Operating System - P90

requirement
mutual exclusion : if one process executing in critical section
no other process can be executing in critical section
progress : decision about which enter critical section next
cannot be postponed indefinitely
current process in remainder section cannot participate
bounded wait : limit on how many processes
will first gain access to critical section before given process

synchronization hardware : uniprocessor system can disable interrupt
currently running code will execute without preemption
require elevated privilege level
generally too inefficient on multiprocessor system
OS using this not broadly scalable
modern machine provide special atomic hardware instruction
either test memory word and set value
or swap content of two memory words
atomic : operation guaranteed to happen without interruption
implementation on x86 of hardware locking
XCHG : perform atomic swap (exchange) of two operands
implicitly lock memory location for all processes

semaphore (信号量), integer variable can be initialized to non-negative number
can only be accessed through two standard atomic operations
down/up (wait/signal , wait/release , acquire/release)

busy waiting (忙等待), process waiting to enter critical section in while loop of down operation
spinlock (互斥锁) : semaphore implementing for busy waiting
can be remove by associating waiting queue with each semaphore
process block if semaphore value not positive
happen after process added to waiting queue
process restarted when another process execute signal (up)
process to restart chosen from the queue

Operating System - P91

semaphore
- implemented as system call
  - OS disable interrupt when running up/down
- implemented in thread runtime system
  - must use one of software synchronization method
    - to ensure another up/down operation not initiated

binary semaphore (二元信号量): semaphore value 0/1

counting semaphore (计数信号量): any other semaphore

starvation (饥饿): process keep waiting to enter critical section
- can happen if process waken up
  - from waiting queue in last-in-first-out order

priority inversion (优先级反转): lower priority process has lock on resource
- required by higher priority process

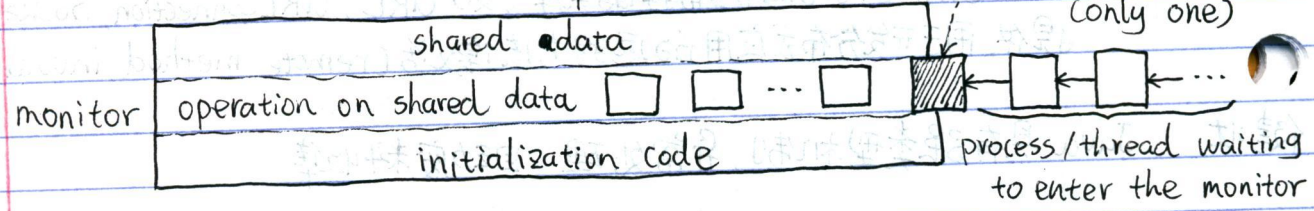
mutex lock (互斥锁): implement in kernel or in thread runtime system
- ownship: only process own the lock can unlock
- modify mutex_lock to remember the process own the lock
  - run the function without blocking
- modify mutex_unlock to execute body only called by owner

futex (fast hybrid mutex operator)
- acquiring lock using TestAndSet/Swap much faster than making system call
- built-in CPU operation don't provide queueing, require busy wait
- implement mutual exclusion operation
  - provide best features of both
  - initially use CPU operation to try to get lock
  - after certain amount of time or number of unsuccessful tries
    - make kernel call to be put on wait queue
  - has advantage of not needing kernel call most of time
    - not trying up the CPU when there is one longer wait

Operating System - P92

condition variable (条件变量) : condition used to block process in semaphore and mutex lock is simple
no check on any variable
wait(cc) : process that invoke is suspended
signal(cc) : resume one of process that invoked wait(cc) (if any)
usage : user code obtain lock on shared variable
user code check condition on variable
release lock on shared variable
call wait/signal depending on result of condition check

monitor (管程) , high-level abstraction provided by language (Java)
provide convenient and effective mechanism for process synchronization
only one process may be active within monitor at one time



condition variable : construct used to suspend execution
while waiting for condition to become true
difference from semaphore
wait() always block the calling process
wait on semaphore may not , semaphore value maybe positive
signal() has no effect if no process waiting on the condition variable
signal on semaphore always increment valeue
and wake up any waiting process
process ready to resume because of signal()
must first acquire lock on monitor

Java monitor , Java class may have synchronized member function
only one thread may execute any synchronized function at one time
other thread block
use wait() / notify() instead of condition variable
special version for synchronized function , not subject to race condition

