資管三作業系統
課程講義

國立中山大學資管系
黃三益教授

1999 年 2 月 22 日

Operating System Overview

What is an operating system?

- The operating system controls and coordinates the use of the hardware among the various applications programs for the various users. (see figure 1.1)
- An operating system is a resource allocator.
- An operating system is a control program.
- An operating system is the one program running at all times on the computer, with all else being application programs.
- Two goals of the operating system: convenience and efficient.

Multiprogramming

- In a multiprogramming environment, when one job needs to wait, the CPU is switched to another job.
- Deciding which job the CPU to execute next is called CPU scheduling.
- Multiprogramming is mainly used to improve CPU utilization.

Time-Sharing

- Time-Sharing allows several users to interactively use computer systems.
- When a user is doing I/O, the CPU is switched to serve another user.
- To give the user the impression that he/she is the only user of the system, CPU may have to be switched to serve another user even when the current user is not doing any I/O.
- A time-sharing operating system does more than the multiprogramming operating system.
    - In a multiprogramming operating system, the system can choose a set of jobs to be fit into memory and switch among them.
    - In a time-sharing operating system, the system has to switch among a set of pre-defined jobs, no matter whether or not they can all fit into main memory.

- To solve the memory limitation problem, virtual memory technique was developed.

A modern computer system

- See figure 2.1

What happened when a computer system is started?

- A bootstrap program, which is located in ROM, is executed and does the following:
  - initialize the system and all controllers.
  - testing all controllers and memory.
  - load the operating system from (the fixed part of) the hard disk.
- The operating system
  - initializes its internal data structures, and
  - invokes a process to execute a pre-defined script which will creates several user processes.
- The operating system will gain control on the request of device controllers or user processes via interrupt.

What does an interrupt do?

- Interrupt can be triggered by hardware by sending a signal to CPU to request some service from the operating system. This is called hardware interrupt.
- Interrupt can also be triggered by a user process by executing an operation of special kind, called system call. This is called software interrupt (or trap).
- Show the figure that contains PC, PS, system stack, and interrupt vector.
- Operations of an interrupt: (done by the hardware)
  1. Save the program counter (PC) register and Program status (PS) registers in the system stack.
  2. Based on the received signal, find the address of the desired interrupt service routine from the interrupt vector.
  3. Put the address in the PC register (thus, the next executing instruction will be

the first instruction in the interrupt service routine.)

- Operations of an interrupt service routine:
  - Save the values of the registers that will be used in the interrupt service routine in the system stack
  - Start doing the service.
  - Pop from the system stack to restore the values for all used registers, PS, and PC.
  - The next executed instruction will be the next instruction in the process that was interrupted.
- In the execution of an interrupt service routine, only interrupt with higher priority is allowed to happen. Others have to wait (also done by hardware).
- Modern operating systems are interrupt driven.
- Interrupt service routine is part of the operating system. It is within the interrupt service routine that the CPU control can be switched to another process.
- Without interrupt mechanisms, it will be very difficult to implement multiprogramming and time-sharing.

How the I/O is conducted?

- The CPU side:
  - The user process traps to request an I/O.
  - The interrupt service routine loads the appropriate registers within the device controller.
  - The operating system can then switch the CPU to another job.
- The device controller side:
  - The device controller examines the contents of its registers to determine what action to take.
  - Once the action is done, it causes an interrupt to get the attention of the CPU.
- The CPU side:
  - The interrupt service routine gets the input data (in case of “I”).
  - The operating system finds which user process is waiting for the I/O, transfers

the input data to the appropriate memory location, and change of the status of the process to “RUNNING”.

- The operating system switches CPU to one of the RUNNING process.

The operating system can use a device-status table to spool I/O requests. See Figure 2.3.

How to prevent too many interrupts:

- Too many interrupts may further deteriorate the CPU performance.
- To solve this problem, direct memory access (DMA) is used for high-speed I/O devices.
- Only one interrupt is generated per block, rather than the one interrupt per byte (or word).

Storage Hierarchy (see Figure 2.6)

- CPU can directly access registers and memory.
- Programs are loaded into main memory and executed one instruction after another.
- Loading one instruction from memory takes several clock cycles.
- To speed up execution, next instructions expected to be executed are copied to the cache.
- Even operand needed by an instruction can be copied to cache in advance. This is called cache management.
- For hard disk structure, see Figure 2.5 at page 33.
- In a hard disk, the read-write head is close to the surface of a platter but does not touch it.
- Floppy disks are coated with a hard surface, so the read-write head can sit directly on the disk surface without destroying the data. But the coating will wear with use.

In a sharing environment (e.g. multiprogramming or timesharing), how to prevent one bug at one program from affecting other processes?

- One erroneous program might modify the program or data of another program, or

event the operating system itself.

- With proper protection, when a user program tries to execute an illegal instruction or to access memory that is not in the user's address space, then the hardware will trap to the operating system.
- An appropriate error message is given, and the memory of the program is dumped.

How is the protection done?

- The hardware supports two modes of operations: **user** mode and **monitor** mode (or called system mode). These two modes are distinguished by a mode bit.
- When the operating system is executing, the machine is in monitor mode.
- Some machine instructions are classified as privileged, indicating that they may cause harm and thus can only be executed in the monitor mode (i.e. only the operating system can execute these instructions).
- If the interrupt vector can be changed, a user program can be in monitor mode.
- All I/O instructions are privileged to prevent users from performing illegal I/O. The user must ask the operating system to do I/O on the user's behalf. Such a request is called a *system call*.
- Memory protection must be done to prevent a user process from updating the interrupt vector, the operating system code, and other user processes.
- This is done by two registers: *base* and *limit*. Once an attempt to access memory address is either less than base or greater than base+limit, a hardware interrupt occurs. See Figure 2.8.
- Loading base or limit registers are done by special privileged instructions.
- Other privileged instructions include halt (i.e. shut down the computer), turning the interrupt on and off, and the change to the mode bit.

How does an operating system implement time-sharing?

- The timer could be set to interrupt every N milliseconds where N is the time-slice each user is allowed to execute before the next user gets control of the CPU.
- When a timer interrupt occurs, the following are done:

- housekeeping work(e.g. accountings, process time accumulation, record current time)
- set the values of registers and other parameters to prepare for the next program to run. (This procedure is called *context switch*.)

System Components:

- Process Management:
  - A process can be thought as a program in execution.
  - A process needs certain resources, e.g. CPU time, memory, files, and I/O, to accomplish its task.
  - All processes can be executed concurrently, by multiplexing the CPU among them.
  - The following activities are performed within process management:
    - The creation and deletion of both user and system processes.
    - The suspension and resumption of processes.
    - The provision of mechanisms for process synchronization.
    - The provision of mechanisms for process communication.
    - The provision of mechanisms for deadlock handling.
- Main Memory Management
  - To improve both the utilization of CPU and the speed of the computer‘s response to its users, we must keep several programs in memory.
  - Memory management usually requires some kind of hardware support.
  - The following activities are performed within main memory management:
    - Keep track of which parts of memory are currently being used and by whom.
    - Decide which processes are to be loaded into memory when memory space becomes available.
    - Allocate and deallocate memory space.
- Secondary Storage Management
  - The secondary storage is used to back up main memory.

- Because secondary storage is used frequently, it must be used efficiently.
- The following activities are performed within disk management:
  - free space management
  - storage allocation
  - disk scheduling

- I/O System Management
  - The I/O system provides a uniform view on different I/O devices, e.g. SCSI, keyboard, mouse, disk, PCI bus device etc.
  - The I/O system consists of:
    - A buffer-caching system
    - A general device-driver interface
    - Drivers for specific hardware devices
- File Management
  - For convenient use of the computer system, the operating system provides a uniform logical view of information storage.
  - A logical storage unit, file, is provided.
  - The following activities are performed within file management
    - the creation and deletion of files
    - the creation and deletion of directories
    - the support of primitives for manipulating files and directories
    - the mapping of files onto secondary storage
    - the backup of files on stable storage media
- Protection System
  - A protection-oriented system provides a means to distinguish between authorized and unauthorized usage.
- Networking

System calls provide the interface between a process and the operating system. See Figure 3.2 for some examples.

System Architecture

- See Figure 3.7 for the system architecture.
- Mach reduces the kernel by moving all nonessentials into systems and even into user-level programs. The resultant kernel is called *microkernel*.

Java Virtual Machine

- See Figure 3.12 for the concept of a virtual machine.
- Java is implemented by a compiler that generates bytecode output.
- The bytecodes are the instructions running on the Java Virtual Machine (JVM).
- When a platform has a JVM running on it, it can execute Java programs.
- IBM PC, Apple Macintosh, UNIX workstations, IBM mini and mainfrome can all run JVM. Web browsers also implement JVM.
- JVM is also implemented on the small JavaOS, which implements JVM directly on the hardware.
- The instructions set of JVM includes the expected arithmetic, logical, data movement, and flow control instructions, as well as other higher level operations such as object creation, manipulation, and method invocation.
- Java program runs faster than interpreted programs, though still slower than the native hardware instructions.
- It can also be implemented on embedded systems through the use of CPUs to execute Java bytecodes.

Chapter 4 PROCESSES

What is a process?

- a program in execution
- the execution of a process must progress in a sequential fashion.
- Physically, a process contains the following:
  - the program code (called text)
  - the value of the program counter
  - the contents of the processor's registers
  - the process stack (that holds temporary data)
  - the data section (that contains global variables)
- a process is an active component, while a program is a passive one.
- Two processes associated with the same program have only one common code, while all the other components of processes are different.
- A process is sometimes called a job.

Process State

- There are the following five states:
  - new
  - running
  - waiting
  - ready
  - terminated
- see Figure 4.1 for the state diagram

Process Control Block

- Each process is represented in the operating system by a process control block (PCB), which contains the following:
  - process state

    - program counter
    - CPU registers (e.g. accumulators, index registers, stack pointers, general purpose registers, program status register)
    - CPU scheduling information
    - memory management information (e.g. base and limit registers, page tables and segment tables for virtual memory)
    - accounting information (CPU, real time used, process number)
    - I/O status information (e.g. a list of used I/O devices and open files)
- See Figure 4.2, 4.3

Process Scheduling

- For scheduling purpose, processes are put in several scheduling queues, waiting for service.
- A ready process is placed in a ready queue.
- A wait process that needs an I/O service is placed in a particular device queue. Each device has its own device queue.
- A life time of a process can be represented as a queuing diagram. See Figure 4.5.

Schedulers

- Three types of schedulers:
    - the long-term scheduler:
        - selects processes from submitted processes and put them into memory for execution.
        - controls the degree of multiprogramming.
        - better chooses a good mix of I/O bound processes and CPU-bound processes.
        - Many time-sharing systems do not have this scheduler, and relies on self-adjusting nature of human users.

- the short-term scheduler: selects among the ready processes and allocate the CPU to it.
- the medium-term scheduler: swaps out and swaps in later some processes to help reduce degree of multiprogramming.

