# Review

## 1 Overview

### 1.1 The definition, objectives and evolution of operating system

An operating system *exploits* and *manages* all kinds of computer *hardware* to *provide a set of services* directly or indirectly to the users. The definition gives two objectives of an operating system:

- managing efficiently all kinds of resources
- providing a friendly interface to the end-users

To better understand the requirements for an operating system, it is useful to consider how operating systems have evolved over the years, from *serial processing systems* involving many manual operations to *automatic multiprogramming batch processing systems* and *time-sharing systems*.

### 1.2 Computer hardware

All the hardware components in a computer system may fall into several categories: *microprocessor*, *main memory*, *I/O modules* and *system bus*.

For exchanging data to or from the main memory and various I/O modules and computation, the microprocessor contains all kinds of registers, including *data registers*, *address registers*, *control and status registers*.

- Different address registers may use different addressing schemes, including *segmented addressing* and *stack addressing* based on *push* and *pop* operations.
- The *program counter* register (PC) contains the address of an instruction to be fetched from main memory and the *instruction register* (IR) contains the instruction most recently fetched.

- A register called *program status word* (PSW) otherwise records the condition codes as part of the result of computations and other status information as well.

The dynamic side of the microprocessor is how it executes instructions. For the execution of each single instruction, there are two cycles, the first of which is called *fetch cycle* and the second *execute cycle*. They together makes an *instruction cycle*. The microprocessor repeats instruction cycles until it halts.

### 1.3 I/O communication techniques

There are three common I/O device access techniques: *Programmed I/O*, *Interrupt-driven I/O*, and *DMA* and the latters are developed to avoid problems in the former approaches.

*Interrupt* is a mechanism by which computer components may interrupt the normal processing of the processor and request the processor to perform a specific action. Whenever the execution of an instruction is finished, the processor will check the availability of any interrupt signal.

### 1.4 Memory hierarchy

Storage components all have some kinds of advantages and disadvantages. It is possible to combine them hierarchically so as to enjoy high speed, large capacity and low price. Typically a cache may be used between the processor and the main memory to have a speedup based on the *principle of locality*.

## 2 Process management

### 2.1 The definition of process

A *process* is an execution of a program. The term is coined to refer to an instance of a program in a multiprogramming environment.

Each process contains *data*, *code*, and its own *stack*. The operating system allocates a *process control block* to record various attributes of each process. All the above together make the *process image*.

### 2.2 State transition

The *state transition model* is used to describe the dynamic behavior of a single process, and the *process queue model* may depict the structure of the process management subsystem in the operating system.

## 3 Thread

### 3.1 The introduction of thread

The characteristics of processes actually fall into two categories:

- **Resource ownership**: Processes may be allocated control or ownership of various resources.
- **Scheduling/execution**: Processes may execute and move from one state to another.

The two categories are independent in fact and thus may be treated separately. Multiple flows of control may coexist within one single process, each called a *thread*.

Similar to process, each thread is associated with *execution state* and various data structures, including a *thread control block* containing thread context, a *stack* and *space for local variables*.

### 3.2 Benefits of multithreading

Multithreading has the following advantages over multiprogramming:

- It takes far less time to create and terminate a thread, and do control switching between threads.
- It is easier to accomplish the communication among execution traces.
- It helps to reflect the real world in a straightforward way.

Two examples, a web server and RPC service, clearly shows the above benefits of multithreading.

### 3.3 Implementation of thread

Threads may be supported in two different ways:

- *At the user-level*: All thread management work is done by the user application, and the operating system is not aware of the existence of threads and typically a *thread library* is available providing routines for manipulating threads and communication between threads as well.
- *At the kernel-level*: The operating system kernel provides direct support for thread management. Both approaches have their advantages and disadvantages.

## 4 Concurrency

### 4.1 Mutual exclusion and synchronization

#### 4.1.1 Introduction

Besides offering benefits, multiprogramming and multithreading also mean more efforts to deal with the interaction among processes, including *mutual exclusion* and *synchronization*, which are required by processes either competing for or cooperating with each other by shared resources or communications.

Mutual exclusion has four requirements:

- Only one process at a time is allowed into its critical section, among the processes that have critical sections for the same resource or shared object.
- When no process is in a critical section, any process that requests entry to its critical section must be permitted to enter without delay.
- A process remains inside its critical section for a finite time only.
- No assumptions should be made about relative process speeds or number of processors.

#### 4.1.2 Software approaches

Based on the assumption that only one access to a memory location can be made at a time, Dekker's algorithm and Peterson's algorithm were developed, so that a user program may implement mutual exclusion and synchronization itself without any support from the operating system or programming languages.

#### 4.1.3 Hardware approaches

Since it is interrupts that lead to the interleaving execution of multiple processes, mutual exclusion can be enforced by *disabling interrupt* temporarily.