Operating System - P93

reader/writer problem (读者/写者问题)
- multiple processes may have simultaneous read-only access to data
- only single process at one time may have write access
  - write access also block all readers
- need solution that prevent deadlock and starvation
- shared resource (memory area / file) protected by critical section
- any number of readers may simultaneously be in critical section
- when writer need access, must have exclusive access
  - no reader/other writer have access
  - writer may be subject to starvation
    - if there is constant stream of readers

deadlock (死锁), a set of blocked processes
- each holding resource and waiting to acquire resource held by another

system model: resource type $R_1, R_2, \dots, R_m$
- CPU cycle, I/O device, mutex lock, semaphore
- resource cannot be preempted
- each resource type R_i has E_i instances
- each process utilize resource

request: wait if request to acquire not granted immediately
use: can operate on the resource
release: no longer operate on the resource

∴ deadlock can arise only when all conditions hold simultaneously

mutual exclusion
- only one process can use one instance at one time

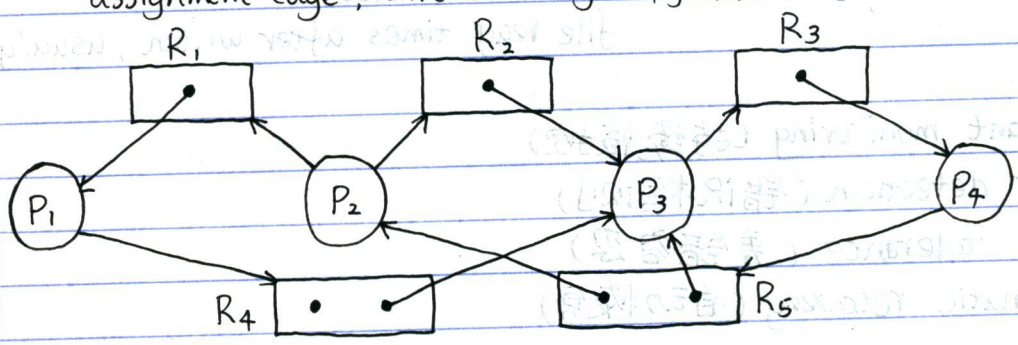
hold and wait
- process holding at least one resource can request additional resource
  - may be held by other process

no preemption
- resource can be released only voluntarily

circular wait
- exist one set $\{P_0, P_1, \dots, P_n\}$ of waiting processes
  - P_i waiting for resource held by $P_{(i+1)\%(n+1)}$

Operating System - P94

resource-allocation graph, vertices partitioned into two types
$P = \{P_1, P_2, \cdots, P_n\}$ consisting of all processes in the system
$R = \{R_1, R_2, \cdots, R_m\}$ consisting of all resource types in the system
edge partitioned into two types
request edge, directed edge (P_i, R_j), P_i wait R_j
assignment edge, directed edge (R_j, P_i), P_i hold R_j



cycle not necessarily a deadlock
graph contain no cycle → no deadlock
graph contain cycle
∧ only one instance per resource type → deadlock
if several instances per resource type, then possibility of deadlock

single instance resources in the system
deadlock exist ↔ cycle in the resource-allocation graph

handling:
ignore and pretend that deadlock never occur in the system
used by most operating system (Linux)
allow system to enter deadlock state and recover
deadlock detection
ensure that system will never enter deadlock state
deadlock avoidance: temporarily withhold resource if lead to deadlock
P deadlock prevention: design system so deadlock is not possible

detection:
allow system to enter deadlock state
examine the state of system to determine if deadlock occured
new request granted / periodically / CPU utilization drop below threshold
recovery scheme: policy on what to do if deadlock occured

Operating System - P9S

possible recovery scheme
- abort all deadlocked processes
- abort one process at one time until deadlock cycle eliminated
- abort process that not deadlocked
  but hold needed resource to break deadlock
- preemption : take resource away from another process
- rollback : return to some safe state (when resource not needed)
  and resume process from that state
  much easier to terminate and restart

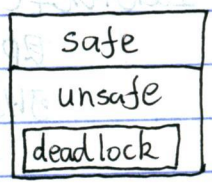
avoidance
require each process declare maximum of each resource type needed
deadlock-avoidance algorithm
- dynamically examine resource-allocation state
- ensure there can never be circular-wait condition
- resource-allocation state : number of available/allocated resource
  maximum demand of process

safe state
$\exists$ safe sequence $< P_1, P_2, \cdots, P_n >$ of all processes in the system
$\forall P_i$, resource P_i may still request can be satisfied by
- currently available resource
- and resource held by all P_j with $j<i$

unsafe : no safe sequence exist in the system
system must decide if immediate allocation leave system in safe state
when process request available resource

execution in safe state
P_i can wait until all P_j ($j<i$) finish if resource needed not immediately met
then P_i can obtain needed resource, execute, return allocated resource, terminate
$P_1, P_2, \cdots, P_i$ terminate, P_{i+1} may obtain resource needed

system in safe state → no deadlock
system in unsafe state → possibility of deadlock
avoidance : ensure system never enter unsafe state