Context Switch

- the task of saving the state of the old process and loading the saved state for the new process is called context switch.
- the speed of context switch ranges from 1 to 1000 microseconds.
- the more complex the operating system, the more work must be done during a context switch.
- Some hardware (CPU) provides several sets of registers to reduce the time for context switch.
- Context switching is becoming a performance bottleneck that new structure (threads) are being proposed to avoid it whenever possible.

Process Creation

- A process may create several new processes (e.g. fork() in UNIX).
- When a process creates a child process, the child process may be able to obtain all or part of the resources from its parent process.
- The child process may execute concurrently with the parent.
- The parent may wait until some or all of its children terminate (e.g. wait() in UNIX).
- The child process may load a new program for execution (e.g. exec() in UNIX).
- See a C program with unix system calls.

Process Termination

- A process may issue a *exit*() system call to ask the operating system to terminate it and deallocate all its resources.

- A process may be terminated by another processes by some system call (e.g. kill() in UNIX).

Cooperating Processes

- Why several processes are created and need to cooperate:
  - information sharing
  - computation speedup
  - modularity
  - convenience
- See page 102 for producer-consumer problem.

Threads

- A thread is also called a light-weight process, in contrast to the conventional (heavy-weight) process that consumes a lot of time for context switch.
- the operating system resources, such as code section, data section, and open files and signals, traditionally allocated to a process are collectively called a task.
- Peer threads of a task share the resources of a task.
- Differences between processes and threads:
  - processes are independent of one another. Different processes may execute different programs.
  - Even if different processes execute the same program, they are probably issued by different users, and the data need to be protected.
  - Threads (of the same task) are not independent of one another. They are probably designed to assist each other.
  - Data shared by peer threads are not protected.
- Context switching between threads still needs to load/save registers, but no memory-management related work needs to be done because data section is also shared.
- Two types of threads:
  - *user level threads*:

        - it is implemented in user level libraries.
        - Operating system does not know the existence of these threads. The operating system sees a group of threads as a single process.
        - It is quick because the context switch does not trap to the operating system.
        - A thread executing a system call will cause the entire task to wait until the system call returns.
        - Without known to the operating system, the scheduling can be unfair. For example, 100 user-level threads that share a process may receive the same time quantum as one user-level thread that share a process.
    - kernel-level threads
        - implemented by most recent operating systems, e.g. Sun Solaris 2, Windows NT, OS/2, and IBM AIX 4.x.

Interprocess Communication

- shared memory
- message system

Message System

- Two primitives: send(message) and receive(message).
- The following are common questions:
    - How are the links established?
    - Can a link be associated with more than two processes?
    - How many links can there be between every pair of processes?
    - What is the capacity of a link? That is, does the link have some buffer space? If so, how much?
    - What is the size of messages? Is it variable-sized or fixed-sized?
    - Is the link unidirectional or bi-directional?

- Direct Communication
  - symmetry

    send(P, message), receive(Q, message)
  - asymmetry

    send(P, message), receive(id, message)
- Indirect Communication
  - send and receive from a mailbox.
  - send(A, message), receive(A, message)

- Buffering
  - zero capacity: The two communicating processes must be synchronized. This synchronization is called *rendezvous*.
  - bounded capacity:
    - At most *n* messages can reside in it.
    - When the buffer is not full, the sender doesn't have to wait.
    - When the buffer is full, the sender has to wait.
    - The sender can ask an acknowledge by doing the following

      send(Q, message1)

      receive(Q, message2), where message2 contains the acknowledge to message1.
    - This is called asynchronous communication.
  - unbounded capacity

- Receiver Blocking
  - the receiver may wait until a message is received.
  - the receiver may try to get a message without waiting.
  - the receiver may set up a time limit for waiting.
- A common type of synchronous communication, remote procedure call (RPC) can be implemented by many low level communication.

Exception Conditions for Interprocess Communication

- Process terminates

  sender terminates: inform the receiver the sender has terminated.

  receiver terminates: inform the sender if the sender may have been blocked.

- Lost messages

  using timeout

- Scrambled messages

  using checksum.

Chapter 5 CPU Scheduling

Why CPU Scheduling?

- The Objective of multiprogramming is to maximize CPU utilization.
- Scheduling is a fundamental operating system function.
- CPU is one of the primary computer resources.

CPU Burst and IO Burst

- The execution of a process consists of a sequence of CPU execution and I/O wait, which are called CPU burst and IO burst respectively.
- The duration of most CPU burst is small. An I/O bound program typically has many very short CPU bursts, while a CPU-bound program might have a few very long CPU bursts.
- See Figure 5.1 and 5.2 for graphical representation.

What is the ready queue for a short term scheduler like?

- The short term scheduler (or called CPU scheduler) chooses a process to execute from the ready queue.
- The ready queue stores the PCBs of ready processes.
- The ready queue might be implemented as a FIFO queue, a priority queue, a tree, or simply an unordered linked list.

When might the CPU scheduling take place?

1) When a process switches from the running state to the waiting state.
2) When a process switches from the running state to the ready state.
3) When a process switches from the waiting state to the ready state.
4) When a process terminates.

What is **nonpreemptive** scheduling?

- Under nonpreemptive scheduling, a process releases the CPU only when it is voluntary to do so (i.e. when it terminates or waits for an IO).
- When scheduling takes places only under circumstances 1 and 4, it is called nonpreemptive scheduling.
- Microsoft Windows 3.1 uses nonpreemptive scheduling

What is preemptive scheduling?

- Under preemptive scheduling, scheduling takes place under all four circumstances.
- Without proper implementation, preemptive scheduling may cause inconsistency to some shared data (e.g. data accessed by system calls).
- One possible way is to disallow context switch when a process is accessing some kernel data structure.
- For some data that might be accessed by multiple processes simultaneously, the protection of the data is done by putting a lock before use and releasing the lock after use.

Possible Scheduling Criteria

- There are many scheduling algorithms. We need some criteria to compare them.
- Different criteria might result in different conclusion about which scheduling algorithm is the best.
- Possible scheduling criteria:
    - CPU utilization
    - throughput (finished jobs/time unit)
    - turnaround time (interval from the time of submission to the time of completion)
    - waiting time
    - response time (interval from the time of submission to the time the first response is received)
- The measure used in this book is average waiting time.

- Average waiting time cannot measure the fairness.

Scheduling Algorithms

1. First Come, First Served Scheduling (FCFS)
    - It can be implemented as a FIFO queue.
    - The average waiting time may be long.
    - See the example in page 129.
2. Shortest Job First Scheduling (SJF)
    - A more appropriate term is shortest next CPU burst.
    - The SJF is provably optimal in terms of average waiting time.
    - The real difficulty is knowing the length of the next CPU request.
    - The information about the next CPU burst can be supplied by the user in the past.
    - The next CPU burst of a process is estimated from its previous CPU bursts in the modern computer system.
    - A common approach for estimation is shown in page 131.
    - The SJF may be either preemptive or nonpreemptive.
    - A preemptive SJF will preempt the current executing process if a process with shorter CPU burst arrives.
    - Preemptive SJF is sometimes called shortest-remaining-time-first.
    - See example in page 133.
3. Priority Scheduling
    - The SJF is a special case of the general priority scheduling with the priority being the next CPU burst.
    - In priority scheduling, the process with the highest priority is chosen.
    - For convenience, this book assumes 0 (small value) to be the highest. That is, low numbers represent high priority.
    - See example in page 134.

- Priority can be defined either internally or externally.
- Internal priorities use some measures concerning the operating system‘s resources, e.g. time limits, memory requirements, the number of open files, etc.
- External priorities indicate some political factors that are unknown to the operating system, e.g. the importance of the process, the amount of money paid by the user, etc.
- A major problem with priority scheduling is indefinite blocking or starvation.
- A solution to the starvation of low-priority processes is **aging**. Aging will gradually increase the priority of processes that wait in the system.

4. Round-Robin Scheduling (RR)
    - Round-robin scheduling is FCFS scheduling with preemption.
    - A time slice, called time quantum, is defined by the scheduler.
    - The ready queue is treated as a circular queue. The CPU is allocated to each process for a time interval of up to 1 time quantum.
    - The average waiting time under the RR scheduling is often quite long.
    - See example in page 135.
    - The performance of the RR depends heavily on the size of the time quantum.
    - If the time quantum is very large, RR is the same as FCFS.
    - If the time quantum is very small, it is as if each process has its own CPU running at 1/n of the speed of the real CPU.
    - However, in reality, we have to consider the overhead of context switch. The time quantum should be large with respect to the context switch time.
    - A rule of thumb is that 80% of the CPU bursts should be shorter than the time quantum.
5. Multiple Queue Scheduling
    - The ready queue is partitioned into several separate queues, and each process is permanently assigned to one queue.

- For example, foreground jobs and background jobs may be assigned to two different queues.
- Different queue may employ different scheduling algorithm. For example, foreground queue may use RR, while background queue uses SJF.
- How the CPU switches between 2 queues?
  - Each queue has absolutely higher priority over lower priority queues. That is, the CPU is serving a queue only when all the queues with higher priority are empty.
  - Another possibility is to assign time slice between the queues (e.g. 80% to foreground queues and 20% to background queues).
- See Figure 5.6.

6. Multilevel Feedback Queue Scheduling
   - This scheduling allows a process to move between queues.
   - The idea is to separate classes of processes with different CPU bursts.
   - For example, all queues may employ RR scheduling. But queue 0 is given a time quantum of 8 ms, queue 1 is given a time quantum of 16 ms. If a process consumes the entire time quantum, it will be moved to the next queue.
   - In general, a multilevel feedback queue scheduler is defined by the following properties:
     - The number of queues.
     - The scheduling algorithm for each queue.
     - The method used to determine when to upgrade a process.
     - The method used to determine when to degrade a process.
     - The method used to determine which queue will enter when that process needs service.
   - See Figure 5.7.

Real-Time Scheduling

- It can be classified into two categories:

1. Hard real-time systems.

2. Soft real-time systems.

What is a hard real-time system?

- A process is submitted along with a deadline, which indicates by when it must be completed.
- If the process is permitted, then its deadline is guaranteed.
- This guarantee is impossible in a system with secondary storage or virtual memory because it is difficult to foresee how long an instruction execution will take.
- Hard real time systems are composed of special-purpose software running on special hardware.

What is a soft real-time system?

- Each process is also associated with a deadline. But this deadline is transformed into a priority based on the urgency of the deadline.
- The goal of the system is to allocate resources in favor of the processes with higher priority.
- A soft real time system must consist of two components:
  1. priority scheduling (easy to accomplish)
  2. the dispatch latency for high priority process must be small.

How to keep dispatch latency low?

- The bad scenario occurs when a high priority process is ready to run, while a low priority process is executing a system call.
- In most version of UNIX, a system call cannot be preempted until it either finishes or has issued an I/O request.
- There are two possible solutions:
  1. insert preemption points in the safe place of a long-duration system calls.
  2. make the entire kernel preemptible. This is the approach used by Sun Solaris 2.

