Operating System - P79

optimal page replacement algorithm (最优页面置换算法)
- replace page not be used for longest period of time
- not really a feasible algorithm
- useful for how well other algorithm perform

FIFO (First-In First-out, 先进先出)
- always replace page has been in memory for longest time

Belady's Anomaly, increasing number of frames increase number of page fault
- increasing memory not always increase performance

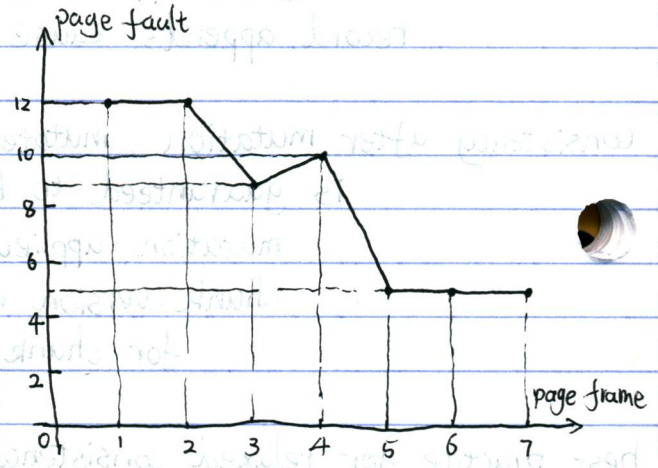
for 1,2,3,4,1,2,5,1,2,3,4,5, in FIFO

	1	2	3	4	1	2	5	1	2	3	4	5
	1	1	1	4	4	4	5	5	5	5	5	5
		2	2	2	1	1	1	1	1	3	3	3
			3	3	3	2	2	2	2	2	4	4

↑ 9 page faults

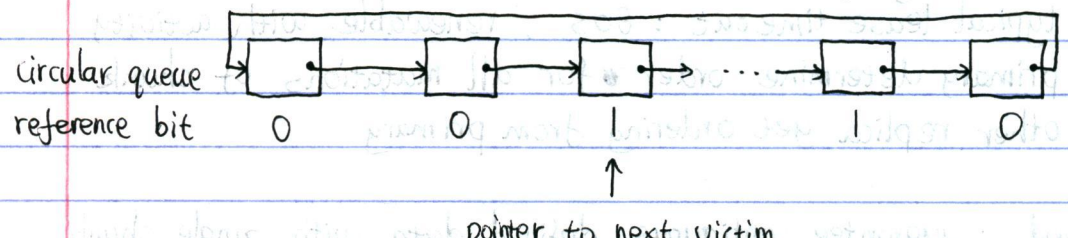
	1	2	3	4	1	2	5	1	2	3	4	5
	1	1	1	1	1	1	5	5	5	5	4	4
		2	2	2	2	2	2	1	1	1	1	5
			3	3	3	3	3	3	2	2	2	2
				4	4	4	4	4	4	3	3	3

↑ 10 page faults



second chance algorithm (第二次机会算法), same as FIFO but use referenced bit
- if referenced bit == 1 when selected using FIFO
  - set referenced bit to 0
  - keep page in memory, move to end of queue
  - move over to next possibility according to FIFO

clock algorithm (时钟页面置换算法), second-chance using circular queue



Operating System - P80

LRU (least recently used page replacement algorithm, 最近最少使用页面置换算法)

time stamp (counter) implementation (not really possible)

system maintain counter incremented after every instruction

effectively time value

every page table entry has field to store the time at last reference

every time page referenced, current counter stored in page table entry

when need to replace, choose the smallest counter value (oldest time stamp)

require search to find least recently used page

most system not provide hardware support true LRU implementation

NFU (not frequently used algorithm, 最不常用算法), LRU approximation algorithm

use reference bit (accessed bit) provided in hardware

maintain software counter for each page, initially set to 0

at each timer interrupt, add page reference bit to its counter

then clear reference bit

on page fault, choose the lowest counter value

NRU (not recently used algorithm, 最近未使用页面置换算法)

OS periodically clear reference (accessed) bit

victim ↓

- class 0 : reference == 0, modified == 0 (swap out not necessary)
- class 1 : reference == 0, modified == 1
- class 2 : reference == 1, modified == 0 (swap out not necessary)
- class 3 : reference == 1, modified == 1

aging algorithm(老化算法), maintain one byte (8-bit) for each page

at timer interrupt, for each page

discard low-order bit of byte

insert reference bit for the page at the left-most position

(byte >> 1) | (reference bit << 7)

then clear reference bit

page with lowest value byte is the least referenced one

not know which page accessed more recently between two interrupts

not know which page accessed before 8 ticks ago

Operating System - P81

demand paging(请求调页), process start with no page in memory
page brought into memory as and when needed
program tend to have locality of reference
program heavily refer to set of pages before moving on to another set

frame allocation, minimum number of frames should be always given to process
number depend on the computer architecture
enough frames to hold different pages
any single instruction can reference

