Unit 2 : Computer and Operating System Structure

Lesson 1 : Interrupts and I/O Structure

1.1. Learning Objectives

On completion of this lesson you will know :

- what interrupt is
- the causes of occurring interrupt
- instruction cycle with interrupt
- I/O structure.

1.2. Interrupts

A method by which other events can cause an interruption of the CPU's normal execution. An Interrupt is a method by which the normal operation of the CPU can be changed. Interrupts are a better solution than polling for handling I/O devices. There are many methods to handle interrupts. Four general classes of interrupts are :

An Interrupt is a method by which the normal operation of the CPU can be changed.

- Program, trap instructions, page faults etc.
- Timer
- I/O devices and
- Hardware failure.

When an interrupt occurs a register in the CPU will be updated. When the CPU finishes the current execute cycle, and when interrupts are enabled, it will examine the register. If the register indicates that an interrupt has occurred and is enabled the interrupt cycle will begin, Otherwise it will be bypassed. The interrupt cycle will call some form of interrupt handler (usually supplied by the operating system) that will examine the type of interrupt and decide what to do. The interrupt handler will generally call other processes to actually handle the interrupt. The CPU follows the simple program outlined in the following diagram.

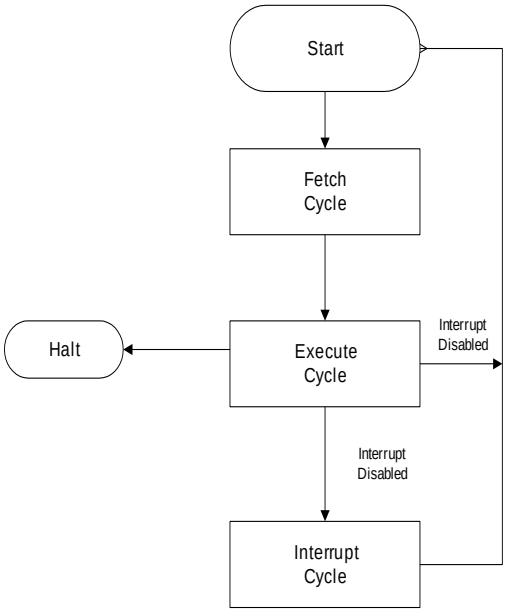


Fig.2.1 : Interrupt cycle with interrupts.

Interrupts are disabled when the operating system wishes to execute some code that must not be interrupted. Examples include interrupt handlers, semaphore operations. The following table describes the causes of occurring interrupts:

Table 2.1: The causes of occurring interrupts.

Interrupt Type	**Caused by...**
Program trap instructions page faults etc.	generated by some condition which is a result of program execution (error condition, system call).
Timer	generated by the system timer.
I/O	generated by I/O controller, signals completion of I/O task (either success or failure).
Hardware failure	power failure, memory parity error.

Simple Interrupt Processing

Steps for processing interrupts are shown below where steps 1 to 5 is done by hardware and from 6 to 9 is done by software

1. Interrupt occurs
2. Processor finishes current instruction
3. processor signals acknowledgment of interrupt
4. processor pushes program status word (Program Status Word) and program counter (Program Counter) onto stack
5. processor loads new Program Counter value based on interrupt

6. save remainder of process information
7. process interrupt
8. restore process state information
9. restore Program Status Word and Program Counter.

1.3. I/O Structure

One of the main functions of an OS is to perform all the computer's I/O devices. It issues commands to the devices, catch interrupts and handle errors. It provide an interface between the devices and the rest of the system. We will discuss the I/O hardware and I/O Software..

1.3.1. I/O Hardware

The I/O hardware is classified as

I/O Hardware

- I/O devices
- Device controllers and
- Direct memory access (DMA).

I/O Devices

Normally all input and output operations in operating system are done through two types of devices; *block oriented devices* and *character oriented devices*. A *block oriented device* is one in which information is stored and transferred at some fixed block size (usually some multiple of 512 bytes), each one with its own address. The block oriented device can read or write each block independently of all other ones out or expand.

I/O Devices

The *character oriented device* is one in which information is transferred via a character stream. It has no block structure. It is not addressable. For example, punched cards, terminals, printers, network interfaces, mouse etc.