Chapter 6 Process Synchronization

Why processes need synchronization

- A set of cooperating processes is designed to fulfill some purpose.
- Each process is a sequential execution.
- These processes may share some data.
- The process here refers to either heavy-weight process or light weight process (thread).
- See page 156 for the producer-consumer programs utilizing *n* slots.

The above program does not function correctly in a system with concurrency.

- The interleaving of "count := count + 1" in the producer process and "count := count := count - 1" in the consumer process may yield incorrect results.
- The problem is that the increment and decrement statements are not atomic. Context switch may occur within the execution of the increment (decrement) statement.
- Race condition: several processes access and manipulate the same data concurrently, and the outcome depends on the particular order of access.
- Note that context switch can occur only between machine instructions.

The critical section problem

- Suppose each process has a segment of code, called critical section. At any time, at most one process can be executing in its critical section.
- One can put the change of common data in a critical section.
- The execution of critical sections by the processes are *mutual exclusive*.
- The critical section problem is to design protocols such that the processes can satisfy the requirements of critical sections.

Requirements of critical sections:

1. mutual exclusion
2. progress: When no one is in the critical section, and some processes are waiting to

enter the critical section, one of the waiting processes should be allowed to enter.

3. bounded waiting: When one process wishes to enter a critical section, the time it takes to wait must be bounded.

General structure of a typical process (see Figure 6.1)

Two processes solutions:

- Suppose there are only two processes Pi and Pj.
- Algorithm 1: see page 159
  - progress requirement is not satisfied
- Algorithm 2: see page 160
  - progress requirement is not satisfied
- Algorithm 3: see page 161
  - mutual exclusion is satisfied
  - progress as well as bounded waiting is satisfied

Multiple process solutions: see page 163

- mutual exclusion: we can prove that if Pi is in its critical section and Pk is waiting, then (number[i], i) < (number[k], k).
- progress and bounded waiting: the progresses enter the critical section in FCFS order.

Easier approaches for solving critical section problems

- Disable interrupt when entering a critical section and enable after it after leaving.
  - Disadvantage: cannot work for a multiple processors architecture.
- Provide special hardware instructions:

1. int Test-and-Set(setting-value)
2. swap(a, b)
   - see page 165
   - Note that machine instructions are atomic
   - See Figure 6.7 for using Test-and-Set for mutual exclusion

- See Figure 6.9 for using swap for mutual exclusion
- See Figure 6.10 for using Test-and-Set to solve critical section problem

Why semaphore?

- Solutions to the critical section problem are not easy to generalize to more complex problem.

What is a semaphore?

- A semaphore is an integer S guarded by two **atomic** operations: wait(S) and signal(S). Originally, wait() is referred to P and signal() is referred to V.
- Classical definition of wait() and signal():

  wait(S): while S <= 0 do no-op;

  S := S - 1;

  signal(S): S:= S+1;

Examples to use semaphore.

- n process critical section problem: semaphore "mutex" (see Figure 6.11).
- Alternating processes (P1 and P2) execution: semaphore "synch".

  P1:

  S1;

  signal(synch);

  P2:

  wait(synch);

  S2;

Another definition for semaphore

- why? Because it only guarantee mutual exclusion and does not provide the properties of progress and bounded waiting.
- Each semaphore is associated with a list of waiting processes:

  type semaphore = record

  value: integer;

  L: list of processes;

```
                end;
```

- Operating systems may store the PCBs of the process waiting for a semaphore S in S.L.
- wait(S) and signal(S) are as follows:

```
wait(S): S.value := S.value - 1;
        if S.value < 0
        then begin
                add this process to S.L;
                block;
        end;

signal(S): S.value := S.value + 1;
        if S.value <= 0
        then begin
                remove a process P from S.L;
                wakeup(P);
        end;
```

- If the semaphore value is negative, its magnitude is the number of processes waiting on that semaphore.
- Bounded waiting can be ensured by employing FIFO policy on the waiting processes.

How to guarantee the atomic execution of wait(S) and signal(S):

- Disable interrupt (not suitable for multiprocessor environment)
- Use the solutions to critical sections discussed before.

Deadlocks and Starvation

- Deadlock may occur if a process might access TWO semaphores:

```
P0              P1
wait(S)         wait(Q)
wait(Q)         wait(S)
```

Binary Semaphore and Counting Semaphore:

- the value in binary semaphore allows only 0 and 1.

  Wait(S):

  ```
  if (S = 0)
  then add this process to S.L;
  else S := S - 1;
  ```

  Signal(S):

  ```
  if some process is waiting in S.L
  then wakeup the first process in S.L
  else S := S+1;
  ```

- the value in counting semaphore allows integer, as discussed before.

How to represent a counting semaphore by binary semaphores:

- See page 171 and page 172.

Classical Problems of Synchronization

- The bounded-buffer problem: see Figure 6.12 and 6.13 in page 173.
- The readers and writers problem:
  - the first readers-writers problem: Priority is given to the readers. Thus, writers may starve.
  - the second readers-writers problem: Priority is given to the writers. Thus, readers may starve.
  - See Figure 6.14 and 6.15 for writer and reader respectively.
- The Dining-Philosophers Problem:
  - Several philosophers spend their time just thinking and eating.
  - These philosophers are sitting on a round table and eat Chinese food using chopsticks.
  - A philosopher pick up only one chopstick one at a time.
  - See Figure 6.17 for the structure of philosopher.
  - Deadlocks may occur to the structure in Figure 6.17
  - Approaches for solving deadlock problem:
    - Allow at most four philosophers to prepare to eat.
    - Allow a philosopher to pick up her chopsticks only if both chopsticks are

available.

Use an asymmetric solution; that is, an odd philosopher picks up first her left chopstick and then her right chopstick, whereas an even philosopher picks up her right chopstick and then her left chopstick.

Why monitors or other higher level language constructs?

- If the designer of a process by mistake or on purpose does not place the right function calls (wait() and signal()) in the right order.

  signal(mutex);
  ... will violate mutual exclusion
  mutex(mutex);

  wait(mutex);
  ... will cause deadlock
  wait(mutex);

See page 182 for the general format of a monitor.

- the local variables of a monitor can be accessed by only the local procedure.
- The monitor construct ensures that only one process at a time can be active within the monitor.
- There is a need for additional mechanisms with monitors for synchronization. These mechanisms are provided by *condition* construct.
  e.g. var x,y: condition;
- The only operations that can be invoked are *wait* and *signal*.
- *x.wait*: the process invoking this operation is suspended until some one invokes *x.signal*.
- *x.signal*: resumes exactly one suspended process. If no process is suspended, then this operation has no effect at all.
- When a process P invokes x.signal to resume the execution of another process Q, there are two possibilities:
  - P either waits until Q leaves the monitor, or wait for another condition.

- Q either waits until P leaves the monitor, or wait for another condition.

- Choice 1 was advocated by Hoare. In the following, we assume choice 1 is adopted.
- See Figure 6.21 in page 185.
- See Figure 6.22 at page 188 for the dining philosopher‘s monitor solution.
- This solution ensures mutual exclusion on the eating of any two neighbors and is deadlock free.
- Suppose dp is an instance of the dining-philosophers monitor, the following is the behavior of the philosopher I:

  dp.pickup(i)

  ...

  eat

  ...

  dp.putdown(i)

How to use semaphores to implement the constructs of monitors:

- see page 186

Solution for Ex. 6.3

We prove that this algorithm satisfies mutual exclusion, progress, and bounded waiting:

1. mutual exclusion

   When one process, say Pi, is in the critical section, flag[i] must be true and flag[j] be false. In addition, flag[i] can only be set by Pi. Thus, if two processes Pi and Pj are both in the critical section, flag[i] = flag[j] = true. This is not possible.

2. progress

   We first consider the case only when one process Pi wants to enter the critical section. In this case, flag[i] = true and flag[j] = false. Thus, Pi will enter the critical section no matter what value ‘turn’ might be.

   Let‘s next consider the case when both processes Pi and Pj want to enter the critical section. flag[i] and flag[j] will be both set to true. Without loss of generality, let turn

= j. In this case, Pi will be waiting in the second while loop (while turn = j do no-op;) and have set flag[I] to false. Thus, Pj can then enter the critical section.

3. bounded waiting

   Suppose process Pj is waiting at the second while loop and Pi is in the critical section. After Pi leave the critical section, turn will be set to j and flag[j] = false. Thus, Pi can then enter the critical section. If Pi wants to enter the critical section again, it will be blocked at the second while loop (with flag[i] = false, and turn = j). Thus Pj can enter the critical section once it gets the CPU.

Chapter 7. Deadlocks

- Types of resources: memory space, CPU cycles, files, and I/O devices. Each type of resources may have several instances.
- If a process requests a resource type, any instance of the type will satisfy the request.
- A process must request a resource before using it, and must release it after using.
- Examples of system calls to request and release a resource:
  open() and close(); malloc() and free()
- Resources may be either physical (i.e. memory space, CPU, printers) or logical (files, semaphores and monitors).

Characterization of Deadlocks

- Necessary Conditions:
  - mutual exclusion
  - hold and wait
  - no preemption
  - circular wait
- Resource-Allocation Graph
  - Pi -> Rj: Process Pi requested an instance of resource type Rj.
  - Rj -> Pi: An instance of resource type Rj has been allocated to process Pi.
  - See Figure 7.1 for an example
  - If the graph contains no cycle, then no deadlock exists
  - If each resource type has only one instance, a cycle implies a deadlock.
  - If each resource type has more than one instance, a cycle does NOT necessarily imply that a deadlock occurred.
  - See Figure 7.2 for a deadlock.
  - See Figure 7.3 for a cycle but no deadlock

Methods for Handling Deadlocks

- Ensuring that the system never enters a deadlock state

- Deadlock prevention: make at least one of the necessary conditions not hold.
- Deadlock avoidance: the operating system be given in advance additional information concerning which resources a process will request and use during its life time.

- Allow the system to enter a deadlock state and then recover.
  - need deadlock detection algorithm and deadlock recovery algorithm.
- Do nothing about it. This is used by most operating systems, including UNIX.

How to do deadlock prevention?

- Drop mutual exclusion
  - The mutual exclusion condition must hold for nonsharable resources. Thus, this approach is not feasible in some cases.
- Drop hold and wait
  - Two approaches:
    - Allocating all resources before a process starts execution.
    - Allocating some resources only when a process holds nothing. e.g. tape -> disk -> printer. This can be done two at a time.
  - Disadvantages:
    - resource utilization is low
    - starvation is possible.
- Drop no preemption
  - When a process P requests some resource that is held by process Q:
    P checks if Q is currently waiting for some other resource.
    then preempt Q
    else wait for Q;
  - This approach is used only for those resources that can be preempted and restored later, e.g. CPU registers, memory space.
- Drop circular wait
  - Number all resources.
  - Each process can request resources only in an increasing order of enumeration.

