Operating System - P57

现代操作系统 (modern operating system):

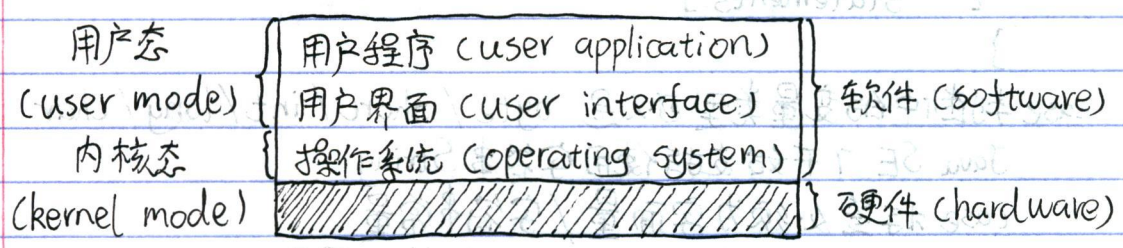
CPU运行模式 (CPU execution mode):

用户态 (user mode, protected mode)

内核态 (kernel mode, privileged mode)

内存保护 (memory protection)

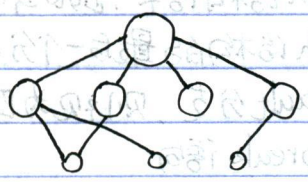
时钟中断 (timer interrupt)



操作系统类型 (operating system zoo)

大型机 main frame	服务器 server	多处理器 multiprocessor	个人计算机 personal computer	掌上计算机 handheld computer
嵌入式 embedded	传感器节点 sensor node	实时 real-time	智能卡 smart card	

单体系统 monolithic system



主程序 (main procedure)

服务过程 (service procedure)

实用过程 (utility procedure)

层次式系统 layered system

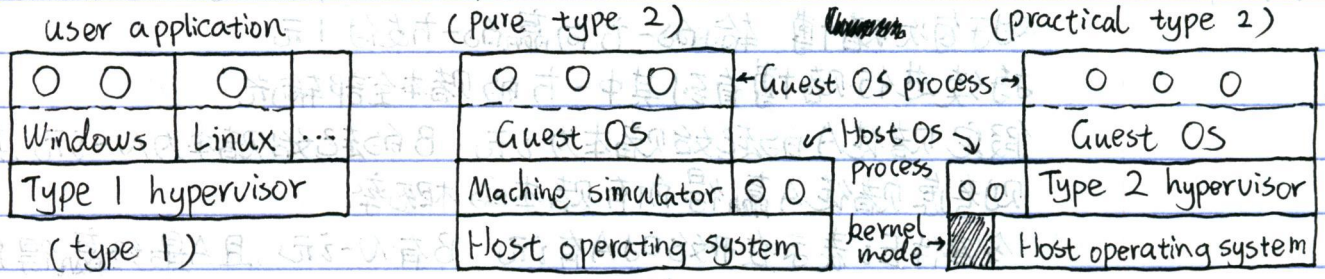
5	操作员	the operator
4	用户程序	user program
3	I/O管理	Input/Output management
2	操作员进程通信	operator-process communication
1	存储器和磁鼓管理	memory and drum management
0	处理器分配和多道程序设计	processor allocation and multiprogramming

微内核 (microkernel)

客户-服务器模式 (client-sever model)

Operating System - P58

虚拟机 (virtual machine)



段地址模式 (segmented addressing mode), 用于 Intel x86 IA-32 architecture

all addresses are modified by a segment register

CS register: all instruction fetches

SS register: all stack references

DS (data segment), ES (extra segment), FS/GS registers

: specified by the user for data references

special instructions (moving) used dedicated references to registers

operation of these segment registers different in 16-bit and 32-bit

new processors boot in 16-bit mode for backward compatibility

segmented mode originated in the x86 architecture

as a way for the 16-bit 8086 processor

to implement 20-bit (1MB) address space

addressing in x86 16-bit non-protected mode

compatible with early 8086/8088/80286 16-bit system

each memory reference is specified as an offset to a segment register

value of segment register is address of the start of segment

注意, 需要除以16, 即 *0x10

offset is 16-bit (64KB)

full address (20-bit) = (segment register) x 0x10 + (offset address)

