SIT 111 2.1P **JASVEENA 224001588**

DATED: 22-04-2024

SUMMARY

Why use operating system?

For a computer to work we require,

1. **CPU** **2. Memory** **3. Peripherals**

Since all this is quite complex we need an operating system. The operating system fills the gap , it manages the hardware for us and gives us ways to access and control the hardware while hiding the low-level details of how the hardware works

WHAT DOES AN OS DO ?

Virtualisation: It provides **every user application** with the illusion of having the entire hardware resources to itself.

Resource sharing: This allows you to run many more applications than the number of available CPUs and the amount of available DRAM.

Protection: the OS *isolates* one application from another, such that faults in one application do not affect the other.

PROGRAMS AND PROCESS

A program is what we write, it involves the code (commands with proper syntax) using a programming language .

When we execute this program it is called a process.

KERNAL MODE AND USER MODE

The CPU executes more than one instruction at a time and in more than one mode, for this there are kernel mode and user mode. The

kernel mode has all the hardware instructions allowed. And the user mode is where all the applications run.

Whatever user applications are running, when certain events occur, **the CPU immediately transition to some kernel code**. These events include:

- Hardware interrupt: timer event, notifications from devices, etc.
- Software interrupt: runtime exception in your program (e.g. divided by 0, out of memory, etc.)
- System call: requests for services from the OS, e.g., to open or to read a file.

Scheduler

A process state, or **context**, captures the necessary information to run the process. It is initialized, saved, and restored by the OS.

Scheduling is the phenomenon of what happens when we enter any command

VIRTUAL MEMORY

virtual memory is an indispensable technical component of modern OS. Virtual memory is a concept originating from the operating system. It gives a process the illusion that it can access the entire memory.

Memory Address Translation

ADDRESSES

Every program accesses data in memory by reading or writing from an address. In almost all modern computers, the memory is byte addressable, meaning that each address stores a byte.

MEMORY MANAGEMENT UNIT

When the CPU wants to read/write data at address *X*, it puts *X* in the Memory Management Unit. It performs a *translation* that maps *X* to a physical address in DRAM. It then sends a signal to DRAM to read or write that physical address.

MEMORY PAGING

The virtual address space is divided into fixed-size chunks called *pages*. the physical address space is divided into chunks of the same size, called *frames*. For each process, there is a data structure called a *translation map*, or **page table** that maps virtual pages to physical.

FILE SYSTEM

It is a method used by os to organise and store data on storage devices such as hard drives and other storage media. It provides a structured way to store, retrieve, and manage files and directories on a folder. Filesystems manage data by organizing it into files which are collections of related information, and directories, which are containers for files and other directories.

Additionally, filesystems typically include metadata such as file attributes (e.g., permissions, timestamps) to manage and control access to files and directories.

TYPICAL FILESYSTEM HERIARCHY IN LINUX :

In Linux, the filesystem hierarchy is organized in a tree-like structure with a root directory ("/") at the top.

Common directories found in the Linux filesystem hierarchy include:

/bin: Contains essential binary files (executable programs) required for system boot and basic system operations.

/lib and /lib64: Contain library files required by programs and shared libraries.

RESOLVING FILES AND DIRECTORIES BY THE OPERATING SYSTEMS

When a user or program references a file or directory, the operating system needs to resolve its location in the filesystem hierarchy. The operating system uses a hierarchical directory structure and maintains a directory tree in memory to quickly locate files and directories. Once the operating system resolves the target file or directory, it can perform various operations such as reading, writing, or executing the file or navigating the directory structure.

REFLECTIONS

How do you know you have achieved the learning goals?

I have thoroughly understood the concepts and summarised them based upon my understanding .

• What is the most important thing you learned from this and why?

The most important thing I learned from this task is how the operating system is so important in a modern computer.

• How does the content or skills learned here relate to things you already know?

I already knew the basics of an operating system this task helped me deepen my understanding.

• Where or when do you think it will be useful?

It will be useful when we will be needing to learn about the hardware.

① Total memory given = 2^{16} bytes
Page size = 4 bytes

1) calculating the number of bits needed in this address :-
⇒ Since the memory size is 2^{16} bytes, it can be represented ~~as~~ with 16 bits.
Answer = **16 bits**

2) Calculating the number of bits in the offset :
The page size is 4 bytes, which can be presented with 2 bits (since $2^2 = 4$).
Answer = 2 bits

3) calculating the number of virtual pages :
Since the page size is 4 bytes, and the total memory size is 2^{16} bytes, the number of virtual pages can be calculated as :-

$$\text{No. of virtual pages} = \frac{\text{Total memory size}}{\text{Page size}}$$

$$= \frac{2^{16}}{4} = 2^{16-2} = \boxed{2^{14}} \text{ virtual pages}$$

② Performing translations for the given addresses :-

i) Address : 0b01000011
- Divide into fields :
- First-level index : 01
- Second-level index : 00
- offset : 0011

• Look up the first-level page table entry for index 01, let's say

it points to entry 1010 in the second-level page table.
- Look up the second-level page table entry for index 00, let's say it points to frame 1100 in physical memory.
- Concatenate the frame number (1100) with the offset (0011) to get the physical address: 11000011

2) Address: 0b111110010
   → Divide into fields:
   → First-level index: 11
   → Second-level index: 11
   → offset: 0010
   → Look up the first-level pages tables entry for index 11, let's say it points to frame 1011 in physical memory.
   → Concatenate the frame number (1011) with the offset (0010) to get the physical address: 10110010

3) Address: 0b00100010
   → Divide into fields:
   → First-level index: 00
   → Second-level index: 10
   → offset: 0010
- Look up the first-level page table entry for index 00, let's say it points to entry 1101 in the second-level page table.
- Look up the second-level page table entry for index 10, let's say it points to frame 0011 in physical memory
- Concatenate the frame number (0011) with the offset (0010) to get the physical address: 00110010.