Deadlock Avoidance

- Deadlock prevention may suffer from low device utilization and throughput.
- If extra information about how resources are to be requested, deadlocks could be prevented in a less pessimistic manner.
- This algorithm defines the *deadlock-avoidance* approach.
- The simplest and most useful model requires that each process declare the maximum number of resources of each type that it may need.
- A state is safe if the system can allocate resources to each process in some order and still avoid deadlock. This order is called *safe sequence*.
- A deadlock state is not a safe state. However, a unsafe state is not always a deadlock state. See Figure 7.4.
- See the example in the middle of Page 218.
- The idea is to ensure that the system always remains in a safe state.
- See page 220 for the banker's algorithm.

Deadlock Detection

- When each resource type has a single instance
  - We can check if there exists a cycle in the resource allocation graph to determine if a deadlock has occurred.
  - The resource allocation graph can be converted to a *wait for graph*.
  - see Figure 7.7 in page 225.
- When each resource type has more than one instance.
  - Use an approach similar to the safety check in the deadlock avoidance algorithm.
  - See Figure 7.2 and 7.3 for ideas.
  - See page 226 for the algorithm.
  - See page 227 for the example.
- How often is a deadlock detection algorithm invoked?
  - Every time a request is issued.
  - in a regular basis.

Recovery from Deadlock

- Aborting a process, and in turn the allocated resources will be released.
- Preempting resources. There are three issues:
  - Select a victim
  - rollback
  - prevent starvation.

Combined Approach to Deadlock Handling

- Partition resources into a number of classes.
- Order these classes.
- Processes must allocate resources in ascending order.
- Deadlock cannot involve more than one class.
- Within each class, the most appropriate technique for handling deadlocks can be used, e.g.
  - classes of internal resources: deadlock prevention through resource ordering.
  - classes of central memory: deadlock prevention through preemption.
  - classes of job resources: deadlock avoidance can be used.
  - classes of swappable space: preallocation can be used.

Chapter 8. Memory Management

Address binding

- When a program is in execution, the address of an instruction or variable must be determined. This process is called address binding.
- The binding of instruction and data to memory addresses can be done at one of the following ways:
    - compile time: If it is known at compile time where the process will reside in memory, then absolute code can be generated, e.g. the MS-DOS .com format programs are absolute code bound at compile time.
    - load time: The compile generates relocatable code. The final binding is delayed until load time.
    - execution time: If the process can be moved during its execution from one memory segment to another, then binding must be delayed until run time. Special hardware is needed for this purpose.

What is dynamic loading:

- a routine is not loaded until it is called. The main program is loaded first.
- The advantage is that an unused routine is never loaded.

What is dynamic linking (or so called DLL):

- Linking (to system library) is postponed until execution time.
- With dynamic linking, a stub is included in the image for each library-routine reference.
- When a stub is executed, it will check if the needed routine is already in memory. If so, the stub will be replaced with the address of the routine.
- Advantage:
    - A popular routine has only one copy in memory in a time-sharing environment.
    - When a library routine is updated, there is no need to recompile all programs that use this routine.

What is overlay?

- Some routines used in a program are mutual exclusion and will not be used at the same time.
- See page 243 and Figure 8.2 in page 244.
- This is usually used by low-end microcomputer, e.g. MS-DOS based PC.

Logical Space v.s. Physical Space

- Logical address: an address generated by the CPU
- physical address: an address seen by the memory unit
- For execution time address-binding scheme, logical address and physical address may differ.
- The run-time mapping from logical to physical addresses is done by the memory-management unit (MMU), which is a hardware device.
- See Figure 8.3 for a simple case.

What is swapping?

- When context switch occurs, another process acquires both CPU and memory. In this case, some process may have to be swapped out to give space to the new process.
- In the environment of compile or loading address-binding, a process that is swapped out has to be swapped back into the same memory space. In execution time address-binding, it is possible to swap a process into a different memory space.
- Swapping requires a backing store, which is commonly a fast disk with enough storage to accommodate copies of all memory images of all concurrent processes.
- Swapping may take time, e.g. 100 ms for a 100K program if the disk bandwidth is 1MB/sec.
- MS Windows let users to decide which process to swap in.

Contiguous Allocation

- The memory needed by a process is allocated in a big continuous space.

- Initially, all memory is available for user processes, and is considered as one large block of available memory. See Figure 8.7 and 8.8 for example.
- How to find a hole to satisfy a request:
  - First fit
  - Best fit
  - Worst fit
- It has been shown that both first fit and best fit are better than worst-fit.
- External Fragmentation: enough total space exists to satisfy a request, but it is not contiguous. For example, in Figure 8.8(c), total space is enough for P5. However, neither hole is big enough.
- Internal Fragmentation: The system maintains a list of trivial holes that can never satisfy any request. See Figure 8.9.
- One solution to external fragmentation is *compaction*. However, compaction cannot be used in compile or loading time address binding scheme.
- See Figure 8.11
- Compaction can be done at swapping time.
- Another possible solution to permit the physical address space of a process to be noncontiguous.

Paging

- Physical memory is broken into fixed-sized blocks called frames. Logical memory is also broken into blocks of the same size called pages.
- Every address generated by the CPU contains two parts: page number (p) and page offset (d). The page number is used as an index into a *page table*. See Figure 8.12 for addressing and Figure 8.13 for paging model.
- With paging scheme, there is no external fragmentation because any free frame can be allocated to a process that needs it.
- However, we may have some internal fragmentation.
- Small page size may decrease the degree of internal fragmentation but may increase the number of entries in the page table.

- Each process has a separate page table. All the page tables must be available to the operating system. For example, if a system call made by the process refers to its logical address space, it must be mapped into physical address by the operating system.
- Besides, the operating system also maintains a frame table, indicating the status of each physical frame (i.e. free or occupied).

What is the structure of the page table?

- the content of the page table must be kept in the PCB.
- case 1: the page table is implemented as a set of dedicated registers. However, this approach is not satisfactory in modern computers since the required page table is usually big.
- case 2: the page table is kept in main memory, and a pointer to the page table is stored in a register called page-table base register (PTBR). When context switch occurs, only PTBR is updated.
- The problem with case 2 is the time required to access a user memory location. In this case, two memory accesses are needed to access a byte.
- The standard solution to this problem is to use a special, small, fast lookup hardware cache called associative registers. Each register consists a key and a value. The number of such registers varies between 8 and 2048.
- When a logical address is generated, its page number is presented to a set of associative registers. If the page is found, its frame number is immediately available. Otherwise, a search to the page table in the main memory is then conducted. See Figure 8.16.
- The percentage of times that a page number is found in the associative registers is called the hit ratio.
- Suppose hit ratio is 98%. It takes 20 ns to search the associative registers and 100 ns to access memory. Then the average time to access a byte is

  0.98 x 120 + 0.02 x 220 = 122 ns.
- For protection, two bits can be associated to each entry in a page table: read-write bit

and valid-invalid bit. See Figure 8 in page 265.

What if the logical address space is large (say 2^{32})?

- If the page size is 4k (2^{12}), then the page table may contain up to 1 million entries ($2^{32}/2^{12}$). Suppose each entry takes 4 bytes, each process will have 4 MB of physical space for the page table alone.
- In this case, we don‘t want to allocate the page table contiguously in main memory.
- One way is to use a two-level paging scheme, in which the page table itself is also paged.
- See Figure 8.18 in page 266 for example.
- For a system with a 64-bit logical address space, a two-level paging scheme is no longer appropriate. Note that if a page is of size 4K, then each page may contain only 2^9 entries since each entry may take 8 bytes.
- The SPARC architecture supports 3 level paging scheme, and the 32 bit MC68030 architecture supports a four-level paging scheme.
- In a three-level paging scheme, accessing a byte may take 4 memory access. However, with high hit ratio in cache, the performance may not be too bad.
- Note that more levels of paging schemes will not decrease the number of required pages for the page table of a process. It only fragments them.

Inverted Page Table for reducing the space needed for a page table.

- The inverted page table has one entry for each real page (frame) of memory.
- See Figure 8.20 in page 270 for example.
- Inverted page tables reduce the amount of physical memory needed to store this information.
- However, it increases the amount of time needed to search the table when a page reference occurs. The problem can be reduced by using hashing table to store the inverted page table.

Advantages of paging

- Elimination of external fragmentation

- Common code sharing
- See Figure 8.21 in page 271.

Segmentation

- In paging scheme, the user‘s view of memory is contiguous. However, in reality, the user may view memory as a collection of variable-sized segments, with no ordering among segments. See Figure 8.22.
- In this case, the user specifies each address by two quantities: a segment name and an offset.
- For simplicity, segments are numbered and a logical address is referred as

  <segment-number, offset>

Hardware support for segment

- The mapping from a logical address to a physical address is done via a *segment table*. Each entry in the segment table has a segment base and a segment limit.
- See Figure 8.23 and Figure 8.24.
- The segment table may be put in the memory with two registers, namely segment-table base register and segment-table length register, for specifying the starting address and the length of the segment table.
- Of course, a small cache can be used for improving performance.

Advantage of segmentation:

1. segment is a better unit for protection, e.g. read and executable for code.
2. segment is a better unit for sharing.
3. See Figure 8.25.

Subtle consideration for segmentation

- If a code segment contain references to itself. what logical address should be used?
- Solutions:
  1. all sharing processes define the shared segment to have the same segment number.

2. Code segments refer to themselves only indirectly.

- Segmentation may cause external fragmentation. Of course, the degree of external fragmentation is not as severe as contiguous allocation.
- Generally, if the average segment size is small, external fragmentation will also be small.

Segmentation with Paging

- The idea is to divide each segment into a set of pages.
- For GE MULTICS, see Figure 8.26 and Figure 8.27.
- OS/2 32 bit version
  - Each process can have up to 16K segments.
  - Each segment can have up to 4GB.
  - each logical address is as follows

    <selector, offset>
  - Each logical address can be converted to a linear address via a descriptor table.
  - The linear address is then mapped into a physical address via a two-level paging scheme.
  - See Figure 8.28 in page 295.

Chapter 9 Virtual Memory

What is virtual memory?

- Virtual memory is a technique that allows the execution of processes not completely in memory.
- This technique frees programmers from concern over memory storage limitation.
- Overlay and dynamic loading ease the restriction, but they require special effort by the programmers. Besides, it does not solve the problem completely.

The entire program is not needed in memory in many cases because

- Some part of the program is rarely executed, e.g., exception handler.
- Some of the allocated data is rarely used, e.g. array allocated with maximum length.
- Besides, not all parts of the program are needed at the same time.

- Virtual memory techniques allow an extremely large virtual memory to be provided to programmers when only a smaller physical memory is available.
- Virtual memory is commonly implemented by *demand paging*.

What is demand paging?

- When a process is to be executed, only some pages which are to be used are swapped in.
- Note that if a page of a process is not in memory, it must be in the backing store of swapping.
- Page table of a process has to record which pages are in memory and which are not.
- The procedure for accessing a word in memory is as follows:
    1. Check the appropriate entry in the page table to determine if the reference is valid or not.
    2. If the reference is invalid, terminate the process. If it is valid but the page is not in memory, we have to page it in.