The above classification scheme is not always true. Some device do not fit in. So, the idea of device driver was introduced. The idea of device driver is to provide a standard interface to all hardware devices. When a program reads or writes a file, OS invokes the corresponding driver in a standardized way, telling it what it wants done, thus decoupling the OS from the details of the hardware.

Device Controller

I/O units consist of mechanical and electronic components.

I/O units consist of mechanical and electronic components. The electronic component is called device controller. It is also called printed circuit card. The operating system deals with the controller.

The controller's job is to convert the serial bit stream into a block of bytes and perform any error connection necessary. The controller for a CRT terminal also works as a bit serial device at an equally low level.

Each controller has a few registers that are used for communicating with the CPU and these registers are part of the regular memory address space. This is a called memory mapped I/O. IBM PC uses a special address space for I/O with each controller allocated a certain portion of it. The following table shows some examples of the controllers and their I/O addresses.

I/O Controller	I/O Address
Keyboard	060 - 063
Hard disk	320 - 32F
Printer	378 - 37F
Floppy disk	3F0 - 3F7

Table 2.2 : Controllers and their addresses.

The operating system performs I/O by writing commands into controller's registers.

Direct Memory Access

Direct memory access

DMA (Direct Memory Access) unit is capable of transferring data straight from memory to I/O devices.

How DMA Works?

First the controller reads the block from the drive serially, bit by bit, until the entire block is in the controller's internal buffer. Next, it computes the checksum to verify that no read errors have occurred. Then the controller causes an interrupt when the operating system starts running, it can read the disk block from the controller's buffer a byte or a word at a time by executing a loop, with each iteration reading one byte or word from a controller device register and storing it in memory.

After the controller has read the entire block from the device into its buffer and verified the checksum, it copies the first byte or word into the main memory at the address specified by the DMA memory address. Then it increments the DMA address and decrements the DMA count by the numbers of bytes just transferred. This process is repeated until the DMA count becomes zero, at which time the controller causes an interrupt.

1.3.2. I/O Software

I/O Software

Let us discuss I/O software. The key idea is to organize the software as a series of layers with lower ones concerned with hardware and upper ones concerned with the interfaces to the users. These goals can be achieved by structuring the I/O software in four layers.

- Interrupt handlers

- Device drivers
- Device independent OS software
- User level software.

Interrupt Handlers

The interrupt handlers will call other processes to handle interrupts that should be hidden away, deep in the bowels of the OS. The best way to hide them is to have every process starting an I/O operation block until the I/O has completed and interrupt occurs.

Device Drivers

The idea of device driver is a program that is used to control each device.

We already know, the idea of device driver is a program that is used to control each device. All hardware components of the computer is called devices. We saw that each controller has one/ more device register used to give it commands. The device drivers issue these commands and check that they are carried out properly.

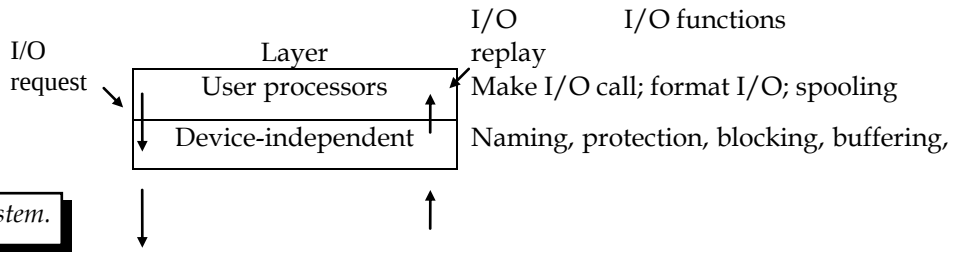
Device Independent I/O Software

Some of I/O software is device specific and others are device independent. The basic function of the device independent software is to perform the I/O functions that are common to all devices and to provide a uniform interface to the user level software. The functions of device independent software are :

- Device naming.
- Device protection.
- Buffering.
- Providing a device independent block size.
- Error reporting.
- Allocating ad releasing dedicated devices.
- Uniform interfacing for the device drivers.

User Level I/O Software

Most of the I/O software is within the OS, a small portion of I/O software consist of library linked user programs and I/O system calls are normally made by library procedures. Beside these, formatting of I/O is done by library procedures. Another function is spooling. Spooling is a way of dealing with dedicated I/O devices in a multiprogramming system e.g. line printer. Fig. 2.2 summarizes the I/O system, showing all the layers and the principal functions of each layer.