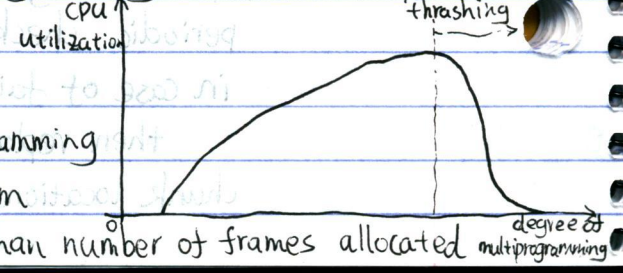
fixed allocation, equal allocation: divide available frames equally amongst processes
proportional allocation: allocate according to the size of process
m: total number of frames
$S = \sum_{i=1}^{m} S_i$: S_i size of process P_i
allocation for P_i : $a_i = \frac{S_i}{S} \cdot m$

priority allocation, proportional allocation scheme using priority rather than size
if process P_i generate page fault,
either select one of P_i's frames
or select one from process with lower priority number

global replacement(全局置换), select replacement frame from set of all frames
one process can take frame from another
process cannot control its page-fault rate

local replacement(局部置换), select from only process own set of allocated frames
not make less used frame available to other process

thrashing(颠簸), state in which most processes busy waiting for pages swapped in or out
high page fault lead to
low CPU utilization
OS think increase degree of multiprogramming
another process added to the system
when size of locality much larger than number of frames allocated



Operating System - P82

working-set model: w(k,t): page used by process in last k references at time t
easier to work with "page used in last Δ time units
process cause thrashing when entire working set not in memory
reduce page fault by prepaging working set
approximate with interval timer and reference bit
need to maintain timer tick count of last use for each page
reference bit cleared on every timer tick (interrupt)
bet set by hardware every time page used
on page fault, examine reference bit and timer tick count
find page not used in last Δ ticks

page replacement algorithm	comment
optimal (最优)	not implementable, but useful as benchmark
NRU (not recently used, 最近未使用)	very crude approximation of LRU
FIFO (fist-in, first-out, 先进先出)	might throw out important page
Second chance (第二次机会)	big improvement over FIFO
Clock (时钟)	realistic
LRU (least recently used, 最近最少使用)	excellent, but difficult to implement exactly
NFU (not frequently used, 最不经常使用)	fairly crude approximation to LRU
aging (老化)	efficient algorithm approximate LRU well
Working-set (工作集)	expensive to implement
WSClock (工作集时钟)	good efficient algorithm

counting algorithm: other page replacement algorithm
keep count of number of references made to each page

LFU algorithm: least frequently used
replace page with smallest count

MFU algorithm: most frequently used
replace page with largest count
argument: page with smallest count
probably just brought in and yet to be used

