

# Operating Systems

သတိ : စာကိုမကျက်ပါနှင့်။  
နားလည်အောင်ဖတ်ပြီးစဉ်းစားပါ။

# Introduction

- ကျွန်တော်တို့ Operating Systems လို့ပြောရင် ပထမဆုံး Microsoft Windows™ ကို ပြေးမြင်မည်ဖြစ်သည်။ ထို့နောက် Linux Operating System ကို သတိရမည်။ ပြီးတော့ Mac OS X™ ကိုလည်း မြင်ယောင်မိမည်။ Android™ ကိုလည်း စဉ်းစားမိမည် ဖြစ်သည်။
- မှန်ပါသည်။ ဒါတွေအားလုံးသည် ကွန်ပျူတာ အမျိုးမျိုးပေါ်မှာ Run နေသော Modern Operating Systems များ ဖြစ်ပါသည်။
- ကျွန်တော်တို့ ထို Operating Systems များ၏ အခြေခံသဘောတရားများကို လေ့လာမည် ဖြစ်ပြီး လက်တွေ့ အနေဖြင့် Linux Operating System ကို လေ့လာကြည့်ပါမည်။
- ဒါကတော့ Linux နှင့် Android Operating Systems များ၏ Kernel များ ဖြစ်သည်။
  - <https://github.com/torvalds/linux>
  - <https://android.googlesource.com/kernel/>
- Kernel ဆိုတာ ဘာလဲလို့ သိချင်ရင်တော့ ဆက်ဖတ်ပါ။

# Lab Setup

- ကျွန်တော်တို့ OS ကိုလေ့လာရာတွင် Lab မှာလက်တွေ့စမ်းသပ်ကြည့်ရင် ပိုနားလည်ပြီး မှတ်သားရလွယ်မယ်လို့ ယုံကြည်ပါတယ်။
- အများစုက Windows OS ကို မိမိကွန်ပျူတာမှာ ရှိပြီးသားဖြစ်တော့ အခြား OS တွေကိုစမ်းသပ်ဖို့ Virtualization Technology ကိုအသုံးပြုဖို့ အကြံပြုချင်ပါတယ်။
- ဒီအတိုင်းတသွေမသိမ်း setup လုပ်ရမယ်မဆိုလိုပါ။
- Virtual Box ရင်းနှီးပြီးသားဆို VirtualBox. VMware ကိုအားသန်ရင် VMWare သုံးလို့ရပါတယ်။
- မိမိ computer က Chromebook လိုမျိုး cloud connected thin client ဆိုရင် cloud computing service က compute service နဲ့ လေ့လာလို့ရပါတယ်။ ဒီနေရာမှာတော့သိပ်အားမပေးပါဖူး။ Template လုပ်ပြီးသားဆိုတော့ ကိုယ်တိုင် configuration လုပ်စရာနည်းသွားပါတယ်။

ဘာလို့ Virtualization နဲ့စမ်းခိုင်းတာလဲ?

သုံးကိုသုံးရမယ် မဟုတ်ပါဖူး။

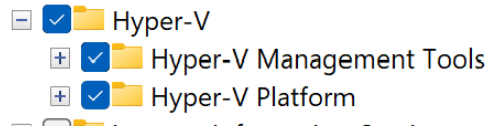
မိမိကျွန်ပျူတာအပိုမှာ install လုပ်ပြီး setup လုပ်တာ သို့မဟုတ် dual boot install လုပ်တာလဲရပါတယ်။

Virtualization ရဲ့အားသာချက်ကတော့ အကြိမ်ကြိမ် snapshot feature လျင်လျင်မြန်မြန် စမ်းသပ်လို့ရတာပါပဲ။

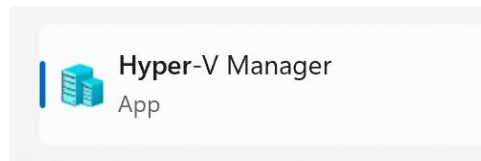
# Lab Setup

- ကျွန်တော်တို့ Windows OS မှာ ပါပြီးသား Hyper-V Virtualization ကိုသုံးကြည့်မှာ ဖြစ်ပါတယ်။ Windows Professional ဖြစ်ဖို့တော့လိုပါတယ်။ Virtual Box သုံးချင်ရင်တော့ ဒီမှာ <https://www.virtualbox.org/wiki/Downloads> ပါ။

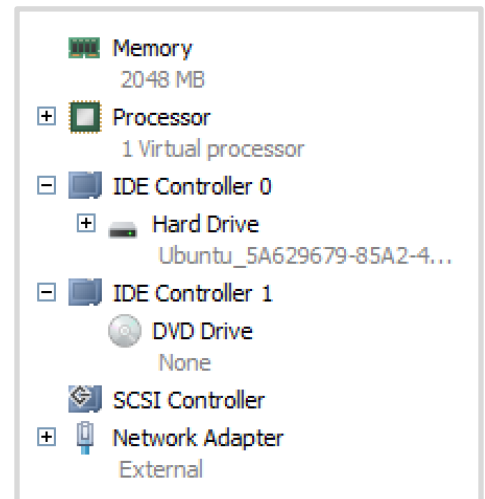
- Control Panel > Programs > Turn Windows features on or off



- Reboot လုပ်ပြီးရင် စသုံးလို့ရပါပြီ။ Application list မှာ Hyper-V Manager App ကိုတွေ့ရမှာပါ။

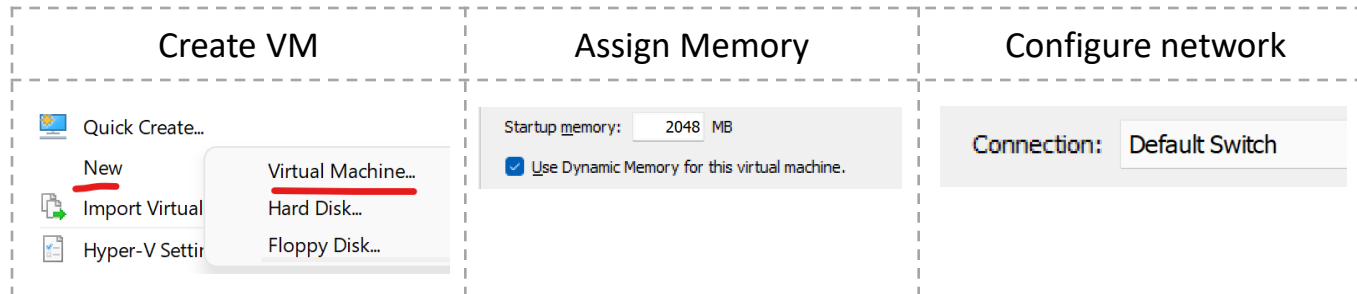


- OS installation and setup ကိုဆွေးမယ်ဆိုရင် hardware configuration ကမဖြစ်မနေပါဝင်လာတာပါပဲ။ OS က hardware resources ကိုသုံးရတာဆိုတော့။
- Virtualization မှာတော့ virtualized hardware တွေပေါ့။
- အနည်းဆုံး Processor, Memory, Storage နဲ့ Network တော့ configure လုပ်ရမှာပါ။

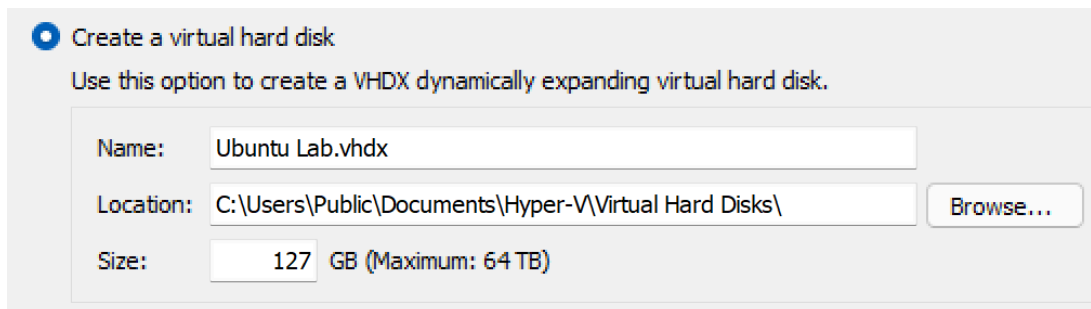


# Linux Installation (Ubuntu)

- ကျွန်တော်တို့ Lab setup အတွက် developer friendly အဖြစ်ဆုံးဖြစ်တဲ့ Ubuntu Linux ကို install လုပ်ပါမယ်။
- ဝေဖန် Ubuntu desktop installer image ISO disk image ကို download လုပ်ထားပါတယ်။
- Hyper-V မှာ VM အနေနဲ့ create လုပ်ပါမယ်။

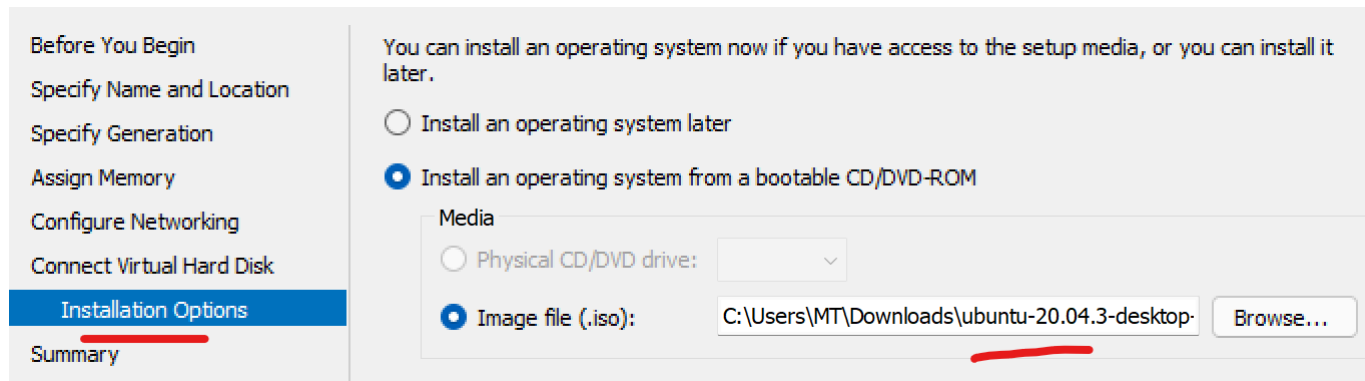


- Configure Storage



# Linux Installation (Ubuntu)

- ပြီးခဲ့တဲ့ slide မှာပြခဲ့တဲ့ screenshots အရ ကျွန်တော်တို့ 2 GB memory assign လုပ်ပြီး storage ကို 127 GB ပေးခဲ့ပါတယ်။
- Ubuntu ISO file ကို installation media အနေနဲ့ ရွေးချယ်မှာဖြစ်ပါတယ်။



- VM hardware တွေကို second ပိုင်းအတွင်း create လုပ်သွားမှာပါ။ စက်ဆင်တာပေါ့။
- Start ကို click လုပ်ပြီး VM ကို on မယ်။ Connect ကို click လုပ်ပြီး virtual monitor ကို connect လုပ်မယ်။



# Linux Installation (Ubuntu)

- Installer boot up ပြီးသွားရင်တော့ LiveOS အနေနဲ့သုံးမှာလား install လုပ်မှာလား မေးမှာပါ။ Ubuntu 20.04 LTS version ကိုအခြေခံထားတာပါ။ Install ကြည့်ရအောင်။
- Language တို့ keyboard တို့ရွေးပြီးရင်တော့ disk format section ကိုရောက်မှာပါ။
- Virtual device ဖြစ်တဲ့အတွက် စိတ်ပူစရာမလိုပဲ default selection ဖြစ်တဲ့ destroy all နဲ့ install လုပ်မှာပါ။
- Dual boot သို့မဟုတ် hardware မှာ setup လုပ်ရင်တော့ မိမိရဲ့ storage device ကို သေချာရွေးရမှာပါ။



## • Erase disk and install Ubuntu

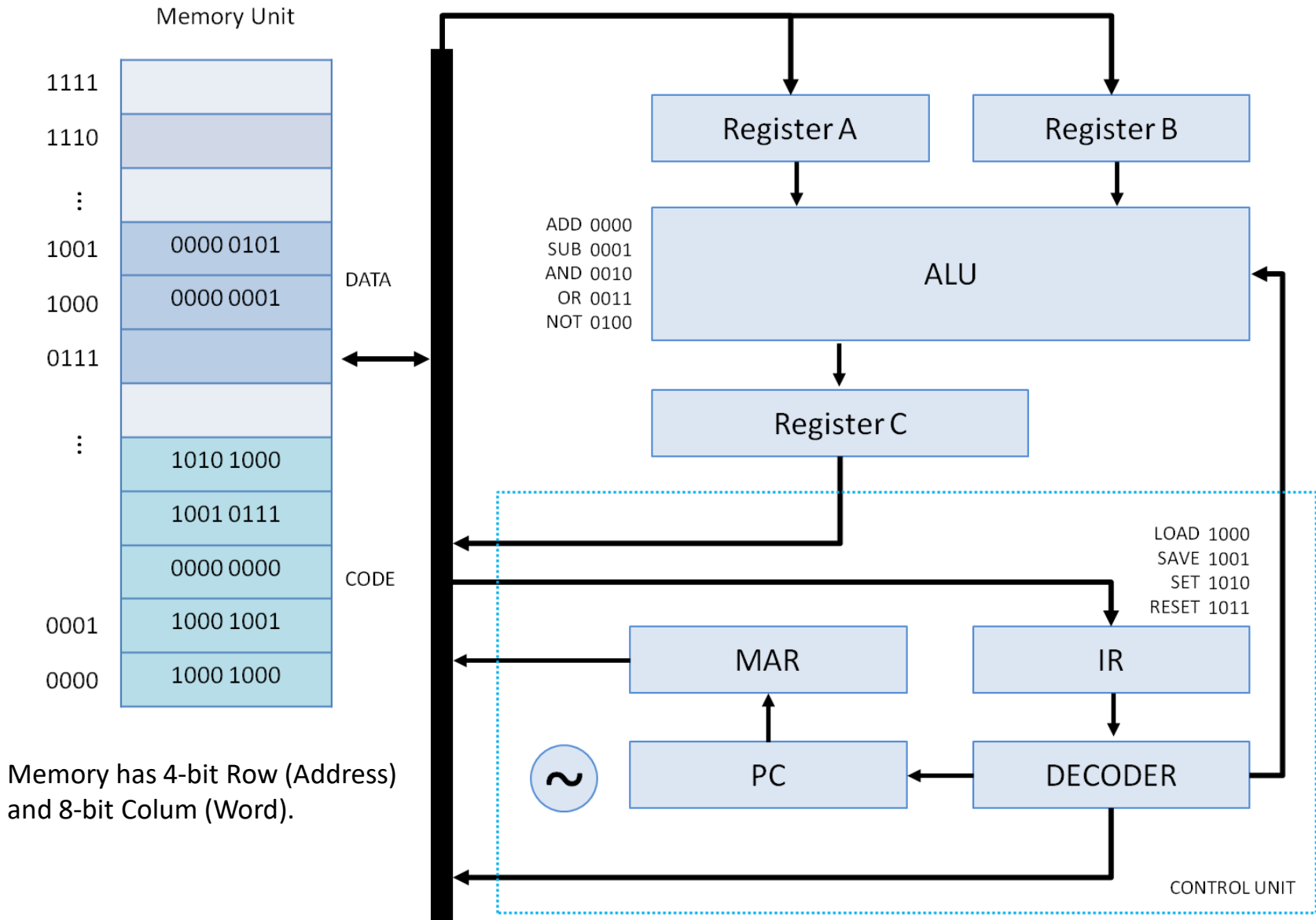
**Warning:** This will delete all your programs, documents, photos, music, and any other files in all operating systems.

Advanced features...

None selected

- မိမိပေးလိုတဲ့ login username နဲ့ password ကိုပေးပြီးပြီးဆိုရင် OS ကို installation စပြီးလုပ်ပါလိမ့်မယ်။
- Reboot လုပ်ပြီးရင် lab တွေမှာသုံးဖို့အတွက် Ubuntu VM ready ဖြစ်ပါပြီ။

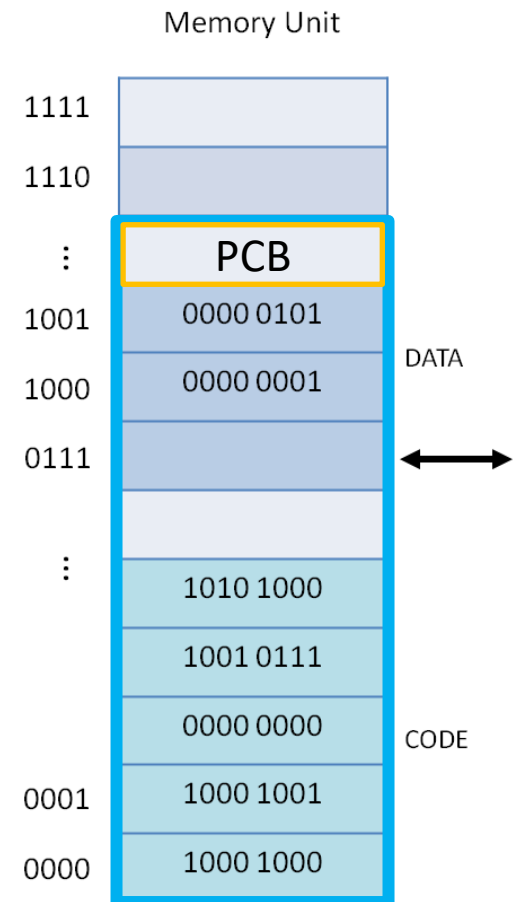
# Simple Computer Architecture





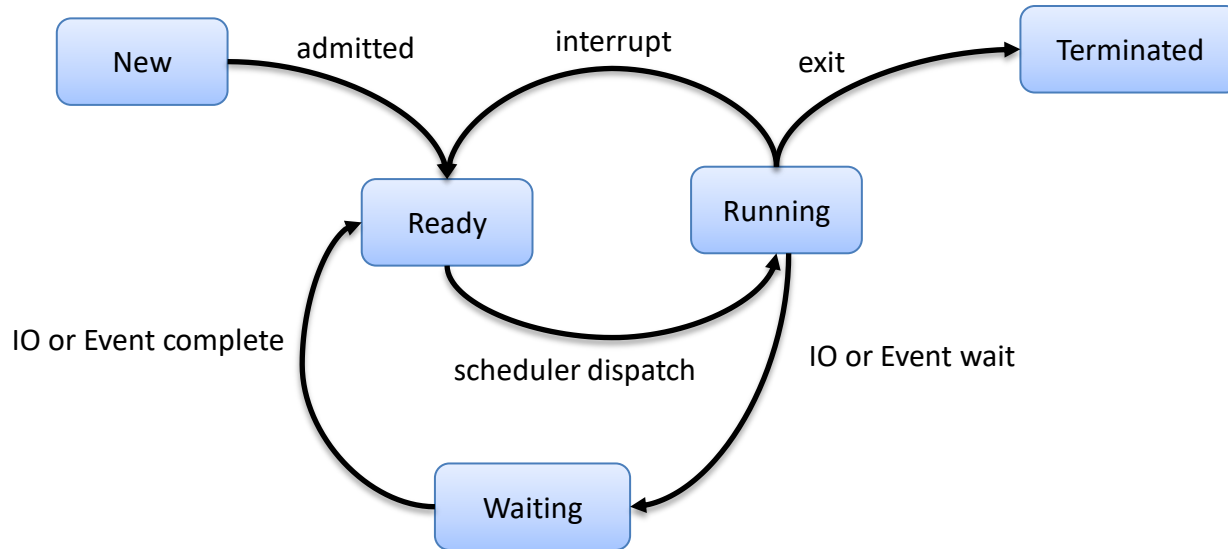
# Process

- ကျွန်တော်တို့ Von Neumann Architecture (Store Program Concept) အရ Memory ပေါ်မှာ Data ရော၊ Code ရော အတူတူထားပြီးတော့ Processor က Fetch, Decode, Execute လုပ်ပါသည်။
- Memory ပေါ်ရှိ Code အပိုင်းကို Code Segment ဟုခေါ်ပြီး Data အပိုင်းကို Data Segment ဟုခေါ်ပါသည်။
- A Unit of Program Execution အား Code Segment နှင့် Data Segment တို့ကိုပေါင်းပြီး Process တစ်ခု အဖြစ် သတ်မှတ်ပါသည်။
- အမှန်တော့ Process တစ်ခုမှာ Code နှင့် Data အပြင် Process ရဲ့ အခြား Information (Id, User, Date and Time of Execution, ...) များကို ဖော်ပြတဲ့ Process Control Block (PCB) ပါပါဝင်ပါသည်။ ဘာလို့လဲ ဆိုတော့ Mr. A ရဲ့ အဖြေကို Mr. B ကို ပေးလို့ မရ။
- ထို့ကြောင့် ပထမဆုံး Operating Systems များသည် Process များကို Manage လုပ်ဖို့ ကြိုးစားလာကြပါသည်။
- ဒါကို Process Management ဟုခေါ်ပါသည်။ Operating Systems အားလုံး၏ အသက်သည် Process Management ဖြစ်သည်။



$$\text{PROCESS} = \text{CODE} + \text{DATA} + \text{PCB}$$

# Process States



- ကျွန်တော်တို့ Processes 10 ခုပါဝင်သော Batch Processing ကို စဉ်းစားကြည့်ပါ။ Process 4 ခု မြောက်ကို Execute လုပ်ပြီး Program Error တက်လာသည် ဆိုပါတော့။ Process States များကိုသာ Manage မလုပ်ရင် အစက အကုန်ပြန်ပြီး Execute လုပ်ရမည် ဖြစ်သည်။
- Process States များ ရှိလာသည့် အခါ Error ရှိသော Process ကို Terminate လုပ်ပြီး အခြား Process များကို ဆက်ပြီး Execute လုပ်လို့ ရပါသည်။
- တစ်နည်းအားဖြင့် Process States များသည် လက်ရှိ Execute လုပ်နေသော Process များကို ဖော်ပြပြီး လိုရင်လိုသလို Execution ကို ပြောင်းလို့ ရပါသည်။ ဒါသည်ပင် Process Management ကို ပိုပြီး အဆင်ပြေသွားစေပါသည်။
- Process States များကို PCB မှာ Store လုပ်ထားပါသည်။

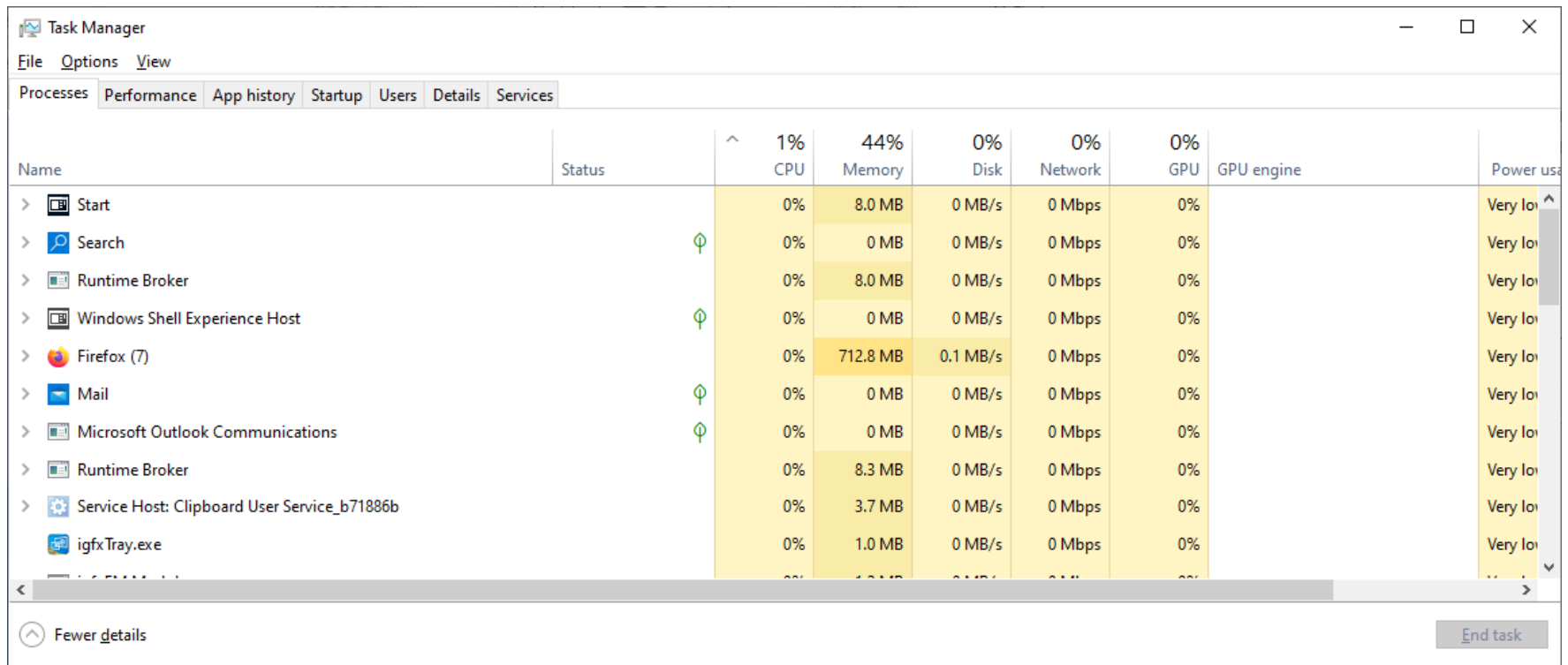
# Process Description

- Process တစ်ခုကို Manage လုပ်ဖို့ လိုအပ်သော Information တွေအားလုံးကို Process Control Block မှာ Store လုပ်ထားပါသည်။
- Program Counter က လက်ရှိ Instruction ရဲ့နေရာကို Point လုပ်ထားပါသည်။
- အကြောင်းတစ်စုံတစ်ရာကြောင့် Process 1 ကို Pause (Waiting) လုပ်လိုက်သည် ဆိုပါတော့။ ဒီ Process 1 ကို Resume လုပ်ချင်ရင် ထိုနေရာမှ ပြန်စမည် ဖြစ်သည်။
- Memory Pointers ကတော့ Process တစ်ခု၏ Data များ၏ Memory Addresses များကို ဖော်ပြဖို့ ဖြစ်ပါသည်။
- ထို့အပြင် PCB မှာ Process နှင့် ပတ်သက်သော အခြား Information များလည်း ပါရှိမည် ဖြစ်သည်။

PROCESS CONTROL BLOCK

Process Id
Process State
Priority
Program Counter
Memory Pointers
...
I/O Status
...