Layers of the I/O system.

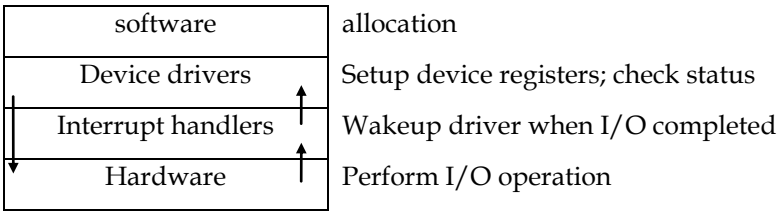


Fig. 2.2 : Layers of the I/O system and the main functions of each layer.

The arrows in Fig. 2.2 show the flow of control. When a user program tries to read a block from a file, for example, the operating system is invoked to carry out the call. The device-independent software looks in the cache, for example. If the needed block is not there, it calls the device driver to issue the request to the hardware. The process is then blocked until the disk operation has been completed.

When the disk is finished, the hardware generates an interrupt. The interrupt handler is run to discover what has happened. It then extracts the status from the device, and wakes up the sleeping process to finish off the I/O request and let the user process continue.

1.4. Exercise

1.4.1. Multiple choice questions

1. Normally I/O devices are divided into

i) 2 categories
ii) 3 categories
iii) 4 categories
iv) 5 categories.

2. A device driver is a

i) program
ii) controller
iii) DMA
iv) interrupt handler.

1.4.2. Questions for short answer

a) What are the four classes of interrupts? Provide one example of each class.
b) What do you understood by interrupt? What are the causes of occurring interrupts?
c) Explain why they are important to an operating system.
d) What are the functions of the device independent software?
e) List some examples of device controllers.
f) What do you mean by memory mapped I/O?
g) What do you know about I/O devices?

1.4.3. Analytical questions

a) Describe the layers of the I/O system and list the main functions of each layer.
b) What do you mean by DMA? Explain how it works.

Lesson 2 : System Calls and System Program

2.1. Learning Objectives

On completion of this lesson you will know:

- system calls
- categorized system calls and system programs
- discuss system program.

2.2. System Calls

User programs communicate with the operating system and request services from it by making system calls. Fundamental services are handled through the use of system calls. The interface between a running programs and the operating system is defined by what is referred to as systems calls. A system call is a special type of function call that allows user programs to access the services provided by the operation system. A system call will usually generate a trap, a form of software interrupt. The trap will force the machine to switch into the privileged kernel mode- that allows access to data structures and memory of the kernel. In other words, system calls establish a well defined boundary between a running object program and the operating system. When a system call appears in a program, the situation is equivalent to a conventional procedure call whereby control is transferred to operating system routine invoked during the run time along with change of mode from user to supervisor. These calls are generally available as assembly language instructions, and are usually listed in the manuals used by assembly language programmers.

A system call is a special type of function call that allows user programs to access the services provided by the operation system.

System calls can be roughly grouped into three major categories: process or job control, device and file manipulation, and information maintenance. In the following discussion, the types of system calls provided by an operating system are presented.

2.2.1. Process and Job Control

A running program needs to be able to halt its execution either normally (end) or abnormally (abort). If the program discovers an error in its input and wants to terminate abnormally.

Process and Job Control

A process or job executing one program may want to load and execute another program. This allows the control card interpreter to execute a program as directed by the control cards of the user job.

If control returns to the existing program when the new program terminates, we must save the memory image of the existing program and effectively have created a mechanism for one program to call another program. If both programs continue

concurrently, we have created a new job or process to be multi-programmed. Then system call (*create process* or *submit job*) are used.

If we create a new job or process, to control its execution, then control requires the ability to determine and reset the attributes of a job or process, including its priority, its maximum allowable execution time, and so on (*get process attributes* and *set process attributes*). We may also want to terminate a job or process that we created (*terminate process*) if we find that it is incorrect or on longer needed.

Having created new jobs or processes, we may need to wait for them to finish execution. We may want to wait for a certain amount of time (*wait time*), but more likely we want to wait for a specific event (*wait event*). The jobs or processes should then signal when that event has occurred (*signal event*).

