System Call Entry Points

jmp   BOOT   ;cold boot
jmp   WBOOT  ;warm boot
jmp   CONST  ;console status
jmp   CONIN  ;console input
...   jmp   HOME  ;disk head to track 0
...   jmp   SETDMA ;set memory transfer address
jmp   READ   ;read sector
jmp   WRITE  ;write sector

Example: Linux System Call API On Intel 80x86

Library System Call Example

ssize_t read (int fd, void *buf, size_t count);
...
int hFile;
ssize_t iCount;
char abBuffer [1024];
iCount = read (hFile, &abBuffer, sizeof (abBuffer));

pushl $1024 ;sizeof (abBuffer)
pushl $abBuffer ;&abBuffer
pushl hFile ;hFile
call read ;call the library
addl $12,%esp ;remove arguments from stack
movl %eax,iCount ;save result

Linux Gate Library Based On INT 80h

__kernel_vsyscall: int $0x80
    ret

Linux Gate Library Based On SYSENTER And SYSEXIT

__kernel_vsyscall: push %ecx
    push %edx
    push %ebp
__resume: mov %esp,%ebp
    sysenter
Example: Linux Syslet API

Syslet Atom Structure

struct syslet_uatom
{
    u32 flags;
    u32 nr;       // what syscall to execute
    u64 ret_ptr;  // where to store return value
    u64 next;     // what is the next atom in the chain
    u64 arg_ptr[6]; // what are the syscall arguments
    u64 private;  // free for application use
};

- SYSLET_STOP_ON_ZERO - termination condition value
- SYSLET_STOP_ON_NONZERO
- SYSLET_STOP_ON_NEGATIVE
- SYSLET_SKIP_TO_NEXT_ON_STOP - conditional branching in atom array

Syslet Usage Example

struct request
{
    u64 filename_ptr;
    u64 fd;

    struct syslet_uatom open_file;
    struct syslet_uatom read_file;
    struct syslet_uatom close_file;
};

request req;

req.open_file.nr = __NR_sys_open;
req.open_file.arg_ptr[0] = (u64)&req.filename_ptr;
...
req.open_file.ret_ptr = (u64)&req.fd;
req.open_file.flags = (u64)SYSLET_STOP_ON_NEGATIVE;
req.open_file.next = (u64)&req.read_file;
...

struct syslet_uatom *done;
done = sys_async_exec(&req.open_file ...);
Example: Windows System Call API On Intel 80x86

Library System Call Example

```c
int MessageBox (  
    HWND hwndOwner,  
    LPCTSTR lpszText,  
    LPCTSTR lpszTitle,  
    UINT fuStyle);  
...
MessageBox (0, zMessageText, zWindowTitle, MB_OK || MB_SYSTEMMODAL || MB_ICONHAND);
```

push MBOK or MB_SYSTEMMODAL or MB_ICONHAND
push offset zWindowTitle
push offset zMessageTest
push 0
call MessageBoxA ;call the library
add esp,16 ;remove arguments from stack

What Is The Interface

Example: Posix Process And Thread API

Posix Process Creation System Calls

```c
pid_t fork (void);
int execve (const char *filename, char *const argv [], char *const envp []);
```
```c
pid_t wait (int *status);
pid_t waitpid (pid_t pid, int *status, int options);
```
```c
void exit (int status);
```

Posix Thread Creation System Calls

```c
int pthread_create (  
    pthread_t *thread,  
    pthread_attr_t *attr,  
    void * (*start_routine) (void *),  
    void *arg);
```
```c
int pthread_join (  
    pthread_t thread,  
    void **return_value);
```
```c
void pthread_exit (  
    void *return_value);
```
```c
int pthread_detach (  
    pthread_t thread);
```
Posix Thread Specific Data Calls

```c
int pthread_key_create (
    pthread_key_t *key,
    void (* destructor) (void *));

int pthread_setspecific (
    pthread_key_t key,
    const void *value);

void *pthread_getspecific (
    pthread_key_t key);
```

Example: Windows Process And Thread API

Windows Process Creation System Calls

```c
BOOL CreateProcess (  
    LPTSTR lpApplicationName,  
    LPTSTR lpCommandLine,  
    LPSECURITY_ATTRIBUTES lpProcessAttributes,  
    LPSECURITY_ATTRIBUTES lpThreadAttributes,  
    BOOL bInheritHandles,  
    DWORD dwCreationFlags,  
    LPVOID lpEnvironment,  
    LPCTSTR lpCurrentDirectory,  
    LPPROCESS_INFORMATION lpProcessInformation
);

VOID ExitProcess (  
    UINT uExitCode);

DWORD WaitForSingleObject (  
    HANDLE hHandle,  
    DWORD dwMilliseconds
);
```

Windows Thread Creation System Calls

```c
HANDLE CreateThread (  
    LPSECURITY_ATTRIBUTES lpThreadAttributes,  
    SIZE_T dwStackSize,  
    LPTHREAD_START_ROUTINE lpStartAddress,  
    LPVOID lpParameter,  
    DWORD dwCreationFlags,  
    LPDWORD lpThreadId
);

VOID ExitThread (  
    DWORD dwExitCode);
```
Chapter 2. Process Management

Windows Fiber Creation System Calls

LPVOID ConvertThreadToFiber (LPVOID lpParameter);

LPVOID CreateFiber (SIZE_T dwStackSize, LPFIBER_START_ROUTINE lpStartAddress, LPVOID lpParameter);

VOID SwitchToFiber (LPVOID lpFiber);

VOID DeleteFiber (LPVOID lpFiber);

Windows Thread Local Data Calls

DWORD TlsAlloc (void);
BOOL TlsFree (DWORD dwTlsIndex);

BOOL TlsSetValue (DWORD dwTlsIndex, LPVOID lpTlsValue);
LPVOID TlsGetValue (DWORD dwTlsIndex);

Windows Fiber Local Data Calls

DWORD FlsAlloc (PFLS_CALLBACK_FUNCTION lpCallback);
BOOL FlsFree (DWORD dwFlsIndex);

BOOL FlsSetValue (DWORD dwFlsIndex, PVOID lpFlsValue);
PVOID FlsGetValue (DWORD dwFlsIndex);

Windows Stack Guarantee Call

BOOL SetThreadStackGuarantee (PULONGStackSizeInBytes);

Example: Linux Clone API

Linux Clone Call

int clone (int (*fn) (void *), void *child_stack, int flags, void *arg, ...);
• CLONE_SIGHAND - share signal handler table
• CLONE_FILES - share file descriptor table
• CLONE_FS - share filesystem context (root directory, current directory, file creation mask)
• CLONE_IO - share device scheduler context
• CLONE_VM - share address space

• CLONE_NEWIPC - create new process communication namespace
• CLONE_NEWPID - create new process identifier namespace
• CLONE_NEWUTS - create new host platform namespace
• CLONE_NEWNET - create new network namespace
• CLONE_NEWNS - create new mount namespace

Example: Posix Dynamic Linker API

Posix Dynamic Linker Calls

void *dlopen (const char *filename, int flag);
int dlclose (void *handle);

• RTLD_LAZY - resolve symbols on use when possible
• RTLD_GLOBAL - make symbols available to other libraries

void *dlsym (void *handle, const char *symbol);

• RTLD_DEFAULT - use default symbol lookup order
• RTLD_NEXT - lookup in libraries that follow this

Example: Java Thread API

Java Thread Class

class java.lang.Thread implements java.lang.Runnable {
    java.lang.Thread ();
    java.lang.Thread (java.lang.Runnable);
    void start ();
    void run ();
    void interrupt ();
    boolean isInterrupted ();

    void join () throws java.lang.InterruptedIOException;
    void setDaemon (boolean);
}
Chapter 2. Process Management

boolean isDaemon ();
static java.lang.Thread currentThread ();
static void yield ();
static void sleep (long) throws java.lang.InterruptedIOException;
...

Example: OpenMP Thread API

OpenMP Thread Creation Directives

#pragma omp parallel private (iThreads, iMyThread)
{
    iThreads = omp_get_num_threads ();
    iMyThread = omp_get_thread_num ();
    ...
}

#pragma omp parallel for
    for (i = 0 ; i < MAX ; i++)
        a [i] = 0;

#pragma omp parallel sections
{
    #pragma omp section
        DoOneThing ();
    #pragma omp section
        DoAnotherThing ();
}

Achieving Parallelism

Multiprocessing On Uniprocessors

Processor State

Example: Linux Processor Context Switching

Linux 2.6.8 80x86 Processor Context Switching

#define __SAVE_ALL \
cld; \
pushl %es; \
pushl %ds; \
pushl %eax; \
pushl %ebp; \
pushl %edi; \
pushl %esi; \
pushl %edx; \
pushl %ecx; \

Chapter 2. Process Management

```c
pushl %ebx; 
movl $(__USER_DS), %edx; 
movl %edx, %ds; 
movl %edx, %es;

#define __RESTORE_INT_REGS 
  popl %ebx; 
  popl %ecx; 
  popl %edx; 
  popl %esi; 
  popl %edi; 
  popl %ebp; 
  popl %eax

#define __RESTORE_REGS 
  __RESTORE_INT_REGS; 
  111: popl %ds; 
  222: popl %es; 
  .section .fixup,"ax"; 
  444: movl %0, (%esp); 
  jmp 111b; 
  555: movl %0, (%esp); 
  jmp 222b; 
  .previous; 
  .section __ex_table,"a"; 
    .align 4; 
    .long 111b, 444b; 
    .long 222b, 555b; 
    .previous

#define __RESTORE_ALL 
  __RESTORE_REGS 
  __RESTORE_INT_REGS; 
  333: iret; 
  .section .fixup,"ax"; 
  666: sti; 
  movl $(__USER_DS), %edx; 
  movl %edx, %ds; 
  movl %edx, %es; 
  pushl %11; 
  call do_exit; 
  .previous; 
  .section __ex_table,"a"; 
    .align 4; 
    .long 333b, 666b; 
    .previous

#define SAVE_ALL  
  __SAVE_ALL; 
  __SWITCH_KERNELSPACE;

#define RESTORE_ALL  
  __SWITCH_USERSPACE; 
  __RESTORE_ALL;
```
Chapter 2. Process Management

Example: Kalisto Processor Context Switching

Kalisto MIPS Processor Context Switching

.macro SAVE_REGISTERS base
    sw $zero, REGS_OFFSET_ZERO(base)
    sw $at, REGS_OFFSET_AT(base)
    sw $v0, REGS_OFFSET_V0(base)
    sw $v1, REGS_OFFSET_V1(base)
    sw $a0, REGS_OFFSET_A0(base)
    sw $a1, REGS_OFFSET_A1(base)
    sw $a2, REGS_OFFSET_A2(base)
    sw $a3, REGS_OFFSET_A3(base)
    ...
    sw $gp, REGS_OFFSET_GP(base)
    sw $fp, REGS_OFFSET_FP(base)
    sw $ra, REGS_OFFSET_RA(base)
.endm SAVE_REGISTERS

.macro LOAD_REGISTERS base
    lw $ra, REGS_OFFSET_RA(base)
    lw $fp, REGS_OFFSET_FP(base)
    lw $gp, REGS_OFFSET_GP(base)
    ...
    lw $a3, REGS_OFFSET_A3(base)
    lw $a2, REGS_OFFSET_A2(base)
    lw $a1, REGS_OFFSET_A1(base)
    lw $a0, REGS_OFFSET_A0(base)
    lw $v1, REGS_OFFSET_V1(base)
    lw $v0, REGS_OFFSET_V0(base)
    lw $at, REGS_OFFSET_AT(base)
    lw $zero, REGS_OFFSET_ZERO(base)
.endm LOAD_REGISTERS

switch_cpu_context:

    /* Allocate a frame on the stack of the old thread and update
    the address of the stack top of the old thread. */
    addiu $sp, -CONTEXT_SIZE ;Allocate space on stack
    sw $sp, ($a0) ;Save the old stack
    SAVE_REGISTERS $sp ;Save general registers
    mflo $t0 ;Few other registers that
    mfhi $t1 ;the macro does not handle
    sw $t0, REGS_OFFSET_LO($sp) ;need to be saved as well
    sw $t1, REGS_OFFSET_HI($sp)
Chapter 2. Process Management

```assembly
mfc0 $t0, $status
sw $t0, REGS_OFFSET_STATUS($sp)
la $t1, ~CP0_STATUS_IE_MASK
and $t0, $t1
mtc0 $t0, $status ; Disable interrupts
lw $sp, ($a1) ; Switch to the new stack
lw $t0, REGS_OFFSET_LO($sp) ; Restore the registers in
lw $t1, REGS_OFFSET_HI($sp) ; roughly the opposite
mtlo $t0 ; order to fit the
mthi $t1 ; stack semantics
LOAD_REGISTERS $sp
lw $k0, REGS_OFFSET_STATUS($sp)
addiu $sp, CONTEXT_SIZE ; Free space on stack
j $ra ; Return to the newly
mtc0 $k0, $status ; restored context
```

How To Decide Who Runs

Scheduling Requirements

- Responsiveness - reacting in reasonable time.
- Predictability - scheduling in consistent manner.
- Turnaround - minimizing time to complete a single task.
- Throughput - maximizing number of completed tasks.
- Efficiency - maximizing resource utilization.
- Fairness - treating tasks equally.

Application Classes

- Interactive - mostly waits, needs quick reaction to events.
- Batch - mostly works, benefits from low trashing.
- Realtime - needs guaranteed timing.

Example: Solaris Scheduler

Solaris Real Time Dispatcher Configuration

```
> /usr/sbin/dispadmin -c RT -g
# Real Time Dispatcher Configuration
RES=1000
# TIME QUANTUM PRIORITY
# (rt_quantum) LEVEL
```
Chapter 2. Process Management

Solaris Time Sharing Dispatcher Configuration

```
> /usr/sbin/dispadmin -c TS -g
# Time Sharing Dispatcher Configuration
RES=1000

  # ts_quantum  ts_tqexp  ts_slpret  ts_maxwait  ts_lwait  PRIORITY  LEVEL
  200  0  50  0  50  #  0
  160  1  51  0  51  #  1
  160  2  51  0  51  #  2
  160  3  51  0  51  #  3
  160  4  51  0  51  #  4
  160  5  51  0  51  #  5
  160  6  51  0  51  #  6
  160  7  51  0  51  #  7
  160  8  51  0  51  #  8
  160  9  51  0  51  #  9
  120 10  52  0  52  # 10
  120 11  52  0  52  # 11
  120 12  52  0  52  # 12
  120 13  52  0  52  # 13
  120 14  52  0  52  # 14
  120 15  52  0  52  # 15
  120 16  52  0  52  # 16
  120 17  52  0  52  # 17
  120 18  52  0  52  # 18
  120 19  52  0  52  # 19
  120 20  52  0  52  # 20
  40  45  58  0  59  # 55
  40  46  58  0  59  # 56
  40  47  58  0  59  # 57
  40  48  58  0  59  # 58
  20  49  59  32000 59  # 59
```

Example: Windows Scheduler

Windows Thread Priority Calculation

<table>
<thead>
<tr>
<th></th>
<th>Idle</th>
<th>Normal</th>
<th>High</th>
<th>Realtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>Lowest</td>
<td>2</td>
<td>6</td>
<td>11</td>
<td>22</td>
</tr>
<tr>
<td>Below Normal</td>
<td>3</td>
<td>7</td>
<td>12</td>
<td>23</td>
</tr>
<tr>
<td>Normal</td>
<td>4</td>
<td>8</td>
<td>13</td>
<td>24</td>
</tr>
<tr>
<td>Above Normal</td>
<td>5</td>
<td>9</td>
<td>14</td>
<td>25</td>
</tr>
<tr>
<td>Highest</td>
<td>6</td>
<td>10</td>
<td>15</td>
<td>26</td>
</tr>
<tr>
<td>Time Critical</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>31</td>
</tr>
</tbody>
</table>
Example: Linux Late 2.6.X Series Scheduler

CFS Scheduler Tunables

sched_min_granularity_ns
The minimum time a task will run unless it blocks.

sched_latency_ns
The turn around period of the run queue.

sched_wakeup_granularity_ns
The minimum time a previously sleeping task will run unless it blocks.

Example: Advanced Linux Scheduling Features

Scheduler Control Groups Example

> tree -d /sys/fs/cgroup

/sys/fs/cgroup |
|-- blkio
|-- cpu
| `-- system
|   |-- acpid.service
|   |-- chronyd.service
|   |-- crond.service
|   |-- dbus.service
|   |-- httpd.service
|   |-- mdmonitor.service
|...```
|   |-- sshd.service
|   |-- udev.service
|   `-- upower.service
|-- devices
|-- memory
...

Control Groups Configuration Example

group singlecore {
  perm {
    admin {
      # can change settings (e.g. cpumask)
      # or reset statistics
      uid = root;
    }
    task {
      # can run the tasks
      uid = user;
    }
  }
  cpuset {
    # limit to single CPU
    cpuset.cpus = 3;
    cpuset.mems = 0;
  }
  cpuacct {

25
Chapter 2. Process Management

```bash
cgexec -g cpuset:singlecore /home/user/some_script.sh
```

**Scheduler Control Group Tunables**

cpu.shares
Processor share this group gets relative to other groups.

cpu.cfs_period_us
Defines a period for the purpose of limiting the CFS scheduler bandwidth.

cpu.cfs_quota_us
Limits the CFS scheduler to particular quota within each period above.

cpu.rt_period_us
Defines a period for the purpose of limiting the RT scheduler bandwidth.

cpu.rt_runtime_us
Limits the RT scheduler to particular quota within each period above.