# Processes in Windows



Name	Status	1% CPU	44% Memory	0% Disk	0% Network	0% GPU	GPU engine	Power usage
> Start		0%	8.0 MB	0 MB/s	0 Mbps	0%		Very low
> Search	🟢	0%	0 MB	0 MB/s	0 Mbps	0%		Very low
> Runtime Broker		0%	8.0 MB	0 MB/s	0 Mbps	0%		Very low
> Windows Shell Experience Host	🟢	0%	0 MB	0 MB/s	0 Mbps	0%		Very low
> Firefox (7)		0%	712.8 MB	0.1 MB/s	0 Mbps	0%		Very low
> Mail	🟢	0%	0 MB	0 MB/s	0 Mbps	0%		Very low
> Microsoft Outlook Communications	🟢	0%	0 MB	0 MB/s	0 Mbps	0%		Very low
> Runtime Broker		0%	8.3 MB	0 MB/s	0 Mbps	0%		Very low
> Service Host: Clipboard User Service_b71886b		0%	3.7 MB	0 MB/s	0 Mbps	0%		Very low
igfxTray.exe		0%	1.0 MB	0 MB/s	0 Mbps	0%		Very low

- Windows မှာ Processes များကို ကြည့်မည်ဆိုလျှင် Task Manager ( Ctrl + Alt + Delete ) မှ ကြည့်ပြီး Manage လုပ်လိုရပါသည်။

# Processes in Linux

- ကျွန်တော်တို့ OS ရဲ့ processes တွေအကြောင်းကို ဒီ Lab မှာလေ့လာကြည့်မှာပါ။
- အတိုချုပ်အားဖြင့် process တစ်ခုက program တစ်ခုရဲ့ executed instance တစ်ခုဖြစ်ပါတယ်။
- Linux OS မှာ user နဲ့ system/daemon process ကို user space processes အနေနဲ့တွေ့ရပြီး kernel processes တွေကိုတော့ kernel space အနေနဲ့တွေ့ရမှာပါ။
- ကျွန်တော်တို့ အခု lab မှာတော့ user space processes တွေကို ကြည့်ကျရအောင်။  
`ps aux | head -n 2`

```
root@sysadmin-Virtual-Machine:/home/sysadmin# ps aux | head -n 2
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.1 170736 12760 ?        Ss   12:14   0:01 /sbin/init splash
```

Ubuntu ရဲ့ process ID 1 ပါ။

- Apache web server process ကို daemon process အနေနဲ့ လေ့လာလို့ ရပါတယ်။ Apache service က multiple processes ကို launch လုပ်ပါတယ်။

```
root@sysadmin-Virtual-Machine:/home/sysadmin# ps auxf | grep apache
root          609  0.0  0.0   6704  4512 ?        Ss   12:14   0:00 /usr/sbin/apache2 -k start
www-data      610  0.0  0.0 1211672 5436 ?        Sl   12:14   0:00 \_ /usr/sbin/apache2 -k start
www-data      611  0.0  0.0 1211592 4500 ?        Sl   12:14   0:00 \_ /usr/sbin/apache2 -k start
root          4962  0.0  0.0  17544   664 pts/1    S+   14:00   0:00 \_ grep --color=auto apache
```

# Processes in Linux

- ဘယ် user အနေနဲ့ execute လုပ်တာလဲကို ps ရဲ့ User column ကနေသိနိုင်ပါတယ်။

```
ps auxf | head -n 2
```

```
root@sysadmin-Virtual-Machine:/home/sysadmin# ps auxf | head -n 2
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root      2  0.0  0.0      0     0 ?        S    12:14   0:00 [kthreadd]
```

- Process ရဲ့ status code ကို ps နဲ့ကြည့်လို့ရပါတယ်။

```
ps -el | head -n 5
```

```
root@sysadmin-Virtual-Machine:/home/sysadmin# ps -el | head -n 5
F S      UID      PID      PPID    C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S       0         1         0  0  80   0 - 42684 ep_pol ?           00:00:01 systemd
1 S       0         2         0  0  80   0 -      0 kthrea ?           00:00:00 kthreadd
1 I       0         3         2  0  60 -20 -      0 rescue ?           00:00:00 rcu_gp
1 I       0         4         2  0  60 -20 -      0 rescue ?           00:00:00 rcu_par_gp
```

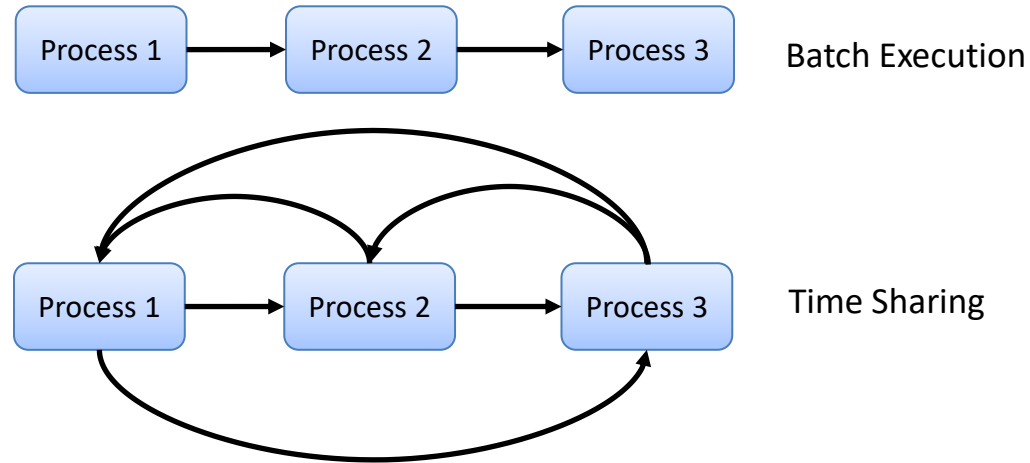
- Status codes တွေရဲ့ definition ကို man ps ရဲ့လေ့လာလို့ရပါတယ်။
- Process တွေရဲ့ CPU နဲ့ Memory ကို top command နဲ့ analyze လုပ်လို့ရပါတယ်။ ဥပမာ memory အများဆုံးသုံးနေတဲ့ program ကိုရှာမယ်ဆိုရင်

```
top -o %MEM
```

```
Tasks: 238 total,  1 running, 237 sleeping,  0 stopped,  0 zombie
%Cpu(s):  0.2 us,  0.3 sy,  0.0 ni, 99.5 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :  7750.7 total,  5409.4 free,  1283.2 used,  1058.1 buff/cache
MiB Swap:  2048.0 total,  2048.0 free,    0.0 used.  6188.6 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1421	sysadmin	20	0	4230612	365392	130684	S	0.0	4.6	0:27.57	gnome-shell
1855	sysadmin	20	0	3265320	343652	158848	S	0.0	4.3	1:14.79	firefox
969	gdm	20	0	3826884	204444	116096	S	0.0	2.6	0:10.59	gnome-shell

# Process Execution



- Multiple Process များကို Single Processor (CPU) မှ Execute လုပ်ရတွင် အကြမ်းအားဖြင့် 2 နည်းရှိပါသည်။
- Batch Execution မှာ Process တစ်ခုကို Execute လုပ်ပြီးမှ နောက်တစ်ခုကို Execute လုပ်ပါမည်။ ဒါက Manage လုပ်ရတာလွယ်ကူပါသည်။ သို့သော် Optimize မဖြစ်ပါ။
- Process 1 က Printing လုပ်ဖို့ Document အခု 100 ရှိသည်ဆိုပါက Printer က Printing မပြီးမချင်း Process 2 ကို Execute လုပ်လို့ မရပါ။ ထို့ကြောင့် အချိန်ပုတ်ပါသည်။ ဥပမာ၊ ထမင်းအိုးတည်လို့ ကျက်မှပြီးမှ ရေချိုးမည်ဆိုလျှင် ထမင်းအိုးတည်ထားတုန်း ရေချိုးတာထက် ပိုကြာမည် ဖြစ်သည်။
- Time Sharing က ဒီပြဿနာကို ဖြေရှင်းဖို့ ကြိုးစားပါသည်။ Process 1 က Printing လုပ်ဖို့ လိုမည်ဆိုပါက Printer ကို Print ထုတ်ခိုင်းထားပြီး Process 1 ကို Pause (Waiting State) ထားပါမည်။ ထို့နောက် Process 2 ကို Execute လုပ်ပါမည်။ Printer က Print လုပ်ပြီးပါက Process 2 ကို Pause လုပ်ပြီး Process 1 ကို Resume ပြန်လုပ်ပြီး Execute လုပ်ပါမည်။
- Time Sharing ဆိုသည်က Process တစ်ခုကို Execute လုပ်လို့ မပြီးသေးခင် အခြား Process များကို Execute လုပ်ခြင်း ဖြစ်သည်။

# Process Scheduling

- ခင်ဗျားမှာ အလုပ် (Job)တွေ 10 ခုလောက် ရှိတယ်ဆိုရင် ဘယ်အလုပ်ကို အရင်ဆုံး လုပ်မလဲ။ သိပ်ပြီး စဉ်းစားစရာ မလိုပါ။ အမြန်ဆုံးပြီးရမည့် အလုပ်ကို အရင်ဆုံး လုပ်မည် ဖြစ်သည်။ ။ အမြန်ဆုံးပြီးဖို့ထဲမှာမှ အရေးကြီးဆုံးကို အရင်ဆုံး လုပ်ရမည် ဖြစ်သည်။
- အလုပ်တစ်ခုကို တစ်ကြိမ်မှာ တစ်ခါပဲ လုပ်လိုရသည် ဆိုလျှင် Importance နှင့် Urgency ပေါ်မူတည်ပြီး Schedule လုပ်ရမည် ဖြစ်သည်။
- ထို့ကြောင့် Batch Execution မှာ Simple Queue သို့မဟုတ် Simple Priority Queue သာ ရှိလျှင် အလုပ်ဖြစ်သည်။ Batch Execution အတွက် Process Scheduling သည် First Come, First Serve Basis သို့မဟုတ် Priority Basis ဖြစ်သည်။
- သို့သော် Time Sharing ကို စဉ်းစားကြည့်ရအောင်ပါ။ Process 1: ထမင်းအိုးတည်ရမည်။ Process 2: ဟင်းချက်ရမည်။ Process 3: ရေချိုးရမည် ဆိုပါတော့။ ဘာကို အရင်လုပ်မလဲ။ Time Sharing မှာတော့ အကြာဆုံးအလုပ်ကို အရင်ဆုံးလုပ်ထားရင် ပိုအဆင်ပြေသည်။
- ဒီလို အလုပ်တွေကို ဘယ်လိုလုပ်ရင် အမြန်ဆုံးပြီးမလဲ ဆိုတာသည် Queuing Theory ကလာတာ ဖြစ်ပါသည်။
- အမှန်တော့ Queuing Theory သည် Process Scheduling အတွက် သာမက အခြား Management များအတွက်ပါ အရေးကြီးပါသည်။

	Not Important	Important
Urgent	Second Priority	First Priority
Not Urgent	Fourth Priority	Third Priority

ခင်ဗျားသာ YouTube™ က Music Video ကို ကြည့်ဖူးရင် စဉ်းစားကြည့်ပါ။ အင်မတန်များပြားသော အလုပ်တွေ ဖြစ်သည်။

Web Browser က Server ကို ချိတ်ဆက်ရမည်။ Server က Network မှတဆင့် Data များကို ခင်ဗျား Computer သို့ ပို့ပေးရမည်။

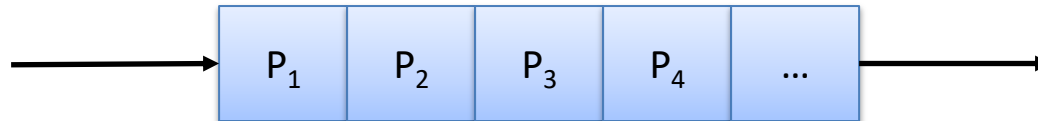
ရလာသော Data များကို အချိန်နှင့် တပြေးညီ Speaker မှ အသံထွက်ပေးရမည်။ Monitor မှ ရုပ်ပုံပြပေးရမည်။

ဒါတွေ အားလုံးကို စက္ကန့်ပိုင်းအတွင်းပြီးအောင် ဘယ်လို Schedule လုပ်မလဲ။

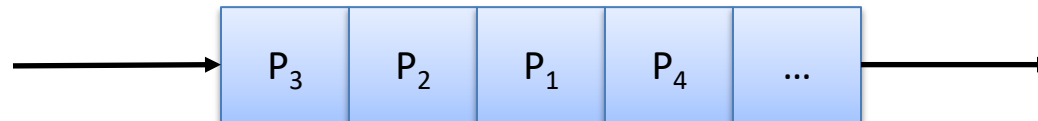


# Queuing Model

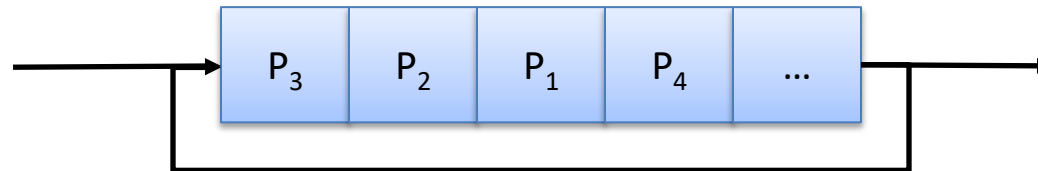
Simple Queue



Priority Queue



Cyclic Priority Queue



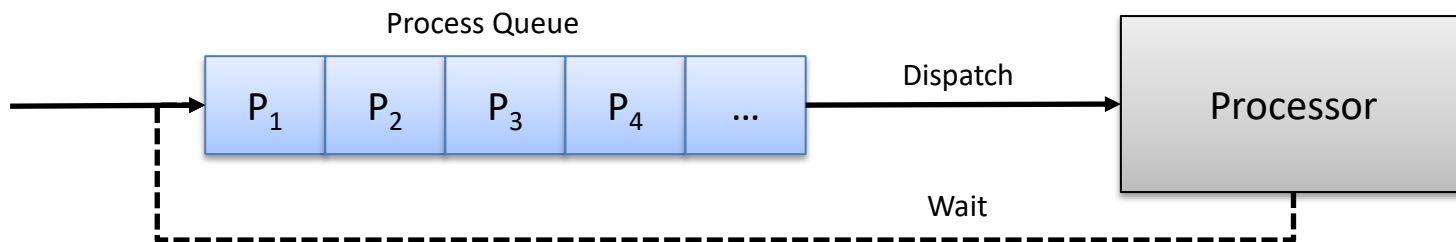
- Simple Process Scheduling များကို Single Simple Queue သို့မဟုတ် Single Priority Queue များဖြင့် တည်ဆောက်ပါသည်။
- သို့သော် ပိုရှုပ်ထွေးသော Complex Process Scheduling များကိုတော့ Multiple Cyclic Priority Queue များဖြင့် တည်ဆောက်ကြပါသည်။

# Process Dispatching

- Multiple Process များကို အများအားဖြင့် Process Queue ထဲတွင် ထည့်ထားပါသည်။ Process Queue မှ Process များကို Execute လုပ်ခြင်းကို Process Dispatching ဟုခေါ်ပါသည်။

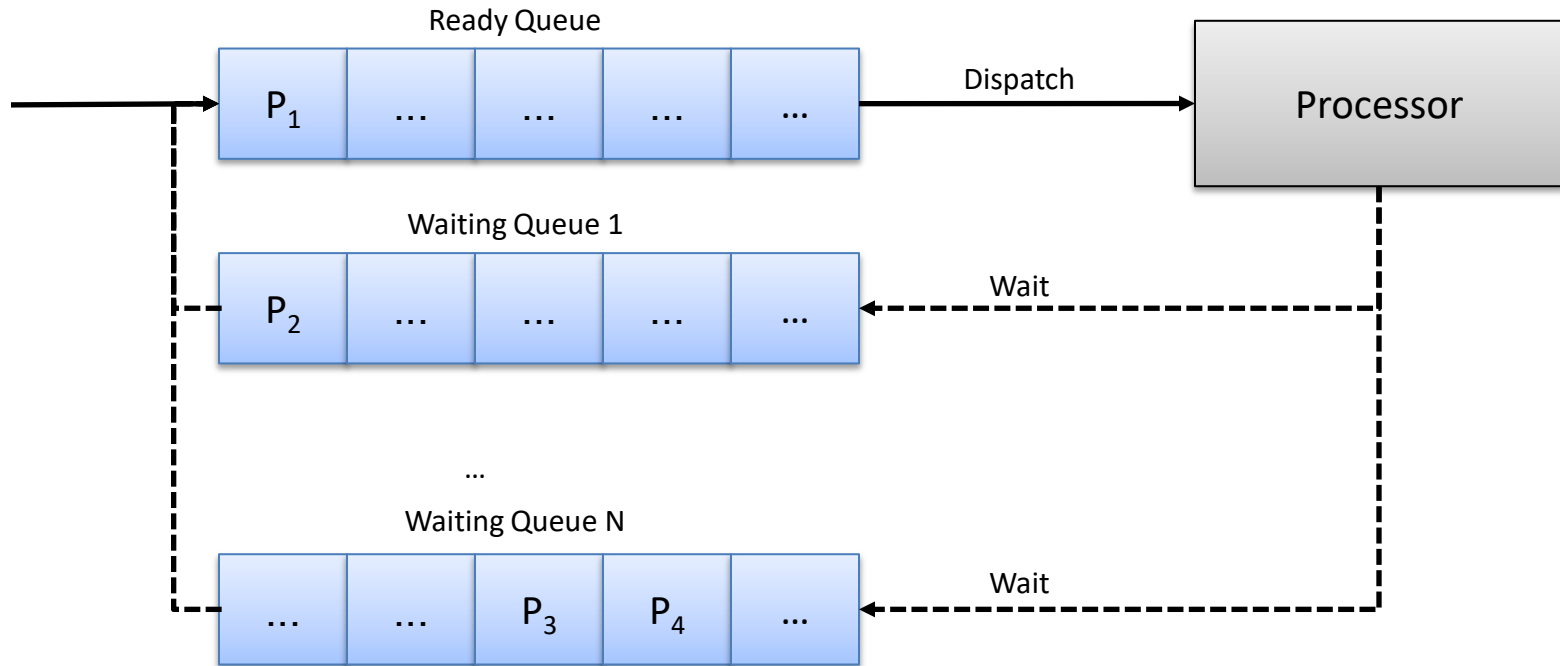


- Batch Execution ဆိုလျှင်တော့ Process 1, 2, 3, 4, ... ကို အစဉ်အတိုင်း Dispatch လုပ်မည် ဖြစ်သည်။ ထို့ကြောင့် Single Priority Queue ဖြင့် အဆင်ပြေပါသည်။



- Time Sharing မှာတော့ နည်းနည်း ပိုရှုပ်ထွေးပါသည်။ Process တစ်ခုက Waiting ဖြစ်သွားပြီ ဆိုလျှင် ထို Process ကို Process Queue ထဲပြန်ထည့်ရပါသည်။ ထို့ကြောင့် အနည်းဆုံး Single Cyclic Priority Queue တစ်ခု လိုအပ်ပါသည်။
- အမှန်တော့ Process Dispatching ကို Multiple Cyclic Priority Queue များဖြင့် တည်ဆောက်ကြပါသည်။

# Process Dispatching



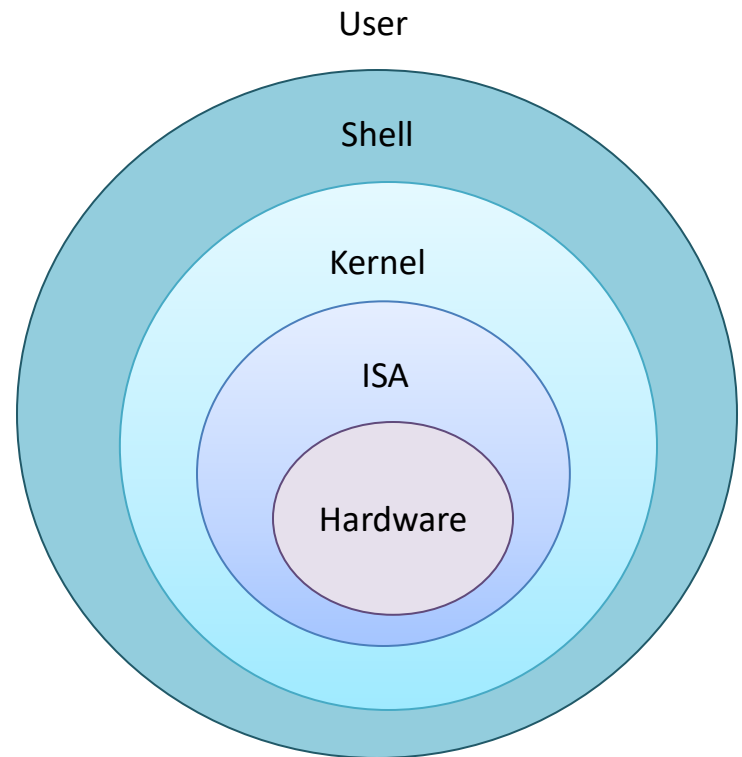
- Time Sharing မှာ Process များသည် တစ်ချိန်တည်းမှာ Multiple Queue များထဲတွင် ရှိနေမည်ဖြစ်သည်။
- Process များကို CPU Bound (CPU Intensive) နှင့် I/O (Input Output) Bound (I/O Intensive) ဆိုပြီး ခွဲခြားနိုင်ပါသည်။  
အများအားဖြင့် I/O Bound Process များကို အရင် Dispatch လုပ်လေ့ရှိပြီး I/O Bound Process များက Wait လုပ်နေချိန်တွင် CPU Bound Process များကို Dispatch လုပ်ပါသည်။
- Process တစ်ခုသည် I/O Operation တစ်ခုကို Wait လုပ်နေပါက Waiting Queue ထဲမှာ ရှိနေမည် ဖြစ်ပြီး I/O Operation Complete ဖြစ်သွားပါက Ready Queue ထဲသို့ ပြန်ရောက်သွားမည် ဖြစ်သည်။
- Ready Queue ထဲမှ Process များကို Priority ပေါ်မူတည်ပြီး Process Dispatching လုပ်မည် ဖြစ်သည်။

# Operating Systems

- Operating Systems များသည် Operator (ကြားလူ) အစား Computer Users များ၏ Computational Program များကို မြန်ဆန်တိကျစွာ ဆောင်ရွက်ပေးနိုင်ဖို့ ပေါ်လာတာ ဖြစ်သည်။
- Operating Systems များ အသက်သည် Process များကို Optimally Execute လုပ်နိုင်အောင် ဆောင်ရွက်ဖို့ ဖြစ်သည်။
- တစ်နည်းအားဖြင့် Operating Systems များ အသက်သည် Process Management ဖြစ်သည်။
- သို့သော် Process Management သည် Process များကိုသာ Manage လုပ်လို့ မရပါ။ Process နှင့် အတူ Memory ကိုလည်း Manage လုပ်ရသည်။ Input/output Device များကိုလည်း Manage လုပ်ရသည်။ External Storage Device (File) များကိုလည်း Manage လုပ်ရသည်။ နောက်ဆုံး Process နှင့် သက်ဆိုင်သော User များကိုလည်း Manage လုပ်ရပါသည်။
- ထို့ကြောင့် Operating Systems များ၏ အဓိက တာဝန်များသည်
  - Process Management
  - Memory Management
  - File Management
  - IO Management
  - Network Management
  - User and Security Management

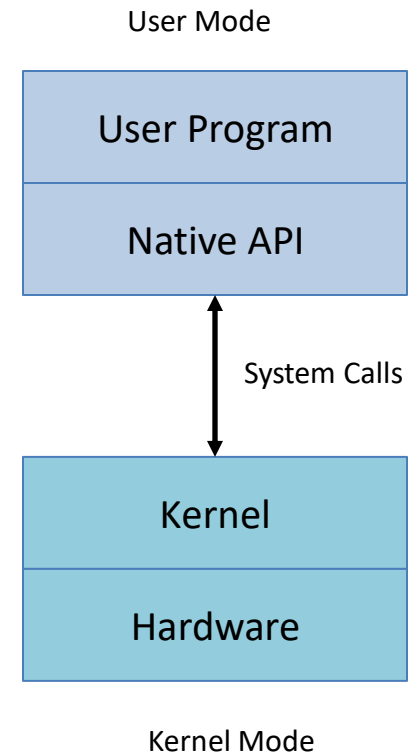
# Kernel and Shell

- အမှန်တော့ Operating Systems များကိုယ်တိုင်သည်ပင် Computer Program များဖြစ်ပါသည်။ ထို Program Code များသည် Process Management, Memory Management စသည့် Operating Systems များ၏ အဓိက တာဝန်များကို လုပ်ဆောင်ဖို့ ဖြစ်သည်။
- ထို့ကြောင့် Operating Systems များ၏ အဓိက တာဝန်များကို လုပ်ဆောင်ဖို့ ရေးသားထားသော Program Code များကို Kernel ဟုခေါ်ပါသည်။
- Kernel ဟူသည် Computer များ၏ Instruction Set Architecture (ISA) နှင့် Instruction Set များကို တိုက်ရိုက်ထိတွေ့ရပြီး Computer များ၏ Resources (Memory, File Storage, IO Devices) များကို Manage လုပ်သော Computer Program များ ဖြစ်ပါသည်။
- Shell ဟူသည် User များနှင့် တိုက်ရိုက်ထိတွေ့ရပြီး UI (User Interface) များ၊ API (Application Programming Interface) များ၊ System Utilities များ ပါဝင်သည်။ Shell သည် Kernel နှင့် User ကို ချိတ်ဆက်ပေးသော Computer Program များ ဖြစ်ပါသည်။



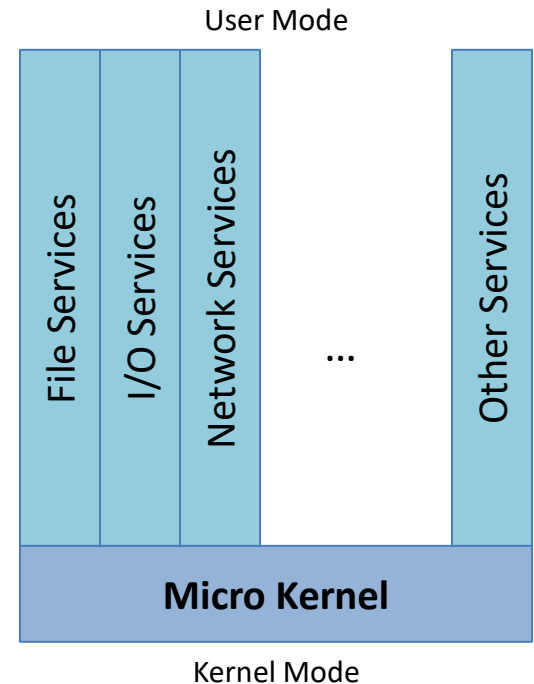
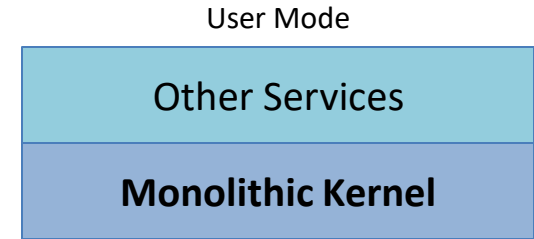
# Kernel and User Mode

- Process များကို Execute လုပ်နေချိန်တွင် Processor မှ Execute လုပ်နေသော Program Code များသည် User ၏ Program Code များ ဖြစ်သည်။
- ဒါဆို Process များကို တစ်ခုမှ တစ်ခုသို့ Switch လုပ်ဖို့၊ Process များ၏ Process States များကို Change လုပ်ဖို့ အတွက်တော့ User ၏ Program Code များကို Execute လုပ်လို့မရပါ။ Operating Systems ၏ Kernel Code များကို Execute လုပ်ရမည် ဖြစ်သည်။
- Operating Systems ၏ Kernel Code များကို Execute လုပ်ခြင်းကို Kernel Mode ဟုခေါ်ပြီး User ၏ Program Code များ ကို Execute လုပ်ခြင်းကို User Mode ဟုခေါ်သည်။
- Kernel Mode သည် User Mode ထက်ပိုပြီး Special Privileges များသည်။
- အများအားဖြင့် User ၏ Program Code များသည် Kernel ကို API (Application Programming Interface) များဖြင့် Interact လုပ်လေ့ရှိပါသည်။ ဥပမာ၊ Windows API (Win32 API, Win64 API)။
- ထိုသို့ User ၏ Program Code များမှ Kernel သို့ Access လုပ်ခြင်းကို System Call ဟုခေါ်ပါသည်။

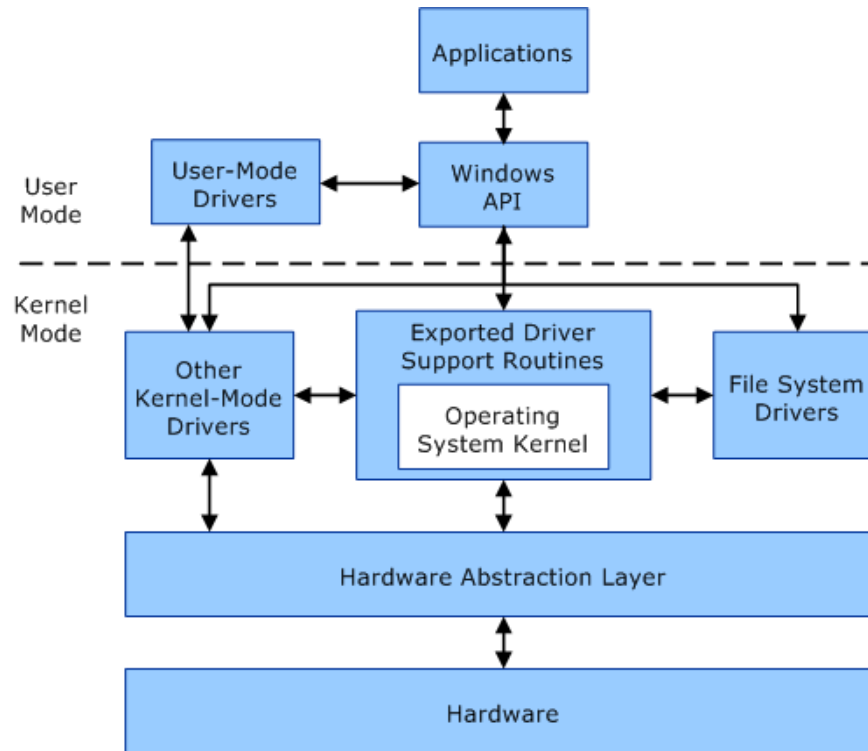


# Micro Kernel and Services

- အရင်တုန်းက Operating Systems တစ်ခု၏ Kernel သည် အကြီးကြီး ဖြစ်သည်။ ဆိုလိုသည် အရင်တုန်းက Kernel တစ်ခုသည် Operating Systems ၏ Function တွေ အားလုံးနီးပါးလောက်ကို Kernel Mode မှာ Execute လုပ်ပါသည်။
- ဒါကို Monolithic Kernel Architecture ဟုခေါ်သည်။ ပြဿနာက Function တွေ အသစ်တိုးတိုင်း Kernel တစ်ခုလုံးကို Update လုပ်ရသည်။
- ထို့ကြောင့် နောက်ပိုင်းတွင် Kernel တစ်ခုကို ကျစ်ကျစ်လစ်လစ် သေးသေးပဲထားပြီး ကျန်သော Functions များကို Services အနေဖြင့် User Mode မှာ Execute လုပ်ပါသည်။
- ဒါကို Micro Kernel Architecture ဟုခေါ်သည်။ Kernel တစ်ခုတွင် အရေးကြီးဆုံးအစိတ်အပိုင်းများသာ ပါတော့သည်။ ကျန်သော Functions များသည် Services များ ဖြစ်လာပြီး လိုရင်လိုသလို ပြောင်းလို့ရလာသည်။ ဥပမာ၊ Windows တစ်ခုကို Safe Mode ဖြင့် Run လျှင် အရေးကြီးသော Services များသာ ပါတော့သည်။
- Linux တွင်တော့ Operating Systems ၏ Services များကို Daemons ဟုခေါ်သည်။



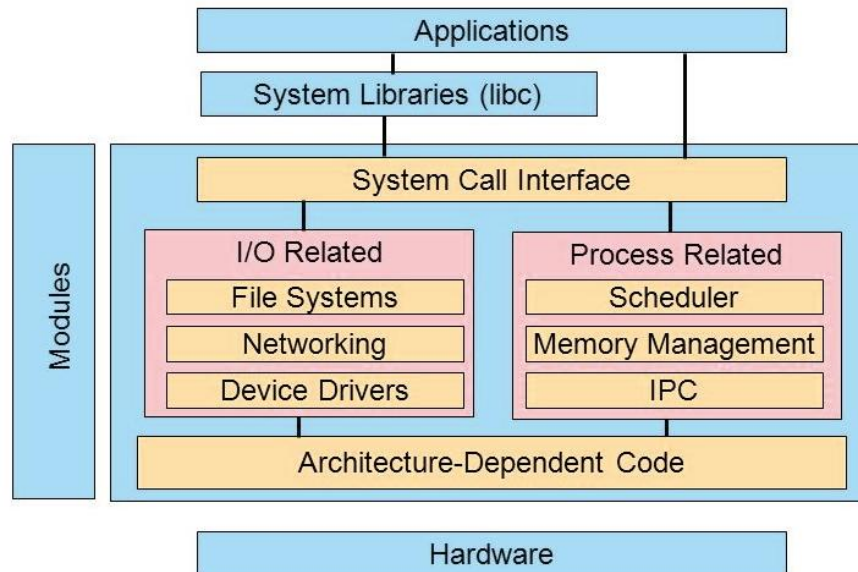
# Windows Operating Systems



- Windows Operation Systems ၏ တည်ဆောက်ပုံ အကြမ်းဖျဉ်း ဖြစ်ပါသည်။ HAL (Hardware Abstraction Layer) က ISA (Instruction Set Architecture) Independent (x86, x64) ဖြစ်အောင် လုပ်ပေးပါသည်။
- User Applications အားလုံးသည် Windows API သို့မဟုတ် Systems Run-Time (Win32, Win64) ကို Interact လုပ်ရပါသည်။

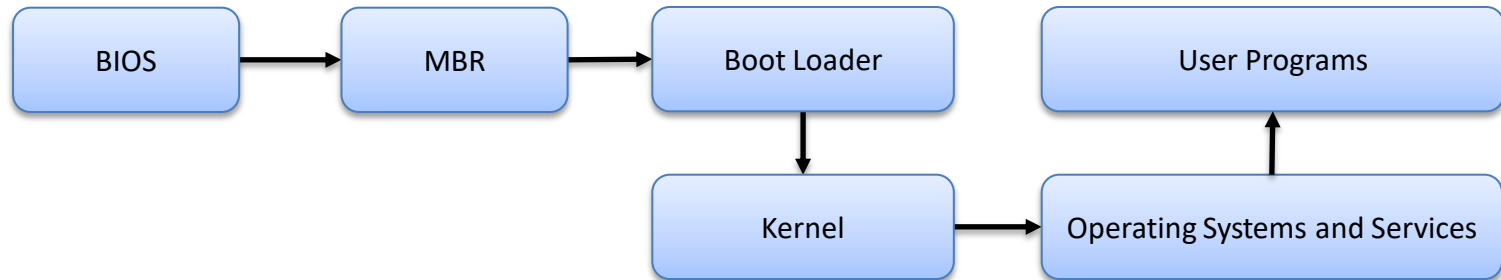


# Linux Operating Systems



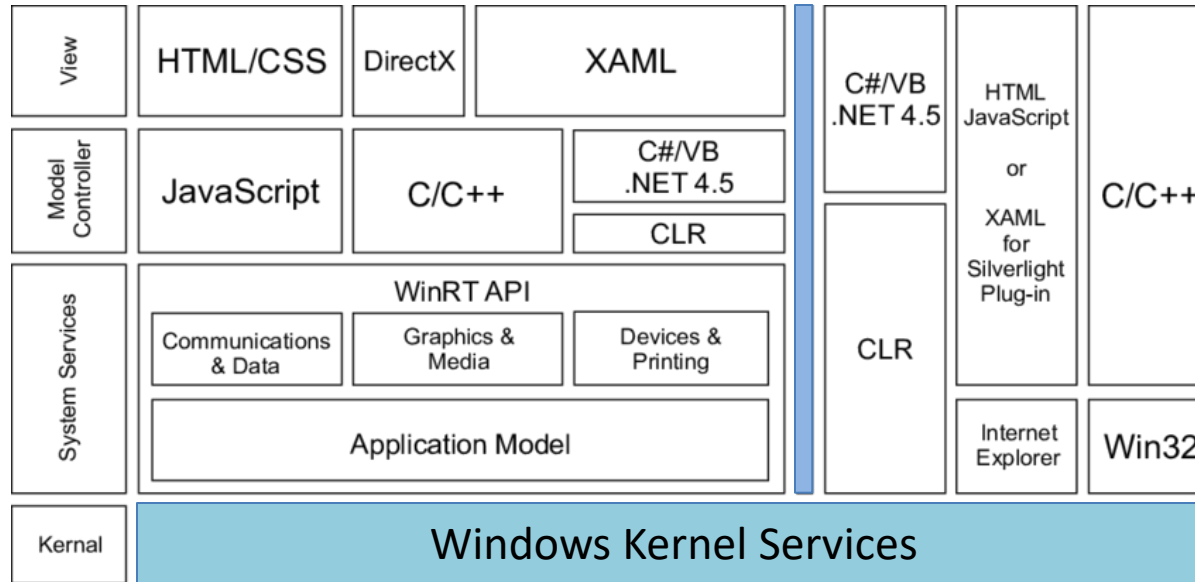
- Linux Operation Systems ၏ တည်ဆောက်ပုံ အကြမ်းဖျဉ်း ဖြစ်ပါသည်။ Linux သည် Unix မှ ဆင်းသက်လာတာ ဖြစ်ပြီး Android OS များပေါ်လည်း များစွာ Influence ဖြစ်ပါသည်။
- Architecture Dependent Code က ISA (Instruction Set Architecture) Independent (x86, x64) ဖြစ်အောင် လုပ်ပေးပါသည်။
- User Applications အားလုံးသည် System Libraries သို့မဟုတ် System Call Interface (POSIX) ကို Interact လုပ်ရပါသည်။

# BIOS (Basic Input Output Systems)



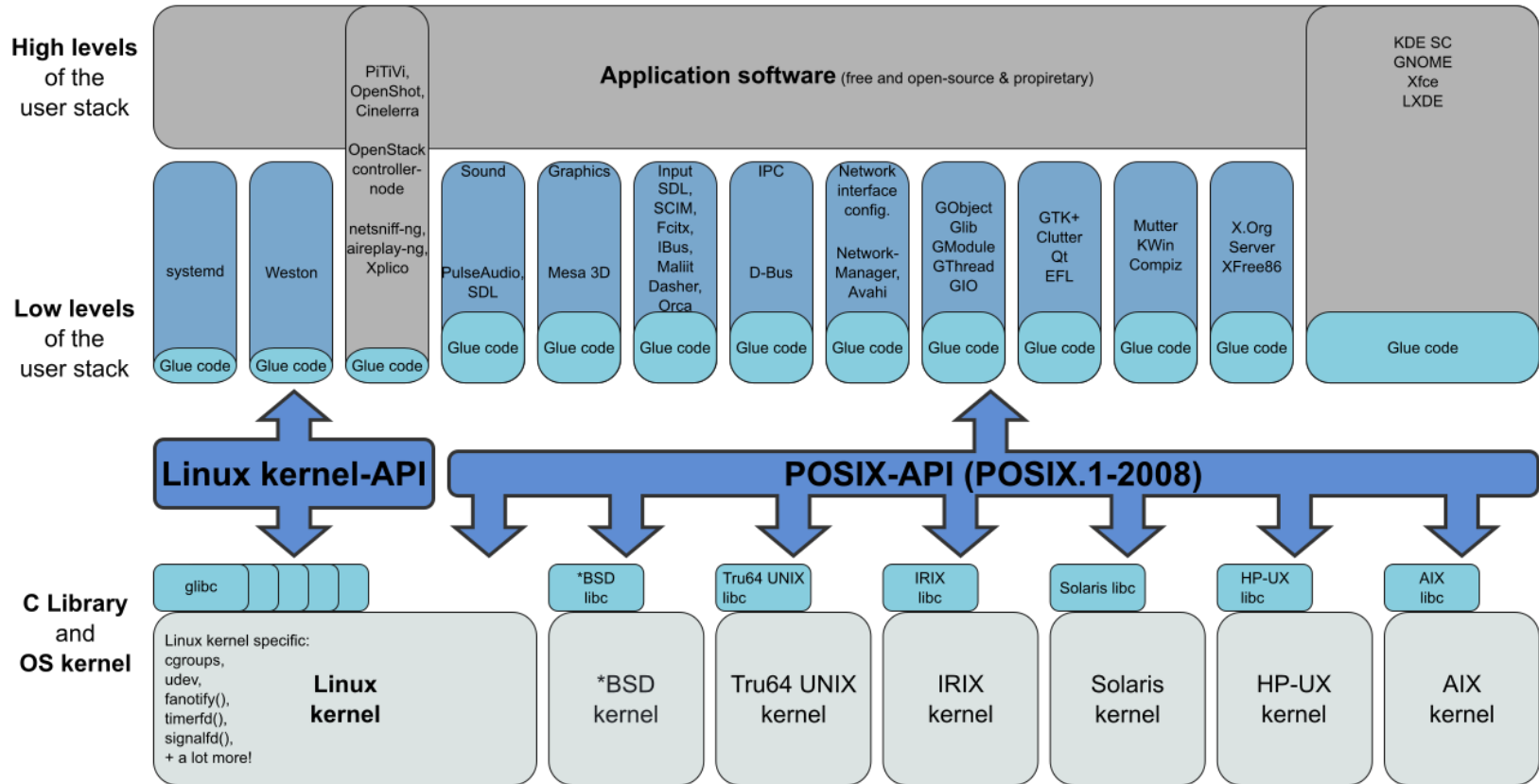
- Computer Manufacturer များမှ BIOS (Basic Input Output Systems) ဟုခေါ်သော Firmware တစ်ခုကို ROM (Read Only Memory) အနေဖြင့် ထည့်သွင်းပေးလေ့ ရှိပါသည်။ BIOS ၏ အဓိက တာဝန်က Chip Set ပေါ်ရှိ Hardware Devices များ Functional ဖြစ်၊ မဖြစ် စစ်ဖို့ နှင့် Operating Systems များကို Boot (Memory ပေါ်တင်ပေးဖို့) လုပ်ဖို့ ဖြစ်ပါသည်။
- Master Boot Record (MBR) သည် Operating Systems များကို Store လုပ်ထားသော Storage Device ၏ Location ဖြစ်ပြီး Boot Loader က Program Loader တစ်ခု ဖြစ်ပါသည်။
- Program Loader ဟူသည် Program Code နှင့် Data များကို Process များ အဖြစ် Memory ပေါ်တင်ပေးသော Firmware (Software) ဖြစ်သည်။
- OS များ မပေါ်ခင် Program Loader က User Program များကို Storage Device မှ တိုက်ရိုက် Load လုပ်ပေးရပါသည်။ OS များ ပေါ်လာပြီးနောက်တော့ Program Loader က Kernel နှင့် Operating Systems Services များကို Storage Device မှ အရင် Load လုပ်ပေးပါသည်။ ပြီးတော့မှ သက်ဆိုင်ရာ User Program များကို Load လုပ်ပါသည်။
- Safe Mode ဟူသည် Kernel နှင့် မရှိမဖြစ် (Essential) Operating Systems Services များကိုသာ Boot Loader မှ Load လုပ်ခြင်းကို ဆိုလိုသည်။ အများအားဖြင့် Operating Systems Trouble Shooting အတွက် ဖြစ်ပါသည်။

# Windows Application Environment



- အမှန်တော့ ကျွန်တော်တို့က Windows ပေါ်မှာ .Net Framework သို့မဟုတ် .Net Core နဲ့ Application ရေးမယ်ဆိုရင် Native Kernel Services များကို (Native Libraries) တိုက်ရိုက် သုံးနိုင်သော်လည်း အများအားဖြင့် CLR (Common Language Runtime) မှတစ်ဆင့် ခေါ်သုံးကြပါသည်။
- ထို့ကြောင့် Application Code များမှ .Net Framework (.Net Core) ၊ ထိုမှတစ်ဆင့် Common Language Runtime ၊ ထိုမှတစ်ဆင့် Native Kernel Services များကို သွားလေ့ရှိပါသည်။
- သို့သော် တခါတလေ .Net Framework နှင့် CLR (Managed API Code) ကို By Pass လုပ်ပြီး Win32 သို့မဟုတ် Win64 Library (Native API Code) များကို တိုက်ရိုက်ခေါ်သုံးရလေ့ ရှိပါသည်။ (အထူးသဖြင့် Performance ပိုကောင်းအောင်လို့ ဖြစ်သည်။)

# Linux Application Environment



- Linux ကို Monolithic Kernel လို့ပြောရမှာ ဖြစ်ပါတယ်။ OS ရဲ့ အခြေခံ Services တွေကို Kernel ထဲမှာထည့်ပေါင်းထားတာကိုတွေ့ရမှာပါ။
- AppArmor လို Mandatory Access Control (MAC) Framework ကို Kernel ရဲ့ Built-in Module အနေနဲ့အလုပ်လုပ်ပါတယ်။
- Net Filter Firewall Module နဲ့ KVM Virtualization Module တွေကို Dynamic Module အနေနဲ့ Load လုပ်မှာ ဖြစ်ပါတယ်။

# Process Management

- အခုဆိုရင် ကျွန်တော်တို့ Process များ၊ Operating Systems Kernel နှင့် Services များ၊ BIOS နှင့် Boot Loader များ အကြောင်းကို နည်းနည်း နားလည်လောက်ပါပြီ။
- ကျွန်တော်တို့ Process Management ကို ပိုပြီး အသေးစိတ်ထပ်ပြီး ဆွေးနွေးသွားပါမည်။ ပြောခဲ့သည့်အတိုင်း Process Management သည် Operating Systems များ၏ အသက်ဖြစ်ပါသည်။
- Process Management တွင်
  - Process Creation and Termination
  - Process Switching (Context Switch and Swapping)
  - Multithreading (Process Execution)
  - Process Synchronization and Communication
  - Parallel and Concurrent Processing
- Process Management ကို အများအားဖြင့် Operating Systems Kernel နှင့် Services များက လုပ်ဆောင်လေ့ရှိပြီး User Program များ အနေဖြင့် API များနှင့် System Calls များကို သုံး၍ အတိုင်းအတာ တစ်ခုအထိ Process Management လုပ်ဆောင်လို့ ရပါသည်။ ဥပမာ၊ Parent Process မှ Child Process များ Fork (Create) လုပ်ခြင်း၊ Execution Thread များကို Create လုပ်ခြင်း။

# Process Creation and Termination

- Process များကို Create လုပ်သော အခြေအနေ 2 မျိုး ရှိပါသည်။ ပထမက Operating Systems က Create လုပ်တာ ဖြစ်ပြီး ဒုတိယက User Application Program က Create လုပ်တာ ဖြစ်ပါသည်။
- Windows သို့မဟုတ် Linux Desktop ပေါ်ရှိ Application Icon ကို နှိပ်ခြင်းဖြင့် User Application Program ၏ Process တစ်ခုကို Operating Systems က Create လုပ်ပါသည်။
- တဖန် User Application Program ၏ Program Code မှလည်း Process အသစ်များကို Spawn လုပ်နိုင်ပါသည်။ ထိုသို့ User Application Program လုပ်သော Process များသည် မူလ Process ၏ Child Process များ ဖြစ်သွားပါသည်။
- ဒီလို User Application Program ၏ Program Code မှ Kernel Services များကို Access လုပ်ခြင်းသည် System Call ဖြစ်သည်။
- User Application Program ၏ Program Code မှ Process အသစ်များကို Spawn လုပ်ပုံကို ကျွန်တော်တို့ .Net Framework (CLR) ဖြင့်ရော Native API (Win32) ဖြင့်ပါ စမ်းကြည့်ပါမည်။ Linux ပေါ်မှာလည်း ဒါကို Python သို့မဟုတ် C ဖြင့် စမ်းကြည့်နိုင်ပါသည်။
- ကျွန်တော်တို့ ပြောခဲ့သလို Process တစ်ခုမှာ Code Segment, Data Segment နှင့် PCB တို့ပါဝင်ပါသည်။

# .NET (CLR) Process Spawn and Kill

- ကျွန်တော်တို့ ဒါက ကျွန်တော် .Net Framework နဲ့ CLR ကို သုံးပြီး Process Create လုပ်ခြင်းဖြစ်ပါသည်။