3. Find a free frame in memory (by taking one from the free-frame list).
4. Schedule a disk operation to read the desired page into the newly allocated frame.
5. When the disk read is complete, modify the appropriate entry of the page table to indicate that the page is now in memory.
6. Restart the instruction that was interrupted by the illegal address trap.

- See Figure 9.4
- In an extreme case, the execution of a process starts with no pages in memory. Pages are brought into memory until it is required. This is called *pure paging*.
- Demand paging may not be as bad as we think because the analysis of running processes show that programs tend to have *locality of reference*.
- Hardware support for demand paging:
  - page table: when a page is not in memory, a (valid/invalid) bit must record this information.
  - secondary storage: This memory holds those pages not in memory. It is known as swap space or backing store.
- A crucial requirement for demand paging is that an instruction that causes a page fault can be restarted after the page fault. If the page fault occurs on the instruction fetch when we are fetching an operand, we must re-fetch the instruction, decode it again, and then fetch the operand. This is no problems in most cases. The following instructions are the difficulties:
  1. moving a block of memory from one place to another: It causes problems when the source block overlaps with destination block.
  2. instructions with auto-decrement or auto-increment:

     MOV (R2)+, -(R3)
- Thus, demand paging cannot be applied to any system. Some hardware support is necessary.

9.3 Performance of Demand Paging

- Let *ma* be the memory access time (between 10 to 200 nanoseconds) and *p* the probability of a page fault. The effective access time is

  $(1-p) \times ma + p \times$ page fault time
- The operation for a page fault is shown in page 297 and 298.
- The page swap in time is

  seek time (15 ms) + latency time (8 ms) + transfer time (1 ms) = 25 ms. Suppose ma = 100 ns. The effective access time is

  $(1-p) \times 100$ ns $+ p \times 25$ ms
- To support 110 ns effective access time, we only allow 1 memory access out of 2,500,000 to page fault.

When to place the process image in the swapping space?

- page in from the swap space (or backing store) is usually done via direct disk I/O rather than the file system.
- Three options:
  1. copy the entire file image into the swapping space at process startup. This requires longer loading time.
  2. Demand pages are brought directly from the file system.
  3. (most common one) Initially demand pages from the file system, but to write the pages to swap space as they are replaced.

9.4 Page Replacement

- When a page is not in memory but there is no free frame, we find one that is not currently being used and free it. We can free a frame by writing its contents to swap space and changing the page table.
- See page 313 for the page fault service routine.
- See Figure 9.5 and page 9.6

- When a frame needs to be freed, it is not necessary to write the originally occupied page back to backing store. A modify (dirty) bit can be used to indicate whether a page has been modified or not since it is paged in.

9.5. Page-Replacement Algorithm

- How to evaluate a page replacement algorithm?
- Running it on a string of memory references and compute the number of page faults.
- Where is the string come from?
  - artificially generated.
  - tracing a given system and recording the address of each memory reference.
- In fact, only the page number of a reference is needed.

1. FIFO Algorithm

- The pages in memory is maintained as a queue. When a page is brought into memory, it is inserted at the tail. The one at the top, if any, is to be replaced if necessary.
- See Figure 9.8
- Belady's anomaly: adding more memory creates more page faults. For example, with string 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5, 3 frames yield 9 page faults while 4 frames yield 10 page faults.

2. Optimal Algorithm

- An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms.
- It replaces the page that will not be used for the longest period of time.
- See Figure 9.10.
- In general, the optimal algorithm cannot be implemented because it needs future knowledge of the reference string.

3. LRU Algorithm.

- It replaces the page that has not been used for the longest period of time.
- See Figure 9.11.
- The LRU policy is quite good in performance from experimentation. The only problem is *how* to implement it *efficiently*.
  - *counter*: Each page-table entry is associated with a time. In this scheme:
    - Each memory access incurs writing the time to the entry in the page table.
    - When it comes to page-replacement, a scan to all entries in the page table is required.
  - stack: The referenced page numbers are doubly-linked as stack. In this scheme:
    - Each memory access involves removing a page and putting it on the top of the stack, which requires changing six pointers at worst.
    - When it comes to page-replacement, the LRU page can be easily determined.
  - Neither implementation of LRU is satisfactory.
- LRU will NOT suffer Belady's anomaly because it is a stack algorithm.
- A replacement algorithm is a stack algorithm if the set of pages in memory for n frames is always a subset of the set of pages that would be in memory with n+1 frames.
- For LRU, the set of pages in memory would be the n most recently referenced pages.
- LRU Approximation Algorithms
  - few systems provide sufficient hardware support for true LRU page replacement.
  - Additional-Reference-Bits Algorithm
    - A 8-bit byte and a reference bit is associated with each page. In this scheme,
      - A memory reference will automatically set its reference bit by hardware.

        - At regular intervals, a timer interrupt transfers control to the operating system. The OS will shift the 8 bits right, move the reference bit to the most significant (8'th) bit, and then reset the reference bit.
        - When it comes to replace, the page with the smallest value on the 8 bits byte will be selected.
    - Second-Chance Algorithm
        - This algorithm basically follows FIFO. However, if a page is encountered with reference bit set. It is skipped in the hope that there exists other page with reference bit unset. In this scheme,
            - A memory reference will automatically set its reference bit by hardware.
            - When a page is swapped in, it is added to the tail of a circular queue.
            - When it comes to page replacement, the os advances the queue until a page with 0 reference bit is found. As it advances, it clears the reference bits. See Figure 9.13.
    - Enhanced Second-Chance Algorithm
        - Modify bit as well as the reference bit is considered.
        - It tries to find out a page with the lowest value on the combination of reference bit and modify bit.
4. LFU Algorithm: The least frequently used algorithm.
5. MFU Algorithm: The most frequently used algorithm.

Neither LFU nor MFU are common, and their performance are not quite good from experimentation.

What is the maximum number of frames allocated to each process?

- the amount of available physical memory.

What is the minimum number of frames that must be allocated?

- enough frames to hold all different pages that any single instruction can reference.
- For example, mv (m1), (m2) need six frames because this instruction may span two words that reside on two consecutive pages. (m1) and (m2) are both indirect access, and each needs two pages in worst case.
- It is defined by the architecture.

Allocation algorithms

1. equal allocation.
    - Suppose there are totally 62 free frames, and two processes. One needs 10K and the other needs 127K. Each process will be allocated 31 frames.
2. proportional allocation
    - Allocate available memory to each process according to its size.
    - In the previous example, the 10K process will be assigned $10/(127+10) \times 62 = 4$ frames. The 127K process will be assigned $127/(127+10) \times 62 = 57$ frames.
3. priority allocation
    - The same as proportional allocation except that the ratio of frames depends not on the relative sizes of processes but on the processes‘ priority, or the combination of size and priority.

Which page to be replaced, the page of the same process or the page of another process?

1. global replacement: the replaced page may belong to another process.
    - The problem is that a process cannot control its own page-fault rate. The set of pages in memory depends on the behavior of other processes.
2. local replacement: the replaced page must belong to the requesting process.
    - The number of frames allocated to a process doesn‘t change.
    - Some less used pages cannot be available to other processes.

- Global replacement usually performs better.

What is thrashing?

- When the paging activity is high, this phenomenon is called thrashing.
- A process is thrashing if it is spending more time paging than executing.

What causes thrashing?

- Suppose an operating system is monitoring CPU utilization. Whenever it finds CPU utilization is low, the degree of multiprogramming is increased by introducing a new process to the system.
- As more processes are in the system, fewer frames are allocated to each process. Thus, processes incur higher page faults. Most of processes will queue up for the paging device. The CPU ready queue becomes empty, and CPU utilization decreases.
- Global replacement algorithms introduce severe thrashing.
- See Figure 9.14

How do we prevent thrashing?

- We must provide a process as many frames as it needs.

How do we know how many frames a process needs?

- It is observed that the execution of a process moves from locality to locality.
- When a subroutine is executed, memory references include instructions of the subroutine, local variables, and a subset of global variables. When the subroutine ends, the process leaves this locality.
- Local model is the unstated principle behind the caching discussion.
- With enough space for locality set, a process will not fault after all locality pages are in memory.
- See Figure 9.15.

Working Set Model:

- Assume a locality can be covered by the most Δ page references. The set of pages in the most recent Δ page references is called working set. The size of the working set of process i is denoted as WSS_i.
- The accuracy of the working set depends on the selection of Δ. If is too large, it may span several localities, which wastes memory. If it is too small, it will not encompass the entire locality.
- If $\Sigma \mid WSS_i \mid >$ available frames, thrashing will occur.
- When there are extra frames, another process can be initiated.
- The working-set strategy prevents thrashing while keeping the degree of multiprogramming as high as possible.
- See Figure 9.16.

How to decide the working set size of a process (with Δ being 10000 memory references) ?

- Use three reference bits on each page. Timer interrupt occurs every 5000 references. The first bit records whether the page is referenced from the last timer interrupt until now. The second bit records whether the page is referenced between the last and the 2‘nd last interrupt. The third bit records whether the page is referenced between the 2’nd last the 3rd last interrupts. Thus, if a page is 0 for all three bits, it is not referenced within the previous 10,000 references. The working set size can be measured as the number of pages whose three reference bits are not all 0.

Second strategy for preventing thrashing:

- The page-fault frequency strategy monitors the page fault rate of each process to prevent it from becoming too high or too low.
- If the page fault becomes too low, a frame is removed from the process.
- if the page fault becomes too high, the process will be given a free frame. If no free frame is available, the entire process is swapped out.

Chapter 13 Secondary Storage Structure

Disk Structure:

- Information on the disk is referenced by <cylinder#, surface#, sector#>.
- I/O transfers between memory and disk are performed in units of one or more sectors, called blocks.
- Typically, block addresses increase through all blocks on a track, then through all the tracks in a cylinder, and finally from cylinder 0 to the last cylinder on the disk.
- Suppose a block size is the same as a sector size. A disk address of <i, j, k> generates a block number b:

  $$b = k + s \times (j + i \times t),$$

  where s is the number of sectors in a track, and t is the number of tracks in a cylinders.
- To access a block on the disk, the needed time is
  seek time (head movement time) + latency time (head rotation time) + transfer time.

Disk Scheduling Algorithms

1. FCFS Scheduling
   - See Figure 13.1 for an ordered disk queue of
     98, 183, 37, 122, 14, 124, 65 and 67.
   - Since seek time is the most time-consuming part of disk access. The aim is to reduce the total head movement.
2. SSTF (Shortest-seek-time-first) Scheduling
   - It selects the request with the minimum seek time from the current head position.
   - See Figure 13.2.
   - It may cause starvation.
   - SSTF is not optimal. Serving 53->37->14->... creates a shorter total seek time.
3. SCAN Scheduling

- The head starts at one end of the disk and moves toward the other end, serving requests as it reaches each track.
- See Figure 13.3.
- It is also called the elevator algorithm.