Some special machine instructions, which combines two memory accesses, may meet the requirement, such as *test and set instruction* and *exchange instruction*.

#### 4.1.4 Programming languages or operating system support

Since either software or hardware approaches have all kinds of disadvantages, e.g. high complexity for the former and busy waiting for the latter, some advanced facilities supporting mutual exclusion and synchronization have been developed:

- *Semaphores*

  A semaphore can be viewed as an integer variable, whose initial value indicates the number of resources of concern available. Processes may invoke two atomic primitives: *wait* and *signal*. And each semaphore is associated with a queue, which is used to hold processes waiting on the semaphore.

  The implementation of semaphores has to be based on either software approaches or hardware approaches, but semaphores provide an easier interface to the programmers.

- *Monitors*

  The monitor method is proposed based on the development of structural programming and object orientation theory. A monitor is actually a software module consisting of:

  - Local data: the shared resources or the access points leading to those resources
  - Procedures: the interface to the above local data, which is invisible to any external procedure
  - Initialization: code to initialize local data

At any time, at most one process is allowed to own the monitor and invoke one of its procedures, thus mutual exclusion is enforced. *Condition variables* and two primitives, *cwait()* and *csignal()*, may be used inside monitor procedures to provide synchronization support.

#### 4.1.5 Message passing

Message passing, common for processes that communicate with each other, may also be used to support mutual exclusion and synchronization.

Message passing is actually similar to semaphore signaling except that a mailbox is used for relaying messages.

### 4.2 Deadlock

In a multiprogramming environment, a deadlock may happen if and only if:

- *Mutual exclusion*: Only one process may use the shared resource at a time.
- *Hold and wait*: More than one resource is involved and they are to be obtained one by one. That is processes may hold allocated resources while awaiting assignment of others.
- *No preemption*: Once a resource is held by a process, it cannot be forcibly removed from the process.
- *Circular wait*: A closed chain of processes exists, such that each process holds at least one resource needed by the next process in the chain. In terms of graph theory, there is a directed circuit, in which processes and resources alternate exactly one by one.

Deadlock problems may be solved in various ways:

- *Prevention*: The occurrence of one of the above four conditions is prevented in advance.
- *Avoidance*: The progress of resource allocation in the operating system is monitored dynamically and whenever a deadlock is going to happen, some measure is taken to avoid it. The *banker's algorithm* may be used for determining the possibility of a deadlock.
- *Detection and recovery*: Instead of avoiding a deadlock, the operation system may performs an algorithm periodically to detect the existence of circular wait. If a deadlock has indeed occurred, effort may be taken for recovery.

## 5 Memory management

Each process is associated with two address spaces: a *logical address space* and a *physical address space*. The former can be translated into the latter in different ways, either in compile time, load time, or execution time.

Besides satisfying the address mapping, memory management subsystem also need to provide protection and sharing support, and facilitate application organization.

Memory may be allocated to processes in the forms of *partitions*, *pages*, or/and *segments*:

- Partitioning may be *fixed* or *dynamic*, and *internal* and *external fragments* may be caused respectively.
- Process spaces and the main memory may be divided into fixed-size blocks, called pages or frames so that a process image may reside in uncontiguous sections of memory space. A page table helps to track the location of each section and do the address translation in run-time.
- A process may be organized in form of segments as well and a segment table is used instead.

## 6 Virtual memory

Virtual memory is needed to support the execution of a program whose size is beyond the capacity of the main memory so that only part of a process instead the whole thing may be physically reside in the main memory for execution.

To support virtual memory, additional information is needed for either paging or segmentation mechanisms. The two methods may be also combined to benefit the advantages of both.

With virtual memory where paging is involved, various policies should be considered: *fetch policy*, *placement policy*, *replacement policy*, and *cleaning policy*. Various replacement policies, including *optimal*, *LRU*, *FIFO* and *clock*, have been discussed to reduce the occurrence of page faults.

## 7 Uniprocessor scheduling

Three types of scheduling are involved in an operating system: *long-term scheduling*, *medium-term scheduling*, and *short-term scheduling*. The last one is the most common use of scheduling.

With short-term scheduling, various policies, *FCFS*, *round robin*, *shortest process next*, *shortest remaining time*, *highest response ratio next*, and *feedback*, may be used to meet criteria including *turnaround time*, *response time*, *throughput*, *processor utilization*, and *fairness*.

## 8 I/O management

The diversity of I/O devices makes the I/O management the messiest part in an operating system, however it is still possible to deal with the various I/O devices in a uniform way by

organizing the I/O routines hierarchically.

To improve the efficiency of the I/O subsystem in general, the buffering mechanism may be used so that the level of parallelism will be increased and swapping will not be interfered any more.

Disks, as the most often used peripheral devices, may be dealt with elegantly by deploying various scheduling algorithms for disk I/O requests, including *FIFO*, *shortest service time first*, *SCAN*, and *C-SCAN*.