注意, full address can be accessed in multiple ways

如 0x10 x 0x1A30 + 0x2F3C = 0x1D23C

0x10 x 0x1030 + 0xCF3C = 0x1D23C

注意, segment only can start at address are multiple of 16

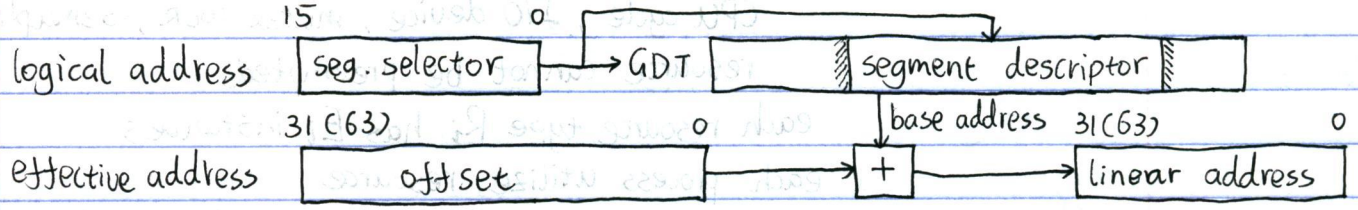
Operating System - P59

Protected mode 32-bit segmented addressing
- system enables protected mode by setting bit 0 to Control Register 0
- address of Global Descriptor Table (GDT) set in GDT register
- [redacted] segment register contains an offset into the GDT
- contents of GDT entry determine segment location in memory
- GDT entry has the segment offset in physical memory and segment length

Linux (most IA-32 OS) use Basic Flat Segmentation mode
- all GDT entries map to segments starting at 0 and mapping entire address space
- paging is used to seperate address spaces of multiple processors and OS

more sophisticated use of GDT : isolation of memory spaces without paging

邏輯地址 logical address specified by a segment register and offset
- translated to linear address (線性地址) by using GDT segment descriptor



segment descriptor format (64-bit) [redacted] as each entry in GDT

	31–24	24	22	21	20	19–16	15	14–13	12	11–8	7–0
4	Base 31:24	G	D/B	L	AVL	Seg. Limit 19:16	P	DPL	S	Type	Base 23:16

	31–16	15–0
0	Base Address 15:00	Segment Limit 15:00

L – 64-bit code mode (IA-32e)
BASE – segment base address
DPL – descriptor privilege level
LIMIT – segment limit
S – descriptor type (0: system, 1: code/data)
AVL – available for system software
D/B – default operation size (0: 16-bit, 1: 32-bit)
G – granularity
P – segment present
TYPE – segment type

type field	11	10	9	8
	0: data, 1: code	E: expand-down, C: conforming	W: 0: read-only / 1: read&write, R: 0: execute-only / 1: execute&read	A: accessed

Operating System - P60

段寄存器格式 (segment register format)

each 16-bit segment register contains a segment selector

15 ... 3	2	1 ... 0
Index	TI	RPL

Index : (>0), index of GDT segment descriptor to use
Index * 8 (0x08) = offset in GDT
Table Indicator (TI) : 0 : use GDT , 1 : use Local Descriptor Table (LDT)
Requested Privilege Level (RPL) : privilege level requesting to access segment
RPL field of CS contains the Current Privilege Level (CPL)
data access uses maximum of RPL and CPL

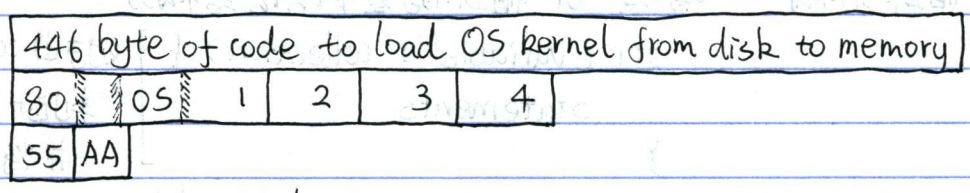
BIOS (Basic Input/Output System), typical x86 system execute on initial power-up
built into the system in read-only or non-volatile rewritable memory
initialize the system hardware
provide basic services, eg reading/writing the disk
read MBR (Master Boot Record) after initializing the hardware
transfer control to the code found in MBR
MBR : sector 0 on the disk

主引导记录 (Master Boot Record, MBR),
contain boot loader and partition table for booting from hard disk
partition table allows the disk to be divided into partitions
may contain different file systems, swap areas, multiple OSs

Byte	content	size	
0 - 445	Code	446 byte	512 byte
446 - 461	Partition table entry 1	16 byte	
462 - 477	Partition table entry 2	16 byte	
478 - 493	Partition table entry 3	16 byte	
494 - 509	Partition table entry 4	16 byte	
510	0x55	1 byte	
511	0xAA	1 byte	