4. C-SCAN (Circular SCAN) Scheduling
    - When it reaches the other end, it immediately returns to the beginning of the disk without serving any requests on the return trip.
5. LOOK Scheduling
    - The head is only moved as far as the last request in each direction.
    - See Figure 13.4.

Which disk scheduling algorithm should be selected?

- SCAN or C-SCAN are more appropriate for systems with heavy load on the disk.
- Whether an algorithm is good is influenced by the file-allocation method.
- Accessing a contiguous file will generate several requests that are close together on the disk.
- Accessing a linked or indexed file may include blocks that are widely scattered on the disk. See Figure 11.6 in page 379 for the UNIX file inode structure.
- Some systems incorporate head scheduling algorithm on the disk control card. Thus, the operating system can do nothing about it.

What is disk format?

- physical format: Break the disk into sectors. Each sector has a header containing the sector number and space for an error correcting code.
- Most hard disks come physically formated from the factory.
- logical format: Writing FAT (file allocation table), inodes, free space lists and other information the system needs to track the disk contents.

How the bootstrap is done?

- When the machine is powered on, the program in ROM is executed. Its execution will transfer a code located in the disk boot blocks into memory and start executing the new code.
- The boot blocks are located at a fixed location on a disk.

File System Implementation

- Most materials are from *Modern Operating Systems*, Chapter 4, by A. S. Tanenbaum.
- This chapter concerns about how files and directories are stored, how disk space is managed, and how to access a file.
- Files can be implemented in three ways: contiguous, linked list, Linked List Allocation Using an Index , and I-nodes.

Contiguous Allocation

- This is the simplest way, in which each file is stored as a contiguous block of data on the disk.
- The advantages with this scheme include 1) easy to implement, and 2) efficient continous data access.
- The disadvantages are 1) it is not feasible to know the maximum file size at the time the file is created, and 2) disk is fragmented.

Linked List Allocation

- Each file is kept as a linked list of disk blocks.
- The directory entry only needs to store the disk address of the FIRST block.
- The disadvantage is that randomly accessing a block incurs sequentially reading blocks starting from the first block.

Linked List Allocation Using an Index

- The pointer part in linked list allocation is taken from each disk block and put in a table or index in memory. See Figure 4-10 and 4-11.
- When randomly accessing a block, though a chain must still be followed, it is much faster because the chain is entirely in memory.
- The primary disadvantage is that the entire table must be in memory all the time to make it work. Consider a 4G disk with 4K block size. There will be 1M blocks. Suppose each block entry takes 4 bytes, the table takes 4 MB memory.

Index Node (*or called I-node*)

- an inode contains
    - owner, group id

    - file type
    - file access permission
    - file access times
    - file size
    - table of contents
- The table of content of an inode has up to three level of indirection and keeps track of which blocks belong to it. See Figure 4-12.
- UNIX uses this scheme.

The main function of the directory system is to map the ASCII name of the file onto the information needed to located the data (consider the above four schemes in deciding what information should be contained in a directory entry).

Directories in MS-DOS
- MS-DOS uses linked list allocation with index for storing files.
- Each directory entry contains file name, the number of the first disk block, among others. See Figure 4-14 for the structure of MS-DOS directory entry.

Directories in UNIX
- UNIX uses i-nodes for storing files.
- Each direcotory entry contains just a file name and its i-node number. All the infomration about the type, size, times, ownership, and disk blocks is contained in the I-node.
- See Figure 4-16 for steps in looking up /usr/ast/mbox.
- It is obvious that looking up a path takes a number of disk access.

File Sharing
- With file sharing, the file system itself will look like a directed acyclic graph, rather than a tree.
- There are two approaches in dealing with file sharing, namely hard link, and soft link (or called symbolic linking)

Hard Link
- In hard link, when a file is shared by another path name, only a directory entry is inserted in each directory in the path. A count is associated with the I-node of each file to record the number of paths that point to the file.

- The advantages include
  - Quick link operation
  - Quick path traversal operation
- The disadvantages include
  - You never know a file is really deleted. (A file is really deleted by the system only when count=0).
  - Owner of a file is obscure.

Soft Link

- In soft link, when a file is shared by another path name, a new I-node of type LINK is created that contains the path name, in addition to a directory entry that is inserted in each directory in the path.
- The advantages include
  - A file has an owner and only owner can delete the file.
  - The path name is resolved at run-time so that soft link can be used to link to any machine in the world.
- The disadvantages include
  - An extra I-node is needed for each soft link
  - Slow path traversal operation.

What happens when a file is opened several times?

- See Figure ?

Cache in File System

- To further accelerate the disk block performance, a buffer cache is usually employed.
- With cache, a potential problem of inconsistency is introduced when a disk block is modified in buffer but not written to the disk.
- UNIX uses an approach that divides the disk blocks into several categories with different priorities, e.g. I-node blocks, directory blocks, indirect blocks, data blocks, etc. More critical blocks are written back more quickly.
- Real LRU can be employed.

When a data block should be written?

- Write-through cache, used by MS-DOS, incurs bad performance.
- Periodically write through (30 sec.), used by UNIX, has better performance.

In addition, to further improve the disk performance, blocks of a file should be allocated to be as close as possible.

Chapter 19 Protection

What is a protection?

- Protection refers to a mechanism for controlling the access of programs, processes, or users to the resources, including files, memory segments, CPU, semaphores, etc.
- A protection-oriented system provides means to distinguish between authorized and unauthorized usage.
- The role of protection is to provide *mechanism* for the enforcement of the *policies*.
- It is important to separate policies (what) from mechanisms how). Policies are likely to change from place to place or time to time. It is hoped that a change in policies require the modification of only some system parameters or tables.

Domain of Protection

- A process operates within a protection domain, which specifies the resources that the process may access.
- Formally, a domain is a collection of access rights, each of which is expressed as a pair <*object-name*, *rights-set*>. Domains need not to be disjoint. See Figure 19.1.
- To allow flexibility, the association between domains and processes may change overtime. For example, a process may need to read an object in phase 1 but write it in phase 2. Providing both read and write to that process provides access rights more than needed.
- A domain can be realized in a variety of ways:
- Each user may be a domain.
- Each process may be a domain.
- Each procedure may be a domain.
- In Unix, switching the domain corresponds to changing the user identification temporarily. Each file is associated with a owner-id and a domain bit, called *setuid* bit. Suppose there are two users A and B, and a program file P with owner-id and setuid being <B, 1>. When user A executes P, the user-id of the process will be temporarily changed to B.

Access Matrix

- The protection can be viewed as a matrix with rows being domains and columns being objects. See Figure 19.3 for an example.

- Policy decisions can be implemented by the access matrix.
- When a new object is created, a new column is added to the access matrix with the appropriate initialization entries dictated by the creator.
- To allow switching from domain D_i to D_j, an entry "switch" is specified on (D_i, D_j). See Figure 19.4 for an example.
- The ability to modify the access matrix can also be specified on the access matrix:
- Copy: (represented as * appended to an access right) allows the copying of the access right only within the column. See Figure 19.5. Two variants exist for copy:
    - Transfer
    - Limited copy
- Owner: owner on access(i, j) indicates that process executing in domain D_i can add and remove any right in any entry in **column** j. See Figure 19.6.
- Control: control on access(I, j) indicates that a process executing in domain Di can remove any access right from **row** j. See Figure 19.7.
- The access-matrix model meets the requirements of dynamic protection.

Implementation of Access matrix

1. Global Table: This is not a good design because this table, though sparse, is too big to be held in main memory.
2. Access List for Objects: Each column is represented as a string called access lists. To further improve the space utilization, defaults settings for all objects are not included in each access list but stored separately.
3. Capability Lists for Domains: Each row is represented as a string called access lists. A capability list cannot be directly accessed by processes executing in the domain. Access to a capability list has to be enabled via operating system.

Chapter 20 Security

- Protection is strictly an internal problem with subjects and objects being both inside the computer.
- Security, on the other hand, requires the consideration of external environment.
- Security violation is caused by either intentional (malicious) or accidental misuse.
- Some security problems are management problems, such as
  - Physical: the site containing the computer system must be physically secured.
  - Human: the chance of authorizing a user who then gives access to an intruder is low.
- This chapter deals with the operating system level security.

Authentication

- Authentication is to ensure that a user correctly identifies himself.
- The most common approach to authenticating a user identity is the use of *password*.
- Password can be revealed by the following ways:
  - Guessing (spouse/child/pet name/birthday, brute-force trying)
  - Accidentally explosed (visual or electronic monitoring)
  - Illegally transferred (multiple users sharing an account)
- If password is given the system, it becomes hard to remember and the user tends to write it down.
- If password is user-selected, it may becomes easy to guess.
- UNIX uses encryption techniques to encode the password entries. That is, the system only stores the encoded password so even the super-user does not know the password of any user. However, there has been some public software in the internet that can be used to GUESS the passwords given the password file. Thus, new versions of UNIX now hide the password entries.

Program Threats

- Trojan Horse: e.g., login simulator
- Trap Door: a hole in the software that benefits some user or damage the corporate. The hole may hide in the source code of either the software of the compiler.

System Threats

- Worms: A worm is a process that uses the spawn mechanism to colbber system performance. See Figure 20.1
- Viruses: A virus is a fragment of code embedded in a legitimate program.

- It is often spreaded by users downloading viral programs from public domain or exchanging floppy disks containing an infection.

Encription

- The most straightforward approach is for the sender to encrypt a message before sending and the receiver to decrypt the message after receiving.
- Both encryption algorithm and decryption algorithm have to agree on a common key in order to operate correctly. The problem then is how both parties get to know the common key.
- The solution to this problem is to use a public key-encryption scheme.
- In this scheme, each user must provide a public key which is know to anyone else in the community. Besides, each user also holds a private key that must be kept secret. When user A want to send a message to user B, the message is first encrypted by using the public key of user B. When user B receives the encrypted message, it is decrypted by using his own private key. The public-key, private-key pair can be defined as follows:

  $E(m)=m^e \bmod n = C$

  $D(C)=C^d \bmod n$,

  Where e and d are public key and private key respectively, and E(), D(), and n are known to everyone.

Computer Security Classifications

- Four divisions A, B, C, and D. Each division can be further divided into a number of classes.
- The classes from strict to loose are A1, B3, B2, B1, C2, C1, and D, where D denotes those that do not satisfy C1.
- C1: provide group protection, most UNIX are C1-class.
- C2: provide individual-level access control, NT and some UNIX are C2-certified.
- Division B: provide sensitivity labels to each object.