```
try
{
    using (Process myProcess = new Process())
    {
        myProcess.StartInfo.UseShellExecute = false;

        myProcess.StartInfo.FileName = "C:\\WINDOWS\\system32\\notepad.exe";
        myProcess.StartInfo.CreateNoWindow = true;
        myProcess.Start();

        Console.WriteLine("Process Id {0}", myProcess.Id);

        Console.WriteLine("Press any key to kill Process Id {0}", myProcess.Id);
        Console.ReadKey();

        myProcess.Kill();
    }
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}

Console.WriteLine("Press any key to continue...");
System.Console.ReadKey();
```

# WIN32 (Native) Process Spawn and Kill

- “kernel32.dll” သည် Windows ၏ system32 (Directory) ထဲမှာ ရှိသော System Calls များအတွက် Native Kernel Library တစ်ခု ဖြစ်ပါသည်။
- ထို “kernel32.dll” သည် C Language Library (Dynamic Link Library) ဖြစ်ပြီး ကျွန်တော်တို့က .Net Application မှ ခေါ်သုံးချင်လျှင် External Library အဖြစ် Import လုပ်ရပါသည်။
- အများအားဖြင့် Native Library များကို Wrapper Class (ဥပမာ၊ MyWrapper32) များ ပြန်ရေးပြီး ထို Wrapper Class များကို ပြန်သုံးရပါသည်။

```
[DllImport("kernel32.dll", SetLastError = true, CharSet = CharSet.Auto)]
public static extern bool CreateProcess(
    string lpApplicationName,
    string lpCommandLine,
    ref SECURITY_ATTRIBUTES lpProcessAttributes,
    ref SECURITY_ATTRIBUTES lpThreadAttributes,
    bool bInheritHandles,
    uint dwCreationFlags,
    IntPtr lpEnvironment,
    string lpCurrentDirectory,
    [In] ref STARTUPINFO lpStartupInfo,
    out PROCESS_INFORMATION lpProcessInformation
);
```



# WIN32 (Native) Process Spawn and Kill

- ကျွန်တော်တို့ ဒါက ကျွန်တော် Native Library (Win32)ကို သုံးပြီး Process Create လုပ်ခြင်းဖြစ်ပါသည်

```
try
{
    const uint NORMAL_PRIORITY_CLASS = 0x0020;

    bool retValue;
    string Application = "C:\\WINDOWS\\system32\\notepad.exe";

    MyWrapperWin32.PROCESS_INFORMATION pInfo = new MyWrapperWin32.PROCESS_INFORMATION();
    MyWrapperWin32.STARTUPINFO sInfo = new MyWrapperWin32.STARTUPINFO();
    MyWrapperWin32.SECURITY_ATTRIBUTES pSec = new MyWrapperWin32.SECURITY_ATTRIBUTES();
    MyWrapperWin32.SECURITY_ATTRIBUTES tSec = new MyWrapperWin32.SECURITY_ATTRIBUTES();

    pSec.nLength = Marshal.SizeOf(pSec);
    tSec.nLength = Marshal.SizeOf(tSec);

    retValue = MyWrapperWin32.CreateProcess(Application, "", ref pSec, ref tSec, false,
        NORMAL_PRIORITY_CLASS, IntPtr.Zero, null, ref sInfo, out pInfo);

    Console.WriteLine("Process ID (PID): " + pInfo.dwProcessId);
    Console.WriteLine("Process Handle : " + pInfo.hProcess);
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}
```

# Process Fork and Kill in Linux

- အစမှာ process type and process တွေကို analyze လုပ်ပုံတွေနဲ့ စခဲ့ပါတယ်။ အခု sectionမှာတော့ process တွေကို fork() function နဲ့ create လုပ်တာကို python နဲ့စမ်းကြည့်မှာပါ။

```
import os

def subprocess():
    print('\nForked Process ', os.getpid())
    os._exit(0)

def main():
    while True:
        newpid = os.fork()

        if newpid == 0:
            subprocess()
        else:
            pids = (os.getpid(), newpid)
            print('main process: %d, forked process: %d\n' % pids)

            reply = input('x to Exit / c for New Forked Processed')

            if reply == 'c':
                continue
            else:
                break;

main()
```

# Process Fork and Kill in Linux

- ဒီ lab script ကို execute လုပ်ကြည့်မယ် ဆိုရင် main process နဲ့ forked processes တွေကို create လုပ်တာတွေ့ရပါမယ်။

```
root@sysadmin-Virtual-Machine:/home/sysadmin# python3 fork.py
main process: 14477, forked process: 14478

x to Exit / c for new forked process
Forked process 14478
c
main process: 14477, forked process: 21074

x to Exit / c for new forked process
Forked process 21074
c
main process: 14477, forked process: 21123
```

- ပုံမှန် exit ကိုတော့ script မှာ exit function နဲ့ထွက်လို့ရပါတယ်။ နောက်ထပ် ကြုံရနိုင်တဲ့ အခြေအနေကတော့ kill utility နဲ့ stop ရတာပါ။ kill <pid> သို့မဟုတ် kill -9 <pid>

```
root@sysadmin-Virtual-Machine:/home/sysadmin# kill 14477
root@sysadmin-Virtual-Machine:/home/sysadmin#
root@sysadmin-Virtual-Machine:/home/sysadmin#

root@sysadmin-Virtual-Machine:/home/sysadmin# python3 fork.py
main process: 14477, forked process: 14478

x to Exit / c for new forked process
Forked process 14478
c
main process: 14477, forked process: 21074

x to Exit / c for new forked process
Forked process 21074
c
main process: 14477, forked process: 21123