Operating System - P61

Linux MBR format



simple MBR may contain one single partition
1st byte of first partition contains 0x80 to indicate bootable
5th byte contains the OS type
other bytes of partition table entry are OS dependent
Linux 0.01 booted from floppy dist (软盘)
entire sector 0 (512 byte) available for code (no MBR)

Start up memory addressing
system starts up in 8086 real mode for compatibility reason
allow addressing of only first 1MB of memory
hardware provides a single line (A20)
toggle (切换) between 1MB (20-bit) addressing
and full address space (32-bit : IA-32, 64-bit : X86-64 processor)
boot.s (Linux 0.01) sets A20 after some initial setup
also set initial GDT
enable protected mode with segments
for the first 8MB of memory before starting the kernel

after the MBR code load the kernel, control transfered to head.s (Linux 0.01)
setup the GDT to use for the running system (8MB total memory)
Linux use flat segment model (code/data segment map to all of memory)
also use paging
after paging code, execute actual Linux kernel C code at entry point "main"

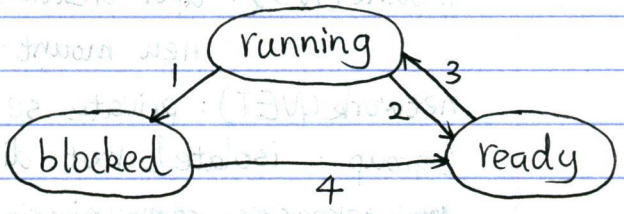
process creation
1. system initialization
2. execution of process creation system call by running process
user request / initiation of batch job (批处理任务)

termination
1. self-termination : return exit code accessible by parent
2. killed by another process : permission required (typically parent)
3. power off / system level

Operating System - P62

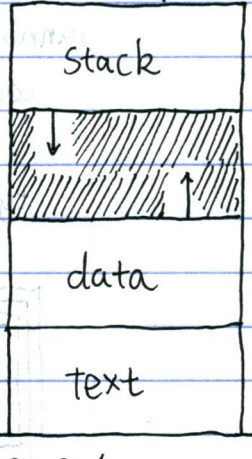
process state
- running (运行态) : actually using the CPU at that instant
- ready (就绪态) : runnable, temporarily stopped to let other process run
- blocked (阻塞态) : unable to run until some external event happens

1. block for input
2. scheduler picks another process
3. scheduler picks this process
4. input becomes available



进程表项 (process table entry)

process management	memory management
register (寄存器, Reg)	pointer (指针) : address space
program counter (程序计数器, PC)	stack segment info (堆栈段)
program status word (程序状态字, PSW)	data segment info (数据段)
stack pointer (堆栈指针)	text segment info (正文段)
process state (进程状态)	
priority (优先级)	
scheduler parameter (调度参数)	
process ID (进程ID)	
parent process (父进程)	
process group (进程组)	
signal (信号)	
time when process started	
CPU time used	
children's CPU time	
time of next alarm	



file management	
root directory	(根目录)
working directory	(工作目录)
file descriptor	(文件描述符)
user ID	(用户ID)
group ID	(组ID)

多道程序设计分析 (analysis of multiprogramming)

p = fraction of time that typical process spends blocked (waiting for event, such as I/O)

p^n = probability that all processes are waiting for event at any given time (for n processes)

CPU utilization = $1 - p^n$: percentage of time that CPU is actively in use

Operating System - P63

hardware
- when an interrupt occurs: 当中断发生时
- stack program counter, etc. 程序计数器等压入堆栈
- load new program counter from interrupt vector 从中断向量装入新程序计数器

assembly language procedure
- save registers 保存寄存器值
- set up new stack 设置新的堆栈

C
- interrupt service run (typically reads and buffers input) 中断服务例程运行
- scheduler decides which process is to run next 决定下一个将运行的进程
- return to the assembly code 返回至汇编代码

assembly language procedure
- start up new current process 开始运行新的当前进程

线程 (thread)

multithreaded process

code	register1	stack1	~
data	register2	stack2	~
file	register3	stack3	~

responsiveness : divided tasks to run concurrently
resource sharing : threads share the process' instruction code and context
economy : threads carry less state information than processes
scalability : multithreaded processes can occupy multiple CPUs

issue in multicore system
- dividing activities : what subtasks can be run in parallel
- balance : are all threads performing an equal amount of work
- data spliting : which thread will operate on what data
- data dependency : does a thread depend on data operated on by another
- testing and debugging : is the program bug-free because it ran fine once

用户线程 (user-level thread), created by using threading API
managed by the thread runtime system (运行时系统)