**What Is The Interface**

**Example: Linux Scheduler API**

**Linux Old Style Scheduler Calls**

```c
int getpriority (int which, int who);
int setpriority (int which, int who, int prio);
```

- PRIO_PROCESS - set or get process priority
- PRIO_PGRP - set priority of all in group, get highest
- PRIO_USER - set priority of all owned by user, get highest

**Linux New Style Scheduler Calls**

```c
int sched_setscheduler (pid_t pid, 
   int policy, 
   const struct sched_param *param);
int sched_setscheduler (pid_t pid);

int sched_setparam (pid_t pid, const struct sched_param *param);
int sched_setparam (pid_t pid, struct sched_param *param);

int sched_getparam (pid_t pid, struct sched_param *param);
int sched_getparam (pid_t pid, const struct sched_param *param);
```

```c
struct sched_param
{
   ...
```
Chapter 2. Process Management

```c
int sched_priority;
...

int sched_yield (void);
```

- SCHED_OTHER - normal time sharing process
- SCHED_BATCH - processor intensive time sharing process
- SCHED_IDLE - time sharing process to be run very rarely
- SCHED_FIFO - static priority with unlimited quantum
- SCHED_RR - static priority with limited quantum

**Linux Scheduler Calls**

```c
int sched_setaffinity (    pid_t pid,    size_t cpusetsize,    cpu_set_t *mask);    int sched_getaffinity (    pid_t pid,    size_t cpusetsize,    cpu_set_t *mask);    
```

**Example: Windows Scheduler API**

**Windows Scheduler Calls**

```c
BOOL SetPriorityClass (    HANDLE hProcess,    DWORD dwPriorityClass);    DWORD GetPriorityClass (    HANDLE hProcess);    
```

```c
BOOL SetThreadPriority (    HANDLE hThread,    int nPriority);    int GetThreadPriority (    HANDLE hThread);    
```

```c
BOOL SetProcessPriorityBoost (    HANDLE hProcess,    BOOL DisablePriorityBoost);    BOOL SetThreadPriorityBoost (    HANDLE hThread,    BOOL DisablePriorityBoost);    
```

```c
BOOL SetProcessAffinityMask (    HANDLE hProcess,    DWORD_PTR dwProcessAffinityMask);    DWORD_PTR SetThreadAffinityMask (    HANDLE hThread,    DWORD_PTR dwThreadAffinityMask);    
```
**Chapter 2. Process Management**

### Windows Thread Pool Calls

PTP_POOL CreateThreadpool (PVOID reserved);  
VOID CloseThreadpool (PTP_POOL ptpp);  
BOOL SetThreadpoolThreadMinimum (PTP_POOL ptpp, DWORD cthrdMic);  
VOID SetThreadpoolThreadMaximum (PTP_POOL ptpp, DWORD cthrdMost);  
VOID SubmitThreadpoolWork (PTP_WORK pwk);  

### Windows Process Timing Call

BOOL GetProcessTimes (HANDLE hProcess, LPFILETIME lpCreationTime, LPFILETIME lpExitTime, LPFILETIME lpKernelTime, LPFILETIME lpUserTime);  

### Process Communication

**Shared Memory**

**Example: System V Shared Memory**

#### ShmGet System Call

int shmget (key_t key, size_t size, int shmflg);  

- IPC_PRIVATE - private key value  
- IPC_CREAT - object with key can be created  
- IPC_EXCL - object with key must not exist  

void *shmat (int shmid, const void *shmaddr, int shmflg);  
int shmdt (const void *shmaddr);  
key_t ftok (const char *pathname, int proj_id);
Chapter 2. Process Management

Shared Memory Listing

> ipcs -m

<table>
<thead>
<tr>
<th>key</th>
<th>shmid</th>
<th>owner</th>
<th>perms</th>
<th>bytes</th>
<th>nattch</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000</td>
<td>12345</td>
<td>root</td>
<td>600</td>
<td>123456</td>
<td>2</td>
<td>dest</td>
</tr>
<tr>
<td>0x00000000</td>
<td>123456</td>
<td>root</td>
<td>600</td>
<td>234567</td>
<td>2</td>
<td>dest</td>
</tr>
<tr>
<td>0x00000000</td>
<td>1234567</td>
<td>nobody</td>
<td>777</td>
<td>345678</td>
<td>2</td>
<td>dest</td>
</tr>
</tbody>
</table>

Example: Windows Shared Memory

CreateFileMapping System Call

HANDLE CreateFileMapping (  
    HANDLE hFile,  
    LPSECURITY_ATTRIBUTES lpFileMappingAttributes,  
    DWORD flProtect,  
    DWORD dwMaximumSizeHigh,  
    DWORD dwMaximumSizeLow,  
    LPCTSTR lpName);

- PAGE_READONLY
- PAGE_READWRITE
- PAGE_WRITECOPY
- SEC_IMAGE

MapViewOfFile System Call

LPVOID MapViewOfFile (  
    HANDLE hFileMappingObject, DWORD dwDesiredAccess,  
    DWORD dwFileOffsetHigh, DWORD dwFileOffsetLow,  
    DWORD dwNumberOfBytesToMap);

LPVOID MapViewOfFileEx (  
    HANDLE hFileMappingObject, DWORD dwDesiredAccess,  
    DWORD dwFileOffsetHigh, DWORD dwFileOffsetLow,  
    DWORD dwNumberOfBytesToMap, LPVOID lpBaseAddress);

- FILE_MAP_READ
- FILE_MAP_WRITE
- FILE_MAP_COPY
- FILE_MAP_ALL_ACCESS
Chapter 2. Process Management

Message Passing

Example: Posix Signals

Standard Signals

<table>
<thead>
<tr>
<th>Name</th>
<th>Number</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGHUP</td>
<td>1</td>
<td>Controlling terminal closed</td>
</tr>
<tr>
<td>SIGINT</td>
<td>2</td>
<td>Request for interrupt sent from keyboard</td>
</tr>
<tr>
<td>SIGQUIT</td>
<td>3</td>
<td>Request for quit sent from keyboard</td>
</tr>
<tr>
<td>SIGILL</td>
<td>4</td>
<td>Illegal instruction</td>
</tr>
<tr>
<td>SIGTRAP</td>
<td>5</td>
<td>Breakpoint instruction</td>
</tr>
<tr>
<td>SIGABRT</td>
<td>6</td>
<td>Request for abort</td>
</tr>
<tr>
<td>SIGBUS</td>
<td>7</td>
<td>Illegal bus cycle</td>
</tr>
<tr>
<td>SIGFPE</td>
<td>8</td>
<td>Floating point exception</td>
</tr>
<tr>
<td>SIGKILL</td>
<td>9</td>
<td>Request for kill</td>
</tr>
<tr>
<td>SIGUSR1</td>
<td>10</td>
<td>User defined signal 1</td>
</tr>
<tr>
<td>SIGSEGV</td>
<td>11</td>
<td>Illegal memory access</td>
</tr>
<tr>
<td>SIGUSR2</td>
<td>12</td>
<td>User defined signal 2</td>
</tr>
<tr>
<td>SIGPIPE</td>
<td>13</td>
<td>Broken pipe</td>
</tr>
<tr>
<td>SIGALRM</td>
<td>14</td>
<td>Timer alarm</td>
</tr>
<tr>
<td>SIGTERM</td>
<td>15</td>
<td>Request for termination</td>
</tr>
<tr>
<td>SIGTERM</td>
<td>16</td>
<td>Illegal stack access</td>
</tr>
<tr>
<td>SIGCHLD</td>
<td>17</td>
<td>Child process status changed</td>
</tr>
<tr>
<td>SIGCONT</td>
<td>18</td>
<td>Request to continue when stopped</td>
</tr>
<tr>
<td>SIGSTOP</td>
<td>19</td>
<td>Request to stop</td>
</tr>
<tr>
<td>SIGTSTP</td>
<td>20</td>
<td>Request for stop sent from keyboard</td>
</tr>
<tr>
<td>SIGTIN</td>
<td>21</td>
<td>Input from terminal when on background</td>
</tr>
<tr>
<td>SIGTOU</td>
<td>22</td>
<td>Output to terminal when on background</td>
</tr>
</tbody>
</table>

Signal Handler Registration System Call

typedef void (*sighandler_t) (int);
sighandler_t signal (int signum, sighandler_t handler);

- SIG_DFL - use default signal handler
- SIG_IGN - ignore the signal

struct sigaction
{
    void (*sa_handler) (int);
void (*sa_sigaction) (int, siginfo_t *, void *);
sigset_t sa_mask;
int sa_flags;
}

struct siginfo_t
{
    int    si_signo; // Signal number
    int    si_errno; // Value of errno
    int    si_code; // Additional signal code
    pid_t   si_pid;  // Sending process PID
    uid_t   si_uid;  // Sending process UID
    int    si_status; // Exit value
    clock_t si_utime; // User time consumed
    clock_t si_stime; // System time consumed
    signal_t si_value; // Signal value
    int    si_int; // Integer value sent with signal
    void * si_ptr; // Pointer value sent with signal
    void * si_addr; // Associated memory address
    int    si_fd;  // Associated file descriptor
}

int sigaction (int signum, const struct sigaction *act, struct sigaction *oldact);

• sa_handler - signal handler with limited arguments
• sa_sigaction - signal handler with complete arguments
• sa_mask - what other signals to mask while in signal handler

• SA_RESETHAND - restore default signal handler after one signal
• SA_NODEFER - allow recursive invocation of this signal handler
• SA_ONSTACK - use alternate stack for this signal handler

Signal Masking System Call

int sigprocmask (int how, const sigset_t *set, sigset_t *oset);
int pthread_sigmask (int how, const sigset_t *set, sigset_t *oset);

• SIG_BLOCK - add blocking to signals that are not yet blocked
• SIG_UNBLOCK - remove blocking from signals that are blocked
• SIG_SETMASK - replace existing mask

Signal Send System Call

int kill (pid_t pid, int sig);
int pthread_kill (pthread_t thread, int sig);

union sigval
{
    int sival_int;
    void *sival_ptr;
}

int sigqueue (pid_t pid, int sig, const union sigval value);
Chapter 2. Process Management

Example: System V Message Passing

MsgSnd And MsgRcv System Calls

```c
int msgsnd (int que, message *msg, int len, int flags);
int msgrcv (int que, message *msg, int len, int type, int flags);
```

MsgGet System Call

```c
int msgget (key_t key, int msgflg);
```

Process Synchronization

Means For Synchronization

Active Waiting

Naive Active Wait For Critical Section

```c
while (bCriticalSectionBusy)
{
    // Active waiting cycle until the
    // bCriticalSectionBusy variable
    // becomes false
}
bCriticalSectionBusy = true;

// Code of critical section comes here
...

bCriticalSectionBusy = false;
```

Improved Active Wait For Critical Section

```c
while (true)
{
    // Indicate the intent to enter the critical section
    bIWantToEnter = true;
    // Enter the critical section if the other
    // process does not indicate the same intent
    if (!bHeWantsToEnter) break;
    // Back off to give the other process
    // a chance and continue the active
    // waiting cycle
    bIWantToEnter = false;
}

// Code of critical section comes here
...

bIWantToEnter = false;
```
**Dekker Algorithm**

// Indicate the intent to enter the critical section
bIWantToEnter = true;
while (bHeWantsToEnter)
{
  // If the other process indicates the same intent and
  // it is not our turn, back off to give the other
  // process a chance
  if (iWhoseTurn != MY_TURN)
  {
    bIWantToEnter = false;
    while (iWhoseTurn != MY_TURN) { }
    bIWantToEnter = true;
  }
}

// Code of critical section comes here ...

iWhoseTurn = HIS_TURN;
bIWantToEnter = false;

**Peterson Algorithm**

// Indicate the intent to enter the critical section
bIWantToEnter = true;
// Be polite and act as if it is not our
// turn to enter the critical section
iWhoseTurn = HIS_TURN;
// Wait until the other process either does not
// intend to enter the critical section or
// acts as if its our turn to enter
while (bHeWantsToEnter && (iWhoseTurn != MY_TURN)) { }

// Code of critical section comes here ...

bIWantToEnter = false;

**Active Wait For Critical Section Using Atomic Swap**

while (AtomicSwap (bCriticalSectionBusy, true))
{
  // Active waiting cycle until the
  // value of the bCriticalSectionBusy
  // variable has changed from false to true
}

// Code of critical section comes here ...

bCriticalSectionBusy = false;
Chapter 2. Process Management

Passive Waiting

*Naive Passive Wait For Critical Section*

```cpp
if (AtomicSwap (bCriticalSectionBusy, true))
{
    // The critical section is busy, put
    // the process into the waiting queue
    oWaitingProcesses.Put (GetCurrentProcess ());
    // Wait until somebody wakes the process
    Sleep ();
}

// Code of critical section comes here ...
...

// See if any process is waiting in the queue
oWaitingProcess = oWaitingProcesses.Get ();
if (oWaitingProcess)
{
    // A process was waiting, let it enter the critical section
    Wake (oWaitingProcess);
}
else
{
    // No process was waiting, mark the critical section as free
    bCriticalSectionBusy = false;
}
```

Memory Models

*Example: Invalid Register Optimization*

```cpp
// Assume a hot loop worth optimizing.
while (...)
{
    ...
    // Protect access to x if multithreaded.
    if (bThreaded) oLock.lock ();

    // Do a lot of work with x here.
    x = f (x);

    // Protect access to x if multithreaded.
    if (bThreaded) oLock.unlock ();
    ...
}

// Since x is used a lot it is kept in some register.
register = x;
while (...)
{
    ...
    if (bThreaded)
    {
        // External call may use x so it is written back.
        x = register;
        oLock.lock ();
        register = x;
    }
```
Chapter 2. Process Management

```c
// A lot of work done efficiently with x in register.
register = f (register);

if (bThreaded)
{
    // External call may use x so it is written back.
    x = register;
    oLock.unlock ();
    register = x;
}
...
}

Example adjusted from literature, see references.

**Example: Invalid Branch Optimization**

// If x is 0 then set y to 0.
// Otherwise leave y unchanged.
if (x == 0) y = 0;

// If x is 0 then set y to 0.
// Otherwise leave y unchanged.
y = (x == 0) ? 0 : y;

Example adjusted from literature, see references.

**Example: Memory Model On Intel 80x86 Processors**

**Example: Effects Of Intel 80x86 Memory Ordering Model**

A dd 0
B dd 0

Executed on one processor:

    mov [A], 1
    mov eax, [B]

Executed on another processor:

    mov [B], 1
    mov eax, [A]

It is possible for both processors to finish with EAX containing 0.

**Example: Memory Model In Java**

**Example: Causality Loops With Happens-Before Consistency**

```java
int A = 0;
int B = 0;

Executed in one thread:

```
Chapter 2. Process Management

if (A == 1) B = 1;

Executed in another thread:
if (B == 1) A = 1;

Example adjusted from literature, see references.

Example: Effects Of Java Memory Ordering Model

int A = 0;
int B = 0;

Executed in one thread:
A = 1;
X = B;

Executed in another thread:
B = 1;
Y = A;

It is possible for threads to finish with both X and Y containing 0.

What Is The Interface

Atomic Operations

Windows Atomic Operations

LONG InterlockedIncrement (PLONG Addend);
LONG InterlockedDecrement (PLONG Addend);
LONG InterlockedExchange (LPLONG lp1Target, LONG lValue);

PSLIST_ENTRY InterlockedPushEntrySList ( 
    PSLIST_HEADER ListHead,
    PSLIST_ENTRY ListEntry);
PSLIST_ENTRY InterlockedPopEntrySList ( 
    PSLIST_HEADER ListHead);

GCC Atomic Operations

type __sync_fetch_and_{add,sub,or,and,xor,nand} (type *ptr, type value, ...);
type __sync_{add,sub,or,and,xor,nand}_and_fetch (type *ptr, type value, ...);

bool __sync_bool_compare_and_swap (type *ptr, type oldval type newval, ...);
type __sync_val_compare_and_swap (type *ptr, type oldval type newval, ...);

type __sync_lock_test_and_set (type *ptr, type value, ...);
void __sync_lock_release (type *ptr, ...)

• implemented for common scalars and pointers
• typically supported by processor instructions
• unsupported operations compiled as function calls

**C++ Atomic Operations**

template<typename T> struct atomic
{
    public:

    bool is_lock_free () { ... }

    T operator= (T i) { store (i); return (i); }
    void store (T i, memory_order m = memory_order_seq_cst) { ... }
    T load (memory_order m = memory_order_seq_cst) { ... }

    T operator++ () { ... }
    T operator-- () { ... }
    T fetch_add (T i, memory_order m = memory_order_seq_cst) { ... }
    T fetch_sub (T i, memory_order m = memory_order_seq_cst) { ... }

    T exchange (T i, memory_order m = memory_order_seq_cst) { ... }
    bool compare_exchange_weak (T& e, T i, memory_order succ, memory_order fail) { ... }
    bool compare_exchange_strong (T& e, T i, memory_order succ, memory_order fail) { ... }

    ...
}

enum memory_order
{
    memory_order_relaxed, // No ordering constraints
    memory_order_consume, // Load will be consume operation (no reordering of dependent data accesses)
    memory_order_acquire, // Load will be acquire operation (no reordering of arbitrary data accesses)
    memory_order_release, // Store will be release operation
    memory_order_acq_rel, // Load-modify-store will be acquire and release operation
    memory_order_seq_cst // Any operation will be totally ordered acquire and release operation
};

**Barriers**

**GCC Barrier Constructs**

asm ("" : : "memory");

• empty assembler statement
• no output and input operands
• indicates memory accessed in undefined manner
• no output operands imply important side effects
• does not include processor barrier operation

// Full hardware memory barrier
__sync_synchronize (...);
Chapter 2. Process Management

Visual C++ Barrier Constructs

void _ReadBarrier (void);
void _WriteBarrier (void);
void _ReadWriteBarrier (void);

• compiler intrinsic functions
• do not impact variables compiler knows are local
• different behavior with different compiler versions
• does not include processor barrier operation

// Full hardware memory barrier
void MemoryBarrier (void);

Posix Barrier Interface

int pthread_barrier_init (  
    pthread_barrier_t *barrier,  
    const pthread_barrierattr_t *attr,  
    unsigned count);
int pthread_barrier_destroy (  
    pthread_barrier_t *barrier);
int pthread_barrier_wait (  
    pthread_barrier_t *barrier);

Locks

Posix Mutex Interface

int pthread_mutex_init (pthread_mutex_t *mutex,  
    const pthread_mutexattr_t *mutexattr);  
int pthread_mutex_destroy (pthread_mutex_t *mutex);
int pthread_mutex_lock (pthread_mutex_t *mutex);  
int pthread_mutex_trylock (pthread_mutex_t *mutex);  
int pthread_mutex_timedlock (pthread_mutex_t *restrict mutex,  
    const struct timespec *abs_timeout);
int pthread_mutex_unlock (pthread_mutex_t *mutex);

• error checking mutex
• recursive mutex
• fast mutex

Posix Spin Lock Interface

int pthread_spin_init (pthread_spinlock_t *lock, int pshared);  
int pthread_spin_destroy (pthread_spinlock_t *lock);
int pthread_spin_lock (pthread_spinlock_t *lock);  
int pthread_spin_trylock (pthread_spinlock_t *lock);
int pthread_spin_unlock (pthread_spinlock_t *lock);

- PTHREAD_PROCESS_PRIVATE - only for threads from the same process
- PTHREAD_PROCESS_SHARED - for any thread that can access the memory with the lock

Linux Futex Interface

int sys_futex (void *futex, int op, int val, const struct timespec *timeout)

- FUTEX_WAIT - verify val and sleep if unchanged
- FUTEX_WAKE - wake at most val processes
- FUTEX_CMP_REQUEUE - wake at most val processes and requeue rest

Trivial Mutex Using Futex

class mutex
{
private:
    // Mutex state variable, zero means free.
    int val = 0;

public:
    void lock ()
    {
        int old;

        // Atomically increment the state and get the old value, which should be zero if mutex was free.
        while ((old = atomic_inc (val)) != 0)
        {
            // The old value was not zero, meaning mutex was not free.
            // Wait unless the value has changed since the increment.
            futex_wait (&val, old + 1);
        }
    }

    void unlock ()
    {
        val = 0;
        // Wake a waiting caller if any.
        futex_wake (&val, 1);
    }
}

- unlock always calls kernel which is slow
- contention causes cache ping pong
- contention causes counter overflow

* Source: Ulrich Drepper
Chapter 2. Process Management

Windows Critical Section Interface