2.2.2. File Manipulation

File Manipulation

The file system will be discussed in more detail in unit 7. We first need to be able to *create* and *delete* files. Once the file is created, we need to open it and use it. We may also *read*, *write*, and *reposition* (rewinding it or skipping to the end of the file). Finally, we need to close the file, indicating that we are no longer using it.

We may need these same sets of operations for directories if we have a directory structure in the file system. In addition, for either files or directories, we need to be able to determine the values of various attributes, and perhaps reset them if necessary. File attributes include the file, name a file type, protection codes, accounting information, and so on. Two system calls, *get file attribute* and *set file attribute* are required for this function.

2.2.3. Device Management

Device Management

Files can be thought of a abstract or virtual devices. Thus many of the system calls for files are also needed for devices. If there are multiple users of the system, we must first request the device, to ensure that we have exclusive use of it. After we are finished with the device, we must *release* it. Once the device has been requested (and allocated to us), we can *read, write*, and (possibly) *reposition* the device, just as with files.

2.2.4. Information Maintenance

Information maintenance

Many system calls are used transferring information between the user program and the operating system. For example, most systems have a system call to return the current time and date. Other system calls may return information about the system, such as the number of current users, the version number of the operating system, the amount of free memory or disk space, and so on.

In addition, the operating system keeps information about all of its jobs and processes, and there are system calls to access this information. Generally, there are also calls to reset it (get process attributes and set process attributes).

The following summarizes the types of system calls normally provided by OS.

i). Process Control

- End, Abort
- Load
- Create Process, Terminate Process
- Get Process Attributes, Set Process Attributes
- Wait for Time
- Wait Event, Signal Event.

ii). File Manipulation

- Create File, Delete File
- Open, Close
- Read, Write, Reposition
- Get File Attributers, Set File Attributes.

iii). Device Manipulation

- Request Device, Release Device
- Read, Write, Reposition
- Get Device Attributes, Set Device Attributes.

iv). Information Maintenance

- Get Time of Date, Set Time or Data
- Get Data System, Set System Data
- Get Processes, File or Device Attributes, Set Process, File Device Attributes.

System calls to the operating system are further classified according to the type of call. There are :

- Normal Termination
- Abnormal Termination
- Status Request
- Resource Request and
- I/O Requests.

2.3. System Program

An important aspect of a modern system is its collection of systems programs to solve common problems and provide a more convenient environment and execution.

Classification of System program

Systems programs can be classified into several categories :

File Manipulation : These programs create, delete, copy, rename, print, dump, list, and generally manipulate files and directories.

Status Information : Some programs simply ask the operating system for the date, time, amount of available memory or disk space, number of users, or similar status information.

File Modification : Several text editors may be available to create and modify the content of files stored on disk or tape.

Programming Language Support : Compilers, assemblers, and interpreters for common programming languages (such as Fortran, Cobol, Pascal, Basic, and so on) are often provided with the operating system. But recently many of these programs are being priced and provided separately.

Program Loading and Execution : Once a program is assembled or compiled, it must be loaded into memory to be executed.

Application Program : Most operating systems come with programs which are useful to solve some particularly common problems, such as compiler-compilers, text formatters, plotting packages, database systems, statistical analysis packages, and so on.

The most important system program for an OS is its command *interpreter*. It is that program which is runs when a job initially starts or user first logs in to a time sharing system.

The view of the operating system seen by most users is thus defined by its systems programs, not by its actual system calls. Consequently, this view may be quite removed from the actual system. The problems of designing a useful and friendly user interface are many, but they are not direct functions of the operating system.

2.4. Exercise

2.4.1. Multiple choice questions

1. System calls provide the interface between

i) a running program and user
ii) a running program and programmer
iii) a running program and the OS
iv) user and hardware.

2.4.2. Questions for short answers

a) What do you understood by system calls?
b) How many types of system calls are there in this lesson?
c) Summarize the system calls provided by OS?

2.4.3. Analytical questions

a) What do you know about system programs?
b) Describe different categories of system programs.

Lesson 3 : Operating System Structure

3.1. Learning Objectives

On completion of this lesson you will know :

- different types of OS system structure
- how a system call can be made
- micro kernel.

3.2. Operating System Structure

A number of approaches can be taken for configuring the components of an operating system, ranging from a monolithic to a virtual machines. To conclude the introduction, we identify several of the approaches that have been used to build OS. There are four different structures of operating system, but in this lesson we will discuss only three of them.