Appendix I: 考古題

- 85 年期中考
- 86 年期中考
- 87 年期中考
- 88 年期中考
- 85 年期末考
- 86 年期末考
- 87 年期末考
- 88 年期末考

作業系統

期中考

民國八十五年五月二日

注意：

1. 你有 120 分鐘的時間做答。
2. 不准做弊，否則後果嚴重。
3. 每一頁請都寫上自己的學號姓名。
4. 本試卷共 7 頁，請檢查。

1. (20%) 名詞解釋
   - process control block (PCB)
   - disk cylinder
   - coherency
   - priviledged instructions
   - thread

2. (20%)

A. Interrupt 是 computer architecture 中非常重要的一部份，請描述 interrupt 的動作。

B. 請敘述 timer interrupt 如何與 context switch 結合以達到 time sharing 的目的。

3. (20%)

The table below shows five processes, Pi, 1<= i <= 5, that need to be scheduled. (With everything else equal, resolve ties by process id. For round-robin scheduling, the incoming process goes to the beginning of the ready queue)

			Waiting Times			
			(i)		(ii)	
Process	Arrival Time	CPU Request	SJF	RR	SJF	RR
P1	0	2				
P2	4	3				
P3	15	4				
P4	3	6				
P5	0	8				
Average						

(i) For this part, ignore the arrival times, i.e. assume all processes have arrived at time 0. Draw Gantt charts for Shortest Job First (SJF) and Round Robin (RR with time slice being 1 unit), and then fill in the first two blank columns for the waiting time for each process and the average waiting time.

(ii) Take the arrival time in the table into consideration. Then do the same as in (i) and fill in the last two blank columns.

4. (15%)

For a semaphore s, define

init[s] = initial value of s

start_P[s] = the number of times wait(s) (i.e. P(s)) has been started

end_P[s] = the number of times wait(s) has been completed

end_V[s] = the number of times signal(s) (i.e. V(s)) has been completed

Determine which of the following equation holds:

1. end_P[s] = min(start_P[s], init[s] + end_V[s])
2. end_P[s] > min(start_P[s], init[s] + end_V[s])
3. end_P[s] < min(start_P[s], init[s] + end_V[s])

Please also briefly explain why you think it is correct.

5. (10%)
The following is an *incorrect* solution of using Test-and-Set to implement the semaphore operations wait(s) and signal(s). Please give a scenarion showing why it is incorrect.

```
procedure wait(s)
begin
    while Test-and-Set(lock) do no-op;
    while (s <= 0) do no-op;
    s := s - 1;
    lock := false;
end

procedure signal(s)
begin
    while Test-and-Set(lock) do no-op;
    s := s + 1;
    lock := false;
end
```

6. (15%)

The following is the solution of dining philosphers problem given in the textbook. To ensure the correct execution, each philosopher(i) must behave as

```
repeat
    pick(i);
    eat;
    pushdown(i);
until false
```

Suppose an uncooperative philosopher, say 0, behave as

```
repeat
    pushdown(0);
    eat;
    pick(0)
until false;
```

Which of the following requirements will be violated?

1. mutual exclusion
2. deadlock

For each violated requirement, give a scenarion showing how it is violated.

作業系統

期中考

民國八十六年五月一日

注意：

1. 你有 120 分鐘的時間做答。
2. 不准做弊，否則後果嚴重。
3. 每一頁請都寫上自己的學號姓名。
4. 本試卷共 3 頁，請檢查。

一. (40%) 簡答題

1. How “interrupt” impacts multiprogramming?
2. What’s the difference between a hard real time system and a soft real time system?
3. How is protection violated if the user is allowed to change the address in the interrupt vector?
4. Draw the process state transition diagram.
5. Why should the long term scheduler select a good process mix of I/O bound and CPU bound processes?
6. How are threads different from processes?
7. What are the advantages and disadvantages of user-level threads (compared to kernel level threads)?
8. What does multilevel feedback queue scheduling do?
9. What are the three requirements for the critical section problem?
10. What does the machine instruction “swap(lock, key)” do?

二. (25%)

The table below shows five processes, Pi, 1<= i <= 5, that need to be scheduled. (With everything else equal, resolve ties by process id.) Consider the scheduling algorithms FCFS, preemptive SJF, and RR (round-robin) with quantum 2. For round-robin scheduling, the incoming process goes to the tail of the ready queue, while the CPU gets the next process from the head of the ready queue.)

Process	Arrival Time	CPU Request
P1	0	2
P2	4	3
P3	15	4
P4	3	6
P5	0	8

(i) Draw Gantt charts for the three scheduling algorithms, and then calculate the **waiting time** for each process and fill in the table like the following. (15%)

Waiting time	FCFS	SJF	RR
P1			
P2			
P3			
P4			
P5			

(ii) Which scheduling algorithm yields the minimum average waiting time? (5%)

(iii) Which scheduling algorithm yields the minimum maximum waiting time? (5%)

三. (15%)

The following is the algorithm to enforce mutual exclusion by using *Test-and-Set()*.

```
repeat
    while Test-and-Set(lock) do no-op;
        critical section
    lock := false;
        remainder section
until false;
```

Give a scenario to illustrate why the above algorithm will not guarantee mutual exclusion if *Test-and-Set()* is not implemented as a single machine instruction.

四. (20%)

The following algorithm is for philosopher *i* in the dining philosopher problem (with totally five philosophers). It may cause starvation. Please modify it to prevent starvation.

```
var chopstick: array[0..4] of semaphore;

repeat
    wait(chopstick[i]);
    wait(chopstick[i+1 mod 5];
        eat
    signal(chopstick[i]);
    signal(chopstick[i+1 mod 5];
        think
until false;
```

作業系統

期中考

八十七年四月二十七日

注意：

1. 你有 100 分鐘的時間做答。
2. 不准做弊，否則後果嚴重。
3. 本試卷共 3 頁，請檢查。

一. (35%) 解釋名詞

1. context switch
2. privileged instructions
3. interrupt vector
4. system calls
5. bootstrap program
6. java virtual machine
7. PCB
8. thread (and what do different threads of the same task share)
9. CPU burst
10. real-time systems

二、(25%)

The table below shows five processes, Pi, 1<=i<=5, that need to be scheduled. (With everything else equal, resolve ties by process ids.) Consider the algorithms non-preemptive SJF, preemptive SJF, and multi-level feedback queue. For multilevel feedback queue, there are three queues, numbered from 0 to 2. The scheduler executes queue i only when queue i-1 is empty, 1<=i<=2. A process entering the ready queue is put in queue 0. The scheduling of processes at queue 1, 2, and 3 employs RR with time quantum 2, 4, and ∞ respectively. If a process in queue i does not complete within the time quantum, it is preempted and put into queue i+1.

Process	Arrival Time	CPU Request
P1	0	2
P2	4	4
P3	16	3
P4	3	8
P5	1	11

(i) Draw Gantt charts for the three scheduling algorithms, and then calculate the **waiting time** for each process and fill in the table like the following. (15%)

Waiting time	Non-preemptive SJF	Preemptive SJF	MLFQ
P1			
P2			
P3			
P4			
P5			

(ii) Which scheduling algorithm yields the minimum average waiting time? (5%)

(iii) Which scheduling algorithm yields the minimum maximum waiting time? (5%)

三、Suppose counter := counter + 1 is implemented in a machine language as follows: (15%)

load R1, *counter*
inc R1
store R1,*counter*

Initially, let the value of *counter* be 10. Without any coordination, it is possible that three concurrent processes that execute "counter:=counter+1" may make the value of *counter* become 11. Show a scenario that produces such an incorrect result.

四、In dining philosopher's problem, suppose we adopt the following strategy to prevent the occurrence of deadlock:

Allow a philosopher to pick up her chopsticks only if both chopsticks are available.

The following algorithm is designed to implement the above strategy. Initially, all the elements of chopstick are initialized to *true*, all the elements of block is initialized to *false*, and mutex is initialized to 1.

```
var chopstick: array[0..4] of boolean;
    mutex: semaphore;
    block: array[0..4] of semaphore;
repeat
    wait(mutex);
    if (chopstick[i] and chopstick[i+1 mod 5]) then
    begin
        chopstick[i] := false;
        chopstick[i+1 mod 5] := false;
        signal(mutex);
    end else
        signal(mutex);
        wait(block[i]);
    …
    eat
    …
    wait(mutex);
    chopstick[i] := true;
    chopstick[i+1 mod 5] := true;
    signal(mutex);
    signal(block[i-1 mod 5];
    signal(block[i+1 mod 5];
until false;
```

A. show a scenario illustrating how the above algorithm yields incorrect results. (10%);

B. Suppose you are allowed to use additional variables, please correct the above algorithm. (15%)

作業系統

期中考

八十八年四月二十二日

注意：

4. 你有 100 分鐘的時間做答。
5. 不准做弊，否則後果嚴重。
6. 本試卷共 3 頁，請檢查。

一. (30%) 簡答題

11. What is DMA and what is its purpose?
12. How does the operating system perform I/O operations via interrupt mechanism?
13. The PCB of a process maintains its stack pointer. Why does a process need a stack?
14. Explain the following UNIX commands: du, mv, and finger
15. What are CPU utilization, throughput, and turnaround time?

二、(20%)

The table below shows five processes, Pi, 1<= i <= 5, that need to be scheduled. (With everything else equal, resolve ties by process id. For round-robin scheduling, the incoming process goes to the beginning of the ready queue)

			Waiting Times			
			(i)		(ii)	
Process	Arrival Time	CPU Request	SJF	RR	SJF	RR
P1	0	2				
P2	4	3				
P3	15	4				
P4	3	6				
P5	0	8				

Average				

(i) For this part, ignore the arrival times, i.e. assume all processes have arrived at time 0. Draw Gantt charts for Shortest Job First (SJF) and Round Robin (RR with time slice being 1 unit), and then fill in the first two blank columns for the waiting time for each process and the average waiting time.
(ii) Take the arrival time in the table into consideration. Then do the same as in (i) and fill in the last two blank columns.

三、(15%)
The following is an algorithm for critical section problem.

```
repeat
    flag[i]:=true;
    while (flag[j] and turn=j) do no-op;
        critical section
    turn:=j;
    flag[i] := false;
        remainder section
until false;
```

Give a scenario to illustrate why the above algorithm is incorrect.

四、Consider the following code for semaphore operations:

```
wait(S):
    while S <= 0 do no-op;
    S:=S-1;
signal(S):
    S:=S+1;
```

Show that, if wait() and signal() operations are not executed atomically, then mutual exclusion may be violated.

五、Consider the following producer and consumer processes:

```
        producer                              consumer
repeat                                repeat
    produce an item in nextp;             wait(mutex);
```

```
        wait(mutex);                        wait(full);
        wait(empty;                         --- remove an item from buffer to
        --- add nextp to buffer ---     nextc;
        signal(mutex);                      signal(mutex);
        signal(full);                       signal(empty);
until false;                                --- consumer the itme in nextc;
                                        until false;
```

What is the problem with the above program? Give a scenario for illustration.