Operating System - P96

avoidance algorithm : single instance of one resource type
resource-allocation graph
multiple instances of one resource type
Banker's algorithm
used when resource can have multiple instances
each process must declare
maximum instance of each resource type needed
when process request resource may have to wait
when process get all resources needed
must return in finite amount of time
need nxm matrix, Need[i,j] = k
P_i need k more instances of R_j to execute
similar in concept to deadlock detection, except: process
look ahead to maximum of each resource needed by each
run algorithm before assigning resource
to determine if still in safe state
Banker's algorithm and other algorithm not generally practical
require knowledge of entire future work load of system
and all resources that will be required by process

deadlock prevention, can be incorporated in system design
design to prevent one or more deadlock conditions
design dicision place limitation on behavior of process
typically implemented internally for the OS
difficult to enforce limitation on user process

eliminating mutual exclusion
not allow process to reserve resource
virtualize access to all resources

eliminating hold and wait
require process to request all resources needed as a package
not assign any resource until all resources available

Operating System - P97

deadlock prevention, eliminating no preemption
possible to implement system policy
allow for preemption of assigned resource instance
system can have mechanism to notify process
not all resource easily preemptible without problem

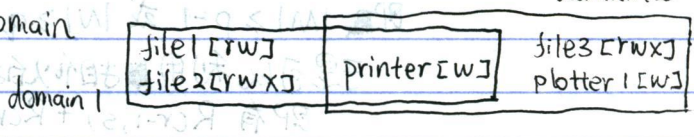
eliminating circular wait
assign unique number to each resource type
require process to request resource in ascending order
used on CDC 6600 Scope OS. for internal system lock

security problem, computer consist of collection of object, hardware or software
each object has one unique name
and accessed through well-defined set of operation
ensure each object accessed correctly
and only by process allowed to access
secure if resource used/accessed as intended under all circumstance

term
intruder (cracker) : attempt to breach security
threat : potential security violation
arising from existence of vulnerability in the system
attack : attempt to breach security
typically by exploiting vulnerability

preserve
confidentiality : release of data to unauthorized user never occur
integrity : unauthorized user should not be able to modify any data
availability : system should be serving the purpose it was designed for

protection domain : set of access-right assigned based on principle of least privilege
access-right = <object-name, right-set>
subset of all valid operation performed on the object
every process run in protection domain



Operating System - P98

domain implementation (UNIX), domain = user-id
- domain switch accomplished via file system
  - each file associated with domain bit (setuid bit)
  - when file executed and setuid = on
    - user-id set to owner of file being executed
  - when execution complete, user-id reset
- domain switch accomplished via password
  - when other domain password provided
  - sudo command temporarily switch to another user's domain
- domain switch via command
  - sudo command prefix execute specified command in another domain
    - if original domain has privilege or password given

protection matrix

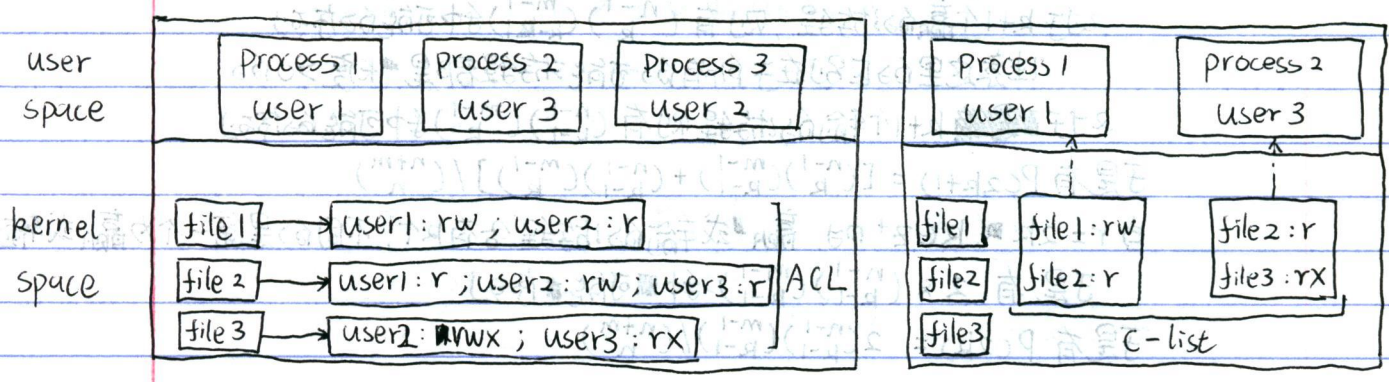
domain \ object	file 1	file 2	printer	plotter	D1	D2
domain 1	RW		W			enter
domain 2		RWX	W	W	enter	

operation "op" must be in cell(i, j)
- if process in domain D_i try to do "op" on object O_j

enter operation : domain can switch to another domain

ACL (access control list), each column implemented as access list for one object
- resulting per-object list consist of ordered pair <domain, right-set>
  - defining all domain with non-empty set of access right for object