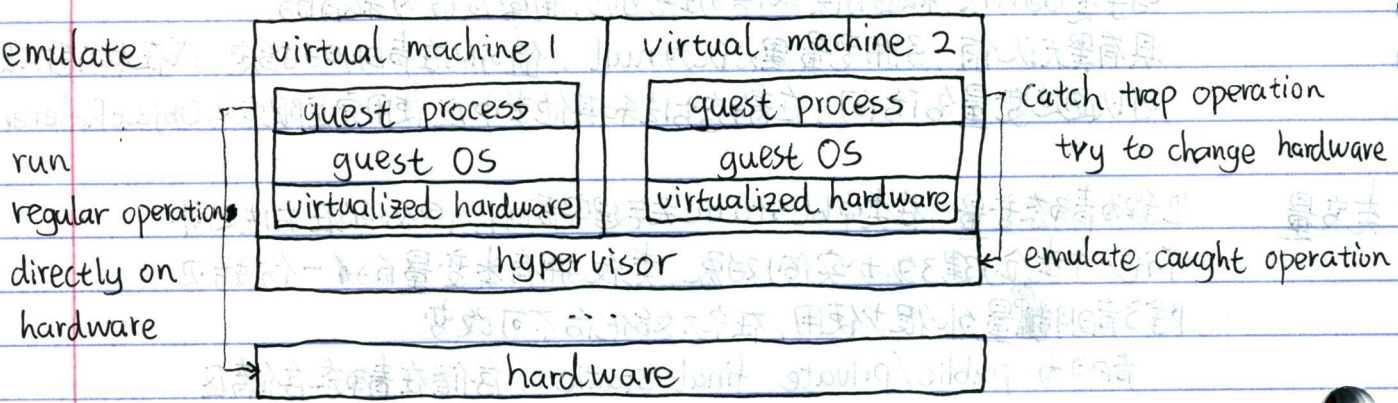
Operating System - P83

Virtualization (仮想化) : single computer host multiple virtual machines (computers)
create illusion of multiple machines on same physical hardware
virtual machine monitor (VMM) : also known as hypervisor
manage each virtual machine
host : OS runs the hypervisor, or hypervisor itself
guest : OS runs in the virtual machine

isolation : failure of one guest doesn't bring down entire system
cost : fewer physical machine mean less money to maintain hardware
less pay for running cost (electricity)
migration : moving system equivalent to
moving memory and disk images (file) in software
legacy support : run legacy application without need for legacy hardware
software development : [redacted] in a single machine
test software written for different operating system

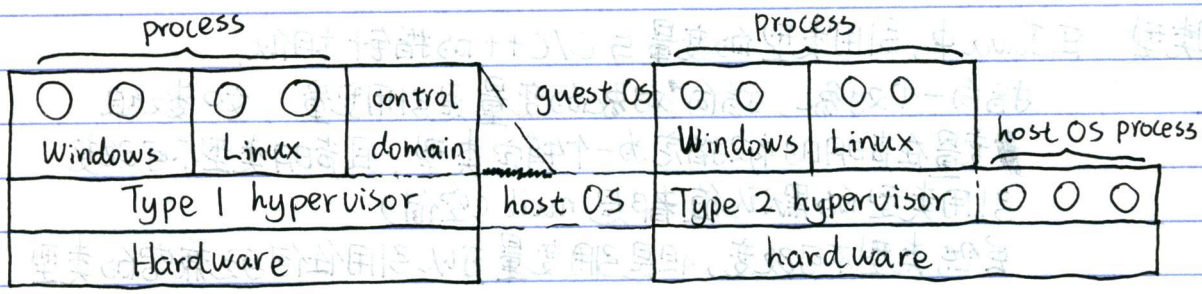
requirement : safety : hypervisor should have full control of virtualized resource
fidelity : behavior of program on virtual machine
should be identical to running on bare hardware
efficiency : much of code in virtual machine
should run without intervention by hypervisor

trap and emulate



instruction : sensitive : instruction work differently in kernel mode / in user mode
privileged : instruction only allowed in kernel mode
executing privileged instruction in user mode result in exception

Operating System - P84

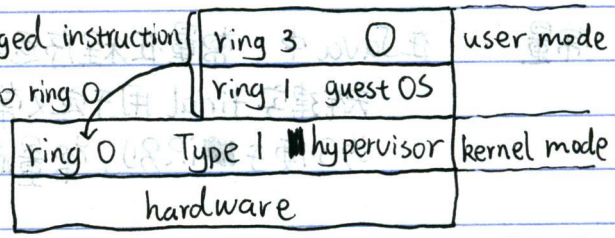


Virtualizing sensitive instruction

privileged instruction in user mode | trap to ring 0

from ring 1 : emulate instruction

from ring 3 : transfer to exception handler in guest OS



full virtualization with binary translation

hardware run sensitive instruction in guest OS process as instruction should be, since issued from ring 3

sensitive instruction executed in guest OS should be run as if issued in kernel mode

ring 1 is not kernel mode : hardware not do this by default

binary translation : sensitive instruction in kernel mode converted to call into hypervisor code

hypervisor code emulate the instruction as if executed in kernel mode

converted code fragments are cached

para-virtualization, guest OS modified : make system call to hypervisor (hypercall) when privileged operation performed