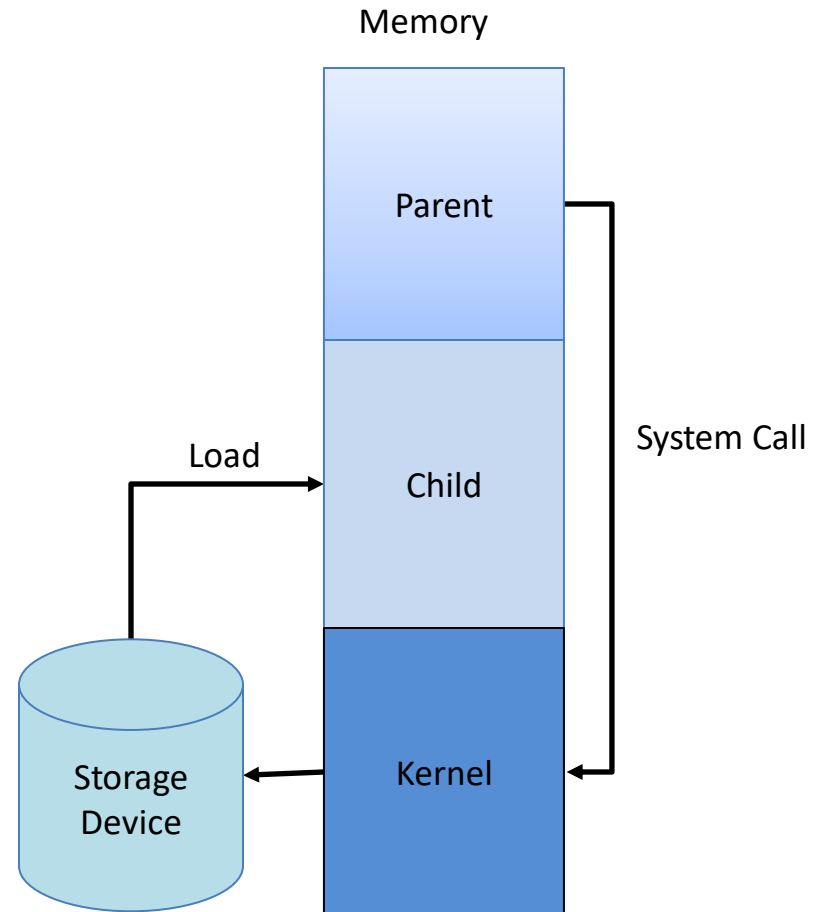
x to Exit / c for new forked process
Forked process 21123
Terminated
root@sysadmin-Virtual-Machine:/home/sysadmin#
```

# Process Switching

- ကျွန်တော်တို့ ပြီးခဲ့တဲ့ Process Creation ရဲ့ Code များကို ကြည့်ရင် User Process (Parent) မှ System Call ဖြင့် Kernel ကို Execute လုပ်ပါမည်။ ပြီးတော့ Kernel မှ Another User Process (Child) ကို ပြန်ပြီး Execute လုပ်ပါသည်။
- ထိုသို့ User Mode မှ Kernel Mode၊ Kernel Mode မှ User Mode သို့ ပြောင်းခြင်းကို Process Switch ဟုခေါ်ပါသည်။
- Processor အနေဖြင့် User Mode နှင့် Kernel Mode ကို ခွဲခြားဖို့ Flag (Process Status Word) ဆိုပြီး ရှိပါသည်။ ထို Flag ပေါ်မူတည်ပြီး User Mode နှင့် Kernel Mode ကို ခွဲခြားပါသည်။ ထို့အပြင် Execution Mode ပေါ်မူတည်ပြီး Privileges များလည်း ကွဲပြားပါသည်။ ဥပမာ၊ User Application များသည် Windows Files များကို Delete လုပ်လို့ မရပါ။
- အမှန်တော့ Process Switching သည် ထင်သလောက် မရိုးရှင်းပါ။ User Process (Parent) က နောက် User Process (Child) ကို Create လုပ်မည်ဆိုပါတော့။ Memory ပေါ်တွင် နေရာ မရှိတော့ဘူး ဆိုရင် ဘယ်လို လုပ်မလဲ။ တစ်ခါ User Application ရဲ့ Code Segment နဲ့ Data Segment (4GB) က Memory (2GB) ထက်များ နေလျှင်ရော ဘယ်လို လုပ်မလဲ။
- New Process အတွက် နေရာမရှိဘူး ဆိုလျှင် Operating Systems က Memory ပေါ်မှာ ရှိပြီးသား Processes များမှ Waiting ဖြစ်နေသော Processes များကို Storage Device ပေါ်ပြန်ဖို့ ပြီးမှ New Process ကို Load လုပ်လို့ ရပါသည်။
- ထို့ကြောင့် Process Switching သည် တော်တော်လေး ရှုပ်ထွေးပြီး Process Management များ၏ အခရာ ဖြစ်ပါသည်။ Process Switching သည် System Calls, Interrupts, Scheduling, Program Error စသည်တို့ကြောင့် ဖြစ်နိုင်ပါသည်။

# Context Switch

- User Mode မှ Kernel Mode၊ Kernel Mode မှ User Mode သို့ပြောင်းခြင်းကို Context Switch ဟုခေါ်ပါသည်။
- Context Switch တစ်ခါ ဖြစ်ပေါ်တိုင်း
  - Process State (PCB) များ ပြောင်းသွားမည် ဖြစ်သည်။
  - Process များ၏ Execution Schedule ပြောင်းသွားမည်ဖြစ်သည်။
  - Process များ၏ Location (Memory သို့မဟုတ် Storage Device) များပြောင်းသွားမည် ဖြစ်သည်။
- ထို့ကြောင့် Context Switch သည် Overhead Cost များလေ့ရှိပါသည်။
- ဒါကြောင့် Context Switch ၏ Overhead Cost ကို Reduce လုပ်ဖို့ နည်းလမ်းများ ကြံဆကြပါသည်။

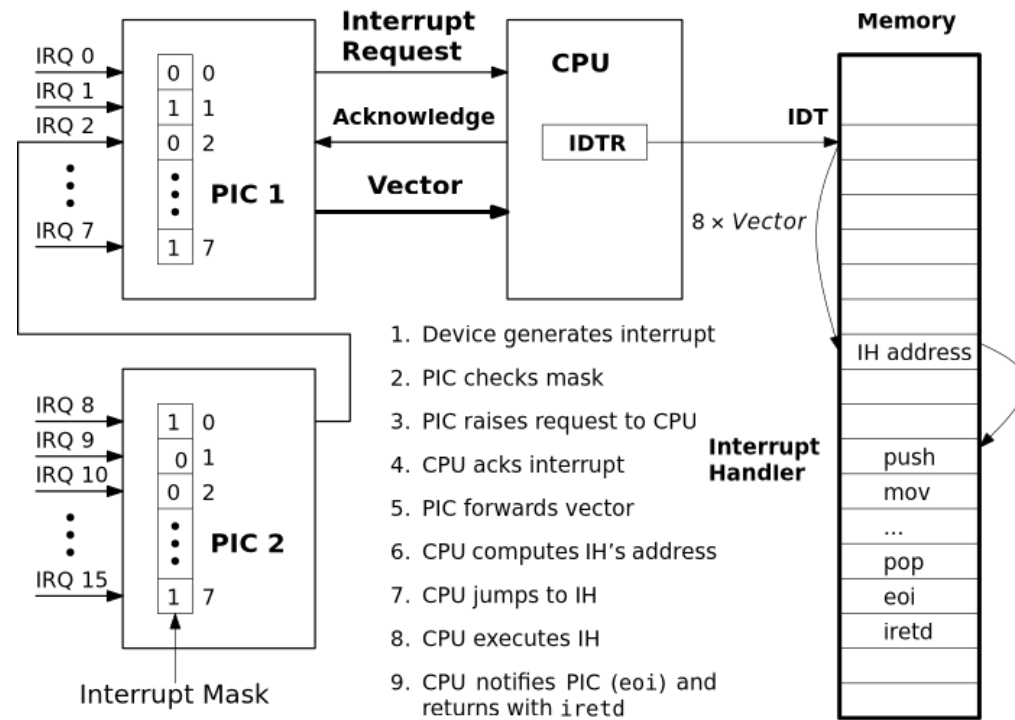


# Swapping

- အရင်ကပြောခဲ့သလို User Process (Parent) က နောက် User Process (Child) ကို Create လုပ်မည်ဆိုပါတော့။ Memory ပေါ်တွင် နေရာ မရှိတော့ဘူး ဆိုရင် ဘယ်လို လုပ်မလဲ။
- တစ်ခါ User Application ရဲ့ Code Segment နဲ့ Data Segment (4GB) က Memory (2GB) ထက်များ နေလျှင်ရော ဘယ်လို လုပ်မလဲ။
- ဒီလို အခြေအနေမျိုးနှင့် ကြုံတွေ့လာရလျှင် Swapping ကို လုပ်ဆောင်ပေးရပါသည်။ အချို့ Memory ပေါ်မှ Process များကို Storage Device မှာ ပြန် Store လုပ်ပြီး အချို့ Process များကို Storage Device မှ Memory ပေါ်သို့ ပြန် Load လုပ်ရပါသည်။
- Swapping သည် အင်မတန်ကို Overhead Cost များပါသည်။ ထို့ကြောင့် Process တစ်ခုလုံးကို Storage Device နှင့် Memory အကြား Swapping လုပ်မည့်အစား Process တစ်ခုကို သေးငယ်သော Chunk (Page) များ အနေဖြင့် ခွဲခြမ်းလိုက်ပြီး ထို Page များကိုသာ Swapping လုပ်ပါသည်။
- ဒါကို Memory Management တွင် အသေးစိတ် ဆွေးနွေးပါမည်။

# Interrupts

- Interrupts များကို အကြမ်းအားဖြင့် Device Level System Events များ အဖြစ် ယူဆနိုင်ပါသည်။
- အထူးသဖြင့် I/O Devices များနှင့် Processor များ Communicate လုပ်ဖို့ အတွက် Interrupt များကို အသုံးပြုကြပါသည်။
- Keyboard မှ Key Press လုပ်လိုက်သော Key Code များ (Text များ) ကို ပထမဆုံး I/O Buffer တစ်ခုအတွင်း သိမ်းထားမည် ဖြစ်သည်။ ထို့နောက် Interrupt Flag တစ်ခု၏ Status ကို Update (IRQ) လုပ်ပါမည်။ Processor မှ နောက် Clock Cycle တွင် Interrupt Flag ကို Check လုပ်ပြီး Status က IRQ = 1 (0x09) ဆိုပါက Keyboard I/O Buffer မှ Memory ပေါ်သို့ Data များကို Store လုပ်မည်ဖြစ်၍ Kernel က Keyboard Event တစ်ခုကို Trigger လုပ်မည် ဖြစ်သည်။ ဒါသည်ပင် Interrupt တစ်ခု အလုပ်လုပ်ပုံ ဖြစ်ပါသည်။
- Interrupts များကို Device Level System Events များအဖြစ် ယူဆနိုင်ပြီး Operating Systems ၏ Kernel က ဒီဟာကို Handle လုပ်ပါသည်။ ဒါကို Interrupt Handler ဟုခေါ်ပါသည်။



IRQ = Interrupt Request

PIC = Priority Interrupt Controller

IH = Interrupt Handler (Kernel Code)

# x86 IRQ (Interrupt Request)

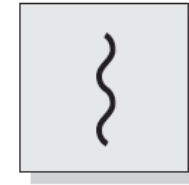
IRQ	DEVICE	VECTOR	IRQ	DEVICE	VECTOR
0	TIMER	0x08	8	REAL TIME CLOCK	0x070
1	KEYBOARD	0x09	9	REPLACE IRQ2	0x071
2	SLAVE 8259	0x0A	10	RESV (RESERVED)	0x072
3	COM1 (SERIAL PORT)	0x0B	11	RESV (RESERVED)	0x073
4	COM2 (SERIAL PORT)	0x0C	12	MOUSE	0x074
5	SOUND CARD (RESV)	0x0D	13	MATH COPROCESSOR	0x075
6	FLOPPY DRIVE	0x0E	14	HARD DRIVE	0x076
7	PARALLEL PORT	0x0F	15	RESV (RESERVED)	0x077

- Interrupts များကို အကြမ်းအားဖြင့် Device Level System Events များ အဖြစ် ယူဆနိုင်ပြီး Operating Systems ၏ Interrupt Handler များကို Handle လုပ်ပါသည်။ ထို့သို့ လုပ်ဆောင်စဉ်အတွင်း Context Switch များကိုလည်း လုပ်ဆောင်ရမည် ဖြစ်သည်။
- Interrupt Handler များမှ User Application များနှင့် သက်ဆိုင်ရာ System Services များသို့ System Events များ အနေဖြင့် Event Bubble Up လုပ်ပါသည်။
- Mouse Click IRQ ကို Interrupt Handler က Handle လုပ်ပြီး Mouse Click Event အနေဖြင့် User Application သို့ Trigger လုပ်မည် ဖြစ်သည်။

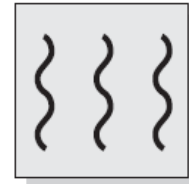


# Multithreading

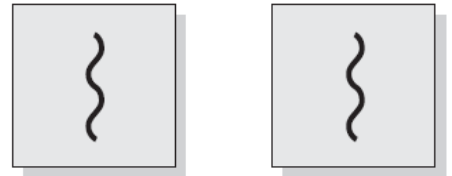
- Context Switch များသည် အရင်ကပြောခဲ့သလို Overhead Cost များပါသည်။
- စဉ်းစားကြည့်ပါ။ YouTube™ က Music Video ကိုကြည့်လျှင် Network, Video, Audio စသည့် Multiple Process များကို အချိန်နှင့် တပြေးညီ Execute လုပ်ရမည် ဖြစ်ပါသည်။ ထိုအခါ Network Process, Video Process, Audio Process စသည်တို့ကို Context Switch သာ တစ်ချိန်လုံးလုပ်နေရပါက အစဉ်မပြေပါ။ Video Update လုပ်နေတုန်း Audio က Waiting ဖြစ်နေမည်ဆိုပါက လုံးဝအဆင်မပြေဘဲ ပြတ်တောင်းပြတ်တောင်း ဖြစ်နေပါမည်။
- ထို့ကြောင့် Process တစ်ခုလုံးကို Switch လုပ်မည့်အစား Execution Thread များကိုသာ Switch လုပ်ခြင်းဖြင့် Context Switch များ၏ Overhead ကိုလျော့ချလိုပါသည်။ ထိုအခါ Application များ၏ Responsiveness များလည်း ပိုကောင်းလာပါသည်။
- Process များကို Processor က တိုက်ရိုက် Execute မလုပ်တော့ဘဲ Executing Thread များကို Create လုပ်ပြီး ထို Thread များကိုသာ Execute လုပ်ပါတော့သည်။ ဒါကို Multithreading ဟုခေါ်ပါသည်။
- ပုံမှန်အားဖြင့် Process တစ်ခုမှာ Execution Thread တစ်ခုရှိသော်လည်း Process တစ်ခုမှာ Multiple Thread များလည်း ရှိနိုင်ပါသည်။
- Processor က Process တစ်ခု သို့မဟုတ် Process များမှ Thread များကို အလှည့်ကျ Execute ပါသည်။



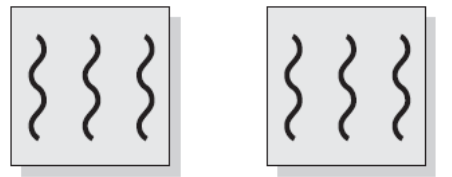
Single Process Single Thread



Single Process Multiple Thread

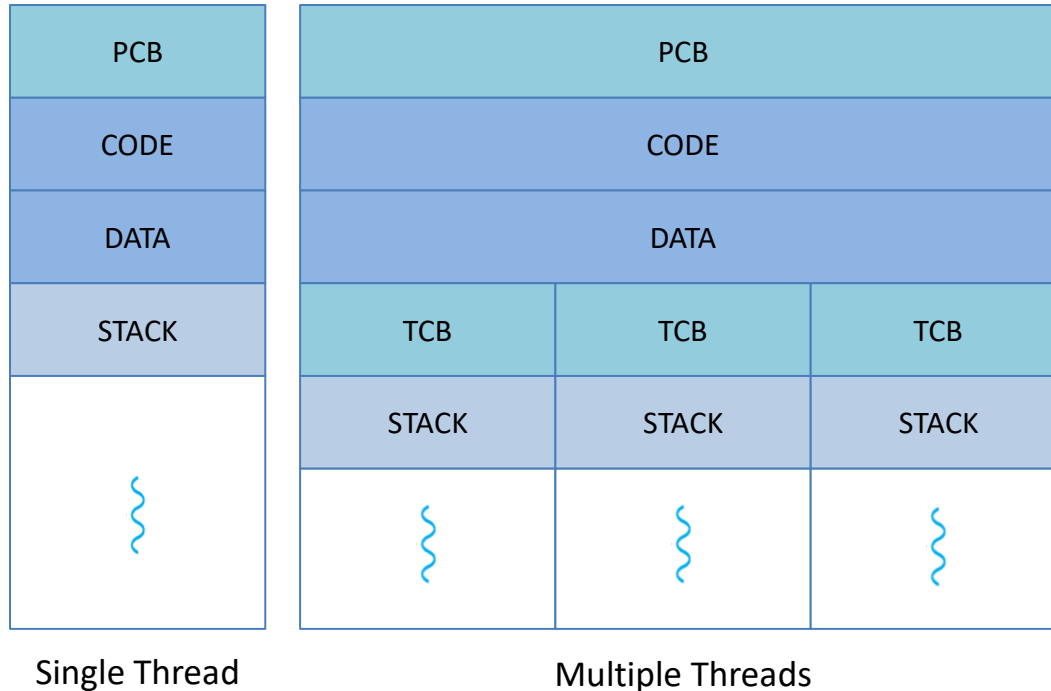


Multiple Process Single Thread



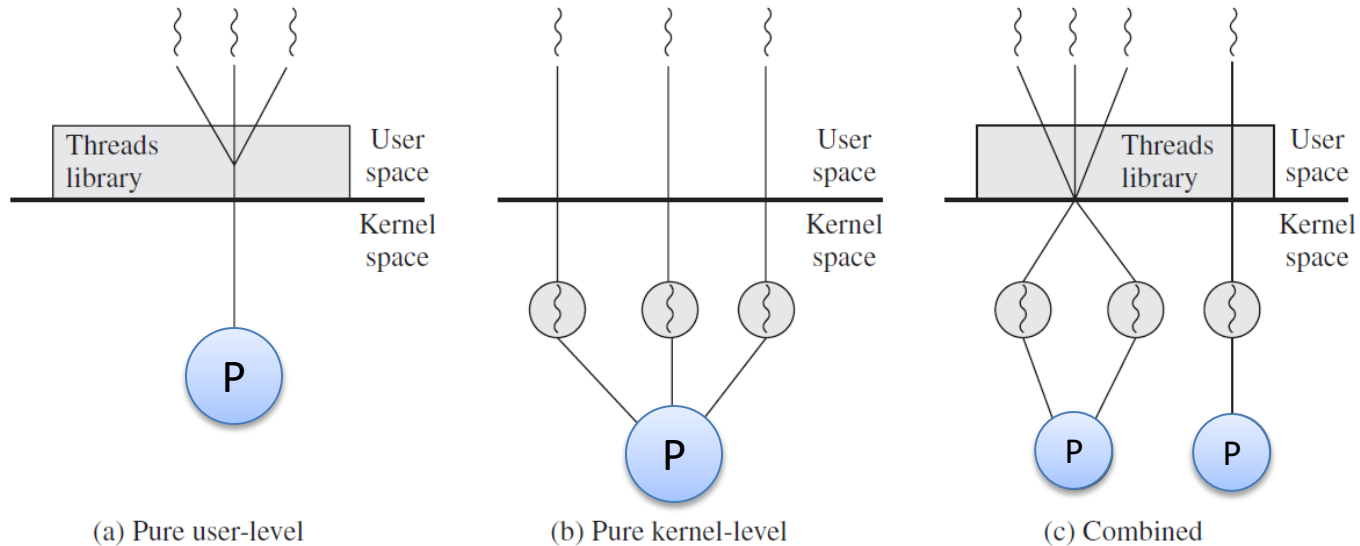
Multiple Process Multiple Thread

# Multithreading Model



- Multithreadingတွင် PCB (Process Control Block) အပြင် TCB (Thread Control Block) များပါ ထပ်ပြီးပါဝင်ပါသည်။ TCB က Executing Thread များ၏ State များနှင့် အခြား Information များကို သိမ်းဆည်းထားမည် ဖြစ်သည်။
- ထို့ကြောင့် Process Switching အစား အများအားဖြင့် Thread Switching ကိုသာ လုပ်ဆောင်ပါတော့သည်။ Thread Switching သည် Process Switching ထက်စာလျှင် Overhead Cost သက်သာရုံသာမက Multiple Processor များအတွက် Multiple Processing (Parallel Processing) ကိုပါ များစွာ အထောက်အကူ ပြုပါသည်။

# User Level and Kernel Level Thread



- ULT (User Level Thread) များကို User Mode မှာသာ Execute လုပ်မည် ဖြစ်ပြီး အများအားဖြင့် User Applications များက Thread Management လုပ်ပါသည်။
- KLT (Kernel Level Thread) များကို Kernel Mode မှာ Execute လုပ်ပြီး Kernel နှင့် Operating System Services များက Thread Management လုပ်ပါသည်။

# Multithreading in Windows

- Multithreading ကို Windows SDK (.Net Framework) က Support လုပ်ပါသည်။ Thread များကို Windows Thread Manager က Kernel Mode တွင် Handle လုပ်သဖြင့် KLT (Kernel Level Thread) များ ဖြစ်ပါသည်။

```
public static void MyThread()
{
    for (int j = 0; j < 10; j++)
    {
        Console.WriteLine("Thread with Parameter: " + j);
        Thread.Sleep(1000);
    }
}

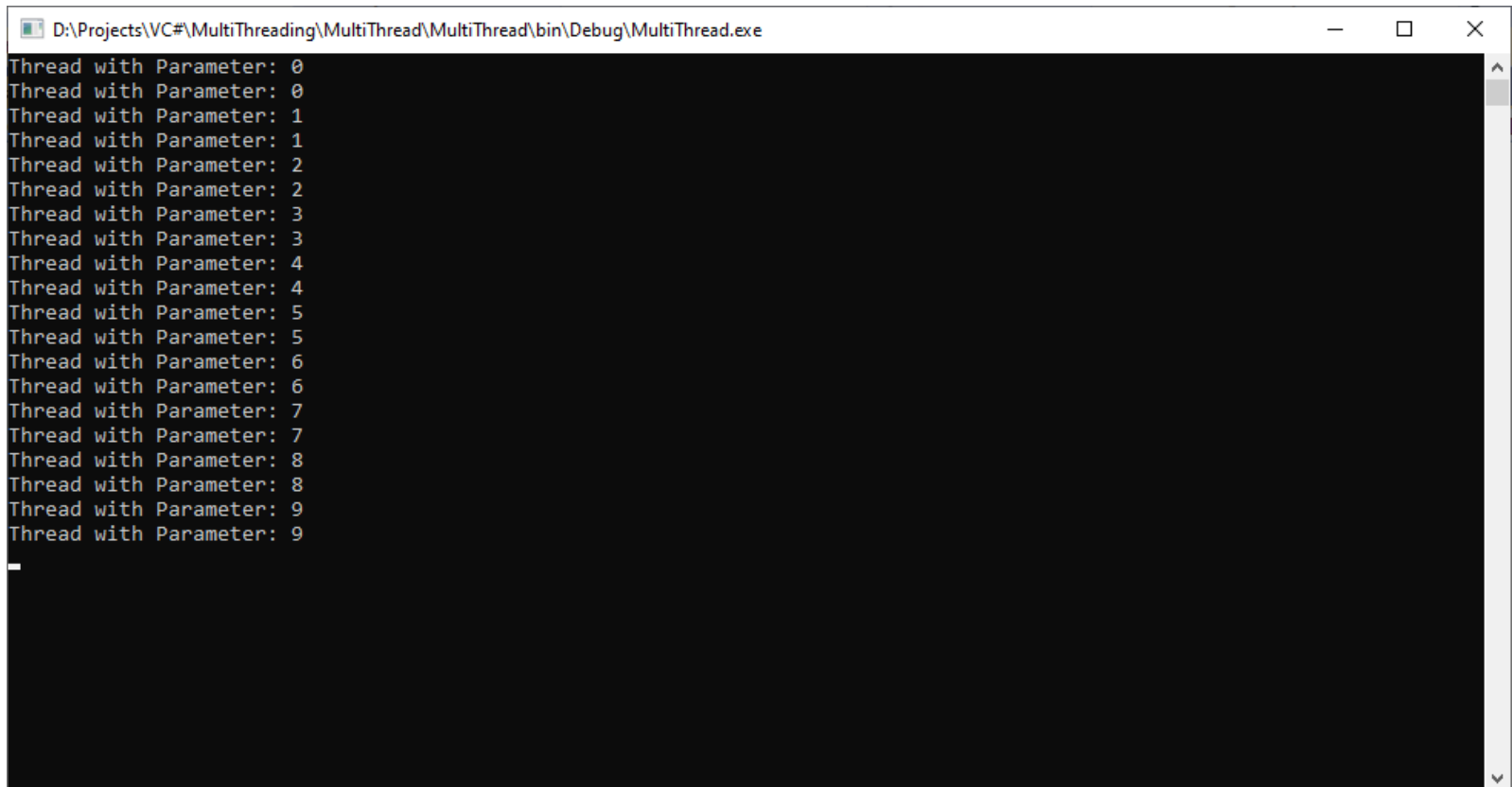
static void Main(string[] args)
{
    Thread th1 = new Thread(MyThread);
    th1.Start();

    Thread th2 = new Thread(MyThread);
    th2.Start();

    Console.ReadKey();
}
```

# Multithreading in Windows

- ကျွန်တော်တို့ ဖော်ပြပါ Code ကို Run ကြည့်လျှင် ဒီလို Result ရရှိမည် ဖြစ်ပါသည်။



```
D:\Projects\VC#\MultiThreading\MultiThread\MultiThread\bin\Debug\MultiThread.exe
Thread with Parameter: 0
Thread with Parameter: 0
Thread with Parameter: 1
Thread with Parameter: 1
Thread with Parameter: 2
Thread with Parameter: 2
Thread with Parameter: 3
Thread with Parameter: 3
Thread with Parameter: 4
Thread with Parameter: 4
Thread with Parameter: 5
Thread with Parameter: 5
Thread with Parameter: 6
Thread with Parameter: 6
Thread with Parameter: 7
Thread with Parameter: 7
Thread with Parameter: 8
Thread with Parameter: 8
Thread with Parameter: 9
Thread with Parameter: 9
```

# Multithreading in Linux

- Ubuntu မှာ Multithreading ကို python script နဲ့ စမ်းကြည့်ပါမယ်။