```c
void InitializeCriticalSection (LPCRITICAL_SECTION lpCriticalSection);
BOOL InitializeCriticalSectionAndSpinCount (
    LPCRITICAL_SECTION lpCriticalSection,
    DWORD dwSpinCount);
void EnterCriticalSection (LPCRITICAL_SECTION lpCriticalSection);
BOOL TryEnterCriticalSection (LPCRITICAL_SECTION lpCriticalSection);
void LeaveCriticalSection (LPCRITICAL_SECTION lpCriticalSection);
```

- `dwSpinCount` - spin count to avoid context switch on multiple processors

Windows Mutex Interface

```c
HANDLE CreateMutex (LPSECURITY_ATTRIBUTES lpsa,
    BOOL fInitialOwner,
    LPTSTR lpszMutexName);
HANDLE OpenMutex (DWORD dwDesiredAccess,
    BOOL bInheritHandle,
    LPCTSTR lpName);
DWORD WaitForSingleObject (
    HANDLE hHandle,
    DWORD dwMilliseconds);
BOOL ReleaseMutex (HANDLE hMutex);
```

Java Lock Support Interface

```java
class java.util.concurrent.locks.LockSupport {
    static void park<8203>();
    static void parkNanos<8203>(long nanos);
    static void parkUntil<8203>(long deadline);
    static void unpark (Thread thread);
    ...
}
```

- thread may possess parking permit
- parking blocks thread until permit becomes available
- unparking provides permit even when thread not parked

Read Write Locks

Posix Read Write Lock Interface

```c
int pthread_rwlock_init (pthread_rwlock_t *rwlock,
    const pthread_rwlockattr_t *attr);
int pthread_rwlock_destroy (pthread_rwlock_t *rwlock);
```
Chapter 2. Process Management

int pthread_rwlock_rdlock (pthread_rwlock_t *rwlock);
int pthread_rwlock_wrlock (pthread_rwlock_t *rwlock);
int pthread_rwlock_tryrdlock ( pthread_rwlock_t *rwlock);
int pthread_rwlock_trywrlock ( pthread_rwlock_t *rwlock);
int pthread_rwlock_unlock ( pthread_rwlock_t *rwlock);

Windows Slim Reader Writer Interface

VOID InitializeSRWLock (PSRWLOCK SRWLock);
VOID AcquireSRWLockShared (PSRWLOCK SRWLock);
VOID AcquireSRWLockExclusive (PSRWLOCK SRWLock);
VOID ReleaseSRWLockShared (PSRWLOCK SRWLock);
VOID ReleaseSRWLockExclusive (PSRWLOCK SRWLock);

Seq Lock

Linux Seq Lock Interface

seqlock_init (seqlock_t *sl);
void write_seqlock (seqlock_t *sl);
void write_sequnlock (seqlock_t *sl);
int write_tryseqlock (seqlock_t *sl);
read_seqbegin (const seqlock_t *sl);
int read_seqretry (const seqlock_t *sl, unsigned start);

• version incremented once before write and once after write
• readers require same version before read and after read
• even version means consistent, odd version means retry

Seq Lock Usage Example

do
{  start = read_seqbegin (&sl);
  ...  
} while (read_seqretry (&sl, start);

Read Copy Update

Linux Read Copy Update Interface

void rcu_read_lock ();
void rcu_read_unlock ();
Chapter 2. Process Management

typeof(ptr) rcu_assign_pointer (ptr, val);
typeof(ptr) rcu_dereference (ptr);
void synchronize_rcu ();

- readers access last consistent version
- new version hidden while inconsistent
- writer atomically installs new version
- writer releases old version when no old version reader exists

Semaphores

Unix Semaphore Interface

int semget (key_t key, int nsems, int semflg);

- IPC_PRIVATE - private key value
- IPC_CREAT - object with key can be created
- IPC_EXCL - object with key must not exist

int semop (int semid, struct sembuf *sops, unsigned nsops);
int semtimedop (int semid, struct sembuf *sops, unsigned nsops, struct timespec *timeout);

- positive - add value to semaphore
- zero - wait for zero value of semaphore
- negative - subtract value from semaphore or wait

key_t ftok (const char *pathname, int proj_id);

Posix Semaphore Interface

int sem_init (sem_t *sem, int pshared, unsigned int value);
int sem_destroy (sem_t *sem);
sem_t *sem_open (const char *name, int oflag, mode_t mode, unsigned int value);
int sem_unlink (const char *name);

int sem_wait (sem_t *sem);
int sem_trywait (sem_t *sem);
int sem_timedwait (sem_t *restrict sem, const struct timespec *abs_timeout);

int sem_post (sem_t *sem);
int sem_getvalue (sem_t *sem, int *sval);
Chapter 2. Process Management

- named - semaphore for synchronization between processes
- unnamed - semaphore for synchronization within a process

Windows Semaphore Interface

HANDLE CreateSemaphore (LPSECURITY_ATTRIBUTE lpsa,
LONG cSemInitial,
LONG cSemMax,
LPTSTR lpszSemName);

HANDLE OpenSemaphore (DWORD dwDesiredAccess,
BOOL bInheritHandle,
LPCSTR lpName);

DWORD WaitForSingleObject (HANDLE hHandle,
DWORD dwMilliseconds);

BOOL ReleaseSemaphore (HANDLE hSemaphore,
LONG cRelease,
LPLONG lplPrevious);

Condition Variables

Posix Condition Variable Interface

int pthread_cond_init (pthread_cond_t *cond, pthread_condattr_t *cond_attr);
int pthread_cond_destroy (pthread_cond_t *cond);

int pthread_cond_signal (pthread_cond_t *cond);
int pthread_cond_broadcast (pthread_cond_t *cond);
int pthread_cond_wait (pthread_cond_t *cond, pthread_mutex_t *mutex);
int pthread_cond_timedwait (pthread_cond_t *cond,
                          pthread_mutex_t *mutex,
                          const struct timespec *abstime);

Windows Condition Variable Interface

VOID InitializeConditionVariable (PCONDITION_VARIABLE ConditionVariable);

BOOL SleepConditionVariableCS (PCONDITION_VARIABLE ConditionVariable,
PCRITICAL_SECTION CriticalSection,
DWORD dwMilliseconds);

BOOL SleepConditionVariableSRW (PCONDITION_VARIABLE ConditionVariable,
PSRWLOCK SRWLock,
DWORD dwMilliseconds,
ULONG Flags);

VOID WakeConditionVariable (PCONDITION_VARIABLE ConditionVariable);

VOID WakeAllConditionVariable (PCONDITION_VARIABLE ConditionVariable);
Chapter 2. Process Management

Events

Windows Event Interface

HANDLE CreateEvent (LPSECURITY_ATTRIBUTES lpsa,
    BOOL fManualReset,
    BOOL fInitialState,
    LPTSTR lpszEventName);

HANDLE OpenEvent (DWORD dwDesiredAccess,
    BOOL bInheritHandle,
    LPTSTR lpName);

- manual reset - remains signalled after set until reset
- automatic reset - returns to unsignalled after thread wake up

BOOL SetEvent (HANDLE hEvent);
BOOL ResetEvent (HANDLE hEvent);
BOOL PulseEvent (HANDLE hEvent);

Monitors

Monitors

class MyClass {
    int a;
    synchronized Foo () {
        a = 1;
    }
    synchronized Bar () {
        a = 2;
    }
}

MyClass b = new MyClass ();
synchronized (b) {
    ...
}

Guards

Guards

task body Foo is
    i, j : integer;
    begin
    ...
    select
    when j > 0 =>
        accept Xyzzy (n : integer) do
            i := n;
        end Xyzzy;
    or
    ...
    end select;
... 
end Foo;

task body Bar is 
begin 
Xyzy (l); 
end Bar;

• execute rendez vous if possible 
• execute conditional branch if possible 
• throw exception if neither of the two is possible
Chapter 3. Memory Management

Management Among Processes

Separating Multiple Processes

Software Implementation

Example: Solaris

Pager Operations

- advise - access optimization hints
- checkprot - check whether access is allowed
- fault - handle page fault
- lockop - lock or unlock a page
- swapout - swap out maximum number of pages
- sync - write out dirty pages

What Is The Interface

MMap And MUnmap System Calls

```c
void *mmap (  
    void *start,  
    size_t length,  
    int prot,  
    int flags,  
    int fd,  
    off_t offset);

int munmap (  
    void *start,  
    size_t length);
```

- MAP_FIXED - require supplied address
- MAP_SHARED - share mapping with other processes
- MAP_PRIVATE - create a copy on write mapping
- MAP_ANONYMOUS - create mapping backed in swap

- MAP_GROWSDOWN - block grows down rather than up
- MAP_POPULATE - fetch pages into memory
- MAP_HUGETLB - allocate large pages
- MAP_LOCKED - lock pages in memory
Chapter 3. Memory Management

**MAdvise System Call**

```c
int madvise (  
    void *addr,  
    size_t length,  
    int advice);
```

- MADV_NORMAL - default treatment
- MADV_RANDOM - accessed in random order (read ahead less useful)
- MADV_SEQUENTIAL - accessed in sequential order (read ahead more useful, good victim after access)
- MADV_WILLNEED - access in near future likely (read ahead useful now)
- MADV_DONTNEED - access in near future unlikely (good victim now)
- MADV_HUGEPAGE - merge pages into huge page where possible
- MADV_MERGEABLE - merge pages with identical content

**MBind System Call**

```c
int set_mempolicy (  
    int mode,  
    unsigned long *nodemask, unsigned long maxnode);
int get_mempolicy (  
    int *mode,  
    unsigned long *nodemask, unsigned long maxnode,  
    unsigned long addr,  
    unsigned long flags);

int mbind (  
    void *addr, unsigned long len,  
    int mode,  
    unsigned long *nodemask, unsigned long maxnode,  
    unsigned flags);
```

- MPOL_DEFAULT - allocate on node that requests memory
- MPOL_BIND - allocate from listed nodes, exhaust node before using next
- MPOL_PREFERRED - allocate as close to first listed node as possible
- MPOL_INTERLEAVE - allocate from listed nodes, cycle across nodes
- MPOL_MF_MOVE - move already allocated pages to satisfy policy
- MPOL_MF_STRICT - fail if already allocated pages do not satisfy policy

**VirtualAlloc System Call**

```c
LPVOID VirtualAlloc (  
    LPVOID lpAddress,  
    SIZE_T dwSize,  
    DWORD flAllocationType,  
    DWORD flProtect);
```
Chapter 3. Memory Management

- MEM_RESET - drop data currently in block
- MEM_COMMIT - reserve address range and storage for block
- MEM_RESERVE - reserve address range but not storage for block
- MEM_TOP_DOWN - allocate with as high address as possible
- MEM_WRITE_WATCH - keep list of pages that were modified

GetWriteWatch System Call

UINT GetWriteWatch (DWORD dwFlags,
                    PVOID lpBaseAddress,
                    SIZE_T dwRegionSize,
                    PVOID *lpAddresses,
                    PULONG_PTR lpdwCount,
                    PULONG lpdwGranularity);

- WRITE_WATCH_FLAG_RESET - stop watching

Allocation Within A Process

Process Memory Layout

Example: Virtual Address Space Of A Linux Process

Process Address Space Layout

> cat /proc/self/maps
00111000-00234000 r-xp 00000000 03:01 3653725 /lib/libc-2.3.5.so
00234000-00236000 r-xp 00123000 03:01 3653725 /lib/libc-2.3.5.so
00236000-00238000 rwxp 00125000 03:01 3653725 /lib/libc-2.3.5.so
00238000-0023a000 rwxp 00238000 00:00 0
007b5000-007cf000 r-xp 00000000 03:01 3653658 /lib/ld-2.3.5.so
007cf000-007d0000 r-xp 00019000 03:01 3653658 /lib/ld-2.3.5.so
007d0000-007d1000 rwxp 0001a000 03:01 3653658 /lib/ld-2.3.5.so
008ed000-008ee000 r-xp 008ed000 00:00 0 [vdso]
08048000-0804d000 r-xp 00000000 03:01 3473470 /bin/cat
0804d000-0804e000 rw-p 00004000 03:01 3473470 /bin/cat
09ab8000-09ad9000 rw-p 09ab8000 00:00 0 [heap]
b7d88000-b7f88000 r-p 00000000 03:01 6750409 /usr/lib/locale/locale-archive
b7f88000-b7f89000 rw-p b7f88000 00:00 0
b7f96000-b7f97000 rw-p b7f96000 00:00 0
bfd81000-bfd97000 rw-p bfd81000 00:00 0 [stack]
Chapter 3. Memory Management

Stack

Example: Stack On Intel 80x86 Processors

Stack Allocation And Access

```c
void SomeProcedure (int anArgument) {
    int aVariable;
    aVariable = anArgument;
}
```

```assembly
SomeProcedure:
push ebp ;save original value of EBP on stack
mov ebp,esp ;store top of stack address in EBP
sub esp,4 ;allocate space for aVariable on stack
mov eax,[ebp+8] ;fetch anArgument into EAX, which is
                ;8 bytes below the stored top of stack
mov [ebp-4],eax ;store EAX into aVariable, which is
                ;4 bytes above the stored top of stack
mov esp,ebp ;free space allocated for aVariable
pop ebp ;restore original value of EBP
ret ;return to the caller
```

Heap

Heap Allocators

Example: dlmalloc Heap Allocator

Allocated Chunk Structure

```c
chunk +------------------------------------------------+---+
| Size of previous chunk (if P = 0) | P |
+-----------------------------------+---+
| Size of this chunk | 1 |
+-------------------+---+
| User data (size - sizeof (size_t) bytes) : |
+-------------------+---+
chunk +------------------------------------------------+---+
| Size of this chunk again | 1 |
+-------------------+---+
| Size of next chunk | U |
+-------------------+---+
```

// Adjusted from dlmalloc source code comments.
Chapter 3. Memory Management

Free Chunk Structure

chunk +-----------------------------------------------+  
| Size of previous chunk (if P = 0)  | +-----------------------------------------------+  
| ---+---+ | P | +-----------------------------------------------+  
| Size of this chunk | 0 | +---+  
| ---+---+  
| Pointer to next chunk in bin list | ---+---+  
| ---+---+  
| Pointer to previous chunk in bin list | ---+---+  
| ---+---+  
| Pointer to left child when in bin trie | ---+---+  
| ---+---+  
| Pointer to right child when in bin trie | ---+---+  
| ---+---+  
| Pointer to parent when in bin trie | ---+---+  
| ---+---+  
| Bin index when in bin trie | ---+---+  
| ---+---+  
| | ---+---+  
| : Free :  
| ---+---+  
chunk +-----------------------------------------------+  
| Size of this chunk again | ---+---+  
| ---+---+  
| Size of next chunk | U | +---+  
| ---+---+  
data +-----------------------------------------------+  

// Adjusted from dlmalloc source code comments.

Example: Posix Heap Allocator Interface

Posix Heap Allocator Interface

void *malloc (size_t size);  
void *calloc (size_t nmemb, size_t size);  
void *realloc (void *ptr, size_t size);  
void free (void *ptr);  

// Aligned allocation for power-of-two alignment  
int posix_memalign (void **memptr, size_t alignment, size_t size);  

// Aligned allocation for power-of-two alignment (deprecated)  
void *memalign (size_t boundary, size_t size);  
// Aligned allocation for page-size alignment (deprecated)  
void *valloc (size_t size);  

Linux Heap Allocator Interface

int mallopt (int param, int value);

• M_CHECK_ACTION - control error reporting
• M_PERTURB - pattern fill on alloc and free

struct mallinfo mallinfo (void);
Chapter 3. Memory Management

```
struct mallinfo {
    int arena; // Allocated space except mapped regions
    int ordblks; // Number of free blocks except fast bins
    int smbblks; // Number of free fast bins
    int hblks; // Number of mapped regions
    int hblkhd; // Space in mapped regions
    int usmblks; // Maximum space allocated so far (only when single thread)
    int fsmblks; // Space in free fast bins
    int uordblks; // Total allocated space
    int fordblks; // Total free space
    int keepcost; // Space at heap end that can be released
};
```

Example: Linux Kernel Slab Allocator

**Slab Cache System Calls**

```
// Create a slab cache
kmem_cache_t * kmem_cache_create (const char *name,
    size_t size,
    size_t align,
    unsigned long flags,
    void (*ctor) (void *, kmem_cache_t *, unsigned long),
    void (*dtor) (void *, kmem_cache_t *, unsigned long));
```

- **name** - slab cache name for debugging
- **size** - object size for this slab cache
- **align** - optional object alignment
- **ctor** - optional object constructor
- **dtor** - optional object destructor

```
// Allocate and free slabs of the cache
void *kmem_cache_alloc (kmem_cache_t *cachep, int flags);
void kmem_cache_free (kmem_cache_t *cachep, void *objp);
```

- **SLAB_DEBUG_FREE** - check correctness of free
- **SLAB_RED_ZONE** - create red zone after objects
- **SLAB_POISON** - fill free objects with poison data
- **SLAB_STORE_USER** - remember return address to last object user
- **SLAB_CACHE_DMA** - allocate memory for use with DMA
- **SLAB_HWCACHE_ALIGN** - align objects on cache lines
Slab Allocator Usage Statistics

```bash
> cat /proc/slabinfo
slabinfo - version: 2.1
# name  <active_objs> <num_objs> <objsize> <objperslab> <pagesperslab> ...
  ext4_inode_cache   49392   49392    1016       16          4
  ext4_free_data     128     128      64         64          1
  ext4_xattr         92      92       88         46          1
  blkdev_requests    273     273     376        21          2
  blkdev_queue       30      30      2080       15          8
  vm_area_struct     3520    3520    184        22          1
  task_struct        160     160     1952       16          8
  inode_cache        11899   11928    568        14          2
  dentry            401373  401373  192        21          1
  ...
```
Chapter 3. Memory Management
Chapter 4. Device Management

Device Drivers

Asynchronous Requests

Example: Linux Tasklets

Soft IRQ Example

// Registers softirq handler
extern void open_softirq (int nr, void (*action)(struct softirq_action*), void *data);

void open_softirq (...)
{
    softirq_vec [nr].data = data;
    softirq_vec [nr].action = action;
}

// Schedules softirq handler
inline fastcall void raise_softirq_irqoff (unsigned int nr)
{
    or_softirq_pending (1UL &lt;&lt; (nr));
    if (!in_interrupt ()) wakeup_softirqd ();
}

• Multiprocessor system can execute multiple soft irqs concurrently.
• One soft irq can execute concurrently on multiple processors.

Tasklet Interface Example

#define DECLARE_TASKLET(name, func, data) \
struct tasklet_struct name = { NULL, 0, ATOMIC_INIT(0), func, data }

// Schedule tasklet for execution on current processor
void tasklet_schedule (struct tasklet_struct *t);

void tasklet_disable (struct tasklet_struct *t);
void tasklet_enable (struct tasklet_struct *t);

• Scheduling guarantees the tasklet will be called at least once.
• Multiprocessor system can execute multiple tasklets concurrently.
• One tasklet never executes concurrently on multiple processors.
Tasklet Handling Example