3.2.1. Monolithic System

The monolithic organization does not attempt to implement the various functions process, file, device and memory management in distinct modules. Instead, all function are implemented within a single module that contains all system routines or process and all operating system data structure.

The monolithic organization does not attempt to implement the various functions process, file, device and memory management in distinct modules.

The operating system is written as a collection of procedures, each can call any of the other ones whenever it needs to. When this technique is used, each procedure in the system has a well defined interface in terms of parameters and results, and each one is free to call any other one, if the latter provides some useful computation that the former needs.

In monolithic systems, it is possible to have at least a little structure. The services (system calls) provided by the operating system are requested by putting the parameters in well-defined places, such as in registers or on the stack, and then executing a special trap instruction known as a kernel call or supervisor call.

This instruction switches the machine from user mode to kernel mode (also known as supervisor mode), and transfers control to the operating system, shown as event 1 in Fig. 2.3. Most CPUs have two modes; kernel mode, for the operating system, in which all instructions are allowed; and user mode, for user programs, in which I/O and certain other instructions are not allowed.

The operating system then examines the parameters of the call to determine which system call is to be carried out, shown as 2 in Fig. 2.3. Next the operating system indexes into a table that contains in slot x a pointer to the procedure that carries out system call x. This operation, shown as 3 in Fig..2.3, identifies the service procedure,

which is then called. Finally, the system call is finished and control is given back to the user program.

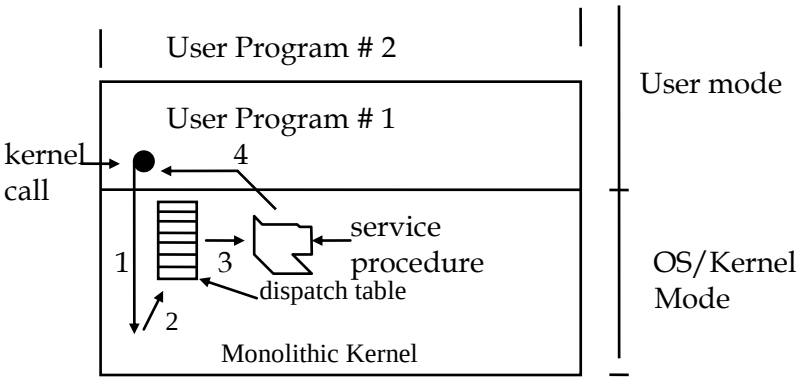


Fig. 2.3 : Method of making system call.

How a system call can be made?

1. User program traps to kernel.
2. OS determines service number required.
3. Service is located and executed.
4. Control returns to user program.

This organization suggests a basic structure for the operating system :

- A main program that invokes the requested service procedure.
- A set of service procedures that carry out the system calls.
- A set of utility procedures that help the service procedures.

In this model, for each system call there is one service procedure that takes care of it. The utility procedures do things that are needed by several service procedures, such as fetching data from user programs.

3.2.2. Client / Server or Micro-Kernel Approach

A micro-kernel is a "new" way of structuring an operating system.

A micro-kernel is a "new" way of structuring an operating system. Instead of providing all operating system services (as do most current kernels) a micro-kernel provides a much smaller subset. Services usually provided are memory management, CPU management and communication primitives. Typically a micro-kernel will provide the mechanisms to perform these duties rather than the policy of how they should be used. Other operating system services are moved into user level processes that use the communication primitives of the micro-kernel to share information. In this system, the OS responsibilities are separated out into separate programs.

The kernel is stripped of much of its functionality, and basically only provides communication between clients and server. The following Fig. 2.4 will clearly illustrates client server model.

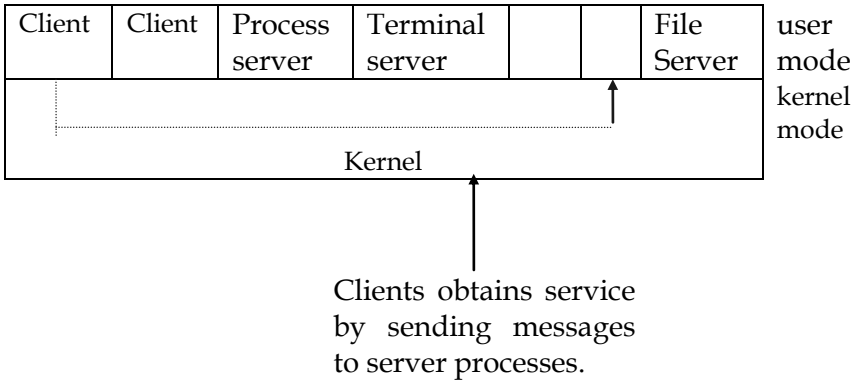


Fig. 2.4 : Client server model.