内核线程 (kernel-level thread), created by the kernel at user process request
- associated with a process
- each process starts with one kernel thread
- OS schedules kernel threads much like processes
- has no knowledge about user-level thread

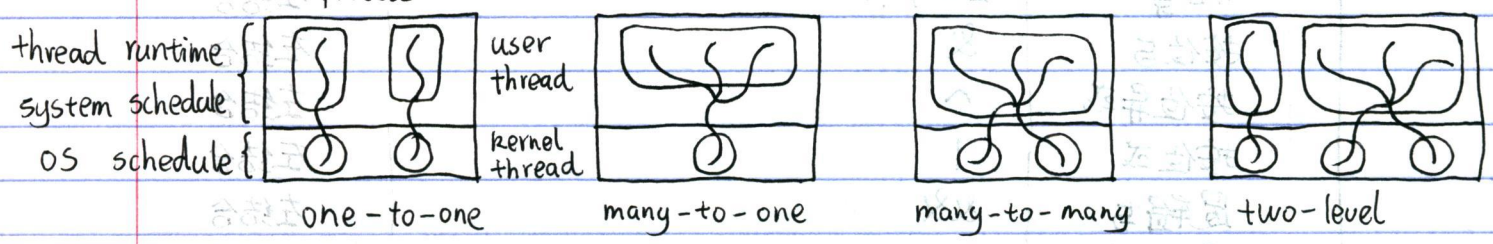
Operating System - P64

mapping thread runtime system schedules user-level threads (lightweight process) on one or more "virtual processor"

vitural processor: kernel thread in reality

OS schedules kernel threads to run on physical processors

mapping is all about virtual processors an application has different levels of concurrency can be achieved by different forms of mapping between user- and kernel-level thread



cancellation (取消) task of terminating a thread before it has completed

asychronous cancellation (异步取消)

one thread immediately terminates the target thread

may result in corrupted data or lost system resources

deferred cancellation (延迟取消)

target thread periodically checks whether it should terminate

thread terminate when they can do so safely (cancellation points)

signal (信号) notify a process about a particular event

can be synchronous (同步) or asynchronous

signal handler (信号处理): used to process signals

can be default one or user-defined

who gets the signal in a multithreaded process

the thread to which the signal applies

every thread in the process

certain threads in the process

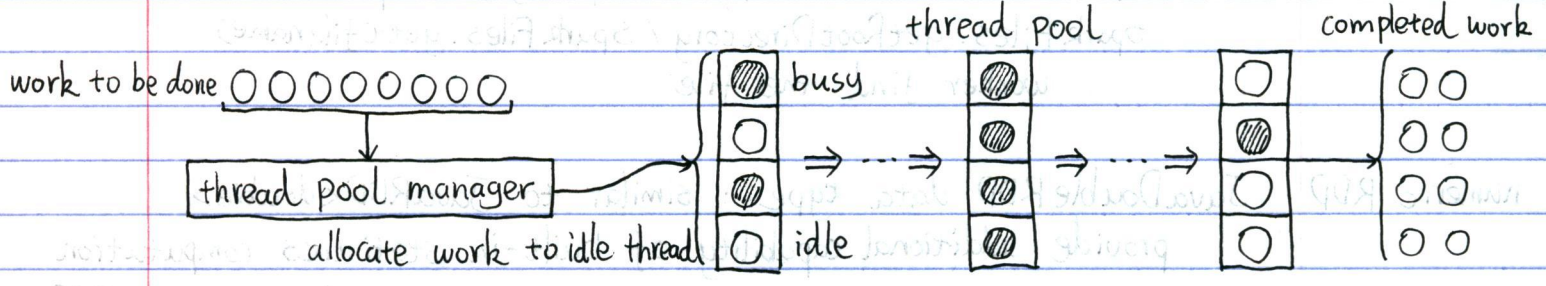
specific thread that receives all signals for the process

thread specific data: allow each thread to have its own copy of data

useful when have no control over the thread creation process

Operating System - P65

thread pool (线程池) : a number of threads in a pool awaiting work
- can use many-to-one or many-to-many mapping
- usually slightly faster to service a request
  - with an existing thread than create a new thread
- allow the number of threads in the application
  - to be bound to the size of the pool



thread library (线程库) : provide the programmer API for creating and managing threads
- library entirely in user space
- kernel-level library supported by OS

POSIX : only a standard define an API for thread creation and management
- Native POSIX Thread Library (NPTL)
  - an implementation of the specification in most Linux system
  - employs the one-to-one model
  - a kernel-level library