```c
static void tasklet_action (struct softirq_action *a)
{
    struct tasklet_struct *list;

    // Get the entire tasklet queue
    local_irq_disable ();
    list = __get_cpu_var (tasklet_vec).head;
    __get_cpu_var (tasklet_vec).head = NULL;
    __get_cpu_var (tasklet_vec).tail = &__get_cpu_var (tasklet_vec).head;
    local_irq_enable ();

    // Go through the queue tasklet by tasklet
    while (list) {
        struct tasklet_struct *t = list;
        list = list->next;

        // Necessary synchronization with other processors
        if (tasklet_trylock (t)) {
            if (!atomic_read (&t->count)) {
                if (!test_and_clear_bit (TASKLET_STATE_SCHED, &t->state))
                    BUG ();
                t->func (t->data);
                tasklet_unlock (t);
                continue;
            }
            tasklet_unlock (t);
        }

        // Put the executed tasklets back into queue
        local_irq_disable();
        t->next = NULL;
        *__get_cpu_var(tasklet_vec).tail = t;
        __get_cpu_var(tasklet_vec).tail = &(t->next);
        __raise_softirq_irqoff(TASKLET_SOFTIRQ);
        local_irq_enable();
    }
}
```

Work Queue Example

```c
#define DECLARE_WORK(name, func, data) \
    struct work_struct name = { data, NULL, func)

// Create a work queue with a kernel thread to serve it
struct workqueue_struct *create_workqueue (const char *name);

// Request executing work by a given work queue
int queue_work (\n    struct workqueue_struct *queue, \
    struct work_struct *work);

int queue_delayed_work (\n    struct workqueue_struct *queue, \
    struct work_struct *work, \
    unsigned long delay);

// Request executing work by the default work queue
int schedule_work (\n    struct work_struct *work);
int schedule_delayed_work (\n    struct work_struct *work, \
    unsigned long delay);
```
void flush_workqueue (struct workqueue_struct *queue);

Example: Windows Deferred Procedure Calls

DPC Example

// Registers DPC for a device
VOID IoInitializeDpcRequest (  
    IN PDEVICE_OBJECT DeviceObject,
    IN PIO_DPC_ROUTINE DpcRoutine
);

// Schedules DPC for a device
VOID IoRequestDpc (  
    IN PDEVICE_OBJECT DeviceObject,
    IN PIRP Irp,
    IN PVOID Context
);

// DPC
VOID DpcForIsr (  
    IN PKDPC Dpc,
    IN struct _DEVICE_OBJECT *DeviceObject,
    IN struct _IRP *Irp,
    IN PVOID Context
);

• DeviceObject - device instance
• DpcRoutine - deferred procedure call routine
• Irp - structure describing the request being processed
• Context - driver context to be passed to the routine

Synchronous Requests

Example: Linux Driver Model

Busses In Linux Driver Model

> ls -R /sys/bus
/sys/bus:
    pci  pci_express  pcmcia  scsi  usb
/sys/bus/pci:
    devices  drivers
/sys/bus/pci/devices:
    0000:00:00.0  0000:00:01.0  0000:00:01b.0  0000:00:01c.4  0000:00:01e.0  0000:01:00.0
/sys/bus/pci/drivers:
    agpgart-intel  ata_piix  ehci_hcd  ohci_hcd  uhci_hcd  ahci  e1000  HDA Intel ...
**Chapter 4. Device Management**

**Devices In Linux Driver Model**

```bash
> ls -R /sys/devices
/sys/devices:
pci0000:00
pci0000:00:00.0
pci0000:00:00.0/0000:00:19.0:
class config device driver irq net power vendor
pci0000:00:00.0/0000:00:19.0/net:
eth0
pci0000:00:00.0/0000:00:19.0/net/eth0:
address broadcast carrier device features flags mtu power statistics ...
```

**Device Insertion Notification In Linux Driver Model**

```bash
> udevmonitor --env
UEVENT[12345.67890] add /devices/pci0000:00/0000:00:1a.7/usb1/1-3/1-3:1.0 (usb)
ACTION=add
DEVPATH=/devices/pci0000:00/0000:00:1a.7/usb1/1-3/1-3:1.0
SUBSYSTEM=usb
DEVTYPE=usb_interface
DEVICE=/proc/bus/usb/001/006
PRODUCT=457/151/100
INTERFACE=8/6/80
MODALIAS=usb:v0457p0151d0100dc00dsc00dp00ic08isc06ip50
```

**Devices**

**Busses**

**Example: SCSI**

**SCSI Inquiry Command**

```
+----+-+----+-+----+-+----+-+----+-+----+-+----+-+----+-+----+-+----+-+----+-+----+-+
| Bit| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Byte |
|-----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
| 0 | Operation Code: Inquiry (12h) |
|-----+-----------------------------------------------------------------------|
| 1 | Logical Unit Number | Reserved | EVPD |
|-----+-----------------------------------------------------------------------|
| 2 | Page Code |
|-----+-----------------------------------------------------------------------|
| 3 | Reserved |
|-----+-----------------------------------------------------------------------|
| 4 | Allocation Length: Inquiry Reply Length (96) |
|-----+-----------------------------------------------------------------------|
| 5 | Control |
+----+-+----+-+----+-+----+-+----+-+----+-+----+-+----+-+----+-+----+-+----+-+----+-+
```
### SCSI Inquiry Response

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Bit| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 |
| 0  | Peripheral Qualifier | Peripheral Device Type |
| 1  | RMB | Device-Type Modifier |
| 2  | ISO Version | ECMA Version | ANSI Version |
| 3  | AENC | TrmIOP | Reserved | Response Data Format |
| 4  | Additional Length (n-4) |
| 5  | Reserved |
| 6  | Reserved |
| 7  | RelAdr | WBus32 | WBus16 | Sync | Linked | Reserved | CmdQue | SftRe |
| 8  | (MSB) | Vendor Identification | ___ |
| 15 | (LSB) |
| 16 | (MSB) | Product Identification | ___ |
| 31 | (LSB) |
| 32 | (MSB) | Product Revision Level | ___ |
| 35 | (LSB) |
| 36 | Reserved |
| 55 | |
| 95 | |
| 96 | Additional Vendor Specific | ___ |
| n  | |

### SCSI Device Listing

```
> cat /proc/scsi/scsi
Attached devices:
Host: scsi0 Channel: 00 Id: 00 Lun: 00
  Type: Direct-Access ANSI SCSI revision: 03
Host: scsi1 Channel: 00 Id: 05 Lun: 00
  Vendor: NEC Model: CD-ROM DRIVE:466 Rev: 1.06
  Type: CD-ROM ANSI SCSI revision: 02
Host: scsi2 Channel: 00 Id: 00 Lun: 00
  Vendor: PLEXTOR Model: DVDR PX-708A Rev: 1.02
  Type: CD-ROM ANSI SCSI revision: 02
```
Chapter 4. Device Management

Example: PCI

PCI Device Listing

```bash
> lspci -t
-0000:00-
+-01.0-[0000:01]-----00.0
+-02.0-[0000:02-03]-----1f.0-[0000:03]-----00.0
|+-0e.0-[0000:04]+-0b.0
| |+-0c.0
| |\-0d.0
+-1f.0
+-1f.1
+-1f.2
+-1f.3
+-1f.4
\-1f.5
```

PCI Brief Device Information

```bash
> lspci
00:00.0 Host bridge: Intel Corp. 82860 860 (Wombat) Chipset Host Bridge (MCH) (rev 04)
00:01.0 PCI bridge: Intel Corp. 82850 850 (Tehama) Chipset AGP Bridge (rev 04)
00:02.0 PCI bridge: Intel Corp. 82860 860 (Wombat) Chipset AGP Bridge (rev 04)
00:1e.0 PCI bridge: Intel Corp. 82801 PCI Bridge (rev 04)
00:1f.0 ISA bridge: Intel Corp. 82801BA ISA Bridge (LPC) (rev 04)
00:1f.1 IDE interface: Intel Corp. 82801BA IDE U100 (rev 04)
00:1f.2 USB Controller: Intel Corp. 82801BA/BAM USB (Hub #1) (rev 04)
00:1f.3 SMBus: Intel Corp. 82801BA/BAM SMBus (rev 04)
00:1f.4 USB Controller: Intel Corp. 82801BA/BAM USB (Hub #2) (rev 04)
00:1f.5 Multimedia audio controller: Intel Corp. 82806AA PCI64 Hub PCI Bridge (rev 03)
01:00.0 VGA compatible controller: ATI Technologies Inc Radeon RV100 QY [Radeon 7000/VE]
02:1f.0 PCI bridge: Intel Corp. 82806AA PCI64 Hub PCI Bridge (rev 03)
03:00.0 PIC: Intel Corp. 82806AA PCI64 Hub Advanced Programmable Interrupt Controller (rev 03)
04:0b.0 Ethernet controller: 3Com Corporation 3c905C-TX/TX-M [Tornado] (rev 78)
04:0c.0 FireWire (IEEE 1394): Texas Instruments TSB12LV26 IEEE-1394 Controller (Link)
04:0d.0 Ethernet controller: Intel Corp. 82544EI Gigabit Ethernet Controller (Copper) (rev 04)
```

PCI Detailed Device Information

```bash
> lspci -vvs 04:0b.0
04:0b.0 Ethernet controller: 3Com Corporation 3c905C-TX/TX-M [Tornado] (rev 78)
    Subsystem: Dell: Unknown device 00d8
    Control: I/O+ Mem+ BusMaster+ SpecCycle= MemWINv+ VGASnoop- ParErr- Stepping- SMM
    Latency: 64 (2500ns min, 2500ns max), Cache Line Size 10
    Interrupt: pin A routed to IRQ 23
    Region 0: I/O ports at dc80 [size=128]
    Region 1: Memory at ff3ffe00 (32-bit, non-prefetchable) [size=128]
    Expansion ROM at ff400000 [disabled] [size=128K]
    Capabilities: [dc] Power Management version 2
      Flags: PMEClk- DSI- D1- D2+ AuxCurrent=0mA PME(D0+,D1+,D2+,D3hot+,D3cold+)
      Status: D0 PME-Enable- DSe=0 DScale=2 PME-
```
Example: USB

USB Device Listing

```bash
> lsusb -t
Bus#  1
  `-Dev#  1 Vendor 0x0000 Product 0x0000
  `-Dev#  2 Vendor 0x046d Product 0xc01b
```

USB Brief Device Information

```bash
> lsusb
Bus 001 Device 002: ID 046d:c01b Logitech, Inc. MX310 Optical Mouse
Bus 001 Device 001: ID 0000:0000
```

USB Detailed Device Information

```bash
> lsusb -vv -s 1:2
Bus 001 Device 002: ID 046d:c01b Logitech, Inc. MX310 Optical Mouse
Device Descriptor:
  bLength 18
  bDescriptorType 1
  bcdUSB 2.00
  bDeviceClass 0 (Defined at Interface level)
  bDeviceSubClass 0
  bDeviceProtocol 0
  bMaxPacketSize0 8
  idVendor 0x046d Logitech, Inc.
  idProduct 0xc01b MX310 Optical Mouse
  bcdDevice 18.00
  iManufacturer 1 Logitech
  iProduct 2 USB-PS/2 Optical Mouse
  iSerial 0
  bNumConfigurations 1
Configuration Descriptor:
  bLength 9
  bDescriptorType 2
  wTotalLength 34
  bNumInterfaces 1
  bConfigurationValue 1
  iConfiguration 0
  bmAttributes 0xa0
    Remote Wakeup
  MaxPower 98mA
  Interface Descriptor:
    bLength 9
    bDescriptorType 4
    bInterfaceNumber 0
    bAlternateSetting 0
    bNumEndpoints 1
    bInterfaceClass 3 Human Interface Devices
    bInterfaceSubClass 1 Boot Interface Subclass
    bInterfaceProtocol 2 Mouse
    bInterface 0
  Endpoint Descriptor:
    bLength 7
    bDescriptorType 5
    bEndpointAddress 0x81 EP 1 IN
    bmAttributes 3
      Transfer Type Interrupt
```

61
Chapter 4. Device Management

<table>
<thead>
<tr>
<th>Synch Type</th>
<th>none</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage Type</td>
<td>Data</td>
</tr>
<tr>
<td>wMaxPacketSize</td>
<td>0x0005 bytes 5 once</td>
</tr>
<tr>
<td>bInterval</td>
<td>10</td>
</tr>
</tbody>
</table>

Disk Storage Devices

Partitioning

Example: IBM Volume Partitioning

IBM Partition Table

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>000h</td>
<td>446</td>
<td>Boot loader code</td>
</tr>
<tr>
<td>1BEh</td>
<td>16</td>
<td>Partition 1</td>
</tr>
<tr>
<td>1CEh</td>
<td>16</td>
<td>Partition 2</td>
</tr>
<tr>
<td>1DEh</td>
<td>16</td>
<td>Partition 3</td>
</tr>
<tr>
<td>1EEh</td>
<td>16</td>
<td>Partition 4</td>
</tr>
<tr>
<td>1FEh</td>
<td>2</td>
<td>Magic (0AA55h)</td>
</tr>
</tbody>
</table>

Example: GPT Volume Partitioning

GPT Partition Table

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>000h</td>
<td>8</td>
<td>Magic (&quot;EFI PART&quot;)</td>
</tr>
<tr>
<td>008h</td>
<td>4</td>
<td>Version</td>
</tr>
<tr>
<td>00Ch</td>
<td>4</td>
<td>Header size (typically 05Ch)</td>
</tr>
<tr>
<td>010h</td>
<td>4</td>
<td>Header CRC32</td>
</tr>
<tr>
<td>014h</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>018h</td>
<td>8</td>
<td>This header LBA address</td>
</tr>
<tr>
<td>020h</td>
<td>8</td>
<td>Backup header LBA address</td>
</tr>
<tr>
<td>028h</td>
<td>8</td>
<td>First data block LBA address</td>
</tr>
<tr>
<td>030h</td>
<td>8</td>
<td>Last data block LBA address</td>
</tr>
<tr>
<td>038h</td>
<td>16</td>
<td>Disk UUID</td>
</tr>
<tr>
<td>048h</td>
<td>8</td>
<td>Partition array LBA address (typically 2)</td>
</tr>
<tr>
<td>050h</td>
<td>4</td>
<td>Partition array entry count</td>
</tr>
<tr>
<td>054h</td>
<td>4</td>
<td>Partition array entry size (typically 128)</td>
</tr>
<tr>
<td>058h</td>
<td>4</td>
<td>Partition array CRC32</td>
</tr>
</tbody>
</table>
Chapter 4. Device Management

---

000h 16 Partition type UUID
010h 16 Partition UUID
020h 8 First data block LBA address
028h 8 Last data block LBA address
030h 8 Flags
038h 72 Partition name (UTF16)

**Example: Linux Logical Volume Management**

**Volume Group Example**

> vgdisplay

```bash
--- Volume group ---
VG Name volumes
System ID
Format lvm2
Metadata Areas 2
Metadata Sequence No 10
VG Access read/write
VG Status resizable
MAX LV 0
Cur LV 3
Open LV 3
Max PV 0
Cur PV 2
Act PV 2
VG Size 1.27 TiB
PE Size 32.00 MiB
Total PE 41695
Alloc PE / Size 24692 / 771.62 GiB
Free PE / Size 17003 / 531.34 GiB
VG UUID fbvtrb-GFbS-Nvf4-Ogg3-J4fX-dj83-ebh39q
```

**Physical Volume Example**

> pvdisplay --map

```bash
--- Physical volume ---
PV Name /dev/md0
VG Name volumes
PV Size 931.39 GiB / not usable 12.56 MiB
Allocatable yes
PE Size 32.00 MiB
Total PE 29804
Free PE 17003
Allocated PE 12801
PV UUID hvfcSD-FvSp-xJn4-lsR3-40Kx-LdDD-wvfGfV

--- Physical Segments ---
Physical extent 0 to 6875:
   Logical volume /dev/volumes/home
   Logical extents 0 to 6875
Physical extent 6876 to 6876:
   Logical volume /dev/volumes/var
   Logical extents 11251 to 11251
Physical extent 6877 to 12800:
   Logical volume /dev/volumes/home
   Logical extents 6876 to 12799
```
Chapter 4. Device Management

Physical extent 12801 to 29803:
FREE

Logical Volume Example

> lvdisplay --map

--- Logical volume ---
LV Name /dev/volumes/home
VG Name volumes
LV UUID OAdf3v-zfII-w5vq-tFVr-Sfgv-yvre-GWFb3v
LV Write Access read/write
LV Status available
LV Size 400.00 GiB
Current LE 12800
Segments 2
Allocation inherit
Read ahead sectors auto
- currently set to 256
Block device 253:2

--- Segments ---
Logical extent 0 to 6875:
  Type linear
  Physical volume /dev/md0
  Physical extents 0 to 6875

Logical extent 6876 to 12799:
  Type linear
  Physical volume /dev/md0
  Physical extents 6877 to 12800

Logical Volume Configuration

contiguous allocation
  allocate adjacent extents

arbitrary allocation
  allocate anywhere at all

normal allocation
  allocate using common sense

cling allocation
  allocate on the same physical volume

linear mapping
  a range of physical extents maps to a range of logical extents in linear order

striped mapping
  physical extents from multiple physical volumes map to a range of logical extents in cyclic order

snapshot volume
  a logical volume that is a copy-on-write image of another logical volume
Chapter 4. Device Management

mirror volume
   a logical volume that is a mirror image of another logical volume

sparse volume
   a logical volume whose storage is mapped upon write
Chapter 5. File Subsystem

File Subsystem

Poskytované abstrakce

• adresáře
• soubory

Základní požadavky

• ukládání velkého počtu i objemu dat
• co nejmenší kapacitní a časová režie
• odolnost proti výpadku systému
• zabezpečení proti neoprávněnému přístupu
• koordinace sdílení dat

Abstractions And Operations

Stream File Operations

Example: Linux Stream File Operations

Open And Close System Calls