```
import threading
import os

def task1():
    print("Task 1 assigned to thread: {}".format(threading.current_thread().name))
    print("ID of process running task 1: {}".format(os.getpid()))

def task2():
    print("Task 2 assigned to thread: {}".format(threading.current_thread().name))
    print("ID of process running task 2: {}".format(os.getpid()))

if __name__ == "__main__":
    print("ID of process running main program: {}".format(os.getpid()))
    print("Main thread name: {}".format(threading.current_thread().name))

    t1 = threading.Thread(target=task1, name='t1')
    t2 = threading.Thread(target=task2, name='t2')

    t1.start()
    t2.start()

    t1.join()
    t2.join()
```

# Multithreading in Linux

- Script ရဲ့ output မှာ thread ၂ ခုလုံးရဲ့ `os.getpid()` process ID တွေတူနေတာကို တွေ့ရမှာဖြစ်ပါတယ်။

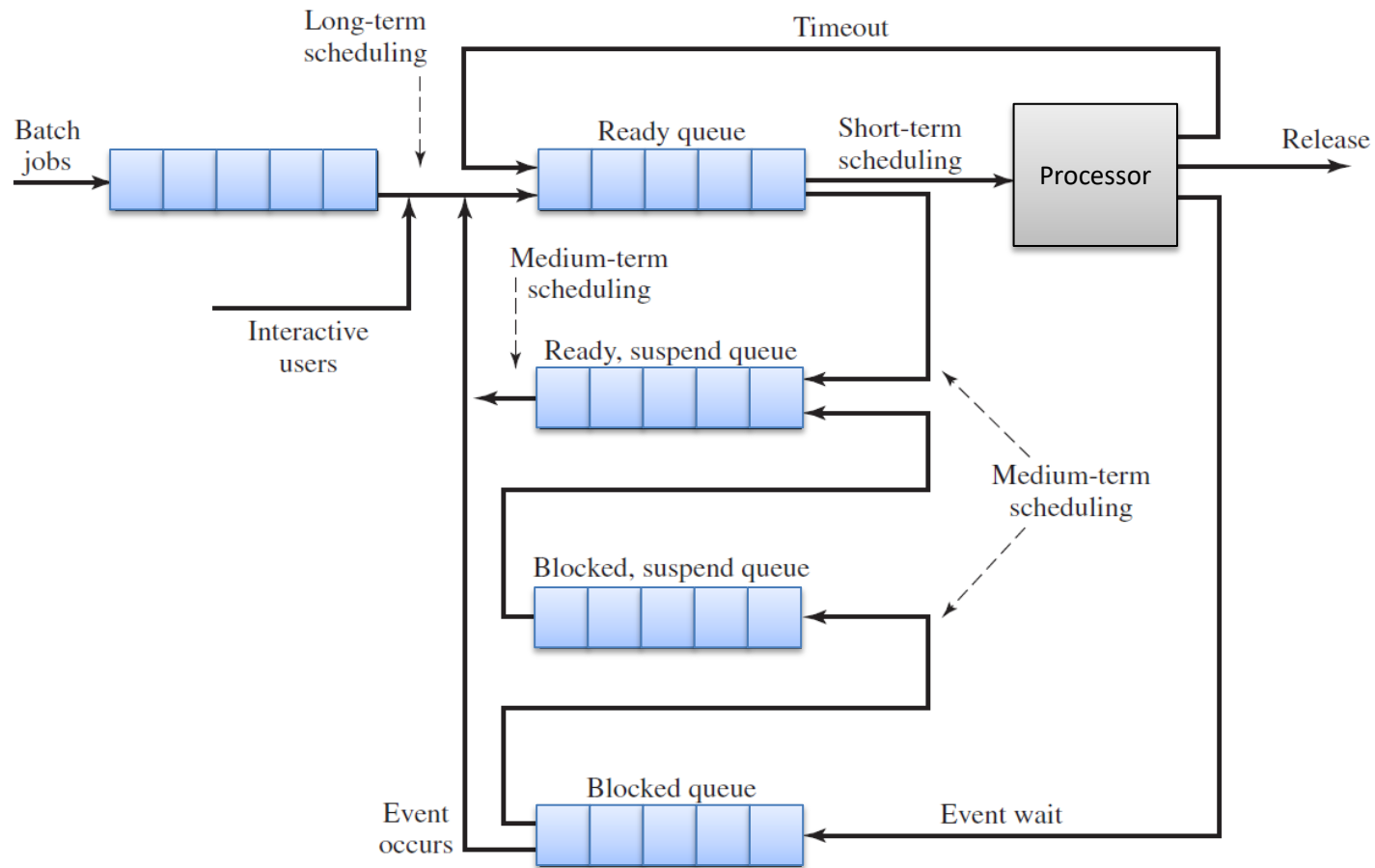
```
root@sysadmin-Virtual-Machine:/home/sysadmin# python3 thread.py
ID of process running main program: 2676
Main thread name: MainThread
Task 1 assigned to thread: t1
Task 2 assigned to thread: t2
ID of process running task 2: 2676
ID of process running task 1: 2676
root@sysadmin-Virtual-Machine:/home/sysadmin#
```

# Process Scheduling

- Process များကို Execute လုပ်ရာတွင် Single Thread ဖြစ်ဖြစ်၊ Multiple Thread ဖြစ်ဖြစ် Process များကို Schedule လုပ်ရမည် ဖြစ်သည်။
- အရင်က Process များကို Schedule လုပ်ရာတွင် Queue နှင့် Priority Queue များကို အသုံးပြုကြောင်း ဆွေးနွေးခဲ့ပါသည်။ အခု ကျွန်တော်တို့နည်းနည်း ပိုပြီး အသေးစိတ်ကို ဆွေးနွေးပါမည်။
- Process Scheduling များကို
  - Long Term Scheduling
  - Medium Term Scheduling
  - Short Term Scheduling
- Long Term Scheduler က တစ်ခါတစ်လေမှ အလုပ်လုပ်ပါမည်။ Long Term Scheduler က ဘယ် Processes များကို Admit လုပ်မည်ကို ဆုံးဖြတ်ပါသည်။ ဥပမာ၊ Windows ကို Run နေလျှင် Windows Explorer Process ကို Memory မှာ တင်ထားရမည် ဖြစ်သည်။
- Medium Term Scheduler က ရံဖန်ရံခါ အလုပ်လုပ်ပါမည်။ Medium Term Scheduler က ဘယ် Processes များကို Swapping လုပ်မည်ကို ဆုံးဖြတ်ပါသည်။ ။ ဥပမာ၊ Admit လုပ်လိုက်သော Process အသစ်များ အတွက် Memory လိုအပ်လျှင် Idle ဖြစ်နေသော Process များကို Swapping လုပ်ပါသည်။
- Short Term Scheduler က မကြာခဏ အလုပ်လုပ်ပါမည်။ Short Term Scheduler က Interrupt များ၊ System Calls များ ကို အဓိက Handle လုပ်ပါသည်။ ဥပမာ၊ User Application က File တစ်ခုကို ဖွင့်မည်ဆိုလျှင် System Call အနေဖြင့် Kernel Mode သို့ Context Switch လုပ်မည် ဖြစ်ပြီး File ကို Access လုပ်ပြီးပါက User Mode သို့ တစ်ခါ Context Switch လုပ်မည် ဖြစ်သည်။



# Process Scheduling



# Process Dispatching

- Process Scheduling ကို အောက်ပါအခြေအနေများတွင် လုပ်ဆောင်လေ့ ရှိပါသည်။
  1. Switches from running to waiting state
  2. Switches from running to ready state
  3. Switches from waiting to ready
  4. Terminates
- အခြေအနေ 1 နှင့် 4 ကို Non-Preemptive Scheduling ဟုခေါ်ပြီး 2 နှင့် 3 ကို Preemptive Scheduling ဟုခေါ်ပါသည်။
- Short Term Scheduler မှ Select လုပ်ထားသော Process ကို Dispatcher Module မှ Context Switch လုပ်ခြင်း၊ Kernel Mode မှ User Mode သို့ပြောင်းခြင်းများကို လုပ်ဆောင်ပါသည်။
- ထိုသို့ Dispatch လုပ်လို့ကြာချိန်ကို Dispatch Latency ဟုခေါ်ပါသည်။

# Scheduling Algorithms

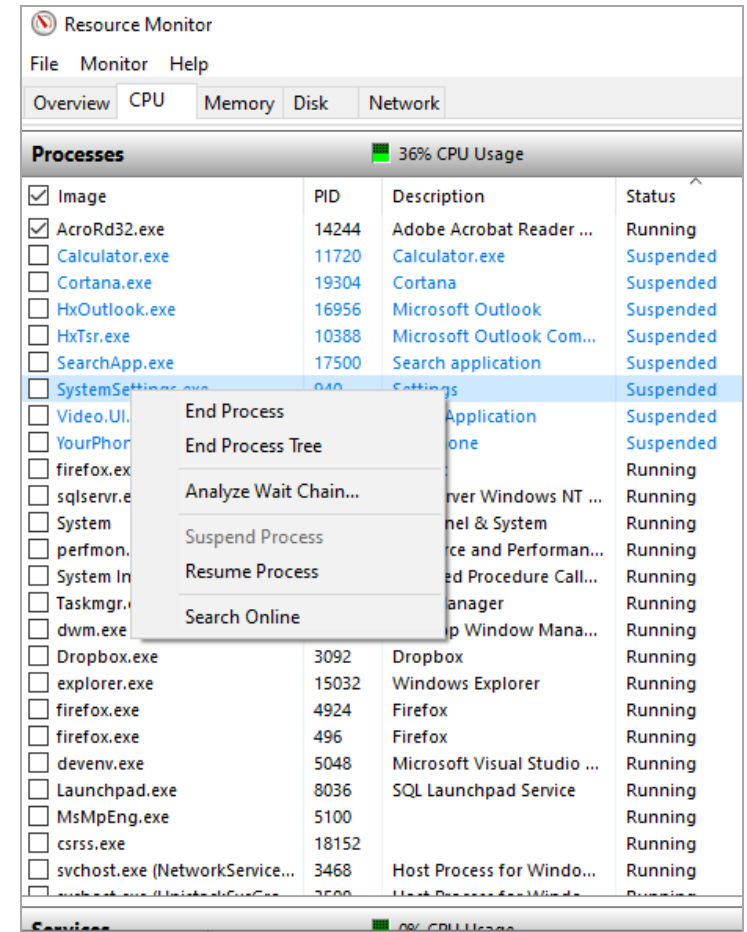
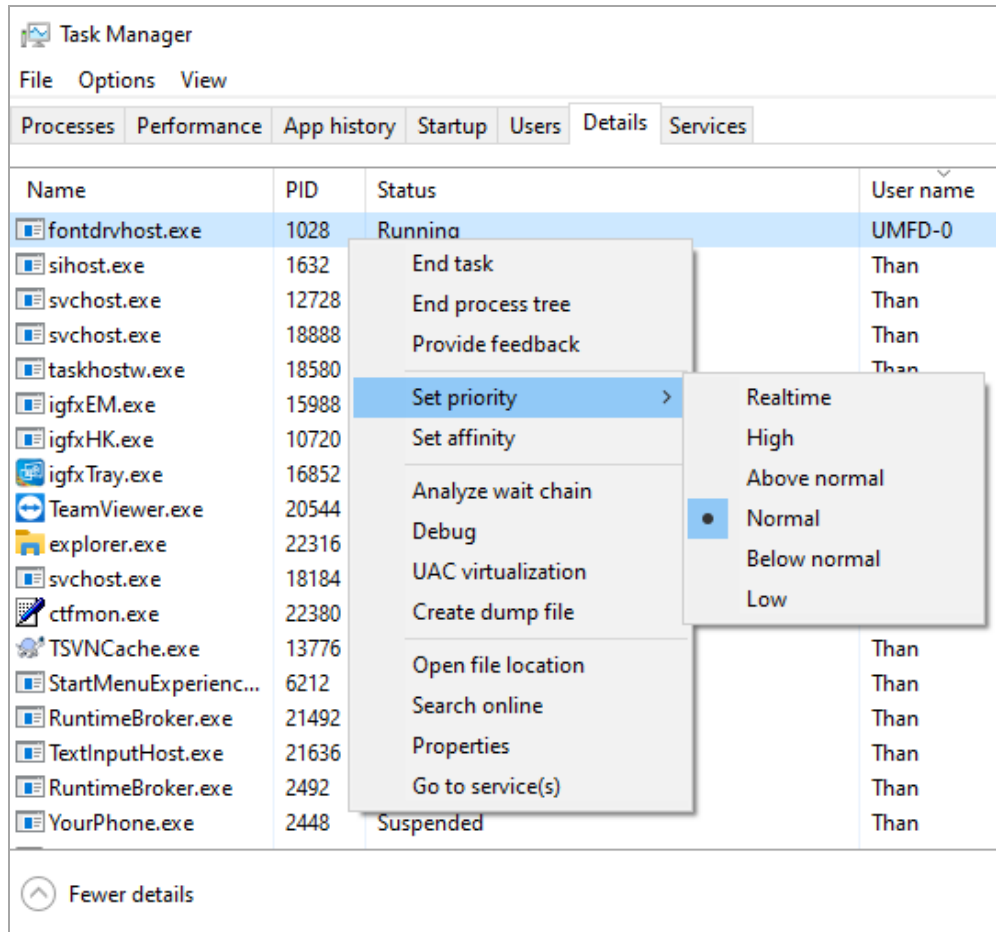
- Process များကို Schedule လုပ်ရာတွင် Algorithm များစွာရှိပါသည်။ Algorithm အများစုသည် Queuing Theory ([https://en.wikipedia.org/wiki/Queueing\\_theory](https://en.wikipedia.org/wiki/Queueing_theory)) အပေါ်အခြေခံပါသည်။
- Scheduling Algorithm များကို Criteria များပေါ်မူတည်ပြီး Performance ကို ဆုံးဖြတ်ပါသည်။
  - Max CPU Utilization
  - Max Throughput
  - Min Turnaround Time
  - Min Waiting Time
  - Min Response Time
- အသုံးများသော Scheduling Algorithm များမှာ
  - FCFS (First Come, First Serve)
  - RR (Round Robin)
  - SPN (Shortest Process Next)
  - SRT (Shortest Remaining Time)
  - HRRN (Highest Response Ratio Next)
  - FB (Feedback)

# Scheduling Algorithms Comparison

	FCFS	Round Robin	SPN	SRT	HRRN	Feedback
<b>Selection Function</b>	$\max[w]$	constant	$\min[s]$	$\min[s - e]$	$\max\left(\frac{w + s}{s}\right)$	variable
<b>Decision Mode</b>	Non-preemptive	Preemptive (at time quantum)	Non-preemptive	Preemptive (at arrival)	Non-preemptive	Preemptive (at time quantum)
<b>Throughput</b>	Not emphasized	May be low if quantum is too small	High	High	High	Not emphasized
<b>Response Time</b>	May be high, especially if there is a large variance in process execution times	Provides good response time for short processes	Provides good response time for short processes	Provides good response time	Provides good response time	Not emphasized
<b>Overhead</b>	Minimum	Minimum	Can be high	Can be high	Can be high	Can be high
<b>Effect on Processes</b>	Penalizes short processes; penalizes I/O bound processes	Fair treatment	Penalizes long processes	Penalizes long processes	Good balance	May favor I/O bound processes
<b>Starvation</b>	No	No	Possible	Possible	No	Possible

Time Quantum ဆိုသည်မှာ အလှည့်ကျ (Round Robin) Execute လုပ်ရာတွင် အချိန်တစ်ခုကြာချိန်ကို ဆိုလိုသည်။

# Processes Scheduling in Windows



- Task Manager နှင့် Resource Monitor မှ Process များကို State ကို တိုက်ရိုက်ပြောင်းနိုင်ပြီး Waiting Time ကိုလည်း Analyze လုပ်နိုင်ပါသည်။

# Processes Scheduling in Linux

- Linux က Dynamic Priority Based Scheduling ကိုသုံးပါတယ်။ လိုအပ်သလို Priority ကို adjust လုပ်တယ်လို့ ပြောရမှာပါ။ Process တွေရဲ့ priority ကို -20 အမြင့်ဆုံး နဲ့ 19 အနိမ့်ဆုံး သတ်မှတ်ထားပါတယ်။ Process အများစုကို default 0 priority နဲ့ schedule လုပ်ထားတာကိုတွေ့ရမှာပါ။

```
sysadmin@sysadmin-Virtual-Machine:~$ ps -el
F S      UID          PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S        0            1        0  0  80   0 -  42684 -        ?        00:00:10 systemd
1 S        0            2        0  0  80   0 -      0 -        ?        00:00:00 kthreadd
1 I        0            3        2  0  60 -20 -      0 -        ?        00:00:00 rcu_gp
1 I        0            4        2  0  60 -20 -      0 -        ?        00:00:00 rcu_par_gp
```

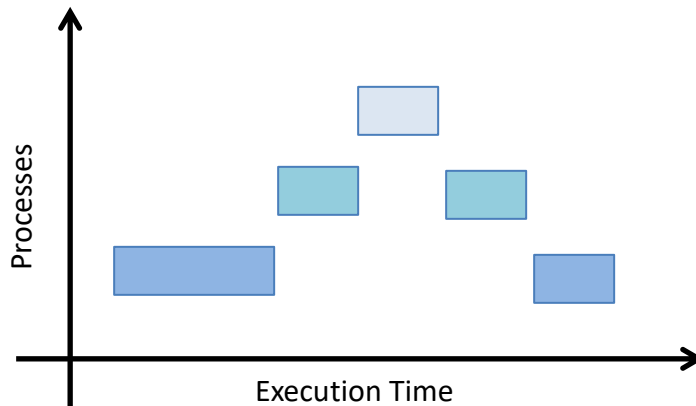
- ps -el output ရဲ့ NI column ကိုလေ့လာနိုင်ပါတယ်။ Effective priority ကိုတော့ PRI column မှာတွေ့နိုင်ပါတယ်။
- Dynamic Priority ကို ကျွန်တော်တို့ renice utility နဲ့ schedule priority ကို ပြင်ကြည့်ရအောင်။ Apache processes တွေကို renice နဲ့ -10 ပြောင်းလိုက်တာပါ။ Schedule priority က 80 ကနေ 70 ဖြစ်သွားတာကို တွေ့ရပါမယ်။

```
root@sysadmin-Virtual-Machine:/home/sysadmin# ps -el | grep apache
1 S        0          609        1  0  80   0 -  1676 poll_s ?        00:00:02 apache2
5 S       33          610       609  0  80   0 - 302918 pipe_r ?        00:00:00 apache2
5 S       33          611       609  0  80   0 - 302898 pipe_r ?        00:00:00 apache2
root@sysadmin-Virtual-Machine:/home/sysadmin# renice -n -10 -p 609 610 611
609 (process ID) old priority 0, new priority -10
610 (process ID) old priority 0, new priority -10
611 (process ID) old priority 0, new priority -10
root@sysadmin-Virtual-Machine:/home/sysadmin# ps -el | grep apache
1 S        0          609        1  0  70 -10 -  1676 poll_s ?        00:00:02 apache2
5 S       33          610       609  0  70 -10 - 302918 pipe_r ?        00:00:00 apache2
5 S       33          611       609  0  70 -10 - 302898 pipe_r ?        00:00:00 apache2
```

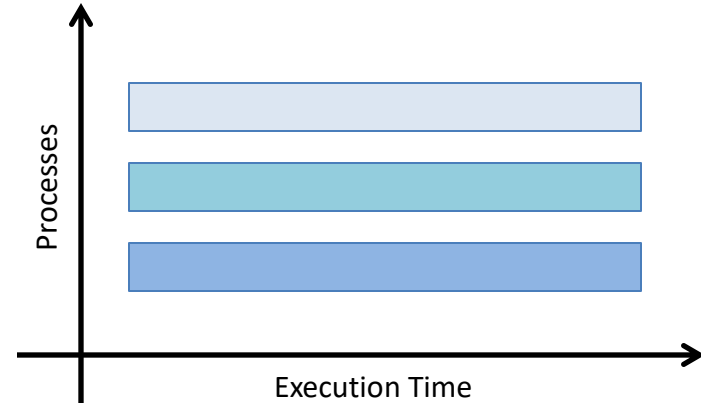
# Process Synchronization and Communication

- Process များကို Execute လုပ်ရာတွင် အများအားဖြင့် Process တစ်ခုထဲကို လုပ်လေ့မရှိ၊ လုပ်လို့မရပါ။ ဥပမာ၊ YouTube™ က Music Video ကြည့်လျှင် Network Process၊ Video Process နှင့် Audio Process များကို တစ်ခုနှင့်တစ်ခု Synchronize နှင့် Communicate လုပ်ရမည် ဖြစ်သည်။
- Process Synchronization နှင့် Communication သည် အင်မတန် အရေးကြီးပြီး Operating Systems များ၏ အဓိက လုပ်ဆောင်ချက်များတွင်ပါဝင်ပါသည်။ တစ်နည်းအားဖြင့် Process များ တစ်ခုနှင့် တစ်ခု ဘယ်လို Cooperate လုပ်မလဲကို အဓိကထား ဖြေရှင်းပါသည်။
- Process များသည် တစ်ခုနှင့် တစ်ခုကို ဒီလိုနည်းများဖြင့် Cooperate လုပ်လို့ရပါသည်။
  - Critical Section (Shared Memory)
  - Messaging (Local and Remote Procedure Call)
- Process အများကြီးကို Execute လုပ်သည့်အခါ Parallel နှင့် Concurrent Processing များလည်း ပါဝင်လာရပါသည်။ အထူးသဖြင့် Processor တစ်ခုပိုသော Multiple Processors များက Multiple Processes များကို ဘယ်လို Cooperate လုပ်မလဲသည် အဓိက ပြဿနာ ဖြစ်လာပါသည်။

# Parallel and Concurrent Processing



Concurrent Processing



Parallel Processing

- Concurrent Processing သည် Multiple Process များကို အများအားဖြင့် Time Sharing ဖြင့် (Time Quantum) Execute လုပ်ခြင်း ဖြစ်သည်။ ဒါကို Multiprogramming ဟုလည်း ခေါ်ကြပါသည်။
- Parallel Processing သည် Multiple Process များကို အများအားဖြင့် Multiple Processors များဖြင့် (Parallel) Execute လုပ်ခြင်း ဖြစ်သည်။ ဒါကို Multitasking ဟုလည်း ခေါ်ကြပါသည်။
- Concurrent မှာရော Parallel Processing မှာပါ Executing Thread များသည် အခရာကျပါသည်။ Thread အပြင် ကျွန်တော်တို့ Task ဆိုတဲ့ Concept တစ်ခုကိုပါ ဆွေးနွေးပါမည်။
- Process, Thread, Task တို့သည် Modern Operating Systems များ၏ Process Management အတွက် အရေးကြီးပါသည်။



# Synchronous and Asynchronous Tasks

- Task တစ်ခုသည် Logical Unit တစ်ခု ဖြစ်ပြီး Program တစ်ခု သို့မဟုတ် Procedure (Method) တစ်ခု ဖြစ်နိုင်ပါသည်။
- Task တစ်ခုမှာ အကြမ်းဖျဉ်းအားဖြင့် နှစ်ပိုင်းပါဝင်ပါသည်။
  - Promise
  - Callback
- တစ်နည်းအားဖြင့် Task တစ်ခုသည် Delegation တစ်ခု ဖြစ်သည်။ ခင်ဗျား ဒါလုပ်လိုက်ပါ၊ လုပ်ပြီးရင် ကျွန်တော့်ကို ပြန်ပြော ဆိုသလိုမျိုးဖြစ်သည်။ ခင်ဗျားက ဒါကိုလုပ်ပါမည်ဆိုပြီး ကျွန်တော့်ကို Promise ပေးရမည် ပြီးလျှင် လုပ်ပြီးကြောင်း သို့မဟုတ် လုပ်လို့အဆင်မပြေကြောင်း ပြန်ပြော (Callback)ရမည်။ ဒါဆိုရင် Task တစ်ခု ဖြစ်သည်။
- Synchronous Task များသည် Blocking Task များ ဖြစ်သည်။ ဆိုလိုသည်က Task တစ်ခုက Callback ပြန်မလုပ်မချင်း ထိုင်စောင့်နေမည် ဖြစ်သည်။
- Asynchronous Task များသည် Non-Blocking Task များ ဖြစ်သည်။ Task တစ်ခုက Callback လုပ်တာကို ထိုင်စောင့်မနေဘဲ အခြားဟာများ လုပ်နေမည် ဖြစ်သည်။
- Task များကြောင့် Multithreading ရော၊ Multiprogramming ရော၊ Multitasking ရော လုပ်ဆောင်ရာမှာ ပိုမိုလွယ်ကူလာပါသည်။

# Asynchronous Tasks in Windows

- ကျွန်တော်တို့ .Net Delegate နှင့် AsyncCallback ကိုသုံးပြီး Asynchronous Task တစ်ခုကို လုပ်ကြည့်ပါမည်။ Delegate သည် အမှန်တော့ Function Pointer တစ်ခုသာ ဖြစ်ပါသည်။

```
delegate int TestDelegate(string parameter);

static void Main(string[] args)
{
    TestDelegate d = new TestDelegate(Promise);

    d.BeginInvoke("Promise", new AsyncCallback(CallBack), d);

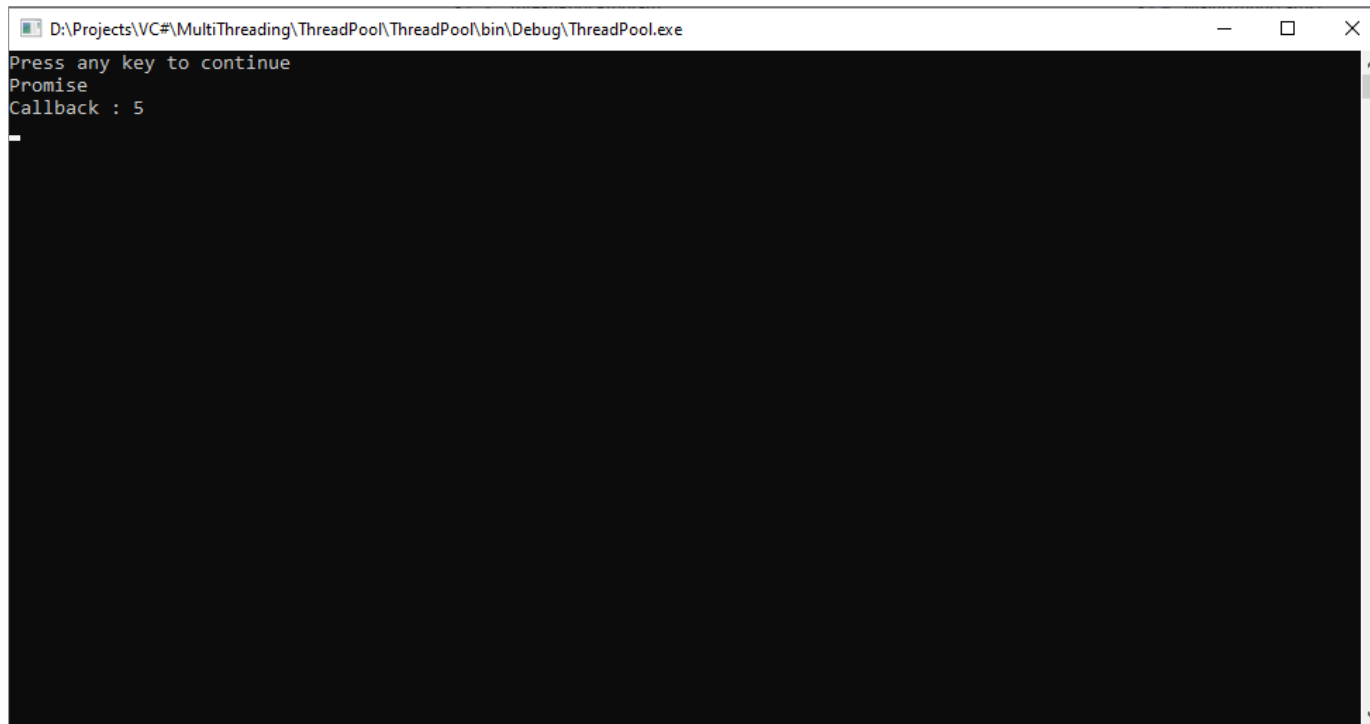
    Console.WriteLine("Press any key to continue");
    Console.ReadKey();
}

static int Promise(string parameter)
{
    Console.WriteLine(parameter);
    return 5;
}

static void CallBack(IAsyncResult ar)
{
    TestDelegate d = (TestDelegate) ar.AsyncState;
    Thread.Sleep(1000);
    Console.WriteLine("Callback : {0} ", d.EndInvoke(ar));
}
```

# Asynchronous Tasks in Windows

- ဒီ Code ကို Run ကြည့်လျှင် ဒီလိုရမည် ဖြစ်သည်။



The screenshot shows a Windows command prompt window with the title bar "D:\Projects\VC#\MultiThreading\ThreadPool\ThreadPool\bin\Debug\ThreadPool.exe". The window contains the following text:

```
Press any key to continue  
Promise  
Callback : 5  
-
```

# Task Parallelism and Data Parallelism

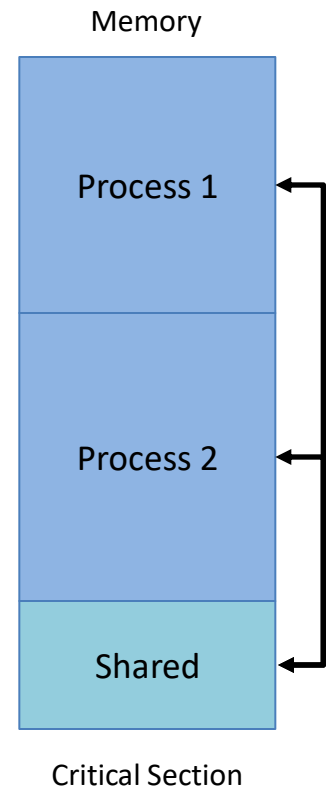
- Parallelism ကို အများအားဖြင့် Task Parallelism နှင့် Data Parallelism ဆိုပြီး ခွဲခြားနိုင်ပါသည်။
- Task Parallelism တွင် Asynchronous Task များကို Parallel Execute လုပ်ခြင်း ဖြစ်သည်။
- သို့သော် အမှန်တော့ အဲဒီလောက် မရိုးရှင်းပါ။ Multiple Task များသည် Data တစ်ခုတည်းကို ဝိုင်းပြီးတော့ “Write” လုပ်နိုင်ပါသည်။ ထိုအခါ ပြဿနာများ ပေါ်လာပါတော့သည်။
- ထို့ကြောင့် Data များကိုပါ Chunk လုပ်ပြီးတော့ Task တစ်ခု အတွက် Data တစ်ခု ဆိုပြီး လုပ်ကြပါသည်။ ဒါကို Data Parallelism ဟုခေါ်ပါသည်။
- နာမည်ကြီးသော Map-Reduce Pipeline သည် Data Parallelism ၏ Algorithm တစ်ခု ဖြစ်ပါသည်။
- သို့သော် တခါတလေ Task များကို Synchronize လုပ်ဖို့နှင့် Communicate လုပ်ဖို့ Data များကို Share လုပ်ဖို့ လိုအပ်လာပါသည်။
- ဒါကြောင့် ကျွန်တော်တို့ Critical Section (Shared Memory) ကို ဆက်ပြီး ဆွေးနွေးသွားပါမည်။

**\*\* Microsoft .Net မှာ Task Parallelism ကိုပိုကောင်းအောင် Support လုပ်ဖို့ TPL (Task Parallel Library) ကို Introduce လုပ်လာပါသည်။**

<https://docs.microsoft.com/en-us/dotnet/standard/parallel-processing-and-concurrency>

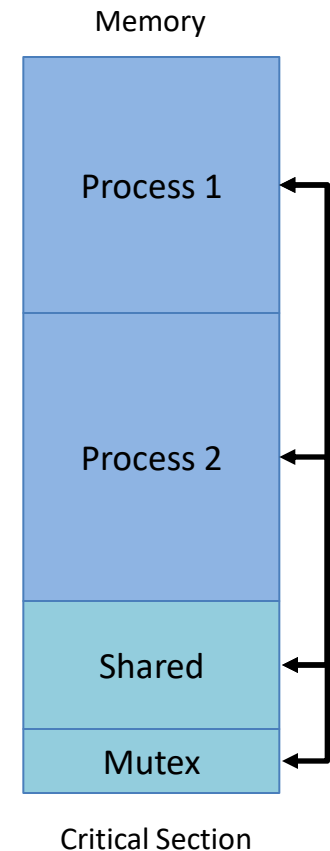
# Critical Section (Shared Memory)

- Process နှစ်ခု သို့မဟုတ် နှစ်ခုထက်ပိုသော Process များသည် Memory Space တစ်ခုကို Share လုပ်ခြင်းဖြင့် Cooperate လုပ်လို့ရပါသည်။
- သို့သော် ထိုသို့ Share Memory လုပ်ခြင်းသည် Cooperate လုပ်ဖို့ အဆင်ပြေသော်လည်း မမျှော်မှန်းနိုင်သော ကိစ္စများ ဖြစ်ပေါ်စေပါသည်။
- အထူးသဖြင့် Multithreading များ၊ Multiprocessing (More than one Processor) များပါဝင်လာသော အခါ ရှုပ်ထွေးမှုများကို ဖြစ်ပေါ်စေပါသည်။
- ထို့ကြောင့် ထို Shared Memory ကို Critical Section ဟုခေါ်ပါသည်။ ဥပမာ၊ Critical Section တစ်ခုကို Process နှစ်ခုက တပြိုင်တည်း “Write” လုပ်လို့မရပါ။ တခါတလေ Process နှစ်ခုက “Write” လုပ်ဖို့ ကြိုးစားရင်း ရှေ့တိုးမရ၊ နောက်ဆုတ်မရ “Deadlock” များ ဖြစ်တတ်ပါသည်။



# Mutex (Mutual Exclusion)

- Process နှစ်ခု သို့မဟုတ် နှစ်ခုထက်ပိုသော Process များသည် Memory Space တစ်ခုကို တစ်ပြိုင်တည်း “Write” မလုပ်ဖို့ အရိုးရှင်းဆုံးသော ကာကွယ်နည်းသည် Mutual Exclusion Lock ဖြစ်သည်။
- Process တစ်ခုက “Write” လုပ်နေလျှင် အခြား Process များက “Write” လုပ်လို့ မရ။ ဒါဆိုရင် ပြဿနာ တော်တော်များများ ပြေလည်သွားသည်။
- Mutex ကို Binary Semaphore ဖြင့် Implement လုပ်ပါသည်။
- Semaphore ဟူသည် Multiple Processes သို့မဟုတ် Multiple Tasks များအကြား Share လုပ်ထားသော Status Flag (Variable) တစ်ခု ဖြစ်ပါသည်။
- Binary Semaphore တွင် 1 နှင့် 0 ရှိပြီး Process တစ်ခုက “Write” လုပ်နေလျှင် Semaphore 1 ဖြစ်နေပြီး “Write” ပြီးသွားလျှင် 0 ပြန်ဖြစ်သွားမည် ဖြစ်သည်။
- ထို့ကြောင့် Critical Section ကို “Write” မလုပ်ခင် အမြဲ Mutex ကို Read လုပ်ရမည် ဖြစ်သည်။



# Mutex in Windows

- ကျွန်တော်တို့ Mutex ကိုသုံးပြီးတော့ Process Instance တစ်ခုကိုပဲ Run လို့ရအောင် လုပ်ပါမည်။ ဒုတိယ Process Instance နောက်တစ်ခုကို ထပ် Run ပါက Process ကို Terminate လုပ်ပါမည်။

```
static void Main(string[] args)
{
    bool firstInstance;

    using (Mutex mutex = new Mutex(true, @"Global\Mutex", out firstInstance))
    {
        if(!firstInstance)
        {
            Console.WriteLine("Other Instances Detected. Aborting...");
            Console.ReadKey();
            return;
        }

        Console.WriteLine("First Instance.");

        for(int i=10; i>0;i--)
        {
            Console.WriteLine(i);
            Thread.Sleep(1000);
        }

        Console.ReadKey();
    }
}
```

# Mutex in Windows

- ပထမ Instance တစ်ခုကိုသာ Run လို့ရပါမည်။ Mutex က Block လုပ်ထားလို့ ဖြစ်သည်။

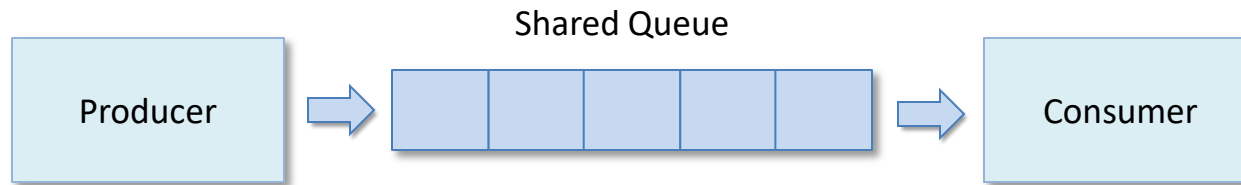


```
D:\Projects\VC#\MultiThreading\Mutex\Mutex\bin\Debug\Mutex.exe
First instance.
10
9
8
7
6
5
4
3
2
1

D:\Projects\VC#\MultiThreading\Mutex\Mutex\bin\Debug\Mutex.exe
Other Instances Detected. Aborting...
```



# Producer Consumer



- Producer Consumer Model (Shared Queue) ကိုအသုံးပြုပြီးတော့လည်း Multiple Process များက ဝိုင်း “Write” လုပ်သည့် ကိစ္စကို ရှင်းကြပါသည်။
- Producer က “Write Only” လုပ်ပြီး Consumer က “Read Only” လုပ်ပါသည်။ သို့သော် Production Rate နှင့် Consumption Rate တူဖို့ လိုပါသည်။ မတူပါက Overflow/Underflow ပြဿနာများ ကြုံရနိုင်ပါသည်။
- ထို့အပြင် Consumer က Queue (0) ကို Read လုပ်နေလျှင် Producer က Queue (0) ကို Write လုပ်လို့မရပါ။ ထို့ကြောင့် Producer မှာ “Write Lock” ရှိပြီး Consumer မှာ “Read Lock” ရှိပါသည်။
- ဒီဟာ (Double Lock)ကို ကောင်းကောင်း Manage မလုပ်ပါက “Deadlock” ဖြစ်စေပါသည်။ Producer ရော၊ Consumer ရော Read လည်း လုပ်မရ Write လည်း လုပ်မရ ဖြစ်သွားပါသည်။

# Deadlock and Starvation

- Deadlock ဟူသည် ရှေ့တိုးမရ၊ နောက်ဆုတ်မရ အခြေအနေမျိုးကို ဆိုလိုတာ ဖြစ်ပြီး အများအားဖြင့် Mutual Lock နှင့် Double Lock များကြောင့် ဖြစ်လေ့ရှိပါသည်။ ထို့ကြောင့် Lock 2 ထပ် ဖြစ်သောအခါ အင်မတန် သတိထားရပါသည်။ မလိုရင် Mutual Lock နှင့် Double Lock များကို မသုံးသင့်ပါ။
- Starvation ဆိုသည်က Priority Queuing သို့မဟုတ် Locking ကြောင့် ဖြစ်လေ့ရှိပါသည်။ Process တစ်ခုက Low Priority ဖြစ်နေပါက ထို Process သည် အများအားဖြင့် Lock အလုပ်ခံထား (ချောင်ထိုးခံထား) ရလေ့ရှိပြီး အခြား Process များကိုသာ ဦးစားပေးလုပ်ဆောင်နေသည်အခါ Starvation ဖြစ်လာပါသည်။
- တခါတရံ Deadlock ကြောင့်လည်း Starvation ဖြစ်ပါသည်။
- ကျွန်တော်တို့ Double Lock ကြောင့်ဖြစ်သော Dead Lock ကို ကြည့်ရအောင်ပါ။

# Deadlock in Windows

- Double Lock ကြောင့်ဖြစ်တဲ့ Deadlock တစ်ခုကို ဖန်တီးပါမည်။ ဖြစ်နိုင်လျှင် Double Lock များကို Avoid လုပ်ပါ။

```
static readonly object Lock1 = new object();
static readonly object Lock2 = new object();

public static void First()
{
    Console.WriteLine("Locking Second");

    lock (Lock2)
    {
        Console.WriteLine("Locked Second");
        Console.WriteLine("Locking First");

        Thread.Sleep(500);

        lock (Lock1)
        {
            Console.WriteLine("Locked First");
        }

        Console.WriteLine("Released First");
    }

    Console.WriteLine("Released Second");
}
```

# Deadlock in Windows

- Lock တစ်ခုကို အပြင်က Lock တစ်ခုထပ်ချခြင်းသည် Deadlock များကို ဖြစ်စေလေ့ရှိပါသည်။

```
public static void Second()
{
    Console.WriteLine("Locking First");
    lock (Lock1)
    {
        Console.WriteLine("Locked First");
        Console.WriteLine("Locking Second");
        lock (Lock2)
        {
            Console.WriteLine("Locked Second");
        }
        Console.WriteLine("Released Second");
    }
    Console.WriteLine("Released First");
}

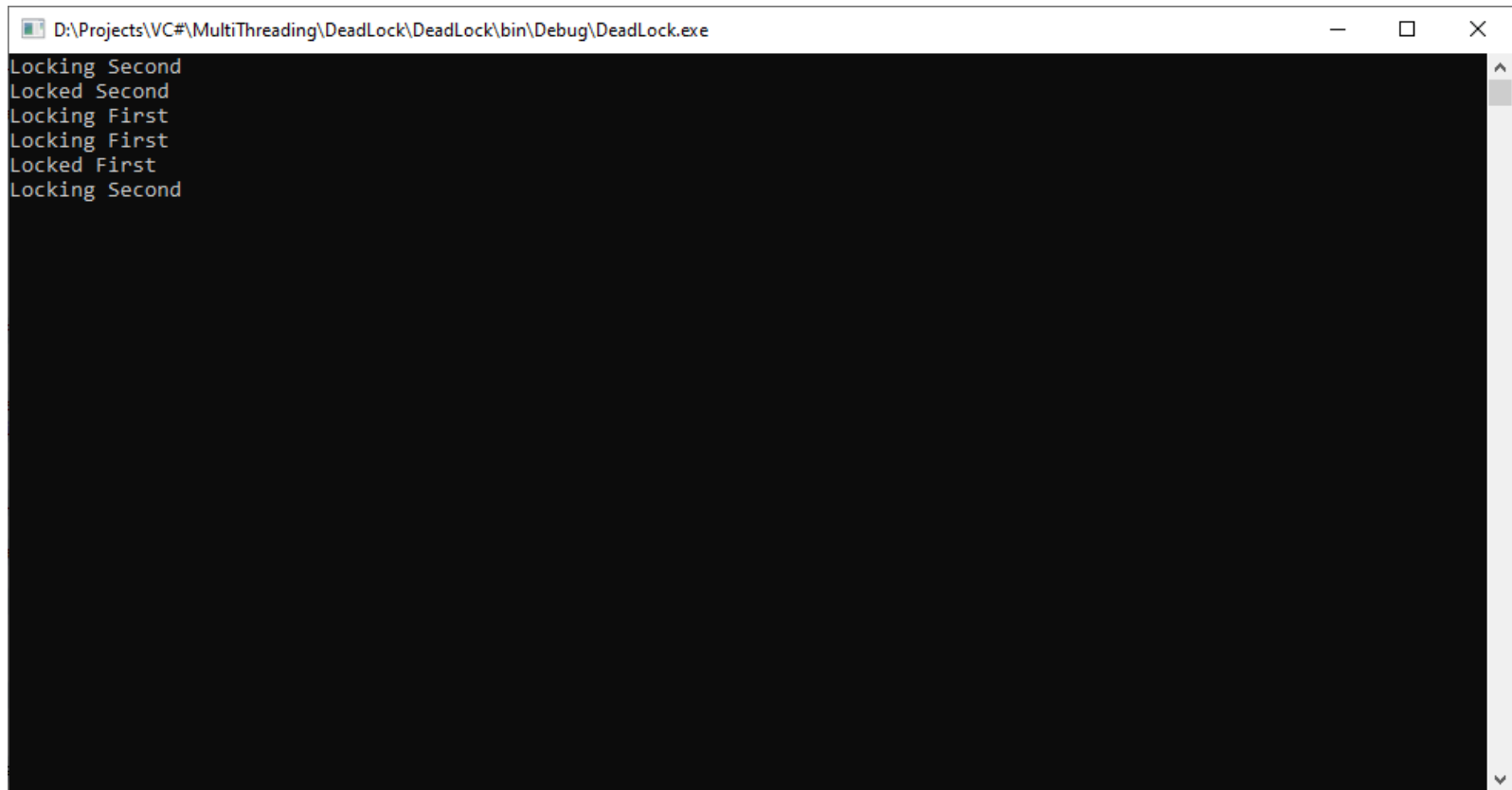
static void Main(string[] args)
{
    Thread th1 = new Thread(First);
    th1.Start();

    Thread th2 = new Thread(Second);
    th2.Start();

    Console.ReadKey();
}
```

# Deadlock in Windows

- Double Lock ကြောင့် First Thread ရော၊ Second Thread ပါ Deadlock ဖြစ်ပြီး Mutual Starvation ဖြစ်သွားပါသည်။ ဒီပြဿနာကို ဘယ်လိုရှင်းမလဲ။

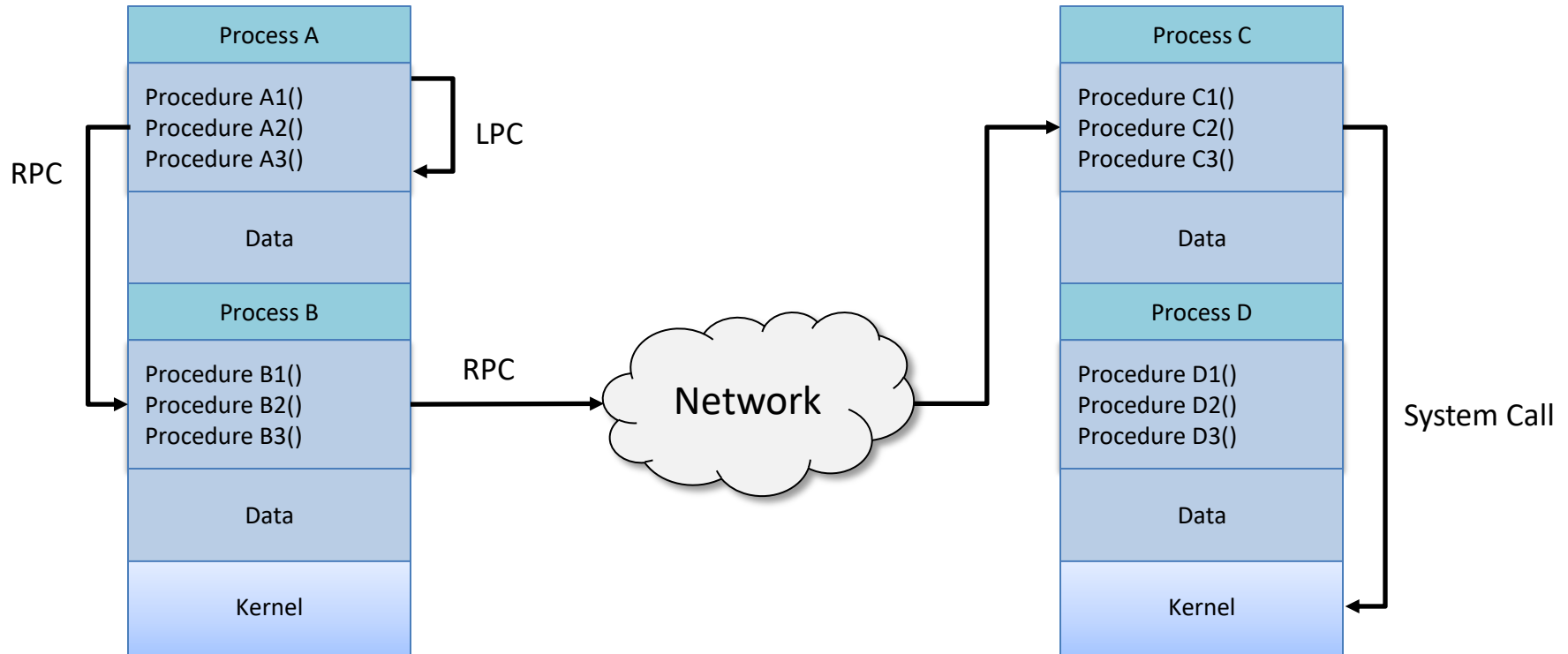


```
D:\Projects\VC#\MultiThreading\DeadLock\DeadLock\bin\Debug\DeadLock.exe
Locking Second
Locked Second
Locking First
Locking First
Locked First
Locking Second
```

# IPC (Interprocess Communication)

- IPC (Interprocess Communication) သည် Critical Section (Shared Memory) အပြင် Process များအချင်းချင်း Cooperate လုပ်ဖို့ ဖြစ်ပါသည်။
- Critical Section ကို အမှန်တော့ Shared Memory သို့မဟုတ် Distributed Memory များပေါ်မှသာ ပြုလုပ်နိုင်ပြီး ပြောခဲ့သည့်အတိုင်း Deadlock နှင့် အခြားပြဿနာများ ရှိပါသည်။
- ထို့ကြောင့် IPC ကို Messaging များ ဖြင့် Implement လုပ်ကြပါသည်။ Messaging ဟူသည် အမှန်တော့ Procedure Call (Method Invocation) များသာ ဖြစ်ပါသည်။
- Kernel Code (Process) များ၏ Procedure များကို Call လုပ်လျှင် System Call ဟုခေါ်ပြီး Application Code (Process) များ၏ Procedure များကို Call လုပ်လျှင် Procedure Call ဟုခေါ်ကြပါသည်။
- Application Process တစ်ခုသည် ကိုယ့် Procedure များကိုသာ Call လုပ်သည်ဆိုလျှင် LPC (Local Procedure Call) ဟုခေါ်ပြီး အခြား Application Process တစ်ခု၏ Procedure များကို Call လုပ်လျှင် RPC (Remote Procedure Call) ဟုခေါ်သည်။

# Local and Remote Procedure Call



- RPC (Remote Procedure Call) များသည် Same Machine (Computer) ပေါ်မှာ ရှိစရာမလိုပါ။ RPC များကို Same Memory (Same Computer) ပေါ်ရှိ အခြား Process များ၏ Procedure များကို Call လုပ်နိုင်သလို Network မှတစ်ဆင့် အခြား Computer ပေါ်ရှိ အခြား Process များ၏ Procedure များကို Call လုပ်နိုင်လာပါသည်။
- ထိုအချက်သည် Client Server Computing ကို ပေါ်ပေါက်လာစေပြီး နောက်ဆုံး Internet ကြီးကို ဖြစ်ပေါ်လာစေပါသည်။

# Client Server Model

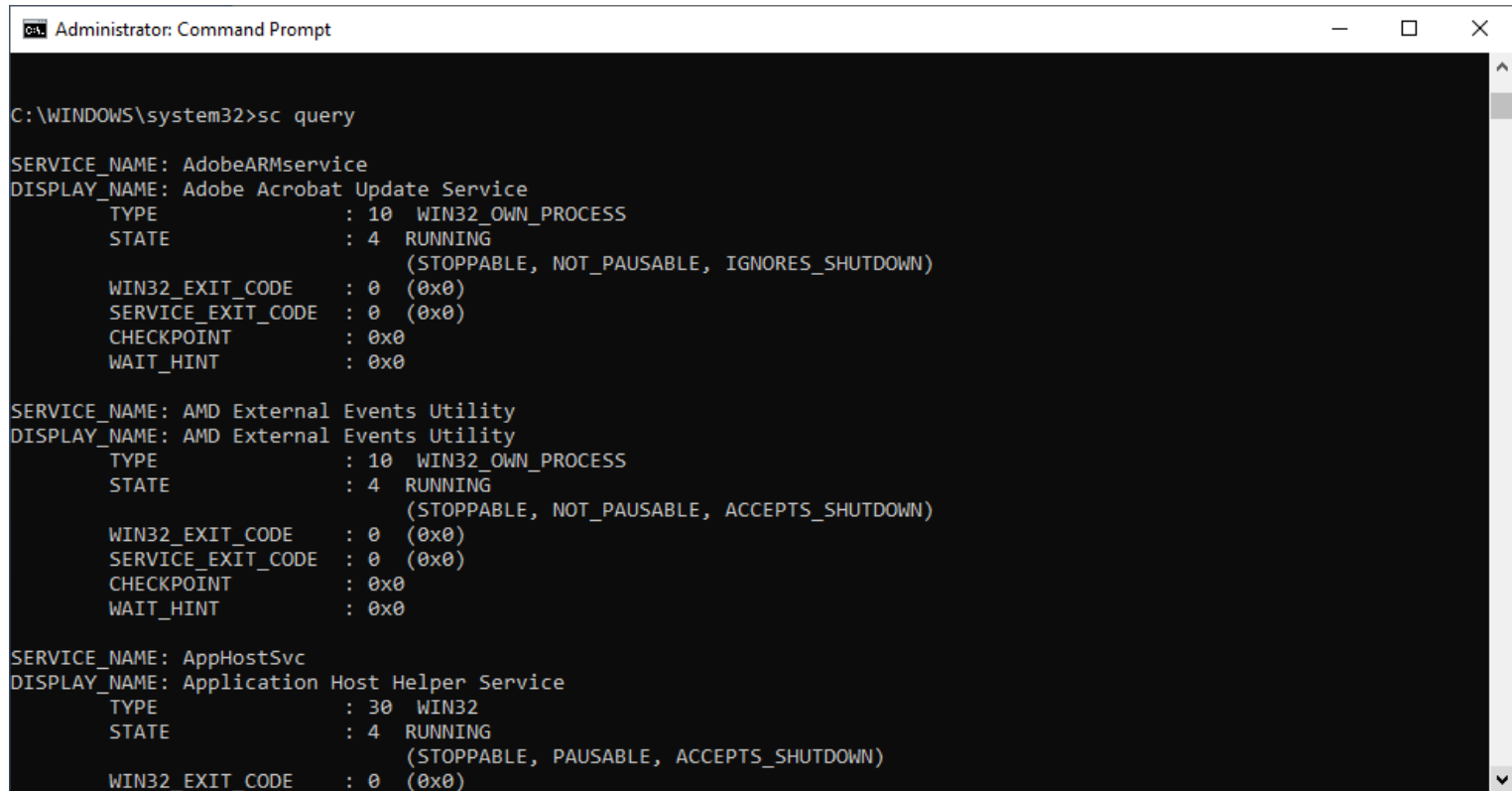
- Process တစ်ခုနှင့် တစ်ခု အကြား Remote Procedure Call လုပ်ပါက အများအားဖြင့် Request/Response ပုံစံဖြင့် လုပ်လေ့ရှိပါသည်။
- Request လုပ်သော Process ကို Client ဟုခေါ်ပြီး Response လုပ်သော Process ကို Server ဟုခေါ်ပြီး Service Process က လုပ်ဆောင်သော Function များကို Service ဟုခေါ်ပါသည်။
- Client Server Model ပေါ်မူတည်ပြီး ကျွန်တော်တို့ Web Server များ၊ Mail Server များ၊ File Server များ တည်ဆောက်ကြခြင်း ဖြစ်သည်။
- Client Server ကို ကျွန်တော်တို့ Signal and Data Communication မှာ ပိုပြီး အသေးစိတ် ဆွေးနွေးပါမည်။ အခုတော့ Operating Systems များ အကြောင်းကိုသာ အဓိက ထားပါမည်။



# Services (Local and Remote)

- Kernel Services များ အပါအဝင် Operating System Services များနှင့် Application Services များသည် Client Server Computing လုပ်ဆောင်ပါသည်။
- အချို့ Services များကို Locally လုပ်ဆောင်နိုင်ပြီး Local Services များဟု ခေါ်ပါသည်။ အချို့ Services များကို Remotely လုပ်ဆောင်နိုင်ပြီး Remote Services (Network Services) များဟု ခေါ်ပါသည်။
- ထိုသို့ Operating Systems များနှင့် Application များကို Services အနေဖြင့် Implement လုပ်ခြင်းကို Service Oriented Architecture (SOA) ဟုခေါ်ပါသည်။
- Distributed Computing နှင့် Cloud Computing တွင် SOA သည် အရေးကြီးသော အခန်းမှ ပါဝင်ပါသည်။ SOA ကြောင့် Web Services များ အပါအဝင် အခြား အရေးကြီးသော Services များကို Internet မှ လုပ်ဆောင်လာနိုင်ခြင်း ဖြစ်ပါသည်။
- ကျွန်တော်တို့ Local Window Services နှင့် Local Linux Services များကို လေ့လာကြည့်ပါမည်။

# Windows Services