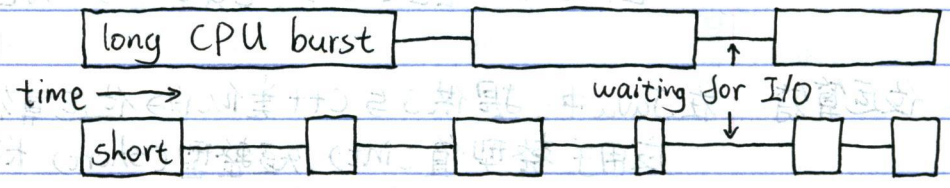
pthread_create	:	create a thread
pthread_join	:	wait for a thread to finish
pthread_cancel	:	terminate another thread
pthread_detach	:	set thread to release resources
pthread_exit	:	exit the thread (same as return)

job queue (工作队列) : all jobs in system, multiple sub-queues
- ready queue : jobs in memory, running or ready to use CPU
- blocked jobs : in memory, waiting for I/O or other events
- waiting to run : scheduled for a particular time
  - waiting for a resource (memory or hardware device)

Operating System - P66

type of scheduling
- batch processing system (批处理系统): throughput and utilization
- network server (网络服务器): response time
- workstation PC (工作站 PC): response time
- personal computer (个人电脑): response time
- mobile device (移动设备): response time and power usage
- real time, hard/soft (实时系统): time-critical scheduling, interrupt response

behavior



goal
- all
  - fairness (公平): giving each process a fair share of the CPU
  - policy enforcement (策略强制): seeing that stated policy is carried out
  - balance (平衡): keeping all parts of the system busy
- batch
  - throughput (吞吐量): maximize jobs per hour
  - turnaround time (周转时间): minimize time between submission and termination
  - CPU utilization (CPU利用率): keep the CPU busy all the time
- interactive
  - response time (响应时间): respond to request quickly
  - proportionality (均衡): meet users' expectation
- real-time
  - meeting deadline (满足截止时间): avoid losing data
  - predictability (可预测性): avoid quality degradation in multimedia system

preemptive scheduling (抢占式调度)
- continue until block or voluntarily release the CPU
- useful primarily for user-level thread

non-preemptive scheduling (非抢占式调度)
- continue until block or time-slice finish due to timer interrupt
- used for kernel-level thread

FCFS (first-come first-served scheduling: 先来先服务调度)
- non-preemptive and simple to implement
- ready queue ordered by time threads become ready to run
- thread run until blocked or complete, may not use system resource efficiently

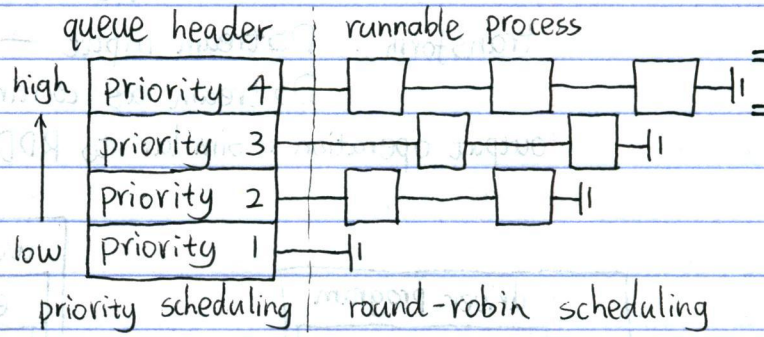
Operating System - P67

SJF (shortest job first, 最短作业优先)
- non-preemptive
- assume knowledge of runtime in advance
- scheduler always pick shortest job in ready queue to execute next
- provably optimal for minimizing average turnaround time

scheduling for interactive system
- preemptive scheduling
- response time is an important consideration

round-robin scheduling (轮转调度)
- scheduler keeps ciralar list (环形列表) of ready process
- scheduler runs periodically or when a process blocks
- time quantum (时间片) : scheduling period
- scheduler gives the CPU to the next process in the ciralar list
- switching processes incurs certain amount of fixed overhead
  - smaller quantum : better response time, reduce CPU efficiency
  - larger quantum : decrease the total amount of process switch overhead

priority scheduling (优先级调度)
- in typical system, all processes may not be of equal importance
- one way to manage priority
  - seperate round-robin list for each priority level
- scheduler works with list at highest level, has job ready to run



absolute (绝对的): only processes in highest priority ready queue get any CPU time assigned by scheduler

priority aging : gradually increase the priority of process has not any CPU time to prevent starvation (饥饿)

proportional (比例的): all level of priority will get some CPU time higher level assigned proportionally higher amount of CPU time