```c
int open (  
    char *pathname,  
    int flags);  
int open (  
    char *pathname,  
    int flags, mode_t mode);  
int creat (  
    char *pathname,  
    mode_t mode);  
int close (int fd);  
```

• O_RDONLY, O_WRONLY, O_RDWR
• O_CREAT, O_EXCL, O_TRUNC, O_APPEND
• O_NONBLOCK, O_SYNC

Seek System Call

```c
off_t lseek (  
    int fildes, off_t offset, int whence);  
```

• SEEK_SET, SEEK_CUR, SEEK_END
Chapter 5. File Subsystem

Synchronous Read And Write System Calls

```c
ssize_t read (int fd,
   void *buf, size_t count);
ssize_t write (int fd,
   void *buf, size_t count);

// File access that ignores seek position
ssize_t pread (int fd,
   void *buf, size_t count,
   off_t offset);
ssize_t pwrite (int fd,
   void *buf, size_t count,
   off_t offset);

// File access with scatter and gather
ssize_t readv (int fd,
   struct iovec *vector, int count);
ssize_t writev (int fd,
   struct iovec *vector, int count);
```

```c
def struct iovec {
   void *iov_base;
   size_t iov_len;
}
```

Asynchronous Read And Write System Calls

```c
// Asynchronous single operation
int aio_read (struct aiocb *aiocbp);
int aio_write (struct aiocb *aiocbp);

// Query asynchronous operation status
int aio_error (struct aiocb *aiocbp);
// Get return status of completed operation
ssize_t aio_return (struct aiocb *aiocbp);

// Wait for completion of any listed operation
int aio_suspend (struct aiocb *aiocbp [],
   int n, struct timespec *timeout);

int aio_cancel (int fd, struct aiocb *aiocbp);

// Submit multiple asynchronous operations
int lio_listio (struct aiocb *aiocbp [],
   int mode, struct sigevent *sig);
```

```c
def struct aiocb {
   int aio_fildes;
   off_t aio_offset;
   void *aio_buf;
   size_t aio_nbytes;
   int aio_reqprio;
   struct sigevent aio_sigevent;
   int aio_lio_opcode;
   ...
}
```
### Advise System Calls

```c
int posix_fadvise (  
    int fd,  
    off_t offset, off_t len,  
    int advice);

int posix_fallocate (  
    int fd,  
    off_t offset, off_t len);
```

- POSIX_FADV_NORMAL - no advice
- POSIX_FADV_SEQUENTIAL - sequential access
- POSIX_FADV_RANDOM - random access
- POSIX_FADV_NOREUSE - data used once
- POSIX_FADV_WILLNEED - data will be used soon
- POSIX_FADV_DONTNEED - data will not be used soon

### Example: Windows Stream File Operations

#### Open And Close System Calls

```c
HFILE OpenFile (  
    LPCSTR lpFileName,  
    LPOFSTRUCT lpReOpenBuff,  
    UINT uStyle);

HANDLE CreateFile (  
    LPCTSTR lpFileName,  
    DWORD dwDesiredAccess,  
    DWORD dwShareMode,  
    LPSECURITY_ATTRIBUTES lpSecurityAttributes,  
    DWORD dwCreationDisposition,  
    DWORD dwFlagsAndAttributes,  
    HANDLE hTemplateFile);

HANDLE ReOpenFile (  
    HANDLE hOriginalFile,  
    DWORD dwDesiredAccess,  
    DWORD dwShareMode,  
    DWORD dwFlags);

BOOL CloseHandle (HANDLE hObject);
```

- OF_CREATE - create or truncate file
- OF_EXIST - open and close, used to test existence
- OF_PARSE - only fill the reopen structure
- OF_PROMPT - open a retry dialog if the file does not exist
- OF_REOPEN - use the reopen structure
- OF_VERIFY - compare timestamp with reopen structure
• FILE_SHARE_READ - allow concurrent reads
• FILE_SHARED_WRITE - allow concurrent writes
• FILE_SHARED_DELETE - allow concurrent deletion

• FILE_FLAG_DELETE_ON_CLOSE - used for temporary files
• FILE_FLAG_NO_BUFFERING - bypass memory manager, access must be sector aligned
• FILE_FLAG_OVERLAPPED - allow asynchronous operation
• FILE_FLAG_WRITE_THROUGH - do not use delayed write back

Seek System Call

DWORD SetFilePointer (  
HANDLE hFile,  
LONG lDistanceToMove,  
PLONG lpDistanceToMoveHigh,  
DWORD dwMoveMethod);

• FILE_BEGIN - distance from beginning of file
• FILE_CURRENT - distance from current position in file
• FILE_END - distance from end of file

Read And Write System Calls

BOOL ReadFile (  
HANDLE hFile,  
LPVOID lpBuffer,  
DWORD nNumberOfBytesToRead,  
LPDWORD lpNumberOfBytesRead,  
LPOVERLAPPED lpOverlapped);

BOOL WriteFile (  
HANDLE hFile,  
LPCVOID lpBuffer,  
DWORD nNumberOfBytesToWrite,  
LPDWORD lpNumberOfBytesWritten,  
LPOVERLAPPED lpOverlapped);

BOOL ReadFileEx (  
HANDLE hFile,  
LPVOID lpBuffer,  
DWORD nNumberOfBytesToRead,  
LPOVERLAPPED lpOverlapped,  
LPOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine);

BOOL WriteFileEx (  
HANDLE hFile,  
LPCVOID lpBuffer,  
DWORD nNumberOfBytesToWrite,  
LPOVERLAPPED lpOverlapped,  
LPOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine);

BOOL ReadFileScatter (  
HANDLE hFile,  
FILE_SEGMENT_ELEMENT aSegmentArray [],  
DWORD nNumberOfBytesToRead,  
LPDWORD lpReserved,
Chapter 5. File Subsystem

LPOVERLAPPED lpOverlapped);
BOOL WriteFileGather (HANDLE hFile,
FILE_SEGMENT_ELEMENT aSegmentArray [],
DWORD nNumberOfBytesToWrite,
LPDWORD lpReserved,
LPOVERLAPPED lpOverlapped);

BOOL WINAPI GetOverlappedResult (HANDLE hFile,
LPOVERLAPPED lpOverlapped,
LPDWORD lpNumberOfBytesTransferred,
BOOL bWait);
BOOL HasOverlappedIoCompleted (LPOVERLAPPED lpOverlapped);

BOOL CancelIo (HANDLE hFile);

typedef struct _OVERLAPPED {
  ULONG_PTR Internal;
  ULONG_PTR InternalHigh;
  union {
    struct {
      DWORD Offset;
      DWORD OffsetHigh;
    };
    PVOID Pointer;
  };
  HANDLE hEvent;
} OVERLAPPED, *LPOVERLAPPED;

typedef union _FILE_SEGMENT_ELEMENT {
  PVOID64 Buffer;
  ULONGLONG Alignment;
} FILE_SEGMENT_ELEMENT, *PFILE_SEGMENT_ELEMENT;

Mapped File Operations

Example: Linux Mapped File Operations

**MMap And MUnmap System Calls**

void * mmap ( void *start, size_t length,
int prot, int flags,
int fd, off_t offset);
int munmap ( void *start, size_t length);

- PROT_READ, PROT_WRITE, PROT_EXEC, PROT_NONE
- MAP_SHARED, MAP_PRIVATE
- MAP_FIXED
- MAP_ANONYMOUS
Chapter 5. File Subsystem

**MRemap System Call**

```c
void *mremap (  
    void *old_address,  
    size_t old_size, size_t new_size,  
    unsigned long flags);
```

- MREMAP_MAYMOVE

**MSync System Call**

```c
int msync (  
    void *start, size_t length, int flags);
```

- MS_SYNC - flush synchronously
- MS_ASYNC - flush asynchronously
- MS_INVALIDATE - invalidate other mappings

**Advise System Calls**

```c
int posix_madvise (  
    void *addr, size_t len, int advice);
```

- POSIX_MADV_NORMAL - no advice
- POSIX_MADV_SEQUENTIAL - sequential access
- POSIX_MADV_RANDOM - random access
- POSIX_MADV_WILLNEED - data will be used soon
- POSIX_MADV_DONTNEED - data will not be used soon

**Example: Windows Mapped File Operations**

**CreateFileMapping System Call**

```c
HANDLE CreateFileMapping (  
    HANDLE hFile,  
    LPSECURITY_ATTRIBUTES lpFileMappingAttributes,  
    DWORD flProtect,  
    DWORD dwMaximumSizeHigh,  
    DWORD dwMaximumSizeLow,  
    LPCTSTR lpName);
```

- PAGE_READONLY, PAGE_READWRITE, PAGE_READCOPY
- SEC_COMMIT, SEC_RESERVE
- SEC_NOCACHE
MapViewOfFile System Call

LPVOID MapViewOfFile (  
    HANDLE hFileMappingObject, DWORD dwDesiredAccess,  
    DWORD dwFileOffsetHigh, DWORD dwFileOffsetLow,  
    DWORD dwNumberOfBytesToMap);

LPVOID MapViewOfFileEx (  
    HANDLE hFileMappingObject, DWORD dwDesiredAccess,  
    DWORD dwFileOffsetHigh, DWORD dwFileOffsetLow,  
    DWORD dwNumberOfBytesToMap, LPVOID lpBaseAddress);

BOOL UnmapViewOfFile (  
    LPCVOID lpBaseAddress);

• FILE_MAP_WRITE, FILE_MAP_READ, FILE_MAP_ALL_ACCESS
• FILE_MAP_COPY

Whole File Operations

Example: Linux Whole File Operations

Send File System Call

ssize_t sendfile (  
    int out_fd,  
    int in_fd,  
    off_t *offset,  
    size_t count);

ssize_t splice (  
    int fd_in,  
    loff_t *off_in,  
    int fd_out,  
    loff_t *off_out,  
    size_t len,  
    unsigned int flags);

Example: Windows Whole File Operations

Copy And Move System Calls

BOOL CopyFile (  
    LPCTSTR lpExistingFileName,  
    LPCTSTR lpNewFileName,  
    BOOL bFailIfExists);

BOOL CopyFileEx (  
    LPCTSTR lpExistingFileName,  
    LPCTSTR lpNewFileName,  
    LPPROGRESS_ROUTINE lpProgressRoutine,  
    LPVOID lpData,  
    LPBOOL pbCancel,
DWORD dwCopyFlags);

BOOL MoveFile (  
    LPCTSTR lpExistingFileName,  
    LPCTSTR lpNewFileName);

BOOL MoveFileWithProgress (  
    LPCTSTR lpExistingFileName,  
    LPCTSTR lpNewFileName,  
    LPPROGRESS_ROUTINE lpProgressRoutine,  
    LPVOID lpData,  
    DWORD dwFlags);

BOOL ReplaceFile (  
    LPCTSTR lpReplacedFileName,  
    LPCTSTR lpReplacementFileName,  
    LPCTSTR lpBackupFileName,  
    DWORD dwReplaceFlags,  
    LPVOID lpExclude,  
    LPVOID lpReserved);

• COPY_FILE_RESTARTABLE - store resume data in target

• REPLACEFILE_WRITE_THROUGH - do not use delayed write back

Backup System Calls

BOOL BackupRead (  
    HANDLE hFile,  
    LPBYTE lpBuffer,  
    DWORD nNumberOfBytesToRead,  
    LPDWORD lpNumberOfBytesRead,  
    BOOL bAbort,  
    BOOL bProcessSecurity,  
    LPVOID* lpContext);

BOOL BackupWrite (  
    HANDLE hFile,  
    LPBYTE lpBuffer,  
    DWORD nNumberOfBytesToWrite,  
    LPDWORD lpNumberOfBytesWritten,  
    BOOL bAbort,  
    BOOL bProcessSecurity,  
    LPVOID* lpContext);

BOOL BackupSeek (  
    HANDLE hFile,  
    DWORD dwLowBytesToSeek,  
    DWORD dwHighBytesToSeek,  
    LPDWORD lpdwLowByteSeeked,  
    LPDWORD lpdwHighByteSeeked,  
    LPVOID* lpContext);

• bAbort - last call, free the context structure

• lpContext - the context structure, allocated on first call
Directory Operations

Example: Linux Directory Operations

Directory Family Of System Calls

DIR *opendir (const char *name);
int closedir (DIR *dir);
struct dirent *readdir (DIR *dir);

Directory Family Of System Calls

int scandir (const char *dir, struct dirent ***namelist,
int (*select) (const struct dirent *),
int (*compar) (const struct dirent **,
    const struct dirent **));

Stat System Call

int stat (char *path, struct stat *buf);

struct stat {
    dev_t st_dev; // File device
    ino_t st_ino; // File inode
    mode_t st_mode; // Access rights
    nlink_t st_nlink;
    uid_t st_uid; // Owner UID
    gid_t st_gid; // Owner GID
    dev_t st_rdev; // Device ID for special files
    off_t st_size; // Size in bytes
    blksize_t st_blksize; // Block size
    blkcnt_t st_blocks; // Size in blocks
    time_t st_atime; // Last access time
    time_t st_mtime; // Last modification time
    time_t st_ctime; // Last status change time
}

Example: Windows Directory Operations

Directory Family Of System Calls

HANDLE FindFirstFile (LPCTSTR lpFileName,
    LPWIN32_FIND_DATA lpFindFileData);
BOOL FindNextFile (HANDLE hFindFile,
    LPWIN32_FIND_DATA lpFindFileData);

typedef struct _WIN32_FIND_DATA {
    DWORD dwFileAttributes;
    FILETIME ftCreationTime;
    FILETIME ftLastAccessTime;
    FILETIME ftLastWriteTime;
    DWORD nFileSizeHigh;
    DWORD nFileSizeLow;
    DWORD nFileAttributes;
**Chapter 5. File Subsystem**

```c
DWORD dwReserved0;
DWORD dwReserved1;
TCHAR cFileName [MAX_PATH (= 260)];
TCHAR cAlternateFileName [14];
} WIN32_FIND_DATA;
```

**Unique File System Calls**

```c
UINT GetTempFileName (
    LPCTSTR lpPathName,
    LPCTSTR lpPrefixString,
    UINT uUnique,
    LPTSTR lpTempFileName
);
```

**Sharing Support**

**Example: Linux Sharing Operations**

**File Locking System Call**

```c
int flock (int fd, int operation);
```

- **LOCK_EX** - exclusive advisory lock
- **LOCK_SH** - shared advisory lock
- **LOCK_UN** - advisory unlock

**Block Locking System Call**

