Tuesday, March 22, 2022 10:33 AM

Kernel = OS

User Space , Kernel Space = RAM Space is divided into two parts

Kernel Space	User Space
All the kernel processes are stored	All the user processes are stored

Micro Kernel , Monolithic Kernel = types of OS

Micro Kernel	Monolithic Kernel
The kernel is split into small modules Sub parts or processes	A kernel is a single HUGE process that includes all parts
More Fault tolerant --- even if any module fails the kernel is still running	Less Fault Tolerant ---- if any part fails then the entire kernel crashes
Communication between the modules is affecting performance ---- SLOWER	Everything is one huge process---but no external communication is needed--- FASTER
Examples - Windows	Example - Linux based OS
find more	Find more

OS ---------- How many OSes are available globally ? Number is between 500 to 800

HW ---------- Give example of each category
GPOS = General Purpose Operating System
MultiUser OS = at a time many users can login in the OS instance
Embedded OS -- embedded systems have their own OS
Mobile OS
Server OS
RTOS === Real time Operating System
Network OS

HDD	SSD
Al disc + magnetic dipoles	Silicon , semiconductor ---- MOSFET
The dipoles store the values of bits	MOSFET retains charge -and it represents bits
Moving parts - prone to wear and tear	No moving parts so not so susceptible to wear and tear
Slower access	Faster access

Ashish ---- Access time of
HDD , 5 to 10 ms
SSD and 0.1 ms
RAM 10 ns

Interrupts -----

What is an interrupt ----- signal given by IO devices to CPU (pins)
by software to CPU (pins)

Whenever an interrupt occurs --------- Interrupt is handled by a function = ISR (Interrupt Service Routine), this is also called as IH --- Interrupt Handler) -- this function code that specified what should be done when that interrupt occurs.

Mapping data structure is maintained by the kernel that keeps track of which handler to which interrupt!!!
This mapping data structure is called as **Interrupt Table** , **Interrupt Vector Table (** IVT)

Interrupt Number	Function Pointer to ISR
1	f1
2	f2
3	..
…	…

Types of Interrupts
Maskable Interrupts = we can configure that the interrupt should not be handled and ignored
NON Maskable Interrupts = no one can ignore it , they must be handled

HW --- give example of maskable and non maskable interrupt

Keyboard Interrupt
Mouse Interrupt
ctrl-c
Timer interrupt
Trap
…
…

Kernel Mode , User Mode = This tells whether the privileged instructions can be executed by the process .

Kernel Mode	User Mode
Permissions are granted to use privileged instructions	Permissions not granted for privileged instructions
Non privileged instructions are permitted	Non privileged instructions are permitted
Kernel process run in Kernel mode	User process run in user mode -- and if it needs to access kernel instructions then it must switch to kernel mode
Examples of privileged Instructions • Turn off all Interrupts • Set the Timer • Context Switching • Clear the Memory or Remove a process from the Memory • Modify entries in the Device-status table	Examples of non privileged instructions • Reading the status of Processor • Reading the System Time • Generate any Trap Instruction • Sending the final printout of Printer

---

V V V IMP --- OS system management task ========> Process Management

What is a process in OS ?

Define OS process = Program in execution is called process !!!

Program not in execution	Is present on secondary memory	Ex --- c:\windows\system32\notepad.exe

program in execution	Is loaded in primary memory	Open a notepad ---- a notepad process1 ---PID 11868
		Open another notepad --- notepad process2 ---PID 5544

Check the processes on our system -- ctrl alt del ---- task manager

Every process created gets a **process Id** that is unique

Every process has a Life cycle = different phases of the process from creation to termination
Phases are called as STATES of the process life cycle .

1. Created State --- a process gets a space in the RAM , it is loaded in the RAM
   A PCB is generated for the process.

   1. Program on secondary memory is loaded in primary memory by a loader system software
   2. Process space is divided into few parts and the process space is also called as Process Address Space.

      C program process

Code segment	Data segment	Stack segment	Extra segment

      Java program process

Class area	Stack area	Heap area

   3. PCB is created ====== Process Control Block
      Process Information is stored in each variable of PCB

```
struct PCB
{
        Process id
        Process state
        Location/address of process space
        Statistical info --- priority
        Context information
}

struct PCB   v1  = (struct PCB) malloc(sizeof(struct PCB) )
        V1.pid = nextpid
        …
        …
```

2. READY State ----- the newly created process waits for its turn to use the CPU
   WHERE ? Waits in Kernel space **Ready Queue**
   PCB variable is added to the ready queue