```
Administrator: Command Prompt
C:\WINDOWS\system32>sc query

SERVICE_NAME: AdobeARMservice
DISPLAY_NAME: Adobe Acrobat Update Service
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE                : 4   RUNNING
                        (STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE       : 0   (0x0)
        SERVICE_EXIT_CODE   : 0   (0x0)
        CHECKPOINT           : 0x0
        WAIT_HINT            : 0x0


SERVICE_NAME: AMD External Events Utility
DISPLAY_NAME: AMD External Events Utility
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE                : 4   RUNNING
                        (STOPPABLE, NOT_PAUSABLE, ACCEPTS_SHUTDOWN)
        WIN32_EXIT_CODE       : 0   (0x0)
        SERVICE_EXIT_CODE   : 0   (0x0)
        CHECKPOINT           : 0x0
        WAIT_HINT            : 0x0

SERVICE_NAME: AppHostSvc
DISPLAY_NAME: Application Host Helper Service
        TYPE               : 30  WIN32
        STATE                : 4   RUNNING
                        (STOPPABLE, PAUSABLE, ACCEPTS_SHUTDOWN)
        WIN32_EXIT_CODE       : 0   (0x0)
```

- Command Prompt (Run as Administrator) မှာ “sc query” ကို ရိုက်ပြီး Local Windows Services များကို ကြည့်နိုင်ပါသည်။

# Creating Services in Windows

- ကျွန်တော်တို့ .Net Framework မှာ Windows Service Project ရွေးပြီး Windows Service တစ်ခုကို စမ်းကြည့်ပါမည်။

 Windows Service (.NET Framework)  
A project for creating Windows Services

C# Windows Desktop Service

```
static void Main()
{
    ServiceBase[] ServicesToRun;

    ServicesToRun = new ServiceBase[] { new Service1() };

    ServiceBase.Run(ServicesToRun);
}
```

- Command Prompt မှာ

```
sc create MyService binPath="C:\Test\MyService.exe"

sc query MyService
sc start MyService
sc stop MyService
sc delete MyService
```

# Creating Services in Windows

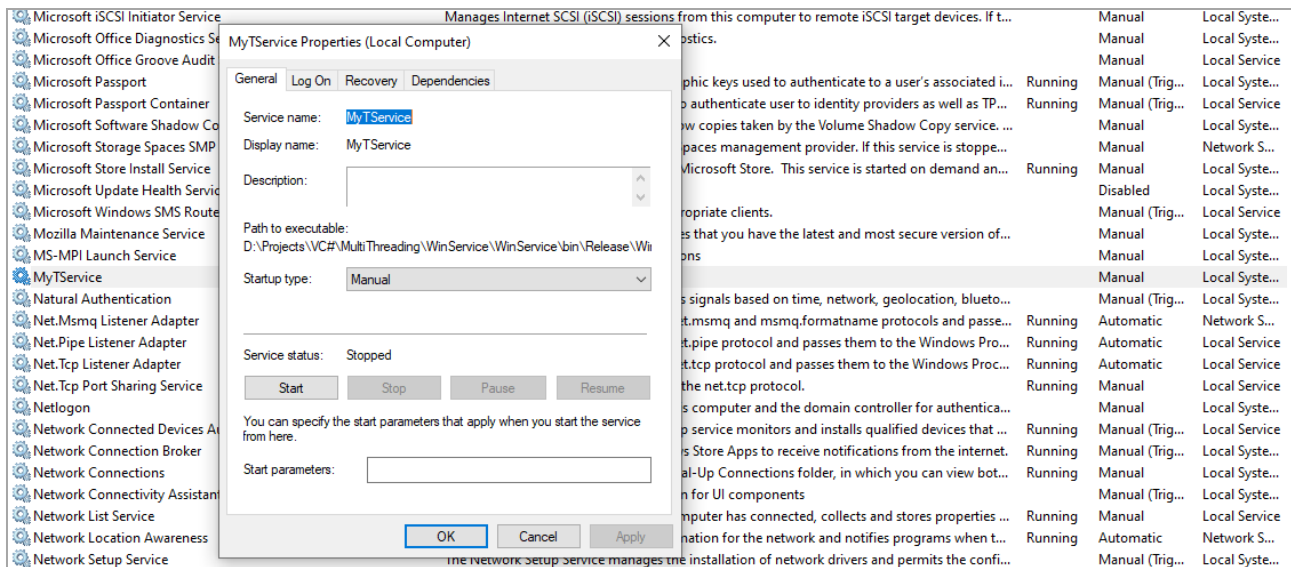
- ကျွန်တော်တို့ ဒါဆိုရင် Windows Service အနေဖြင့် တွေ့ရမည် ဖြစ်သည်။ မလိုအပ်ပါက Service ကို Delete ပြန်လုပ်နိုင်ပါသည်။

```
Administrator: Command Prompt
[SC] CreateService SUCCESS

C:\WINDOWS\system32>sc query MyTService

SERVICE_NAME: MyTService
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE                : 1   STOPPED
        WIN32_EXIT_CODE       : 1077  (0x435)
        SERVICE_EXIT_CODE    : 0     (0x0)
        CHECKPOINT           : 0x0
        WAIT_HINT             : 0x0

C:\WINDOWS\system32>
```



# Linux Services

- ကျွန်တော်တို့ Linux ရဲ့ services တွေနဲ့ သူ့ရဲ့ manager systemd ကိုတခြား lab တွေမှာထိတွေ့ဖူးပါတယ်။
- systemd က OS ရဲ့ serviceတစ်ခု အနေနဲ့ service daemon တွေကို manage လုပ်ပေးပါတယ်။  
systemctl command နဲ့ running state services တွေကိုလုပ်ရပါလိမ့်။

```
root@sysadmin-Virtual-Machine:/home/sysadmin# systemctl list-units --type=service --state=running
```

UNIT	LOAD	ACTIVE	SUB	DESCRIPTION
accounts-daemon.service	loaded	active	running	Accounts Service
acpid.service	loaded	active	running	ACPI event daemon
apache2.service	loaded	active	running	The Apache HTTP Server
avahi-daemon.service	loaded	active	running	Avahi mDNS/DNS-SD Stack
colord.service	loaded	active	running	Manage, Install and Generate Color Profiles
cron.service	loaded	active	running	Regular background program processing daemon
cups-browsed.service	loaded	active	running	Make remote CUPS printers available locally
cups.service	loaded	active	running	CUPS Scheduler
dbus.service	loaded	active	running	D-Bus System Message Bus
gdm.service	loaded	active	running	GNOME Display Manager
irqbalance.service	loaded	active	running	irqbalance daemon
kerneloops.service	loaded	active	running	Tool to automatically collect and submit kernel crash signatures
ModemManager.service	loaded	active	running	Modem Manager
networkd-dispatcher.service	loaded	active	running	Dispatcher daemon for systemd-networkd
NetworkManager.service	loaded	active	running	Network Manager
plymouth-start.service	loaded	active	running	Show Plymouth Boot Screen
polkit.service	loaded	active	running	Authorization Manager
rsyslog.service	loaded	active	running	System Logging Service
rtkit-daemon.service	loaded	active	running	RealtimeKit Scheduling Policy Service
snapped.service	loaded	active	running	Snap Daemon
ssh.service	loaded	active	running	OpenBSD Secure Shell server
switcheroo-control.service	loaded	active	running	Switcheroo Control Proxy service
systemd-journald.service	loaded	active	running	Journal Service
systemd-logind.service	loaded	active	running	Login Service
systemd-resolved.service	loaded	active	running	Network Name Resolution
systemd-timesyncd.service	loaded	active	running	Network Time Synchronization
systemd-udev.service	loaded	active	running	udev Kernel Device Manager
udisks2.service	loaded	active	running	Disk Manager
unattended-upgrades.service	loaded	active	running	Unattended Upgrades Shutdown
upower.service	loaded	active	running	Daemon for power management
user@1000.service	loaded	active	running	User Manager for UID 1000
user@125.service	loaded	active	running	User Manager for UID 125
whoopsie.service	loaded	active	running	crash report submission daemon
wpa_supplicant.service	loaded	active	running	WPA supplicant

# Linux Services

- Apache web serverကို stop, start, restart, reload စသဖြင့် manage လုပ်လို့ရပါတယ်။

```
root@sysadmin-Virtual-Machine:/home/sysadmin# systemctl stop apache2
root@sysadmin-Virtual-Machine:/home/sysadmin# systemctl start apache2
root@sysadmin-Virtual-Machine:/home/sysadmin# systemctl restart apache2
root@sysadmin-Virtual-Machine:/home/sysadmin# systemctl reload apache2
root@sysadmin-Virtual-Machine:/home/sysadmin# systemctl status apache2
● apache2.service - The Apache HTTP Server
   Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2022-01-04 20:15:40 AWST; 11s ago
     Docs: https://httpd.apache.org/docs/2.4/
   Process: 6086 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUCCESS)
   Process: 6149 ExecReload=/usr/sbin/apachectl graceful (code=exited, status=0/SUCCESS)
  Main PID: 6090 (apache2)
    Tasks: 55 (limit: 2221)
   Memory: 4.9M
   CGroup: /system.slice/apache2.service
           └─6090 /usr/sbin/apache2 -k start
             └─6153 /usr/sbin/apache2 -k start
               └─6154 /usr/sbin/apache2 -k start

Jan 04 20:15:40 sysadmin-Virtual-Machine systemd[1]: Starting The Apache HTTP Server...
Jan 04 20:15:40 sysadmin-Virtual-Machine apachectl[6089]: AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 127.0.1.1. Set the 'ServerName' directive globally to suppress this message
Jan 04 20:15:40 sysadmin-Virtual-Machine systemd[1]: Started The Apache HTTP Server.
Jan 04 20:15:45 sysadmin-Virtual-Machine systemd[1]: Reloading The Apache HTTP Server.
Jan 04 20:15:46 sysadmin-Virtual-Machine apachectl[6152]: AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 127.0.1.1. Set the 'ServerName' directive globally to suppress this message
Jan 04 20:15:46 sysadmin-Virtual-Machine systemd[1]: Reloaded The Apache HTTP Server.
```

# Linux Services

- မိမိရဲ့ custom program ကို systemd service အဖြစ် register လုပ်မယ်ဆိုရင်တော့ `/etc/systemd/system/myprogram.service` အဖြစ် file create လုပ်ရပါမယ်။

```
[Unit]
Description=MyProgram service
After=network.target

[Service]
ExecStart=/usr/bin/nc -l 8080
WorkingDirectory=/home/sysadmin
StandardOutput=inherit
StandardError=inherit
Restart=always
User=root

[Install]
WantedBy=multi-user.target
```

- ဒီ service ကို start လုပ်မယ်ဆိုရင် netcat နဲ့ port 8080 မှာ listen လုပ်မှာပါ။
- `systemctl` နဲ့ `daemon-reload` လုပ်ပြီး start ကြည့်မယ်။

```
root@sysadmin-Virtual-Machine:/home/sysadmin# systemctl daemon-reload
root@sysadmin-Virtual-Machine:/home/sysadmin# systemctl start myprogram
```

# Linux Services

- `service status` ကြည့်မယ်။ `telnet` နဲ့ `connect` လုပ်ကြည့်မယ်။

```
root@sysadmin-Virtual-Machine:/home/sysadmin# systemctl status myprogram
● myprogram.service - MyProgram service
   Loaded: loaded (/etc/systemd/system/myprogram.service; disabled; vendor preset: enabled)
   Active: active (running) since Tue 2022-01-04 20:30:43 AWST; 3s ago
     Main PID: 6465 (nc)
       Tasks: 1 (limit: 2221)
      Memory: 196.0K
        CGroup: /system.slice/myprogram.service
                └─6465 /usr/bin/nc -l 8080

Jan 04 20:30:43 sysadmin-Virtual-Machine systemd[1]: Started MyProgram service.
```

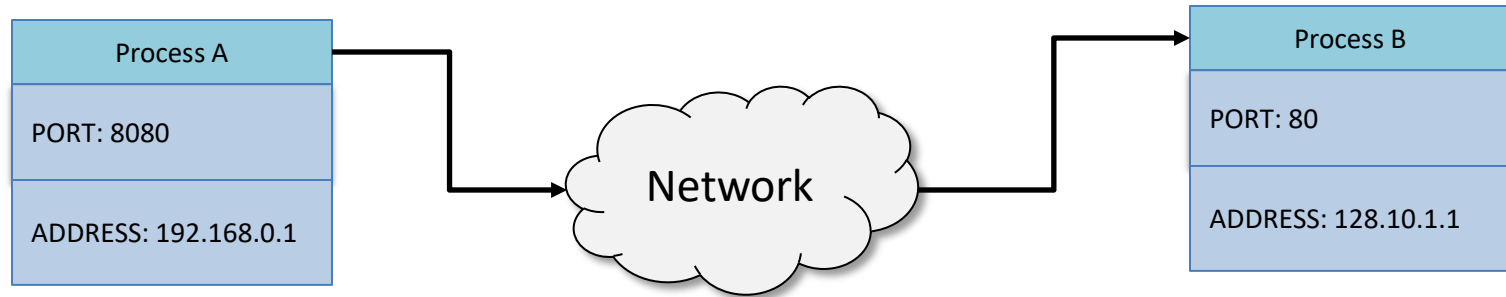
```
root@sysadmin-Virtual-Machine:/home/sysadmin# telnet 127.0.0.1 8080
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
```

- Custom serviceကို OS boot up ဖြစ်တာနဲ့ start စေချင်ရင်တော့ `enable` option နဲ့ `activate` လုပ်ပေးထားရပါမယ်။