```c
int fcntl (int fd, int cmd, struct flock *lock);
struct flock {
    ...
    short l_type; // F_WRLCK, F_RDLCK, F_UNLCK
    short l_whence; // SEEK_SET, SEEK_CUR, SEEK_END
    off_t l_start; // Starting offset for lock
    off_t l_len; // Number of bytes to lock
    pid_t l_pid; // Who blocks our lock
    ...
};
```

- **cmd** - F_GETLK, F_SETLK, F_SETLKW - advisory or mandatory lock get, set, set with wait
- **l_type** - exclusive, shared, unlock
- **l_whence** - how to interpret starting offset
Example: Windows Sharing Operations

Block Locking System Calls

BOOL LockFile (
    HANDLE hFile,
    DWORD dwFileOffsetLow,
    DWORD dwFileOffsetHigh,
    DWORD nNumberOfBytesToLockLow,
    DWORD nNumberOfBytesToLockHigh);

BOOL UnlockFile (
    HANDLE hFile,
    DWORD dwFileOffsetLow,
    DWORD dwFileOffsetHigh,
    DWORD nNumberOfBytesToUnlockLow,
    DWORD nNumberOfBytesToUnlockHigh);

Consistency Support

Example: Windows Transaction Operations

Transaction System Calls

HANDLE CreateTransaction (    
    LPSECURITY_ATTRIBUTES lpTransactionAttributes, 
    LPGUID UOW, 
    DWORD CreateOptions, 
    DWORD IsolationLevel, 
    DWORD IsolationFlags, 
    DWORD Timeout, 
    LPWSTR Description);

BOOL CommitTransaction (    
    HANDLE TransactionHandle);

BOOL RollbackTransaction (    
    HANDLE TransactionHandle);

HANDLE CreateFileTransacted (    
    LPCTSTR lpFileName, 
    DWORD dwDesiredAccess, 
    DWORD dwShareMode, 
    LPSECURITY_ATTRIBUTES lpSecurityAttributes, 
    DWORD dwCreationDisposition, 
    DWORD dwFlagsAndAttributes, 
    HANDLE hTemplateFile, 
    HANDLE hTransaction, 
    PUSHORT pusMiniVersion, 
    PVOID pExtendedParameter);

BOOL DeleteFileTransacted(    
    LPCTSTR lpFileName, 
    HANDLE hTransaction);

BOOL CreateDirectoryTransacted (...);

BOOL RemoveDirectoryTransacted (...);

BOOL MoveFileTransacted (...);

BOOL CopyFileTransacted (...);
Chapter 5. File Subsystem

File Subsystem Internals

Disk Layout

Example: FAT File System

**FAT Boot Sector**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>3</td>
<td>Boot loader code</td>
</tr>
<tr>
<td>03h</td>
<td>8</td>
<td>System vendor ID</td>
</tr>
<tr>
<td>0Bh</td>
<td>2</td>
<td>Bytes per sector</td>
</tr>
<tr>
<td>0Dh</td>
<td>1</td>
<td>Sectors per cluster</td>
</tr>
<tr>
<td>0Eh</td>
<td>2</td>
<td>Number of reserved sectors before FAT</td>
</tr>
<tr>
<td>10h</td>
<td>1</td>
<td>Number of FAT copies</td>
</tr>
<tr>
<td>11h</td>
<td>2</td>
<td>Number of root entries</td>
</tr>
<tr>
<td>13h</td>
<td>2</td>
<td>Number of sectors for small partitions</td>
</tr>
<tr>
<td>15h</td>
<td>1</td>
<td>Media descriptor</td>
</tr>
<tr>
<td>16h</td>
<td>2</td>
<td>Sectors per FAT</td>
</tr>
<tr>
<td>18h</td>
<td>2</td>
<td>Sectors per head</td>
</tr>
<tr>
<td>1Ah</td>
<td>2</td>
<td>Heads per cylinder</td>
</tr>
<tr>
<td>1Ch</td>
<td>4</td>
<td>Number of hidden sectors before boot</td>
</tr>
<tr>
<td>20h</td>
<td>4</td>
<td>Number of sectors for large partitions</td>
</tr>
<tr>
<td>24h</td>
<td>474</td>
<td>Boot loader code</td>
</tr>
<tr>
<td>1FEh</td>
<td>2</td>
<td>Magic (0AA55h)</td>
</tr>
<tr>
<td>24h</td>
<td>4</td>
<td>Number of sectors per FAT</td>
</tr>
<tr>
<td>28h</td>
<td>2</td>
<td>File system flags</td>
</tr>
<tr>
<td>2Ah</td>
<td>2</td>
<td>File system version</td>
</tr>
<tr>
<td>2Ch</td>
<td>4</td>
<td>First cluster of root directory</td>
</tr>
<tr>
<td>30h</td>
<td>2</td>
<td>Sector with file system information</td>
</tr>
<tr>
<td>32h</td>
<td>2</td>
<td>Sector with boot sector copy</td>
</tr>
<tr>
<td>34h</td>
<td>12</td>
<td>Reserved</td>
</tr>
<tr>
<td>40h</td>
<td>1</td>
<td>Physical drive number</td>
</tr>
<tr>
<td>41h</td>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>42h</td>
<td>1</td>
<td>Magic (28h or 29h)</td>
</tr>
<tr>
<td>43h</td>
<td>4</td>
<td>Volume serial number</td>
</tr>
<tr>
<td>47h</td>
<td>11</td>
<td>Volume label</td>
</tr>
<tr>
<td>52h</td>
<td>8</td>
<td>File system ID</td>
</tr>
<tr>
<td>5Ah</td>
<td>420</td>
<td>Boot loader code</td>
</tr>
</tbody>
</table>

**FAT Directory Entry**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>8</td>
<td>File name padded with spaces or 00h or 1Eh</td>
</tr>
<tr>
<td>08h</td>
<td>3</td>
<td>File extension padded with spaces</td>
</tr>
<tr>
<td>0Bh</td>
<td>1</td>
<td>File attributes (archive, dir, vol, sys, hidden, read only)</td>
</tr>
<tr>
<td>0Ch</td>
<td>10</td>
<td>Reserved</td>
</tr>
<tr>
<td>16h</td>
<td>2</td>
<td>Last modification time</td>
</tr>
<tr>
<td>18h</td>
<td>2</td>
<td>Last modification date</td>
</tr>
<tr>
<td>1Ah</td>
<td>2</td>
<td>Starting cluster</td>
</tr>
<tr>
<td>1Ch</td>
<td>4</td>
<td>File size</td>
</tr>
<tr>
<td>0Ch</td>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>0Dh</td>
<td>3</td>
<td>Creation time</td>
</tr>
<tr>
<td>10h</td>
<td>2</td>
<td>Creation date</td>
</tr>
<tr>
<td>12h</td>
<td>2</td>
<td>Last access date</td>
</tr>
<tr>
<td>14h</td>
<td>2</td>
<td>Starting cluster high word</td>
</tr>
</tbody>
</table>
Chapter 5. File Subsystem

00h 1 Sequential number of long name fragment and last fragment flag
01h 10 Long name (5 characters UNICODE)
0Bh 1 File attributes (0Fh means vol, sys, hidden, read only)
0Ch 1 Reserved
0Dh 1 Checksum of short name
0Eh 12 Long name (6 characters UNICODE)
1Ah 2 Reserved
1Ch 4 Long name (2 characters UNICODE)

Example: HPFS File System

HPFS F-Node

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>4</td>
<td>Magic (0F7E40AAEh)</td>
</tr>
<tr>
<td>04h</td>
<td>4</td>
<td>Sequential read history (not implemented)</td>
</tr>
<tr>
<td>08h</td>
<td>4</td>
<td>Fast read history (not implemented)</td>
</tr>
<tr>
<td>0Ch</td>
<td>1</td>
<td>Name length</td>
</tr>
<tr>
<td>0Dh</td>
<td>15</td>
<td>Last 15 characters of name</td>
</tr>
<tr>
<td>1Ch</td>
<td>4</td>
<td>Sector of containing directory</td>
</tr>
<tr>
<td>20h</td>
<td>4</td>
<td>Size of external access control list</td>
</tr>
<tr>
<td>24h</td>
<td>4</td>
<td>Sector of external access control list</td>
</tr>
<tr>
<td>28h</td>
<td>2</td>
<td>Size of internal access control list</td>
</tr>
<tr>
<td>2Ah</td>
<td>1</td>
<td>Indicates whether access control list is large tree</td>
</tr>
<tr>
<td>2Bh</td>
<td>1</td>
<td>History bit count (not implemented)</td>
</tr>
<tr>
<td>2Ch</td>
<td>4</td>
<td>Size of external extended attributes</td>
</tr>
<tr>
<td>30h</td>
<td>4</td>
<td>Sector of external extended attributes</td>
</tr>
<tr>
<td>34h</td>
<td>2</td>
<td>Size of internal extended attributes</td>
</tr>
<tr>
<td>36h</td>
<td>1</td>
<td>Indicates whether extended attributes is large tree</td>
</tr>
<tr>
<td>37h</td>
<td>1</td>
<td>Directory or file F node flag</td>
</tr>
<tr>
<td>38h</td>
<td>1</td>
<td>Indicates whether runs are large tree</td>
</tr>
<tr>
<td>39h</td>
<td>3</td>
<td>Padding</td>
</tr>
<tr>
<td>3Ch</td>
<td>1</td>
<td>Number of free array entries</td>
</tr>
<tr>
<td>3Dh</td>
<td>1</td>
<td>Number of used array entries</td>
</tr>
<tr>
<td>3Eh</td>
<td>2</td>
<td>Offset of first free array entry</td>
</tr>
<tr>
<td>40h</td>
<td>4</td>
<td>Offset of this run in file</td>
</tr>
<tr>
<td>44h</td>
<td>4</td>
<td>Number of sectors in run</td>
</tr>
<tr>
<td>48h</td>
<td>4</td>
<td>Starting sector of this run</td>
</tr>
<tr>
<td>40h</td>
<td>4</td>
<td>Offset of this subtree in file</td>
</tr>
<tr>
<td>44h</td>
<td>4</td>
<td>Starting sector of this subtree</td>
</tr>
<tr>
<td>0A0h</td>
<td>4</td>
<td>File length</td>
</tr>
<tr>
<td>0A4h</td>
<td>4</td>
<td>Number of vital extended attributes</td>
</tr>
<tr>
<td>0A8h</td>
<td>16</td>
<td>User identity (not implemented)</td>
</tr>
<tr>
<td>0B8h</td>
<td>2</td>
<td>Offset of first access control list or extended attribute entry</td>
</tr>
<tr>
<td>0BAh</td>
<td>10</td>
<td>Reserved</td>
</tr>
<tr>
<td>0C4h</td>
<td>316</td>
<td>Access control list and extended attribute entries</td>
</tr>
</tbody>
</table>

HPFS Directory Entry

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>4</td>
<td>Magic (77E40AAEh)</td>
</tr>
<tr>
<td>04h</td>
<td>4</td>
<td>Offset of first free directory entry</td>
</tr>
<tr>
<td>08h</td>
<td>4</td>
<td>Tree root indication</td>
</tr>
<tr>
<td>0Ch</td>
<td>4</td>
<td>Sector of tree parent block</td>
</tr>
<tr>
<td>10h</td>
<td>4</td>
<td>Sector of this tree node block</td>
</tr>
</tbody>
</table>
Chapter 5. File Subsystem

14h 2 Directory entry size
16h 1 Flags
17h 1 Attributes (read only, hidden, sys, dir, archive, long name)
18h 4 Sector of F node
1Ch 4 Last modification time
20h 4 File size
24h 4 Last access time
28h 4 Creation time
2Ch 4 Size of extended attributes in F node
30h 1 Size of access control list
31h 1 Code page index
32h 1 Name size
33h X Name
33h+X X Access control list
Padding to multiple of 4 bytes

Example: EXT2 And EXT3 And EXT4 File Systems

EXT2 Inode Structure

```c
struct ext2_inode {
    __u16 i_mode; /* File mode */
    __u16 i_uid; /* Owner ID */
    __u32 i_size; /* Size in bytes */
    __u32 i_atime; /* Access time */
    __u32 i_ctime; /* Creation time */
    __u32 i_mtime; /* Modification time */
    __u32 i_dtime; /* Deletion Time */
    __u16 i_gid; /* Group ID */
    __u16 i_links_count; /* Links count */
    __u32 i_blocks; /* Blocks count */
    __u32 i_blocks; /* Blocks count */
    __u16 i_flag; /* File flags */
    __u32 i_block [EXT2_N_BLOCKS]; /* Ptrs to blocks */
    __u32 i_version; /* File version for NFS */
    __u32 i_file_acl; /* File ACL */
    __u32 i_dir_acl; /* Directory ACL */
    __u32 i_faddr; /* Fragment address */
    __u8 1_i_frag; /* Fragment number */
    __u8 1_i_fsize; /* Fragment size */
};
```

#define EXT2_DIR_BLOCKS 12
#define EXT2_IND_BLOCK EXT2_DIR_BLOCKS
#define EXT2_DIND_BLOCK (EXT2_IND_BLOCK + 1)
#define EXT2_TIND_BLOCK (EXT2_DIND_BLOCK + 1)
#define EXT2_N_BLOCKS (EXT2_TIND_BLOCK + 1)
#define EXT2_SECRM_FL 0x00000001 /* Secure del */
#define EXT2_SYNC_FL 0x00000008 /* Sync update */
#define EXT2_IMMUTABLE_FL 0x00000010 /* Immutable */
#define EXT2_APPEND_FL 0x00000020 /* Only ap */
**EXT2 Directory Entry Structure**

```c
struct ext2_dir_entry_2 {
    __u32 inode; /* Inode number */
    __u16 rec_len; /* Directory entry length */
    __u8 name_len; /* Name length */
    __u8 file_type; /* File type */
    char name [EXT2_NAME_LEN]; /* File name */
};
```

```c
#define EXT2_NAME_LEN 255
#define EXT2_FT_REG_FILE 1
#define EXT2_FT_DIR 2
#define EXT2_FT_CHRDEV 3
#define EXT2_FT_BLKDEV 4
#define EXT2_FT_SYMLINK 7
```

**EXT2 I-Node Usage**

```bash
> df -i /
Filesystem Inodes IUsed IFree IUse% Mounted on
/dev/hda1 24903680 1007827 23895853 5% /
```

**EXT2 Superblock Information**

```bash
> tune2fs -l /dev/hda1
tune2fs 1.35 (28-Feb-2004)
Filesystem volume name: /
Last mounted on: (not available)
Filesystem UUID: 7404a4b8-84f5-11d6-9629-9bbda41ad84
Filesystem magic number: 0xEF53
Filesystem revision #: 1 (dynamic)
Filesystem features: has_journal ext_attr dir_index filetype
                      needs_recovery sparse_super large_file
Default mount options: (none)
Filesystem state: clean
Errors behavior: Continue
Filesystem OS type: Linux
Inode count: 24903680
Block count: 49785427
Reserved block count: 1024
Free blocks: 8828311
Free inodes: 23895853
First block: 0
Block size: 4096
Fragment size: 4096
Blocks per group: 32768
Fragments per group: 32768
Inodes per group: 16384
Inode blocks per group: 512
Last mount time: Tue Mar 15 21:21:31 2005
Last write time: Tue Mar 15 21:21:31 2005
Mount count: 28
Maximum mount count: -1
Check interval: 0 (none)
Reserved blocks uid: 0 (user root)
Reserved blocks gid: 0 (group root)
First inode: 11
Inode size: 128
Journal inode: 8
```
Chapter 5. File Subsystem

First orphan inode: 328593
Default directory hash: tea
Directory Hash Seed: b099544d-7257-456c-8666-4c646f123e16
Journal backup: inode blocks

Example: NTFS File System

MFT Entry

typedef struct MftEntry
{
    char Signature [4]; // Magic "FILE"
    ushort FixupOffset;
    ushort FixupSize;
    ulong Sequence; // Sequence number in MFT
    ulong HardLinks; // Hard link count
    ushort AttribOffset; // Offset of attributes
    ushort Flags;
    ulong RecLength; // True record size
    ulong AllLength; // Allocated record size
    cluster BaseMftRec; // Base entry of this entry or 0
    ushort MinIdentificator;
    ushort FixupPattern;
    ushort FixupList[];
};

typedef struct AttributeEntry
{
    ulong Type; // Type of attribute
    ushort Length; // Length of attribute entry
    ushort Residency; // Resident or nonresident
    byte NameLen; // Length of name
    ushort Offset; // Offset of name or data
    byte Compressed; // Compressed or uncompressed
    byte Identificator;
    union
    {
        ResidentAttribEntry Resident;
        NonresidentAttribEntry NonResident;
    };
};

typedef struct ResidentAttribEntry
{
    ushort Size; // Size of attribute
    ushort Offset; // Offset of value
    ushort IndexFlag;
};

typedef struct NonResidentAttribEntry
{
    cluster SegFirst; // First cluster in this segment
    cluster SegLast; // Last cluster in this segment
    USHORT Offset; // Offset of run list
    USHORT ComprEngine; // Compression engine
    USHORT
Chapter 5. File Subsystem

USHORT
XLONG Allocated;  // Allocated disk space
XLONG Size;  // Size of uncompressed attribute
XLONG Compressed;  // Size of compressed attribute
);

typedef struct DirectoryEntry
{
    xlong RecordNumber;  // This MFT record number
    ushort Length;  // Length of directory entry
    ushort byte Flags;
    byte xlong Parent;  // Parent MFT record number
    time Create;  // Creation time
    time ModifyFile;  // Last file modification time
    time ModifyEntry;  // Last entry modification time
    time Access;  // Last access time
    xlong Alloc;  // Allocated size
    xlong Size;  // Real size
    xlong byte NameLen;  // Name length in words
    byte byte NameType;  // Type of filename
    ushort Filename [];  // Variable length filename
};

Multiple Streams

Stream Enumeration System Calls

HANDLE FindFirstStreamW ( 
    LPCWSTR lpFileName,
    STREAM_INFO_LEVELS InfoLevel,
    LPVOID lpFindStreamData,
    DWORD dwFlags);
BOOL FindNextStreamW ( 
    HANDLE hFindStream,
    LPVOID lpFindStreamData);

typedef enum _STREAM_INFO_LEVELS { 
    FindStreamInfoStandard
} STREAM_INFO_LEVELS;

typedef struct _WIN32_FIND_STREAM_DATA ( 
    LARGE_INTEGER StreamSize;
    WCHAR cStreamName [MAX_PATH + 36];
) WIN32_FIND_STREAM_DATA;

Example: CD File System

ISO9660 Primary Volume Descriptor

Offset   Length  Contents
-----------------------------------------------
0        8        Magic (1, "CD001", 1, 0)
8        32       System identifier
28h      32       Volume identifier
Chapter 5. File Subsystem

ISO9660 Directory Entry

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>Record size in bytes (must be even)</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>Number of sectors in extended attribute record</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>Number of bytes of file data or length of directory</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>Date and time</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>Flags (HID, DIR ..., LAST for multiple sections)</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>Interleaved file unit size or 0</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>Interleaved file gap size or 0</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Volume sequence number (1)</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>Length of name</td>
</tr>
<tr>
<td>N</td>
<td></td>
<td>Name</td>
</tr>
<tr>
<td>P</td>
<td></td>
<td>Padding</td>
</tr>
</tbody>
</table>

Integration Of Multiple File Subsystems

Example: Linux Virtual File System

Linux VFS Layer

- Purpose
  Unified interface to user processes
  Kernel abstraction for different implementations
- Function
Service file/file system syscalls
Manage file/file system data structures
Provide generic routines for common operations
Interact with file system implementations

Basic VFS Objects

- superblock
  Mounted instance of a file system.
- dentry
  Directory entry in the filesystem hierarchy.
- inode
  In-kernel representation of a file.
- file
  Open file descriptor.

Super Block

- Handles metadata
  Retrieves and stores metadata from media
- Holds file system instance data
  device, block size, dirty flags
  root inode, list of dirty inodes
  superblock operations

Super Block Structure

```c
struct super_block {
    struct list_head s_list;    /* Keep this first */
    dev_t s_dev;                /* search index; _not_ kdev_t */
    unsigned long s_blocksize;
    unsigned long s_old_blocksize;
    unsigned char s_blocksize_bits;
    unsigned char s_dirt;
    unsigned long long s_maxbytes; /* Max file size */
    struct file_system_type *s_type;
    struct super_operations *s_op;
    struct dquot_operations *dq_op;
    struct quotactl_ops *s_qcop;
    struct export_operations *s_export_op;
    unsigned long s_flags;
    unsigned long s_magic;
    struct dentry *s_root;
    struct rw_semaphore s_umount;
    struct semaphore s_lock;
    int s_count;
    int s_syncing;
    int s_need_sync_fs;
};
```
atomic_t s_active;
void *s_security;
struct xattr_handler **s_xattr;

struct list_head s_inodes; /* all inodes */
struct list_head s_dirty; /* dirty inodes */
struct list_head s_io; /* parked for writeback */
struct hlist_head s_anon; /* anonymous dentries for (nfs) exporting */
struct list_head s_files;

struct block_device *s_bdev;
struct list_head s_instances;
struct quota_info s_dquot; /* Diskquota specific options */

int s_frozen;
wait_queue_head_t s_wait_unfrozen;

char s_id[32]; /* Informational name */
void *s_fs_info; /* Filesystem private info */

/*
 * The next field is for VFS only. No filesystems have any business
 * even looking at it. You had been warned.
 */
struct semaphore s_vfs_rename_sem; /* Kludge */

/* Granularity of c/m/atime in ns.
 * Cannot be worse than a second */
u32 s_time_gran;


Super Block Operations

struct super_operations {
    struct inode *(*alloc_inode) (struct super_block *sb);
    void (*destroy_inode) (struct inode *);

    void (*read_inode) (struct inode *);
    void (*dirty_inode) (struct inode *);
    int (*write_inode) (struct inode *, int);
    void (*put_inode) (struct inode *);
    void (*drop_inode) (struct inode *);
    void (*delete_inode) (struct inode *);

    void (*put_super) (struct super_block *);
    void (*write_super) (struct super_block *);

    int (*sync_fs) (struct super_block *, int wait);
    void (*write_super_lockfs) (struct super_block *);
    void (*unlockfs) (struct super_block *);

    int (*statfs) (struct super_block *, struct kstatfs *);
    int (*remount_fs) (struct super_block *, int *, char *);

    void (*clear_inode) (struct inode *);
    void (*umount_begin) (struct super_block *);

    int (*show_options) (struct seq_file *, struct vfsmount *);
    ssize_t (*quota_read) (struct super_block *, int, char *, size_t, loff_t);
Chapter 5. File Subsystem

ssize_t (*quota_write) (struct super_block *, int, const char *, size_t, loff_t);

Directory Entry

- Name to inode translation structure
- Cached aggressively by the VFS (dcache)
- Eliminates private FS lookups, caching
- Negative dentries

Directory Entry Structure