3. Execute State / Running State ----- when CPU is free the next process from ready queue loads instruction in the CPU, the instruction is executed , then next instruction is loaded , and so on …..
    1. When the code reaches the IO instruction
        i) Then the PROCESS changes to WAIT STATE and leaves the CPU
    2. When the code reaches the last line
        i) Then the PROCESS changes to TERMINATE STATE and leaves the cpu
    3. When externally some interrupt occurs ----
        i) ISR loads in the CPU and current PROCESS changes to READY STATE

4. Wait state ----

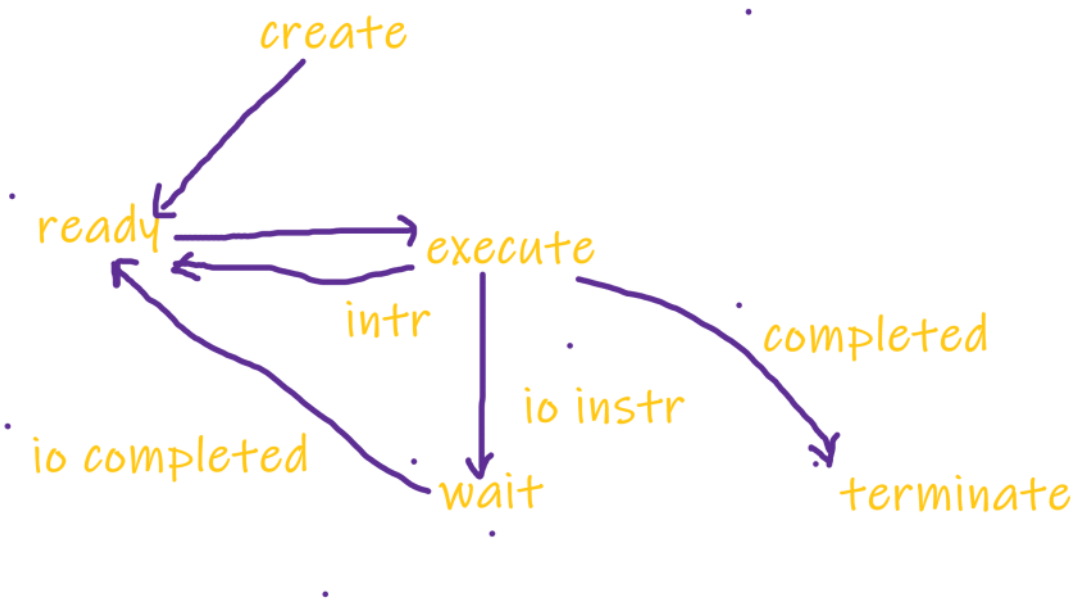
IO wait state --------- process is waiting for the DMA to complete the data transfer between primary memory and IO devices. It is not ready for execution, so it is not in ready queue.

1. When data transfer is done then the process changes to READY STATE

Suspended state ---- process is suspended for few ms
Waiting for lock to open
Sleeping for some ms

5. TERMINATE STATE ------- process ends , process space is deallocated(freed) , pcb variable for the process is destroyed(freed)

LIFE CYCLE OF A PROCESS



Process Scheduling =
Few CPUs and TOO many processes
4 COREs (4 CPUs) ---------- 200 processes

At a time only 4 processes can run on each Core !!!! Remaining process are in ready queue

Kernel must allocate processes to the CPU once it is free
Questions --- which processes should be given a chance?
for how much time the process should be allowed to use a CPU ?

These decisions are taken by a kernel module = CPU Schedular = Short Term Schedular = Low Level Schedular = Schedular

Factors have to IMPROVE ------ these should be considered by the CPU Schedular ??

1. Average Wait Time of a system ----------
   Wt = wait time of a process = time spent by the process in the READY QUEUE
   Average Wt = (Wt1 + Wt2 +Wt3 +….+Wtn)/n

   To improve Wt = As LOW as possible

2. Turn around time of a system ----------
   Ta = Turn around time of a process = time required to complete process life cycle (excluding wait state, IO instr).
   Ta = Wt + CPUt (wait time in ready queue + CPU BURST TIME (total time needed to execute all cpu instructions)

   Average Ta = (Ta1 + Ta2 + Ta3 + Ta4+…Tan)/n

   TO improve Ta = As LOW as possible

3. Throughput of a system ---------- number of processes completed in unit time
   For example Throughput = 20 processes / sec
   To improve throughput = As HIGH as possible

4. Response Time of a process ---------- time required by the process to respond to user request
   competent response time ---------- As LOW as possible

Common philosophies / thought processes used in any **CPU schedular** ---- common / basic **scheduling algorithms**

1. FIFO ,FCFS
2. SJF
3. Priority
4. Round Robin