C-list (capability list), each process has capability list for domain
- list of object together with operation allowed



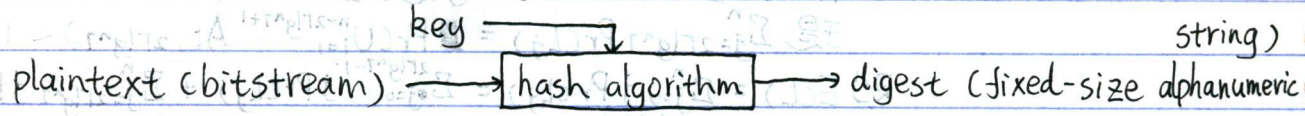
Operating System - P99

comparison

right check for operation
- ACL : check through long list of <domain, right>
- C-list : only check capability list of process

revocation : remove right for object
- ACL : search access list of object
  - and remove entry corresponding to the right
  - can be selective
- C-list : search all C-list for object with particular right, then remove
  - selective removal is difficult

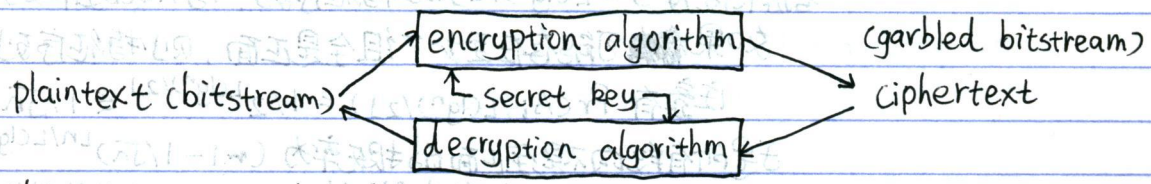
cryptography, constrain potential sender (source) and/or receiver (destination) of message
- based on secret (key)
- enable : receipt only by certain destination
  - confirmation of source



attacker can pre-compute digest for possible password and compare

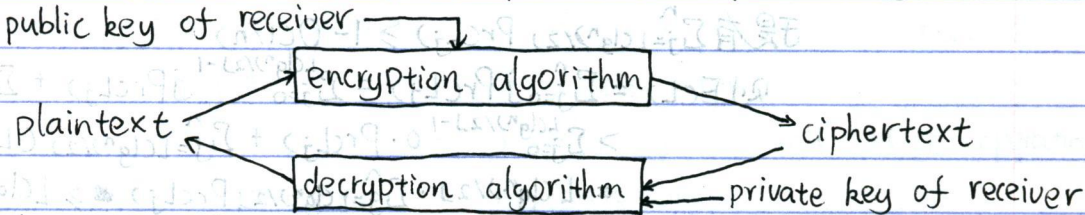
"salt" should be random, large number (>12-bit and different for each
- stored and accessed for password checking

symmetric encryption



attacker know the algorithm and can see ciphertext
- obtaining secret key from ciphertext computationally infeasible

asymmetric encryption



attacker know algorithm, public key and can see cipher text
- obtaining private key from ciphertext and public key is computationally infeasible

Operating System - P100

asymmetric encryption (非对称加密)
- public-key encryption based on each user having two keys
  - public key : published key used to encrypt data
  - private key : key known only to individual user used to decrypt data
- encryption scheme can be made public without making easy to figure out the decryption scheme
- most common : RSA block cipher
- efficient algorithm for testing prime number
- no efficient algorithm for finding prime factor of large number

digital signature
- message + digest (hash) → encryption algorithm ← private key (sender)
- sender / receiver
- cipher-text
- message + d (d' = digest of message) ← decryption algorithm ← public key (sender)
- message not tampered with during transit if d = d'
  - then sender also authenticated

key distribution
- delivery of symmetric key is huge challenge
  - sometimes done out-of-band
- asymmetric key can proliferate
  - public key no secret
  - asymmetric key distribution need care
    - man-in-the-middle attack

digital certificate : proof of ownership of public key
- trusted party receive proof of identification from user
  - certificate public key belong to user
    - user's public key encrypted / signed with trusted party's private key
  - signature legitimate
- certificate authorities are trusted parties
  - public key included with web browser distribution
  - vouch for other authorities via digitally signing their keys