guest OS aware running in virtual machine

poor portability, but easier than full virtualization

less overhead than emulating privileged instruction

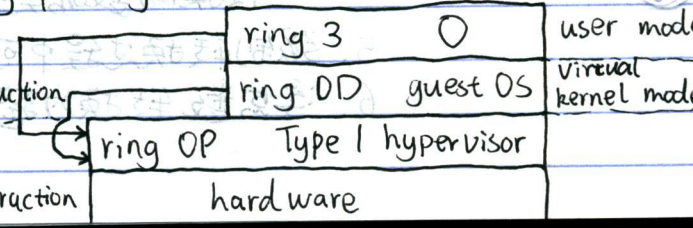
hardware assisted virtualization

program in guest OS run

sensitive / privileged instruction

guest OS run

sensitive / privileged instruction



Operating System - P85

Type 2 hypervisor run as user process

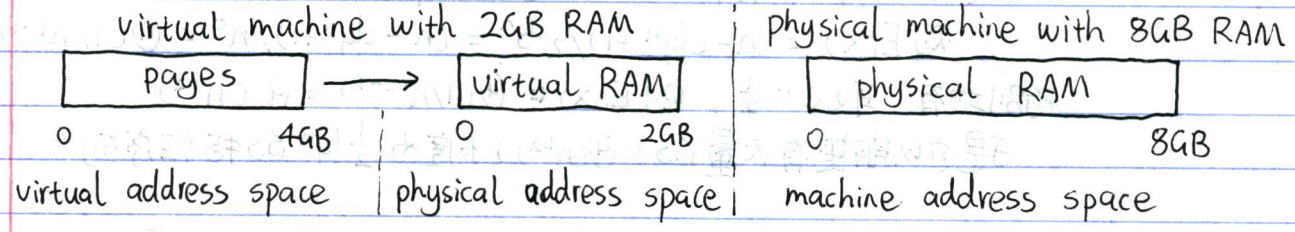
either : perform binary translation of all sensitive/privileged code

or : install module in ring 0 take care of loading virtual machine

set up to run similar to full virtualization in type 1 hypervisor

ring 0 module "clean" CPU state when other host OS process running

memory in virtual machine



memory virtualization, efficiency : the less involve hypervisor, the faster code run

if guest page table map page to frame

use page table directly in hardware for efficiency

frame problem : in use by the hypervisor or the host

in use by another guest

= shadow page table : actual mapping between page and frame

= hypervisor maintain another set of page tables for each virtual machine

== when guest OS try to load page table, trap to hypervisor

== loading page table is privileged operation

hypervisor make MMU use shadow page table instead of guest page table

guest Os : remove an existing mapping

need to invalidate the entry in the TLB

invalidating TLB is privileged operation

hypervisor know and update shadow page table

remap page to different frame

TLB invalidation should be performed

map new page to frame

simple update in memory belong to guest OS

no sensitive/privileged operation necessary to do

Operating System - P86

virtual machine (VM) in data center
- mobility : VM can easily be moved to alternate physical machine
  simply by copying file
- ease of configuration : service run on multiple machines or data centers
  can use common VM image
- response to load change : when load on service heavy
  additional VM can share the load
  can be shut down when load become lighter

VM security in data center
- running job in separate virtual machines
  increase isolation between different users
- user file typically distributed file system
  accessible from any physical machine with proper credential
- network typically organized into encrypted Virtual Private Network
  running on the same physical network
  isolate network traffic from different users' VM
  supply security without need for separate physical network

deploying application, organization frequently on : multiple computers
cloud computing provider

create challenge : having correct computing environment for application
OS / APIs / dependent software
messy configuration option
interference with/from other application
library version conflict

deployment strategy : install application on host system (oldest strategy)
may not have expected environment

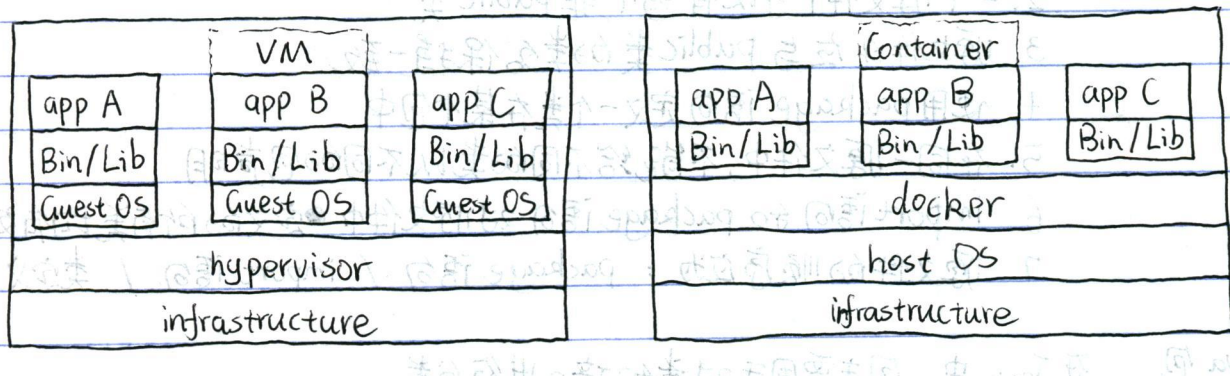
run application in virtual machine (VM)
entire OS, application, dependency encapsulated in VM

run application in Container
Container has only app and dependency, no OS or file system