```
struct dentry {
    atomic_t d_count;
    unsigned int d_flags; /* protected by d_lock */
    spinlock_t d_lock; /* per dentry lock */
    struct inode *d_inode; /* Where the name belongs to - NULL is negative */
    /*
    * The next three fields are touched by __d_lookup. Place them here
    * so they all fit in a 16-byte range, with 16-byte alignment.
    */
    struct dentry *d_parent; /* parent directory */
    struct qstr d_name;
    struct list_head d_lru; /* LRU list */
    struct list_head d_child; /* child of parent list */
    struct list_head d_subdirs; /* our children */
    struct list_head d_alias; /* inode alias list */
    unsigned long d_time; /* used by d_revalidate */
    struct dentry_operations *d_op;
    struct super_block *d_sb; /* The root of the dentry tree */
    void *d_fsdata; /* fs-specific data */
    struct rcu_head d_rcu;
    struct dcookie_struct *d_cookie; /* cookie, if any */
    struct hlist_node d_hash; /* lookup hash list */
    int d_mounted;
    unsigned char d_iname[DNAME_INLINE_LEN_MIN]; /* small names */
};
```

Directory Entry Operations

```
struct dentry_operations {
    int (*d_revalidate) (struct dentry *, struct nameidata *);
    int (*d_hash) (struct dentry *, struct qstr *);
    int (*d_compare) (struct dentry *, struct qstr *, struct qstr *);
    int (*d_delete) (struct dentry *);
    void (*d_release) (struct dentry *);
    void (*d_iput) (struct dentry *, struct inode *);
};
```
Index Node

- VFS abstraction for a file
- Held in inode cache by VFS
- Hold inode and default file operations
- Contain FS specific areas

Index Node Structure

```c
struct inode {
    struct hlist_node     i_hash;
    struct list_head      i_list;
    struct list_head      i_sb_list;
    struct list_head      i_dentry;
    unsigned long         i_ino;
    atomic_t              i_count;
    umode_t               i_mode;
    unsigned int          i_nlink;
    uid_t                 i_uid;
    gid_t                 i_gid;
    dev_t                 i_rdev;
    loff_t                i_size;
    struct timespec       i_atime;
    struct timespec       i_mtime;
    struct timespec       i_ctime;
    unsigned int          i_blkbits;
    unsigned long         i_blocksize;
    unsigned long         i_version;
    unsigned long         i_blocks;
    unsigned short        i_bytes;
    unsigned char         i_sock;
    spinlock_t            i_lock; /* i_blocks, i_bytes, maybe i_size */
    struct semaphore      i_sem;
    struct rw_semaphore   i_alloc_sem;
    struct inode_operations *i_op;
    struct file_operations *i_fop; /* former ->i_op->default_file_ops */
    struct super_block    *i_sb;
    struct file_lock      *i_flock;
    struct address_space  *i_mapping;
    struct address_space  *i_data;
    #ifdef CONFIG_QUOTA
    struct dquot          *i_dquot[MAXQUOTAS];
    #endif
    struct list_head      i_devices;
    struct pipe_inode_info *i_pipe;
    struct block_device   *i_bdev;
    struct cdev           *i_cdev;
    int                    i_cindex;
    __u32                   i_generation;
};
```


```c
#if defined CONFIG_DNOTIFY
unsigned long i_dnotify_mask; /* Directory notify events */
struct dnotify_struct *i_dnotify; /* for directory notifications */
#endif

unsigned long i_state;
unsigned long dirtied_when; /* jiffies of first dirtying */
unsigned int i_flags;
atomic_t i_writecount;
void *i_security;
union {
  void *generic_ip;
} u;
#endif
```

### Index Node Operations

```c
struct inode_operations {
  int (*create) (struct inode *, struct dentry *, int, struct nameidata *);
  struct dentry * (*lookup) (struct inode *, struct dentry *, struct nameidata *);
  int (*link) (struct dentry *, struct inode *, struct dentry *);
  int (*unlink) (struct inode *, struct dentry *);
  int (*symlink) (struct inode *, struct dentry *, const char *);
  int (*mkdir) (struct inode *, struct dentry *, int);
  int (*rmdir) (struct inode *, struct dentry *);
  int (*mknod) (struct inode *, struct dentry *, int, dev_t);
  int (*rename) (struct inode *, struct dentry *, struct inode *, struct dentry *);
  int (*readlink) (struct dentry *, char __user *, int);
  int (*follow_link) (struct dentry *, struct nameidata *);
  void (*put_link) (struct dentry *, struct nameidata *);
  void (*truncate) (struct inode *);
  int (*permission) (struct inode *, int, struct nameidata *);
  int (*setattr) (struct dentry *, struct iattr *);
  int (*getattr) (struct vfsmount *, struct dentry *, struct kstat *);
  int (*setxattr) (struct dentry *, const char *, const void *, size_t, int);
  ssize_t (*getxattr) (struct dentry *, const char *, void *, size_t);
  ssize_t (*listxattr) (struct dentry *, char *, size_t);
  int (*removexattr) (struct dentry *, const char *);
};
```
File Structure

struct file {
  struct list_head f_list;
  struct dentry *f_dentry;
  struct vfsmount *f_vfsmnt;
  struct file_operations *f_op;
  atomic_t f_count;
  unsigned int f_flags;
  mode_t f_mode;
  int f_error;
  loff_t f_pos;
  struct fown_struct f_owner;
  unsigned int f_uid, f_gid;
  struct file_ra_state f_ra;

  size_t f_maxcount;
  unsigned long f_version;
  void *f_security;

  /* needed for tty driver, and maybe others */
  void *private_data;

#ifdef CONFIG_EPOLL
  /* Used by fs/eventpoll.c to link all the hooks to this file */
  struct list_head f_ep_links;
  spinlock_t f_ep_lock;
#endif /* #ifdef CONFIG_EPOLL */
  struct address_space *f_mapping;
};

File Operations

struct file_operations {
  struct module *owner;

  loff_t (*llseek) (struct file *, loff_t, int);
  ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
  ssize_t (*aio_read) (struct kiocb *, char __user *, size_t, loff_t);
  ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
  ssize_t (*aio_write) (struct kiocb *, const char __user *, size_t, loff_t);
  int (*readdir) (struct file *, void *, filldir_t);
  unsigned int (*poll) (struct file *, struct poll_table_struct *);

  int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
  long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
  long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
  int (*mmap) (struct file *, struct vm_area_struct *);
  int (*open) (struct inode *, struct file *);
  int (*flush) (struct file *);
  int (*release) (struct inode *, struct file *);
  int (*fsync) (struct file *, struct dentry *, int datasync);
  int (*aio_fsync) (struct kiocb *, int datasync);
  int (*fasync) (int, struct file *, int);
  int (*lock) (struct file *, int, struct file_lock *);

  ssize_t (*readv) (struct file *, const struct iovec *, unsigned long, loff_t *);
  ssize_t (*writev) (struct file *, const struct iovec *, unsigned long, loff_t *);
};
Chapter 5. File Subsystem

ssize_t (*sendfile) (  
    struct file *, loff_t *, size_t, read_actor_t, void *);  
ssize_t (*sendpage) (  
    struct file *, struct page *, int, size_t, loff_t *, int);  
unsigned long (*get_unmapped_area) (  
    struct file *, unsigned long, unsigned long,  
    unsigned long, unsigned long);  
int (*check_flags)(int);  
int (*dir_notify)(struct file *filp, unsigned long arg);  
int (*flock) (struct file *, int, struct file_lock *);}  

Auxiliary VFS Objects

• filesystem type  
  Filesystem constructor/destructor.  
• vfsmount  
  A subtree in the filesystem hierarchy.  
• nameidata  
  A result of a directory lookup.  
• address space  
  Mapping between file and disk blocks.

File System Type Structure

struct file_system_type {  
    const char *name;  
    int fs_flags;  
    struct super_block *(*get_sb) (  
        struct file_system_type *, int, const char *, void *);  
    void (*kill_sb) (struct super_block *);  
    struct module *owner;  
    struct file_system_type * next;  
    struct list_head fs_supers;  
};

File System Type Generics

struct super_block * get_sb_xxx (  
    struct file_system_type *fs_type,  
    int flags, const char *dev_name, void *data,  
    int (*fill_super) (struct super_block *, void *, int))  

• get_sb_nodev ()  
• get_sb_bdev ()  
• get_sb_single ()  
• get_sb_pseudo ()
Chapter 5. File Subsystem

void kill_xxx_super (struct super_block * sb)

- kill_block_super()
- kill_anon_super()
- kill_litter_super()

VFS Mount Structure

struct vfsmount {
    struct list_head mnt_hash;

    struct vfsmount *mnt_parent; /* fs we are mounted on */
    struct dentry *mnt_mountpoint; /* dentry of mountpoint */
    struct dentry *mnt_root; /* root of the mounted tree */
    struct super_block *mnt_sb; /* pointer to superblock */
    struct list_head mnt_mounts; /* list of children, anchored here */
    struct list_head mnt_child; /* and going through their mnt_child */

    atomic_t mnt_count;
    int mnt_flags;
    int mnt_expiry_mark; /* true if marked for expiry */
    char *mnt_devname; /* Name of device e.g. /dev/dsk/hda1 */

    struct list_head mnt_list;
    struct list_head mnt_fslink; /* link in fs-specific expiry list */

    struct namespace *mnt_namespace; /* containing namespace */
};

Directory Lookup Result Structure

struct nameidata {
    struct dentry *dentry;
    struct vfsmount *mnt;
    struct qstr last;
    unsigned int flags;
    int last_type;
    unsigned depth;
    char *saved_names[MAX_NESTED_LINKS + 1];

    /* Intent data */
    union {
        struct open_intent open;
    } intent;
};

Address Space Structure

struct address_space {
    struct inode *host; /* owner: inode, block_device */

    struct radix_tree_root page_tree; /* radix tree of all pages */
    spinlock_t tree_lock; /* and spinlock protecting it */
unsigned int i_mmap_writable; /* count VM_SHARED mappings */
struct prio_tree_root i_mmap; /* tree of private and shared mappings */
struct list_head i_mmap_nonlinear; /* list VM_NONLINEAR mappings */
spinlock_t i_mmap_lock; /* protect tree, count, list */
unsigned int truncate_count; /* Cover race condition with truncate */
unsigned long nrpages; /* number of total pages */
pgoft_t writeback_index; /* writeback starts here */
struct address_space_operations *a_ops; /* methods */
unsigned long flags; /* error bits/gfp mask */
struct backing_dev_info *backing_dev_info; /* device readahead, etc */
spinlock_t private_lock; /* for use by the address_space */
struct list_head private_list; /* ditto */
struct address_space *assoc_mapping; /* ditto */
};

Address Space Operations

struct address_space_operations {
  int (*writepage) (struct page *page, struct writeback_control *wbc);
  int (*readpage) (struct file *, struct page *);
  int (*sync_page) (struct page *);

  /* Write back some dirty pages from this mapping. */
  int (*writepages) (struct address_space *, struct writeback_control *);

  /* Set a page dirty */
  int (*set_page_dirty) (struct page *page);

  int (*readpages) (struct file *filp, struct address_space *mapping,
                   struct list_head *pages, unsigned nr_pages);

  /*
   * ext3 requires that a successful prepare_write() call be followed
   * by a commit_write() call — they must be balanced
   */
  int (*prepare_write) (struct file *, struct page *, unsigned, unsigned);
  int (*commit_write) (struct file *, struct page *, unsigned, unsigned);

  /* Unfortunately this kludge is needed for FIBMAP. Don’t use it */
  sector_t (*bmap) (struct address_space *, sector_t);

  int (*invalidatepage) (struct page *, unsigned long);
  int (*releasepage) (struct page *, int);
  ssize_t (*direct_IO) (int, struct kiocb *, const struct iovec *iov,
                        loff_t offset, unsigned long nr_segs);
};

References

References

Chapter 5. File Subsystem


Chapter 6. Network Subsystem

Abstractions And Operations

Sockets

Socket System Call

\[
\text{int socket (int domain, int type, int protocol);}\
\]

Domain specifies socket protocol class:

- PF_UNIX - local communication
- PF_INET - IPv4 protocol family
- PF_INET6 - IPv6 protocol family
- PF_IPX - IPX protocol family
- PF_NETLINK - kernel communication
- PF_PACKET - raw packet communication

Type specifies socket semantics:

- SOCK_STREAM - reliable bidirectional ordered stream
- SOCK_RDM - reliable bidirectional unordered messages
- SOCK_DGRAM - unreliable bidirectional unordered messages
- SOCK_SEQPACKET - reliable bidirectional ordered messages
- SOCK_RAW - raw packets

Protocol specifies socket protocol:

- 0 - class and type determine protocol
- other - identification of supported protocol

Bind System Call

\[
\text{int bind (int sockfd, struct sockaddr \*my_addr, socklen_t addrlen);}\
\]

#define __SOCKADDR_COMMON(sa_prefix) \
    sa_family_t sa_prefix##family

struct sockaddr_in
{
    __SOCKADDR_COMMON (sin_);
    in_port_t sin_port;
    struct in_addr sin_addr;
    unsigned char sin_zero [sizeof (struct sockaddr) - 
        __SOCKADDR_COMMON_SIZE -}
Chapter 6. Network Subsystem

```c
sizeof (in_port_t) -
sizeof (struct in_addr)];

struct sockaddr_in6
{
    __SOCKADDR_COMMON (sin6_);
    in_port_t    sin6_port;
    uint32_t     sin6_flowinfo;
    struct in6_addr sin6_addr;
    uint32_t     sin6_scope_id;
};
```

Listen System Call

```c
int listen (int sockfd, int backlog);
```

Accept System Call

```c
int accept (int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

Connect System Call

```c
int connect (int sockfd,
            const struct sockaddr *serv_addr,
            socklen_t addrlen);
```

Send Family Of System Calls

```c
ssize_t send (int sockfd, const void *buf, size_t len, int flags);
ssize_t sendto (int sockfd, const void *buf, size_t len, int flags,
                const struct sockaddr *to, socklen_t tolen);
ssize_t sendmsg (int sockfd, const struct msghdr *msg, int flags);
```

```c
struct msghdr
{
    void       *msg_name;  // optional address
    socklen_t   msg_namelen; // optional address length
    struct iovec *msg_iov;  // array for scatter gather
    size_t      msg_iovlen; // array for scatter gather length
    void       *msg_control; // additional control data
    socklen_t   msg_controllen; // additional control data length
    int         msg_flags;
};
```

Recv Family Of System Calls

```c
ssize_t recv (int sockfd, void *buf, size_t len, int flags);
ssize_t recvfrom (int sockfd, void *buf, size_t len, int flags,
                  const struct sockaddr *from, socklen_t *fromlen);
ssize_t recvmsg (int sockfd, struct msghdr *msg, int flags);
```

```c
struct msghdr
{
    void       *msg_name;  // optional address
```
Chapter 6. Network Subsystem

socklen_t msg_namelen; // optional address length
struct iovec *msg_iov; // array for scatter gather
size_t msg_iovlen; // array for scatter gather length
void *msg_control; // additional control data
socklen_t msg_controllen; // additional control data length
int msg_flags;

Select And Poll System Calls

int select (int setsize,
    fd_set *readfds,
    fd_set *writefds,
    fd_set *exceptfds,
    struct timeval *timeout);

int poll (struct pollfd *ufds,
    unsigned int nfds,
    int timeout);

struct pollfd
{
    int fd;
    short events; // requested events
    short revents; // returned events
}

Socket Options System Calls

int getsockopt (int sockfd, int level,
    int optname, void *optval, socklen_t *optlen);

int setsockopt (int sockfd, int level,
    int optname, const void *optval, socklen_t optlen);

Example: Unix Sockets

Socket Address Structure

struct sockaddr_un
{
    sa_family_t sun_family; // set to AF_UNIX
    char sun_path [PATH_MAX]; // socket name
};

Socket Pair System Call

int socketpair (int domain,
    int type,
    int protocol,
    int sockets [2]);
Chapter 6. Network Subsystem

Example: Sending File Descriptor Array

```c
struct msghdr message;
int descriptors [LEN];
char buffer [CMSG_SPACE (sizeof (descriptors))];

// Prepare message header.
message.msg_control = buffer;
message.msg_controllen = sizeof (buffer);
struct cmsghdr *control = CMSG_FIRSTHDR (&message);

// Fill in protocol identifier and protocol specific type.
// Use of SOL_SOCKET instead of AF_UNIX for legacy reasons.
control->cmsg_level = SOL_SOCKET;
control->cmsg_type = SCM_RIGHTS;
control->cmsg_len = CMSG_LEN (sizeof (int) * LEN);

// Fill in protocol specific payload.
memcpy (CMSG_DATA (control), descriptors, sizeof (descriptors));

// Complete message header.
message->msg_controllen = control->cmsg_len;
```

Socket Listing

```
> netstat --unix --all (servers and established)
Proto RefCnt Flags Type State Path
unix 2 [ ACC ] STREAM LISTENING /var/run/acpid.socket
unix 2 [ ACC ] STREAM LISTENING /tmp/.font-unix/fs7100
unix 2 [ ACC ] STREAM LISTENING /tmp/.gdm_socket
unix 2 [ ACC ] STREAM LISTENING /tmp/.X11-unix/X0
unix 2 [ ACC ] STREAM LISTENING /tmp/.ICE-unix/4088
unix 2 [ ACC ] STREAM LISTENING /var/run/dbus/system_bus_socket
unix 3 [ ] STREAM CONNECTED /var/run/dbus/system_bus_socket
unix 2 [ ] DGRAM @/var/run/hal/hotplug_socket
unix 2 [ ] DGRAM @/dev
unix 2 [ ACC ] STREAM LISTENING /tmp/xmms_ceres.0
unix 3 [ ] STREAM CONNECTED /tmp/.X11-unix/X0
unix 3 [ ] STREAM CONNECTED /tmp/.ICE-unix/4088
```

Example: Linux Netlink Sockets

Netlink Families

- NETLINK_ARPD - ARP table
- NETLINK_ROUTE - routing updates and modifications of IPv4 routing table
- NETLINK_ROUTE6 - routing updates and modifications of IPv6 routing table
- NETLINK_FIREWALL - IPv4 firewall
- ...
Network Subsystem Internals

Queuing Architecture

Example: Linux SK Buff Structure

SK_Buff Structure

```c
struct sk_buff *alloc_skb (unsigned int size, int priority);
void skb_reserve (struct sk_buff *skb, unsigned int len);
int skb_headroom (const struct sk_buff *skb);
int skb_tailroom (const struct sk_buff *skb);

unsigned char *skb_put (struct sk_buff *skb, unsigned int len);
unsigned char *skb_push (struct sk_buff *skb, unsigned int len);
unsigned char *skb_pull (struct sk_buff *skb, unsigned int len);
void skb_trim (struct sk_buff *skb, unsigned int len);
```

Packet Filtering

Example: Linux Packet Filter

Network Filter Tables And Chains

The `filter` table is for normal packets:

- INPUT - chain for incoming packets
- OUTPUT - chain for outgoing packets
- FORWARD - chain for packets that pass through

The `nat` table is for packets that open new connections:

- PREROUTING
- OUTPUT
- POSTROUTING

The `mangle` table is for packets that need special modifications:

- PREROUTING
- INPUT
- OUTPUT
- FORWARD
- POSTROUTING
Chapter 6. Network Subsystem

Network Filter Actions
The basic actions:

- ACCEPT - accept packet
- DROP - discard packet
- QUEUE - forward to application filter
- RETURN - continue previous chain

Examples of the extended actions:

- CONNMARK - mark connection associated with packet
- LOG - log packet and continue packet processing
- MARK - mark packet and continue packet processing
- MASQUERADE - enable address translation for connection
- REDIRECT - deliver packet locally
- REJECT - discard packet with notification
- ROUTE - apply given routing rules to packet
- TARPIT - hold connection associated with packet

Network Filters Example: Router

```bash
> cat /etc/sysconfig/iptables
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
:INPUT_FROM_LOCAL - [0:0]
:INPUT_FROM_WORLD - [0:0]
:FORWARD_FROM_LOCAL - [0:0]
:FORWARD_FROM_WORLD - [0:0]

# Sort traffic
-A INPUT -i lo -j INPUT_FROM_LOCAL
-A INPUT -i eth0 -j INPUT_FROM_LOCAL
-A INPUT -i tun0 -j INPUT_FROM_LOCAL
-A INPUT -i tun1 -j INPUT_FROM_LOCAL
-A INPUT -j INPUT_FROM_WORLD
-A FORWARD -i lo -j FORWARD_FROM_LOCAL
-A FORWARD -i eth0 -j FORWARD_FROM_LOCAL
-A FORWARD -i tun0 -j FORWARD_FROM_LOCAL
-A FORWARD -i tun1 -j FORWARD_FROM_LOCAL
-A FORWARD -j FORWARD_FROM_WORLD

# Input from local machines
-A INPUT_FROM_LOCAL -j ACCEPT

# Input from world machines
-A INPUT_FROM_WORLD -p tcp --dport ssh -j ACCEPT
-A INPUT_FROM_WORLD -p tcp --dport http -j ACCEPT
-A INPUT_FROM_WORLD -p tcp --dport smtp -j ACCEPT
-A INPUT_FROM_WORLD -m state --state ESTABLISHED,RELATED -j ACCEPT
-A INPUT_FROM_WORLD -j REJECT

# Forward from local machines
```
Chapter 6. Network Subsystem

-A FORWARD_FROM_LOCAL -j ACCEPT

# Forward from world machines
-A FORWARD_FROM_WORLD -m state --state ESTABLISHED,RELATED -j ACCEPT
-A FORWARD_FROM_WORLD -j REJECT

COMMIT

*nat

:PREROUTING ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A PREROUTING -s 192.168.0.128/25 -p tcp --dport http -j REDIRECT --to-ports 3128
-A PREROUTING -s 192.168.0.128/25 -p tcp --dport smtp -j REDIRECT --to-ports 25
-A POSTROUTING -o ppp0 -s 192.168.0.128/25 -j MASQUERADE

COMMIT

Network Filters Example: Port Knocking

> cat /etc/sysconfig/iptables

*filter

...  

# Rules to dispatch between port knock state machine states
# INSTEP1 to INSTEP3 are (arbitrary) names of address lists
-A INPUT -m recent --name INSTEP3 --rcheck -j STEP3
-A INPUT -m recent --name INSTEP2 --rcheck -j STEP2
-A INPUT -m recent --name INSTEP1 --rcheck -j STEP1
-A INPUT -j STEP0

# Machine state STEP0
# Jump to STEP1 if first knock comes, otherwise discard and stay in STEP0
-A STEP0 -p udp --dport 1111 -m recent --name INSTEP1 --set -j REJECT
-A STEP0 -j REJECT

# Machine state STEP1
# Jump to STEP2 if second knock comes, otherwise jump to STEP0
-A STEP1 -p udp --dport 2222 -m recent --name INSTEP2 --set -j REJECT
-A STEP1 -j STEP0

# Machine state STEP2
# Jump to STEP3 if third knock comes, otherwise jump to STEP0
-A STEP2 -p udp --dport 3333 -m recent --name INSTEP3 --set -j REJECT
-A STEP2 -j STEP0

# Machine state STEP3
# Accept one SSH connection attempt, otherwise jump to STEP0
-A STEP3 -m recent --name INSTEP3 --remove
-A STEP3 -p tcp --dport ssh -j ACCEPT
-A STEP3 -j STEP0

COMMIT

Example adjusted from literature, see references.
Example: Linux Packet Scheduling

Network Schedulers Example

```c
# Root qdisc is prio with 3 bands
tc qdisc add dev ppp0 root handle 1: prio bands 3

# Band 1 qdisc is sfq and filter is ICMP & SSH & DNS & outbound HTTP
tc qdisc add dev ppp0 parent 1:1 sfq perturb 16
tc filter add dev ppp0 parent 1: protocol ip prio 1 u32 match ip protocol 1 0xff flowid 1:1
tc filter add dev ppp0 parent 1: protocol ip prio 1 u32 match ip sport 22 0xffff flowid 1:1
tc filter add dev ppp0 parent 1: protocol ip prio 1 u32 match ip dport 22 0xffff flowid 1:1
tc filter add dev ppp0 parent 1: protocol ip prio 1 u32 match ip sport 53 0xffff flowid 1:1
tc filter add dev ppp0 parent 1: protocol ip prio 1 u32 match ip dport 53 0xffff flowid 1:1
```

```
# Band 2 qdisc is sfq and filter is anything unfiltered
```

```
tc qdisc add dev ppp0 parent 1:2 sfq perturb 16
```

```
# Band 3 qdisc is tbf and filter is outbound SMTP
```

```
tc qdisc add dev ppp0 parent 1:3 tbf rate 128kbit buffer 100000 latency 100s
```

```
tc filter add dev ppp0 parent 1: protocol ip prio 1 u32 match ip dport 25 0xffff flowid 1:3
```

Network Subsystem Applications

File Systems

Example: Network File System

Basic Protocol Structures

```c
const MNTPATHLEN = 1024; /* maximum bytes in a pathname argument */
const MNTNAMLEN = 255; /* maximum bytes in a name argument */
const FHSIZE = 32; /* size in bytes of a file handle */

typedef opaque fhandle [FHSIZE];
typedef string name <MNTNAMLEN>;
typedef string dirpath <MNTPATHLEN>;

union fhstatus switch (unsigned fhs_status) {
    case 0:
        fhandle fhs_fhandle;
    default:
        void;
};
```

Mount Protocol Interface

```c
typedef struct mountbody *mountlist;
struct mountbody {
    name ml_hostname;
    dirpath ml_directory;
    mountlist ml_next;
};
```
typedef struct groupnode *groups;
struct groupnode {
    char gr_name;
    groups gr_next;
};

typedef struct exportnode *exports;
struct exportnode {
    char ex_dir;
    groups ex_groups;
    exports ex_next;
};

program MOUNTPROG {
    version MOUNTVERS {
        MOUNTPROC_NULL (void) = 0;
        fhstatus MOUNTPROC_MNT (dirpath) = 1;
        mountlist MOUNTPROC_DUMP (void) = 2;
        void MOUNTPROC_UMNT (dirpath) = 3;
        void MOUNTPROC_UMNTALL (void) = 4;
        exports MOUNTPROC_EXPORT (void) = 5;
        exports MOUNTPROC_EXPORTALL (void) = 6;
    } = 1;
    } = 100005;

NFS Protocol LOOKUP Interface Function

program NFS_PROGRAM {
    version NFS_VERSION {
        NFSPROC_LOOKUP (diropargs) = 4;
        } = 2;
    } = 100003;

struct diropargs {
    nfs_fh dir;  /* directory file handle */
    filename name;  /* file name */
};

union diropres switch (nfsstat status) {
    case NFS_OK:
        diropokres diropres;
    default:
        void;
};

struct diropokres {
    nfs_fh file;
    fattr attributes;
};

struct fattr {
    ftype type;  /* file type */
    unsigned mode;  /* protection mode bits */
    unsigned nlink;  /* number of hard links */
    unsigned uid;  /* owner user id */
    unsigned gid;  /* owner group id */
    unsigned size;  /* file size in bytes */
    unsigned blocksize;  /* preferred block size */
    unsigned rdev;  /* special device number */
    unsigned blocks;  /* used size in kilobytes */
    unsigned fsid;  /* device number */
}
unsigned fileid; /* inode number */
nfstime atime; /* time of last access */
nfstime mtime; /* time of last modification */
nfstime ctime; /* time of last change */
};

struct nfs_fh {
    opaque data [NFS_FHSIZE];
};

enum nfsstat {
    NFS_OK=0, /* No error */
    NFSERR_PERM=1, /* Not owner */
    NFSERR_NOENT=2, /* No such file or directory */
    NFSERR_IO=5, /* I/O error */
    NFSERR_NXIO=6, /* No such device or address */
    NFSERR_ACCES=13, /* Permission denied */
    NFSERR_EXIST=17, /* File exists */
    NFSERR_NODEV=19, /* No such device */
    NFSERR_NOTDIR=20, /* Not a directory*/
    NFSERR_ISDIR=21, /* Is a directory */
    NFSERR_FBIG=27, /* File too large */
    NFSERR_NOSPC=28, /* No space left on device */
    NFSERR_ROFS=30, /* Read-only file system */
    NFSERR_NAME_TOO_LONG=63, /* File name too long */
    NFSERR_NOTEMPTY=66, /* Directory not empty */
    NFSERR_DQUOT=69, /* Disc quota exceeded */
    NFSERR_TIMEOUT=70, /* Stale NFS file handle */
    NFSERR_WFLUSH=99 /* Write cache flushed */
};

NFS Protocol READ Interface Function

program NFS_PROGRAM {
    version NFS_VERSION {
        ...
        readres NFSPROC_READ (readargs) = 6;
        ...
    } = 2;
    } = 100003;

struct readargs {
    nfs_fh file; /* handle for file */
    unsigned offset; /* byte offset in file */
    unsigned count; /* immediate read count */
    unsigned totalcount; /* total read count (from this offset)*/
};

union readres switch (nfsstat status) {
    case NFS_OK:
        readokres reply;
    default:
        void;
};

struct readokres {
    fattr attributes; /* attributes */
    opaque data <NFS_MAXDATA>;
};

struct fattr {
    ftype type; /* file type */
    unsigned mode; /* protection mode bits */
    unsigned nlink; /* number of hard links */
}
unsigned uid; /* owner user id */
unsigned gid; /* owner group id */
unsigned size; /* file size in bytes */
unsigned blocksize; /* preferred block size */
unsigned rdev; /* special device number */
unsigned blocks; /* used size in kilobytes */
unsigned fsid; /* device number */
unsigned fileid; /* inode number */
fstime atime; /* time of last access */
fstime mtime; /* time of last modification */
fstime ctime; /* time of last change */
};

struct nfs_fh {
opaque data [NFS_FHSIZE];
};

eenum nfsstat {
NFS_OK=0, /* No error */
NFSERR_PERM=1, /* Not owner */
NFSERR_NOENT=2, /* No such file or directory */
NFSERR_IO=5, /* I/O error */
NFSERR_NXIO=6, /* No such device or address */
NFSERR_ACCES=13, /* Permission denied */
NFSERR_EXIST=17, /* File exists */
NFSERR_NODEV=19, /* No such device */
NFSERR_NODIR=20, /* Not a directory*/
NFSERR_ISDIR=21, /* Is a directory */
NFSERR_FBIG=27, /* File too large */
NFSERR_NOSPC=28, /* No space left on device */
NFSERR_ROFS=30, /* Read-only file system */
NFSERR_NAME_TOO_LONG=63, /* File name too long */
NFSERR_NOTEMPTY=66, /* Directory not empty */
NFSERR_DQUOT=69, /* Disc quota exceeded */
NFSERR_STALE=70, /* Stale NFS file handle */
NFSERR_WFLUSH=99 /* Write cache flushed */
};
Chapter 7. Security Subsystem

Authentication

Linux PAM Example

PAM Functions

- Account management
- Authentication management
- Password management
- Session management

PAM Configuration Example

```bash
> cat /etc/pam.d/login
auth required pam_securety.so
auth required pam_stack.so service=system-auth
auth required pam_nologin.so
account required pam_stack.so service=system-auth
password required pam_stack.so service=system-auth
session required pam_stack.so service=system-auth
session optional pam_console.so
```

PAM Usage Example

```c
#include <security/pam_appl.h>
#include <security/pam_misc.h>

static struct pam_conv conv = { misc_conv, NULL };

int main(int argc, char *argv[])
{
    pam_handle_t *pamh = NULL;
    char *user;
    int retval;

    // ...

    retval = pam_start ("check_user", user, &conv, &pamh);
    if (retval == PAM_SUCCESS)
        retval = pam_authenticate (pamh, 0); // Is user really himself ?
    if (retval == PAM_SUCCESS)
        retval = pam_acct_mgmt (pamh, 0); // Is user account valid ?
    if (retval == PAM_SUCCESS)
        // ...

    pam_end (pamh, retval);
}
```
Chapter 7. Security Subsystem

Authorization

Example: Security Enhanced Linux

SELinux File Contexts

```bash
> ls -Z /
  system_u:object_r:bin_t:s0 bin
  system_u:object_r:boot_t:s0 boot
  system_u:object_r:device_t:s0 dev
  system_u:object_r:etc_t:s0 etc
  system_u:object_r:home_root_t:s0 home
  system_u:object_r:lib_t:s0 lib
  system_u:object_r:lib_t:s0 lib64
  system_u:object_r:mnt_t:s0 media
  system_u:object_r:mnt_t:s0 mnt
  system_u:object_r:usr_t:s0 opt
  system_u:object_r:proc_t:s0 proc
  system_u:object_r:admin_home_t:s0 root
  system_u:object_r:var_run_t:s0 run
  system_u:object_r:bin_t:s0 sbin
  system_u:object_r:var_t:s0 srv
  system_u:object_r:sysfs_t:s0 sys

... > semanage fcontext -l
SELinux fcontext type Context
/ directory system_u:object_r:root_t:s0
/.* all files system_u:object_r:default_t:s0
/bin all files system_u:object_r:bin_t:s0
/bin/* all files system_u:object_r:bin_t:s0
/bin/bash regular file system_u:object_r:shell_exec_t:s0
/bin/dmesg regular file system_u:object_r:dmesg_exec_t:s0
/bin/ip regular file system_u:object_r:ifconfig_exec_t:s0

... /dev directory system_u:object_r:device_t:s0
/dev/* character device system_u:object_r:mouse_device_t:s0
/dev/[0-9].* character device system_u:object_r:usb_device_t:s0
/dev/[shmxv]d[^/]* block device system_u:object_r:fixed_disk_device_t:s0

... /home directory system_u:object_r:home_root_t:s0
/home/* directory unconfined_u:object_r:user_home_dir_t:s0
/home/*+ all files unconfined_u:object_r:httpd_user_content_t:s0

...>
```

SELinux Process Contexts

```bash
> ps -Z
LABEL PID TTY TIME CMD
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 4891 pts/0 00:00:00 ps
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 5124 pts/0 00:00:00 bash
> id -z
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```
SELinux Enforcement Rules

> semanage module -l
Module Name    Priority Language
abrt          100 pp
accountsd     100 pp
acct          100 pp
afs           100 pp
aiccu         100 pp
aide          100 pp
ajaxterm      100 pp
alsa          100 pp
amanda        100 pp
...

> sesearch -A -t sshd_key_t -p write
allow ssh_keygen_t sshd_key_t:file { append create getattr ioctl link lock open read rename setattr unlink write };
...
allow files_unconfined_type file_type:file { append audit_access create execute execute_no_trans getattr ioctl link lock map mounton open quotaon read relabelfrom relabelto rename setattr swapon unlink write };
...
allow ftpd_t non_security_file_type:file { append create getattr ioctl link lock open read rename setattr unlink write }; [ ftpd_full_access ]:True
allow kernel_t non_security_file_type:file { append create getattr ioctl link lock open read rename setattr unlink write };
...
allow sysadm_t non_security_file_type:file { append create getattr ioctl link lock open read rename setattr unlink write };

SELinux Booleans

> getsebool -a
antivirus_can_scan_system --> off
antivirus_use_jit --> off
...
daeemons_dump_core --> off
daeemons_enable_cluster_mode --> off
daeemons_use_tcp_wrapper --> off
daeemons_use_tty --> off
...
ftpd_anon_write --> off
ftpd_full_access --> off
ftpd_use_nfs --> off
...
git.cgi_enable_homedirs --> off
git.cgi_use_nfs --> off
...
httpd_anon_write --> off
httpd_builtin_scripting --> on
httpd_can_check_spam --> off
httpd_can_connect_ftp --> off
httpd_can_network_connect --> off
httpd_can_network_memcache --> off
httpd_can_sendmail --> off
httpd_enable_cgi --> on
httpd_enable_homedirs --> off
httpd_use_nfs --> off
...

Chapter 7. Security Subsystem
Chapter 7. Security Subsystem

SELinux Audit Log

> tail /var/log/audit/audit.log
type=AVC msg=audit(1515657259.550:620585): avc: denied { open } for pid=8358 comm="sudo"
...  
> audit2allow < /var/log/audit/audit.log
#============= nagios_t ==============
allow nagios_t initrc_var_run_t:file open;
...
> ls -Z /run/utmp
system_u:object_r:initrc_var_run_t:s0 /run/utmp

SELinux Policy Sources

policy_module(ssh, 2.4.2)
gen_tunable(allow_ssh_keysign, false)
gen_tunable(ssh_sysadm_login, false)

attribute ssh_server;
attribute ssh_agent_type;

type ssh_t;
type ssh_exec_t;
type ssh_home_t;
type sshd_exec_t;
...

allow ssh_t self:capability { setuid setgid ... };
allow ssh_t self:tcp_socket create_stream_socket_perms;
allow ssh_t self:unix_dgram_socket { create_socket_perms sendto };
allow ssh_t self:unix_stream_socket { create_stream_socket_perms connectto };
...

allow ssh_t sshd_key_t:file read_file_perms;
allow ssh_t sshd_tmp_t:dir manage_dir_perms;
allow ssh_t sshd_tmp_t:file manage_file_perms;
...

tunable_policy ('allow_ssh_keysign',
    domain_auto_trans (ssh_t, ssh_keysign_exec_t, ssh_keysign_t)
    allow ssh_keysign_t ssh_t:fd use;
    allow ssh_keysign_t ssh_t:process sigchld;
    allow ssh_keysign_t ssh_t:fifo_file rw_file_perms;
')
...