The *advantages* of this structure is as follows :

- better way to write software
- easier to distribute across many machines.

The main *disadvantage* is the slow in speed.

Difference between monolithic and micro kernel system

Difference between monolithic and micro kernel system

A monolithic operating system contains all the necessary code in the one kernel. This means that if any changes are made to the kernel the whole system must be rebooted for the changes to take effect.

A micro-kernel operating system contains a much reduced set of code in the kernel of the operating system. Most of the service provided by the OS are moved out into separate user level processes.

All communication within a micro-kernel is generally via message passing whereas a monolithic kernel relies on variables and local procedure calls. These attributes of a micro-kernel mean:

- that it is easier to develop the user level parts of the micro-kernel as they can be built on top of a fully working operating system using programming tools,
- the user level processes can be recompiled and installed without rebooting the machine,
- different service can be moved to totally different machines due to the message passing nature of communication in s micro-kernel operating system.

3.2.3. Virtual Machine

In the following discussion we will discuss the structure of a virtual machine named VM/370 with CMS expand.

The first releases of OS/360 were batch systems. Many 360 users wanted to have timesharing, so various groups, decided to write timesharing systems for it. The official IBM timesharing system, TSS/360 was delivered late and was eventually abandoned.

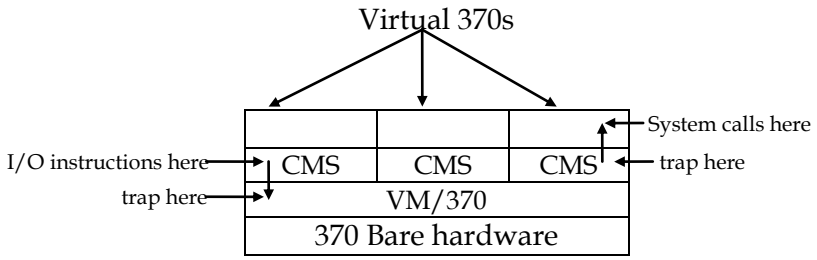


Fig. 2.5 : The structure of VM/370 with CMS.

The next system was called CP/CMS and now called VM/370. The virtual machine monitor, runs on the bare hardware and does the multiprogramming, providing several machines to the next layer up, as shown in Fig. 2.5. Unlike all other operating systems, these virtual machines are not extended machines, but they are exact copies of the bare hardware, including kernel/user mode, I/O, interrupts, and everything else the real machine has. Each virtual machine is identical to the true hardware, each one can run any operating system that will run directly on the hardware. Different virtual machines can run different operating systems. Some run one of the descendants of OS/360 for batch processing, while other ones run a single-user, interactive system called CMS (Conversational Monitor System) for timesharing users.

When a CMS program executes a system call, the call is trapped to the operating system in its own virtual machine, not to VM/370 just as it would if it were running on a real machine instead of a virtual one. CMS then issues the normal hardware I/O instructions for reading its virtual disk. These I/O instructions are trapped by VM/370, which then performs them as part of its simulation of the real hardware. By making a complete separation of the functions of multiprogramming and providing an extended machine, each of the pieces can be much simpler, more flexible, and easier to maintain.

3.3. Exercise

3.3.1. Multiple choice questions

1. Most CPUs have two modes

i) kernel mode and monolithic mode
ii) kernel mode and supervisor mode
iii) kernel mode and user mode
iv) None of the above.

2. CMS stands for

i) computer mail system.
ii) conversational monitor system.
iii) computer message system.
iv) computer mail system.

3.3.2. Questions for short answers

a) What is a micro-kernel?
b) Explain the differences between a monolithic operating system and one based on a micro-kernel architecture.
c) How a system call can be made?
d) What are the advantages and disadvantages of micro-kernel system?

3.3.3. Analytical questions

a) What do you know about virtual machine? Describe briefly.