Operating System - P87

application deployment by virtual machine
- include entire OS, file system, application, all dependencies
- preserve state of file system
  - deploying multiple copies require configuration of each copy
- can run on any host OS
- app memory/disk isolated from other app running on same physical hardware
- require resource on host system
  - virtual disk storage
  - memory for VM OS and hypervisor
  - multi-layer virtual memory page table
- CPU scheduling and memory allocation trickily ensure adequate performance

application deployment by Container
- allow app to run using host OS kernel, memory, file system
  - much like normal host OS process
- namespace : provided by Linux kernel
  - allow app isolated from other app running on same machine
  - overhead of running in Container much less than in VM
- docker : software take care of the setup for Container
  - build image : programmer specify file needed
    - then loaded into container to run



container security
- sandbox : container commonly run in
- restrict : how containerized application interact with host OS and other process
- other application of sandbox
  - browser run web page in sandboxed process for security

Operating System - P88

Linux namespace : run with isolated resource to support Container concept

process ID (PID) : process use own set of process ID number within Container
new process get ID number within namespace

mount (MNS) : upon creation, inherit system mounted file system
new mount visible only with the namespace

network (NET) : private set of IP address, routing table, socket

Cgroup : isolated root directory

interprocess communication and shared memory (IPC)
isolate shared memory and other IPC feature inside namespace

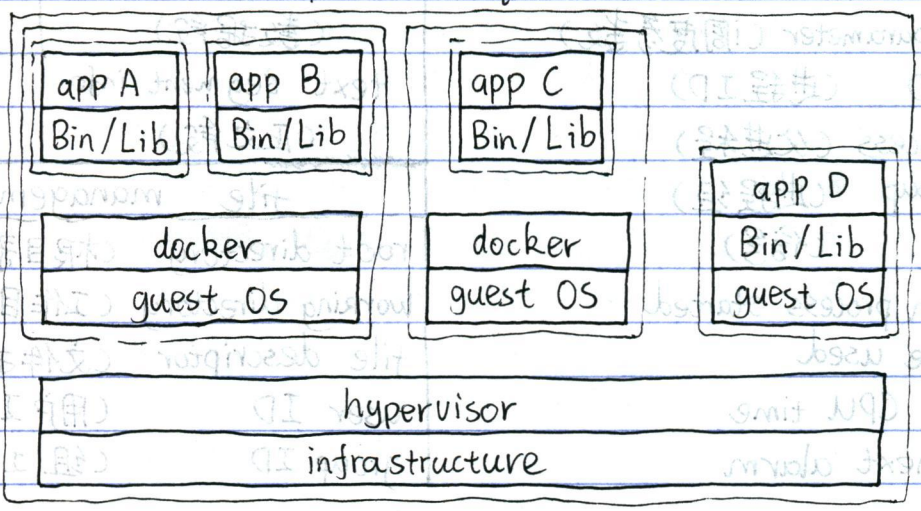
User : user and group ID

UTS : host name and domain name

running Container within VM

container cannot be run directly on a system
with different OS than one needed by the Container

typically accomplished by running one or more Container
inside VM provide required OS



image

built to run on particular OS kernel

other needed OS component packaged into Container

executable and library required by application packaged into Container

Java JVM, JRE and additional library jar file

C/C++ standard library and additional shared library

Python interpreter

Operating System - P89

incremental image building : docker support incremental image build
Start with existing image, add additional file and code
docker provide repository with standard image
organization can also set up own repository

running Container : running copy of image
running image create fresh container
state change during running don't affect original image
docker create isolated container and load image into container to run
original image not altered by running container
same image can be used to launch multiple copies of application

storage and communication for Container
application can be set up to have access to common shared storage volume
can be shared by network across multiple machines
application can access network-based shared storage
app can be configured to accept and request network connection
to other container or non-container app

synchronization (同步) : concurrent access to shared data may result in data inconsistency
maintaining data consistency
require mechanism to ensure orderly execution of cooperating process
all about prevention of race condition

race condition (竞争条件) : multiple process/thread access and manipulate same data concurrently
outcome of execution depend on particular order of access

critical section (临界区) : segment of code process may be changing shared data
mutual exclusion (互斥) : process should not be execution in critical section at same time
time independent : no assumption about speed or number of CPU
well defined blocking condition :
no process outside critical section may block other process
no starvation : no process should have to wait forever to enter critical section