作業系統

期末考

民國八十五年六月二十七日

注意：

1. 你有 120 分鐘的時間做答。
2. 不准做弊，否則後果嚴重。
3. 每一頁請都寫上自己的學號姓名。
4. 本試卷共 6 頁，請檢查。

1. (24%) 多選題
   1. Which of the following schemes restrict that a process that is swapped out must be swapped back into the same memory space that it occupied previously?
      A. compile time address binding
      B. load time address binding
      C. execution time address binding
      D. none of the above
   2. Which of the following memory management approaches may incur external fragmentation?
      A. contiguous allocation
      B. paging
      C. segmentation
      D. none of the above
   3. Which of the following memory management approaches may incur internal fragmentation?
      A. contiguous allocation
      B. paging
      C. segmentation
      D. none of the above
   4. Which of the following schemes can help reduce the space required for the page table:
      A. multi-level paging
      B. TLB

C. inverted page table

D. none of the above

5. Which of the following algorithm may introduce Belady‘s anomaly?

   A. FIFO

   B. Optimal

   C. LRU

   D. LRU approximation algorithms

6. Which of the following disk scheduling algorithms may cause starvation?

   A. FCFS

   B. SSTF

   C. SCAN

   D. C-SCAN

   E. none of the above

7. As a system administrator, when you notice that the CPU utilization in your system is low, what can you do to increase the CPU utilization?

   A. buy more memory

   B. increase the degree of multiprogramming

   C. decrease the degree of multiprogramming

   D. replace the disk by a faster one

   E. none of them will always work

8. Which of the following approaches can be used to reduce the chance of illegal use of your data?

   A. protect your files/directories appropriately

   B. use a dictionary word as your password to help memory

   C. Incorporate at least a punctuation character in your password

   D. use a short password

2. (16%)

Use Banker‘s algorithm to determine if the following state is safe or unsafe. If it is safe, show a sequence of transitions by which all processes complete. If it is unsafe, show how it is possible for deadlock to occur. System contains four resources and five processes

STATE: Available = (1 1 0 1).

Process	Current Allocation	Remaining Need
P1	0 1 4 3	2 3 3 0
P2	2 3 1 2	4 2 2 2
P3	2 0 2 0	0 0 0 1
P4	0 1 3 2	7 3 1 4
P5	2 1 1 1	0 1 2 0

3. (20%)

In a system with 2-level page table, suppose the memory access is 80 ns, and the probability that a page number is found is the associative registers is 95%. Besides, it takes 20 ns to search the associative registers.

(1) What is the effective memory access time?

(2) Assume this system also employs virtual memory. The average page-fault service time is 30 ms. If we require the average memory access be no more than 120 ns, what should the page fault rate be?

4. (20%)

1. 試簡述以下各機制的好處：(15%)
   (1) dynamic linking
   (2) separation of logical and physical space
   (3) multi-level page table
   (4) LRU approximation algorithms
   (5) authentication
2. In a virtual memory system, what are the minimum number and maximum number of pages required by each process? (5%)

5. (20%)

Four options, *switch*, *copy*, *owner*, and *control* can be incorporated into the accss matrix to allow controlled change to the contents of access matrix. What do they mean? Give a simple example in illustrating each option.

作業系統

期末考

民國八十六年六月十九日

注意：

1. 你有 120 分鐘的時間做答。
2. 不准做弊，否則後果嚴重。
3. 本試卷共 3 頁，請檢查。

一. (20%) 名詞解釋

1. UNIX shared memory
2. UNIX wait/signal
3. UNIX pipe
4. NT heaplock
5. NT thread

二. (20%) 填充題

1. Necessary conditions for deadlocks: _____,
   _____,
   _____, and
   _____.
2. Which fragmentation(s) (external or internal) can paging scheme incur: _____.
3. Name two page replacement algorithms that do not incur Belady's anomaly: _____ and _____.
4. What is the maximum number of indirect blocks needed to access a block in a UNIX file system? _____.
5. Name two operations to control the change to the contents of the access matrix entries: _____ and _____.

三. (10%) Why are the page sizes always power of 2?

四. (15%) Consider a demand-paged computer system where the degree of

multiprogramming is currently fixed at four. The system was recently measured to determine utilization of CPU and the paging disk. The results are one of the following alternatives. For each case, what is happening? Can the degree of multiprogramming be increased to increase the CPU utilization?

a. CPU utilization 13%; disk utilization 97%

b. CPU utilization 87%; disk utilization 3%

c. CPU utilization 13%; disk utilization 3%

五．(15%) Suppose that the head of a moving-head disk with 200 tracks numbered 0 to 199, is currently serving a request at track 143 and has just finished a request at track 125. The queue of requests is kept in the FIFO order:

86, 147, 91, 177, 94, 150, 102, 175, 130

What is the total number of head movements needed to satisfy these requests for the following three disk-scheduling algorithms?

a. FCFS

b. SSTF

c. SCAN

六．(10%) Consider a computing environment where a unique number is associated with each process and each object in the system. Suppose that we allow a process with number n to access an object with number m only if $n > m$. What type of protection structure do we have?

七．(10%) Suppose you are the director of our departmental computer center. One day, an assistant bursts into your office screaming: “Some students have discovered the algorithm our workstations (running UNIX) use for encrypting passwords and posted it on the BBS.” What should you do?

作業系統

期末考

民國八十七年六月十七日星期三

注意：

1. 你有 100 分鐘的時間做答。
2. 不准做弊，否則後果嚴重。
3. 本試卷共 3 頁，請檢查。

一. (30%) 問答題

1. What is NT thread, and what operations are provided by NT to manipulate threads?
2. What is NT semaphore, and what operations are provided by NT to manipulate semaphores?
3. What is NT heap, and what operations are provided by NT to manipulate heaps?
4. What are MS ActiveX and DCOM designed for?
5. What are the differences between Java script and Java applet, and what are the advantages of one against the other?

二. (32%) 多選題

1. Which of the following resources can be preempted and restored later?
   A. CPU registers
   B. Memory space
   C. Printers
   D. Tape drives
2. Which of the following descriptions about resource allocation graphs are true?
   A. If each resource type has only one instance, a deadlock implies a cycle.
   B. If each resource type has only one instance, a cycle implies a deadlock.
   C. If each resource type has more than one instance, a deadlock implies a cycle.
   D. If each resource type has more than one instance, a cycle implies a deadlock.
3. For dynamic linking (or called DLL), when is the linking conducted?
   A. Compile time
   B. Load time

C. Execution time

4. Which of the following allocation scheme may incur external fragmentation?
   A. Contiguous allocation
   B. Paging
   C. Segmentation
5. In a paging system with two level page tables, suppose hit ratio is 98% and it takes 20 ns to search the associative registers and 100 ns to access memory. What is the average to access a word?
   A. 120 ns
   B. 122 ns
   C. 124 ns
   D. 126 ns
   E. none of the above
6. Which of the following descriptions about LRU are correct?
   A. It is a stack algorithm.
   B. It has the lowest page-fault rate of all algorithms.
   C. Without any hardware support, it is difficult to implement efficiently.
   D. Few systems provide sufficient hardware support for the true LRU page replacement.
7. If the working-set window, Δ, is set to cover more than one locality, what will happen?
   A. thrashing
   B. low degree of multiprogramming
   C. high degree of multiprogramming
   D. high disk utilization
8. In a disk system, suppose each cylinder has 10 tracks, and each track has 100 sectors. What is the disk address of the block 2975?
   A. <2, 9, 75>
   B. <29, 7, 5>
   C. <2, 97, 5>
   D. <2, 5, 97>

三. (20%) Consider a demand-paging system with a paging disk that has an average access and transfer time of 20 miliseconds. Addresses are translated through a (1-level) page table in main memory, with an access time of 1 microsecond per memory access. Thus, each memory reference through the page table takes two access. To improve this

time, we have added an associative memory that reduces access time to one memory reference, if the page-table entry is in the associative memory. Assume 80% of the accesses are in the associative memory and that, of the remaining, 10% (or 2% of the total) cause page faults. What is the effective memory access time?

四. (18%) Suppose that the head of a moving-head disk with 200 tracks numbered 0 to 199, is currently serving a request at track 143 and has just finished a request at track 125. The queue of requests is kept in the FIFO order:

86, 94, 150, 147, 91, 177, 102, 175, 130

What is the total number of head movements needed to satisfy these requests for the following three disk-scheduling algorithms?

a. FCFS

b. SSTF

c. SCAN

作業系統

期末考

八十八年六月二十二日

注意：

7. 你有 100 分鐘的時間做答。
8. 不准做弊，否則後果嚴重。
9. 本試卷共 2 頁，請檢查。

一. (30%) 簡答題

16. What are the advantages of dynamic linking over dynamic loading?
17. What are the advantages of 2-level paging over 1-level paging?
18. What are the advantages of inverted page table scheme over paging scheme?
19. What are the advantages of i-node over linked list allocation using an index?
20. What are the advantages of public-private key scheme over common key scheme?

二、(20%)

In a system with 2-level page table, suppose the memory access time is 80 ns, and the probability that a page is found in the associative memory is 95%. Besides, it takes 20 ns to search the associative memory.

(1) (5%) What is the effective memory access time?

(2) (15%) Assume this system also employs virtual memory. The average page-fault service time is 30 ms. When a page is not found in the associative memory, the probability that this page is NOT is main memory is 0.1%. What is the average memory access time in this case?

三、(20%)

Consider the two-dimensional array A:

var A: array[1..100] of array[1..100] of integer;

where A[1][1] is at location 100, in a paged memory system with page size 100. Assume an integer takes one unit of space. A small process is in page 0 (location 0 to 99) for manipulating the matrix; every instruction fetch will be from page 0. From three page frames, how many page faults are generated by the following array-initialization loops,

using LRU replcaement, and assuming page frame 1 has the process in it, and the other two are initially empty? Note we assume array A is allocated in a row-majored manner.

(1) For j:=1 to 100 do
    for i:=1 to 100 do
        A[i][j]:=0;

(2) For i:=1 to 100 do
    for j:=1 to 100 do
        A[i][j]:=0;

四、(15%)

Suppose you have a very slow disk in your computer. The disk has only one surface and one moving head, and the capacity and speed information is as follows:

1 sector = 2 KB.
1 track = 200 sectors
1 surface = 1000 tracks.
transfer rate = 1 MB/sec
seek time from one cylinder to another = 10 ms
rotation time = 0 (i.e. this can be ignored)

(1) (5%) What is the size of the capacity of your disk (in terms of Mega bytes)?

(2) (10%) Suppose you have a video data 10MB stored continuously on the disk. The required playback rate to the video data is 2MB/sec. Obviously, your hard disk cannot serve this request in real time. One way to serve the request is to first prefetch enough video data to the main memory and then start playing. Suppose your computer has enough main memory to buffer the video data. Initially, the buffer is empty. After you issue the request to playback the video, how much time at least do you have to wait before the video playing can be started? (Please ignore all other factors such as communication time.)

五、(15%)

Suppose the table of content in a i-node can hold 64 entries, and each indirect block can hold up to 256 entries. Each page is 1KB. In a UNIX system with three level indirect blocks, what is the maximum size of a file?

六、

(1) Give an example on how to use Java for inheritance.

(2) Give an example on how to use Java to throw an exception.

(3) Give an example on how to use Java to invoke security manager.

(4) Give the html syntax on how to invoke a Java applet.

(5) What is the system call in UNIX (NT) to create a thread?