```
root@sysadmin-Virtual-Machine:/home/sysadmin# systemctl enable myprogram
Created symlink /etc/systemd/system/multi-user.target.wants/myprogram.service → /etc/systemd/system/myprogram.service.
root@sysadmin-Virtual-Machine:/home/sysadmin#
```



# Socket



Local Socket = 127.0.0.1:8080

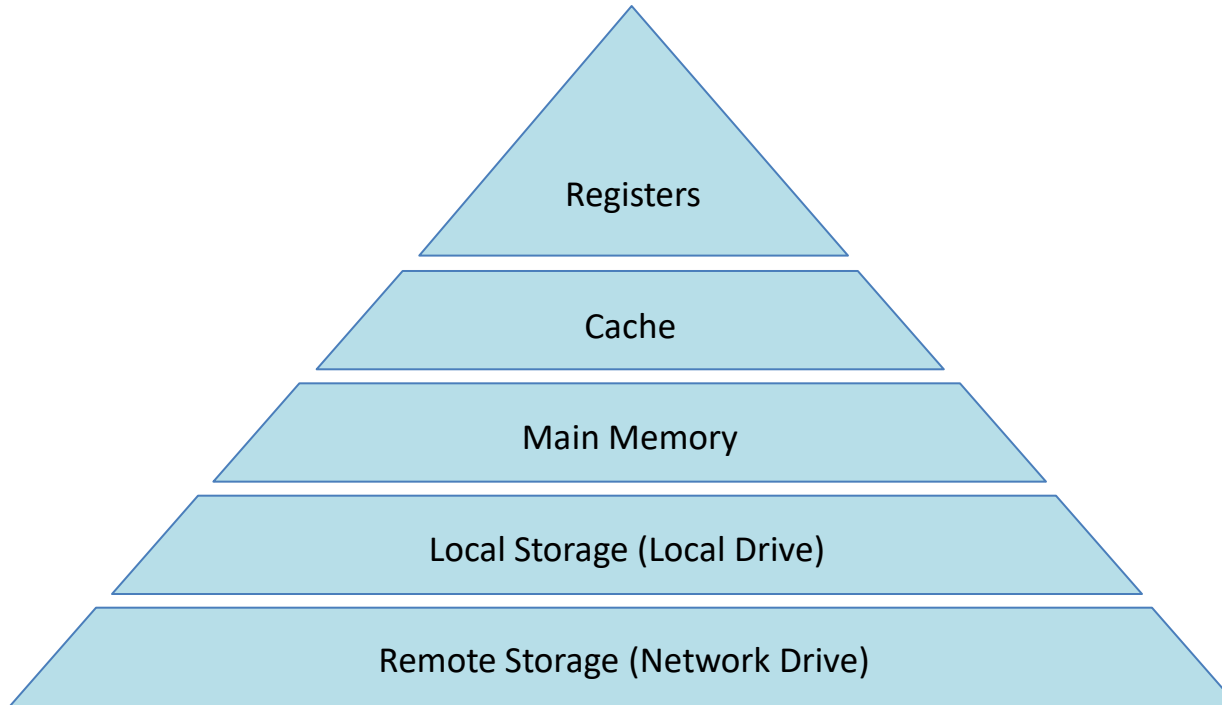
Remote Socket = 128.10.1.1:80

- Operating Systems ၏ Services များကို အများအားဖြင့် Socket များဖြင့် Identify လုပ်လေ့ရှိပါသည်။
- Socket ဆိုသည်မှာ Address (Network Address) နှင့် Port တို့ပါဝင်သော Identification တစ်ခု ဖြစ်ပါသည်။
- Local Service များကို Local Address or Loopback Address (127.0.0.1) ဖြင့် ဖော်ပြနိုင်ပြီး Remote Services များကိုတော့ သက်ဆိုင်ရာ Public Remote Address (128.10.1.1) ဖြင့် ဖော်ပြနိုင်ပါသည်။
- Local Socket နှင့် Remote Socket ကို Connect (Communicate) လုပ်ခြင်းဖြင့် Remote Procedure Call များကို Network ပေါ်မှာ လုပ်ဆောင်ပါသည်။
- ပုံမှန်အားဖြင့် Local Sockets များကို Firewall ဖြင့် ပိတ်ထားလေ့ရှိပါသည်။ ဒီအကြောင်းကိုတော့ Users and Security ရောက်မှ အသေးစိတ် ထပ်ပြီးဆွေးနွေးပါမည်။

# Memory Management

- ကျွန်တော်တို့ Process Management များအကြောင်းကိုတော့ ထိုက်သင့်သလောက် ဆွေးနွေးပြီးသွားပါပြီ။
- အမှန်တော့ Operating Systems တစ်ခုအနေဖြင့် Process Management ကို သူ့ချည်းသက်သက် လုပ်ဆောင်လို့မရပါ။ အခြား Memory Management, File Management များကိုပါ Process Management နှင့် တွဲဖက်လုပ်ဆောင် ရပါသည်။
- Memory Management ၏ အဓိက လုပ်ဆောင်မှုသည် Process, Thread နှင့် Task များကို Memory ပေါ် Allocation လုပ်ပေးခြင်းနှင့် လိုအပ်ပါက Memory နှင့် Storage Device (File) အကြား Swap လုပ်ခြင်း ဖြစ်ပါသည်။
- ထို့ကြောင့် အခု ကျွန်တော်တို့ Memory Management ကို ဆက်ပြီး ဆွေးနွေးပါမည်။

# Memory Organization



- Memory Organization တစ်ခုကို ဒီလို Hierarchical Organization ဖြင့် ဖော်ပြနိုင်ပါသည်။
- Process များ၏ Code နှင့် Data များသည် ဒီ Memory Organization အတွင်းတွင် ရှိနေမည် ဖြစ်သည်။ တစ်နည်းအားဖြင့် Memory Organization သည်ပင် Computer တစ်ခု၏ States များ ဖြစ်သည်။

# Physical and Logical Address

- ကျွန်တော်တို့ Program များ ရေးသည့်အခါ Variable များကို ဒီလို Declare လုပ်ပါသည်။ ဒီ Variable များသည် Memory ရှိ Data Segment ၏ Address များကို ကိုယ်စားပြုမှန်းတော့ သိပါသည်။

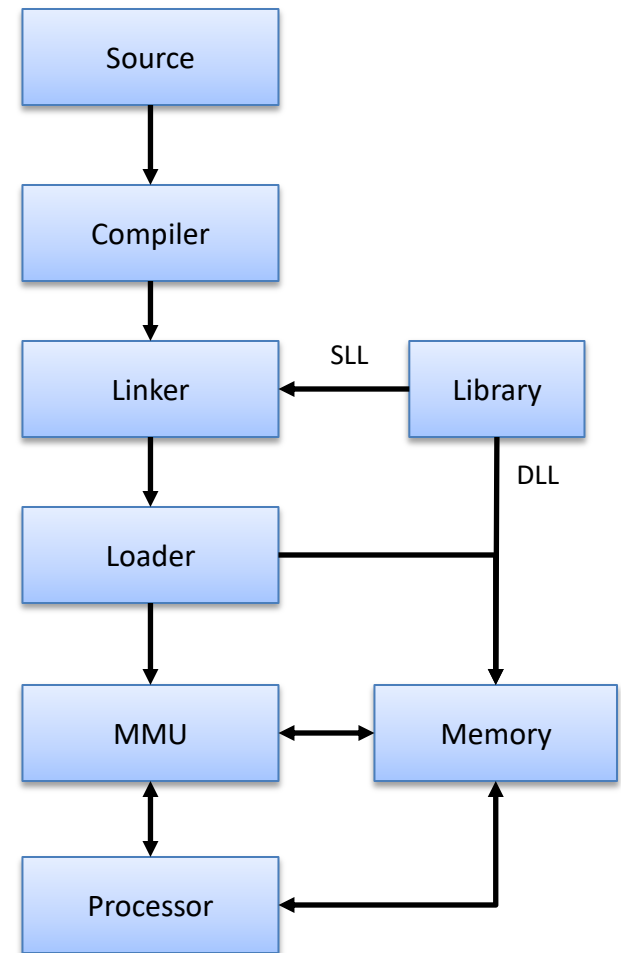
```
int x = 4;  
double y = 0.34;
```

- သို့သော် Process များသည် တချိန်လုံးနီးပါး Context Switching (Relocation) လုပ်ခြင်းခံရမည် ဖြစ်ပြီး Memory Address များအမြဲပြောင်းနေမည် ဖြစ်သည်။
- ထို့ကြောင့် Variable များသည် Memory နှင့် File Storage များ၏ Physical Address တစ်နေရာတည်းကို အမြဲဖော်ပြဖို့ မဖြစ်နိုင်ပါ။
- သို့ဖြစ်၍ Physical Address များအစား Memory Organization ကို ဖော်ပြနိုင်သော Logical Address များလိုအပ်လာပါသည်။
- အများအားဖြင့် Logical Address များသည် Relative Address များဖြစ်ပါသည်။ တစ်နည်းအားဖြင့်

$$\text{Logical Address} = \text{Base Address (Physical Address)} + \text{Offset}$$

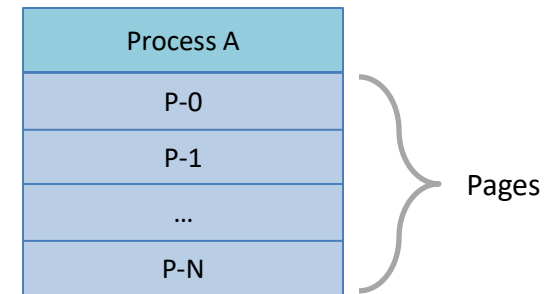
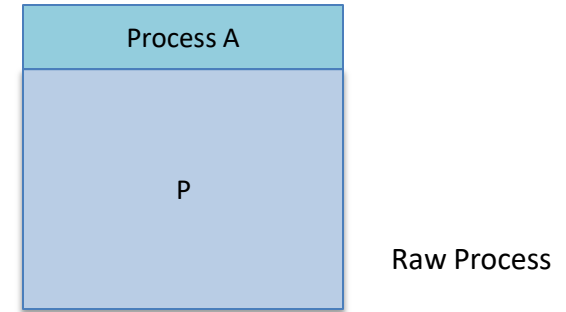
# Memory Management Unit

- Memory Management Unit သည် အများအားဖြင့် Physical Address များကို Logical Address သို့ပြောင်းပေးပါသည်။
- Source Program File မှ Procedure Code များကို Storage Device များမှ Memory ပေါ်သို့တင်ပေးသော Unit များကို Loader (Program Loader) ဟုခေါ်ပါသည်။ Boot Loader သည် Kernel ကို Load လုပ်ပေးသော Special Loader တစ်ခု ဖြစ်သည်။
- Source Program File မှ Procedure Code များအပြင် Library File များမှ Procedure Code များကိုပါ Memory ပေါ်သို့ ပေါင်းတင်ခြင်းကို Linking လုပ်သည်ဟုခေါ်ပါသည်။
- Source Program တစ်ခုလုံးကို Loader မှ Load မလုပ်ဘဲ လက်ရှိဘယ်အပိုင်းကို Execute လုပ်နေလဲမူတည်ပြီး လိုအပ်သော Procedure Code များကိုသာ Load လုပ်ခြင်းကို Dynamic Loading ဟုခေါ်ပါသည်။ Dynamic Linking သည်လည်း Library File များမှ လိုအပ်သော Procedure Code များကိုသာ Execution Time (Run Time) မှာ Link လုပ်ခြင်းဖြစ်သည်။
- Dynamic Linking ကို Support လုပ်သော Library များကို Dynamic Link Library (DLL) ဟုခေါ်သည်။ DLL များကို Execution Time (Run Time) မှာ Link လုပ်ပါသည်။
- ထိုသို့ Loading နှင့် Linking လုပ်သည်နှင့် Process ၏ လက်ရှိ Memory Address ပေါ်မူတည်ပြီး Memory Unit က Logical Address သို့ပြောင်းပေးပါသည်။



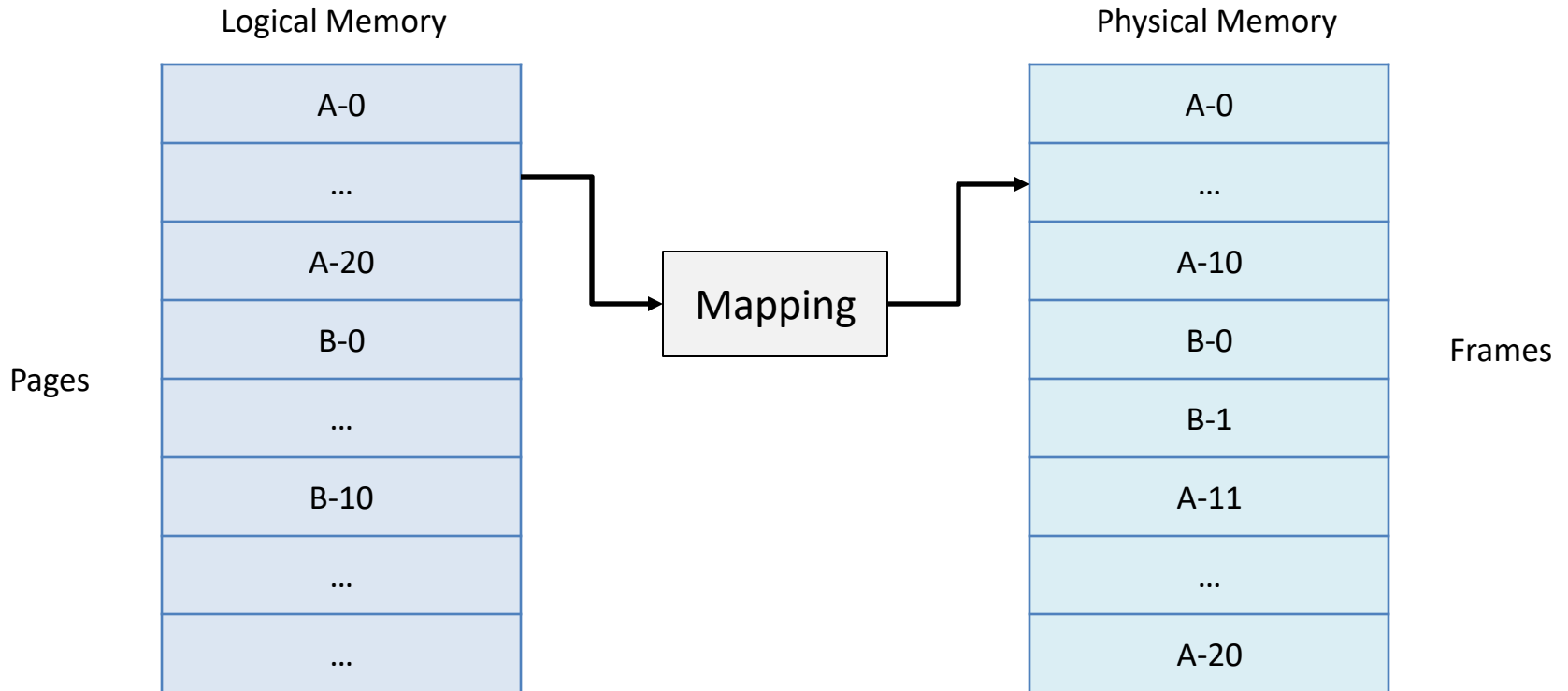
# Paging

- Process များသည် Code နှင့် Data ပေါ်မူတည်၍ Size များ ကွဲပြားမည် ဖြစ်သည်။ ထိုသို့ အရွယ်အစား အမျိုးမျိုးရှိသော Process များကို Memory မှာ Allocation လုပ်သောအခါ External Fragmentation (Process များ နေရာလွတ်များကျန်ခြင်း) ဖြစ်ပေါ်စေပါသည်။
- သို့ဖြစ်၍ Process များကို Fixed Sized Partitions များခွဲထုတ်ပါသည်။ ဒီလို Process Partitions များကို Page ဟုခေါ်ပါသည်။ Physical Memory ကိုလည်း Fixed Sized Partitions များခွဲထုတ်ပါသည်။ Memory Partitions များကိုတော့ Frame ဟုခေါ်ပါသည်။
- ထို့ကြောင့် Process တစ်ခုကို Page များအနေဖြင့် ခွဲထုတ်လိုက်ပြီး ထို Page များကို Page Table မှာ Frame များနှင့် Map လုပ်ထားပါမည်။
- ပြီးလျှင် Memory ပေါ်သို့ Process တစ်ခုလုံးမတင်တော့ဘဲ Page များကိုသာ တင်ပြီး Execute လုပ်မည် ဖြစ်သည်။ N Pages ရှိတော့ Process တစ်ခု အတွက် N Frames လိုမည် ဖြစ်ပြီး N Frames များသည် တဆက်တည်း (Continuous) ဖြစ်စရာမလိုတော့ပါ။
- ထို့ကြောင့် External Fragmentation (နေရာလွတ်များကျန်ခြင်း) ဖြစ်ခြင်းကို လျော့နည်းစေပါသည်။



Logical Address = Base Address (Physical Address) + [Page \* Page Size] + Offset

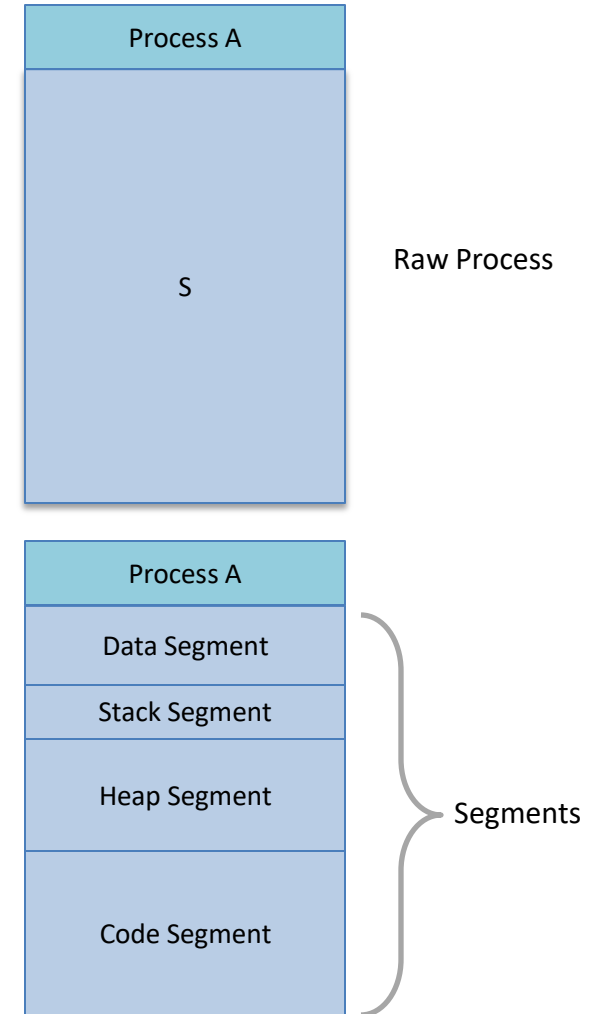
# Paging



- Logical Address (Page) များသည် တဆက်တည်း ဖြစ်နေမည် ဖြစ်သော်လည်း Physical Memory ပေါ်တွင် တဆက်တည်း ဖြစ်နေစေရမလိုတော့ပါ။
- ထို့ကြောင့် Physical Memory ပေါ်တွင် 1 Frame စာနေရာလွတ်သည်နှင့် 1 Page တစ်ခုကို Allocate လုပ်လိုရလာပါသည်။
- Free Frames, Allocated Frames, Frame Address, Page Address များကို Page Table မှာ Store လုပ်ထားပြီး Mapping ပြန်လုပ်မည် ဖြစ်သည်။

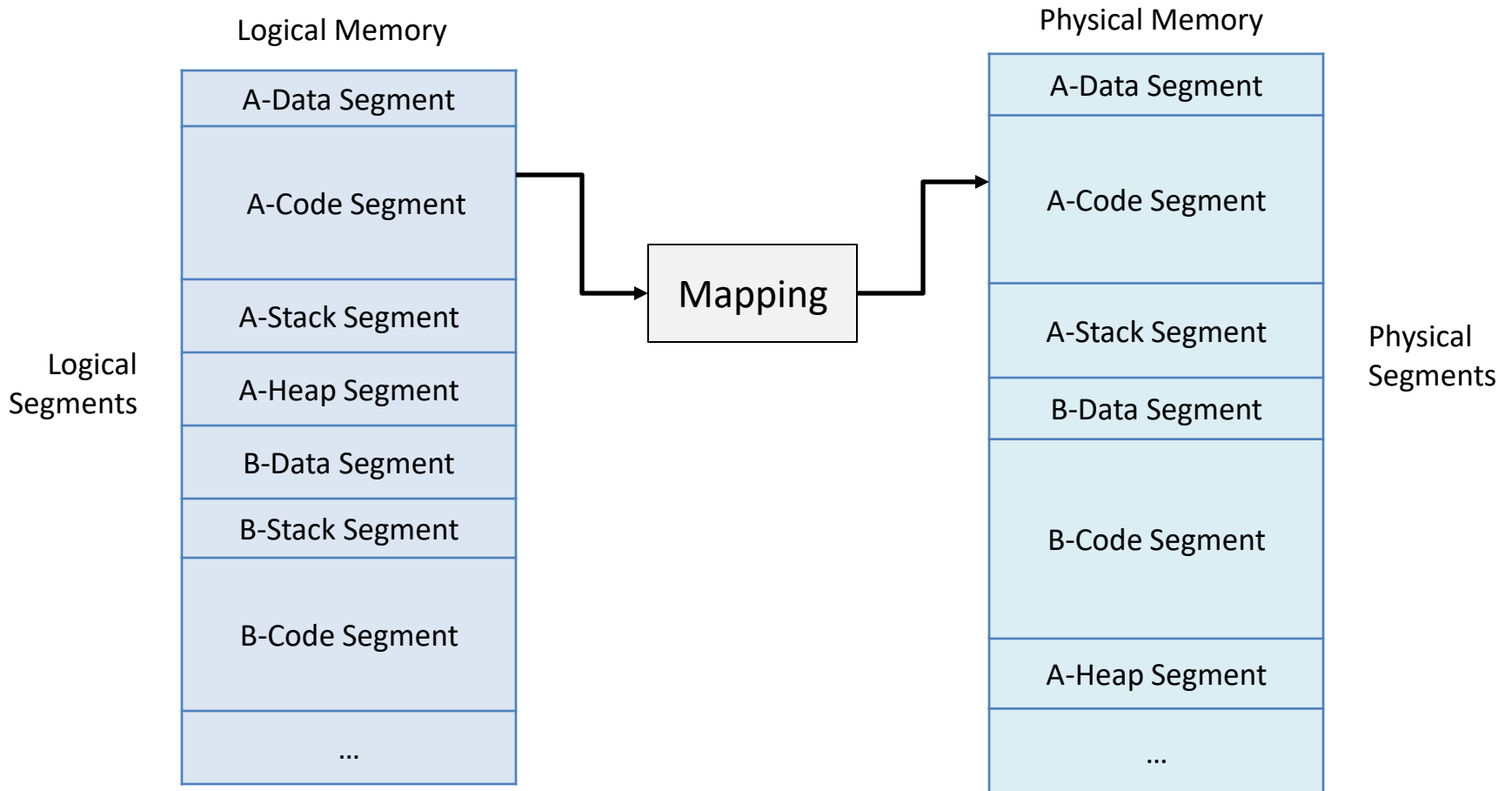
# Segmentation

- Paging နည်းတူ Segmentation သည်လည်း Process များကို Partition များခွဲထုတ်ပါသည်။
- သို့သော် Paging နှင့် မတူသည့် အချက်များ ရှိပါသည်။ ပထမ Segment များကို Code Segment, Data Segment, Stack Segment (Local Variables), Heap Segment (Global Variables and Pointers) စသည်ဖြင့် ခွဲထုတ်ပါသည်။ ဒုတိယ ထို Segment များသည် Variable Size ဖြစ်ပါသည်။
- Process တစ်ခုကို Segment များအနေဖြင့် ခွဲထုတ်လိုက်ပြီး ထို Segment များကို Segment Table မှာ Physical Address များနှင့် Map လုပ်ထားပါသည်။
- ပြီးလျှင် Memory ပေါ်သို့ Process တစ်ခုလုံးမတင်တော့ဘဲ Segment များကိုသာ တင်ပြီး Execute လုပ်မည် ဖြစ်သည်။ Segments များသည် တဆက်တည်း (Continuous) ဖြစ်စေရမလိုတော့ပါ။
- Variable Sized Segment များသည် External Fragmentation (Process များအကြား နေရာလွတ်များကျန်ခြင်း) ဖြစ်ခြင်းကို ဖြစ်ပေါ်စေနိုင်သော်လည်း Internal Fragmentation (Segment များအကြား နေရာလွတ်ကျန်ခြင်း) လျော့နည်းစေပါသည်။
- သို့သော် Segment များသည် Variable Size ဖြစ်သည့်အတွက် Mapping များက ပိုရှုပ်ထွေးပါသည်။





# Segmentation

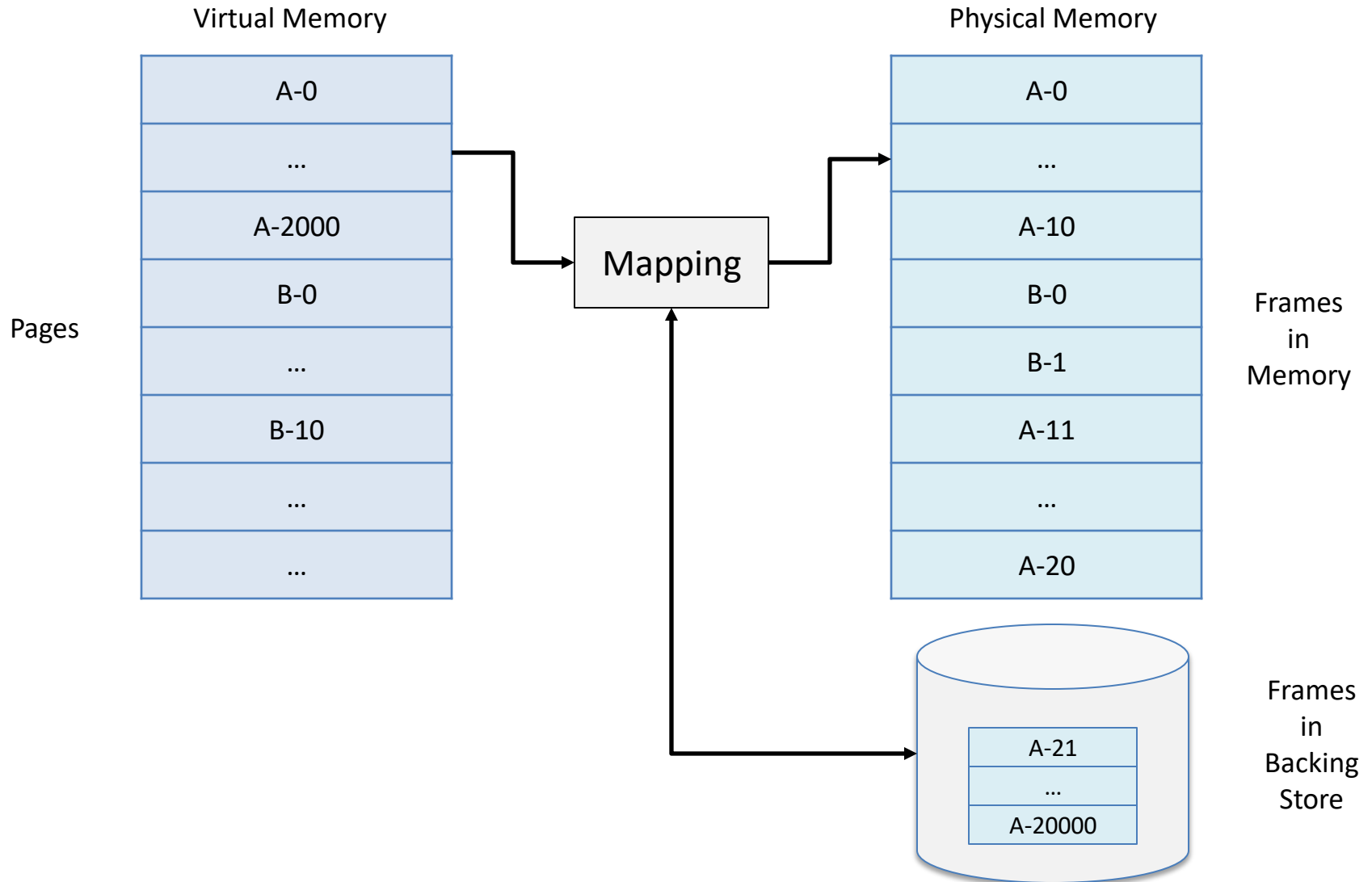


- Logical Address (Segment) များသည် တဆက်တည်း ဖြစ်နေမည် ဖြစ်သော်လည်း Physical Memory ပေါ်တွင် တဆက်တည်း ဖြစ်နေစေရမလိုတော့ပါ။

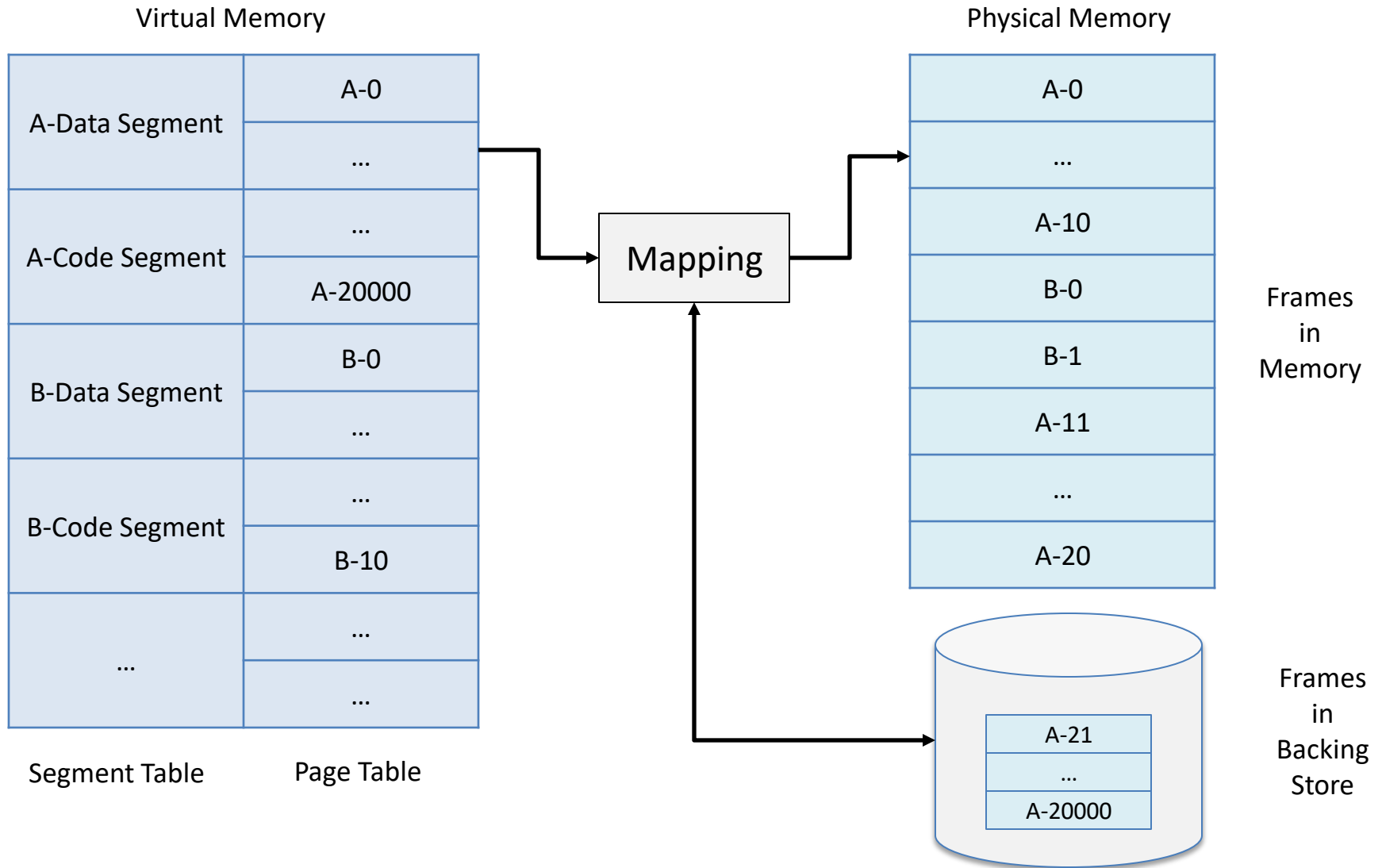
# Virtual Memory

- Logical Address, Physical Address, Page နှင့် Segmentation များသည် Memory Allocation ပြဿနာ နှင့် Swapping ပြဿနာအများစုကို ရှင်းနိုင်သော်လည်း အဓိက ပြဿနာ တစ်ခု ကျန်နေပါသေးသည်။
- ဒါကတော့ Process တစ်ခုက Available Physical Memory ထက်ကြီးနေပါက ထို Process ကို ဘယ်လို Execute လုပ်မလဲ ဖြစ်သည်။
- ကံကောင်းသည်က Process တစ်ခုကို Execute လုပ်နေလျှင် ထို Process ၏ Code များအားလုံးကို တချိန်တည်း တပြိုင်တည်း Execute မလုပ်နေပါ။ ဥပမာ၊ Line 100 ကို Execute လုပ်နေလျှင် Line 100000 ကို Execute လုပ်နေဦးမည် မဟုတ်ပါ။
- ဒီအချက်ပေါ်မူတည်ပြီး On Demand Paging နှင့် On Demand Segmented Paging (Segment များကို Page များပြန်ခွဲခြင်း) များကို Implement လုပ်ကြပါသည်။
- သို့နှင့် Logical Memory Space သည် Available Physical Memory Space ထက်ပိုပြီးလာပါသည်။ ဒါကို Virtual Memory ဟုခေါ်ပါသည်။

# On Demand Paging

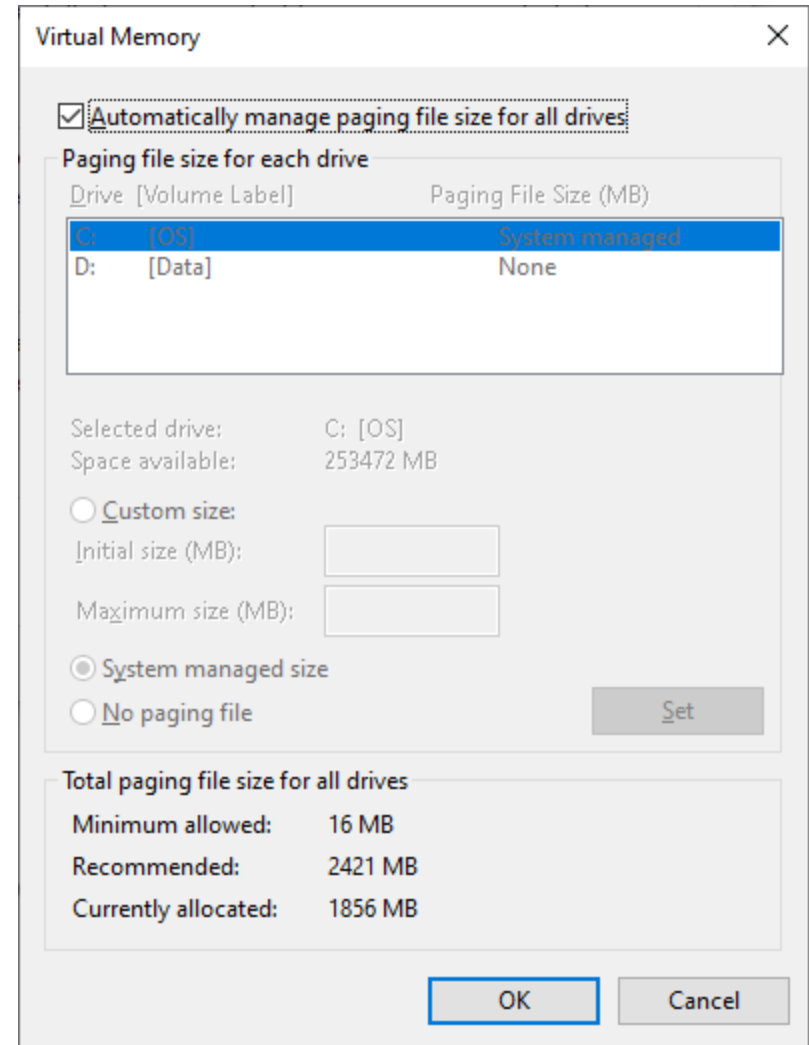


# On Demand Segmented Paging



# Virtual Memory in Windows

- ကျွန်တော်တို့ Windows ၏ Virtual Memory ကို Page Size ကို ပြင်ခြင်းဖြင့် Manage လုပ်လို့ ရပါသည်။
- Page Size နည်းလျှင် Memory Space ကိုပိုပြီး Optimized ဖြစ်စေသော်လည်း Performance ကို ပိုနွေးစေမည် ဖြစ်သည်။
- ထို့ကြောင့် Page Size ကို Optimal (Not Minimum or Maximum) ထားသင့်ပါသည်။



# Virtual Memory in Linux

- Linux OS မှာ virtual memory ကို swap space လို့ သုံးနှုန်းပါတယ်။ Swap ကို dedicated partition သို့မဟုတ် file တစ်ခု အနေနဲ့ configure လုပ်လို့ရပါတယ်။  
ဥပမာ ပိုပြီးမြန်တဲ့ SSD လို disk မှာ file အနေနဲ့ swap ကို create လုပ်တာ ကိုတွေ့ရမှာပါ။

- swap ကို အခုလို file အနေနဲ့ create လုပ်လို့ရပါတယ်။

```
root@sysadmin-Virtual-Machine:/home/sysadmin# fallocate -l 2G /swapfile-2G
root@sysadmin-Virtual-Machine:/home/sysadmin# dd if=/dev/zero of=/swapfile-2G bs=1024 count=2097152
```

- လုံခြုံရေးအတွက် permission change ပါမယ်။ superuser root ဝဲ ပေးသုံးမှာပါ။

```
root@sysadmin-Virtual-Machine:/home/sysadmin# chmod 600 /swapfile-2G
```

- mkswap နဲ့ create လုပ်ပါမယ်။

```
root@sysadmin-Virtual-Machine:/home/sysadmin# mkswap /swapfile-2G
Setting up swapspace version 1, size = 2 GiB (2147479552 bytes)
no label, UUID=53c6493e-68a5-492d-9a69-5d5486175f16
root@sysadmin-Virtual-Machine:/home/sysadmin# swapon /swapfile-2G
```

# Virtual Memory in Linux

- လက်ရှိ swap ကိုကြည့်ရင် 2GB ပါ။ ဒါကိုထပ်ပြီး 2GB တိုးမှာပါ။

```
root@sysadmin-Virtual-Machine:/home/sysadmin# free -m
              total        used        free      shared  buff/cache   available
Mem:           4285          477        3010           4         797        3579
Swap:          2047           0         2047
```

- swapon နဲ့ activate လုပ်ပြီး free နဲ့ကြည့်ရအောင်။

```
root@sysadmin-Virtual-Machine:/home/sysadmin# swapon /swapfile-2G
root@sysadmin-Virtual-Machine:/home/sysadmin# free -m
              total        used        free      shared  buff/cache   available
Mem:           4313          479          930           4        2904        3577
Swap:          4095           0         4095
```

# File Management

- အရင်တုန်းက Data များကို Magnetic Storage Devices များတွင် Permanently Store လုပ်ကြပါသည်။
- ထို့အပြင် Virtual Memory ကို Implement လုပ်ဖို့ Page များကို Store လုပ်ဖို့ Backing Store လိုပါသည်။ Backing Store ကို Storage Device (Drive) ပေါ်မှာ Store လုပ်ရခြင်း ဖြစ်သည်။
- ထို့ကြောင့် Storage Device (Drive) များကို Manage လုပ်ဖို့ အရေးကြီးပါသည်။ Storage Device များကို Computer System တစ်ခု၏ Peripheral Devices များအနေဖြင့် ယူဆနိုင်ပါသည်။ တစ်နည်းအားဖြင့် Storage Device များသည် I/O Device များလည်း ဖြစ်ပါသည်။
- သို့ဖြစ်၍ အခု ကျွန်တော်တို့ File Management များကို ပေါင်းပြီး ဆွေးနွေးပါမည်။



# File

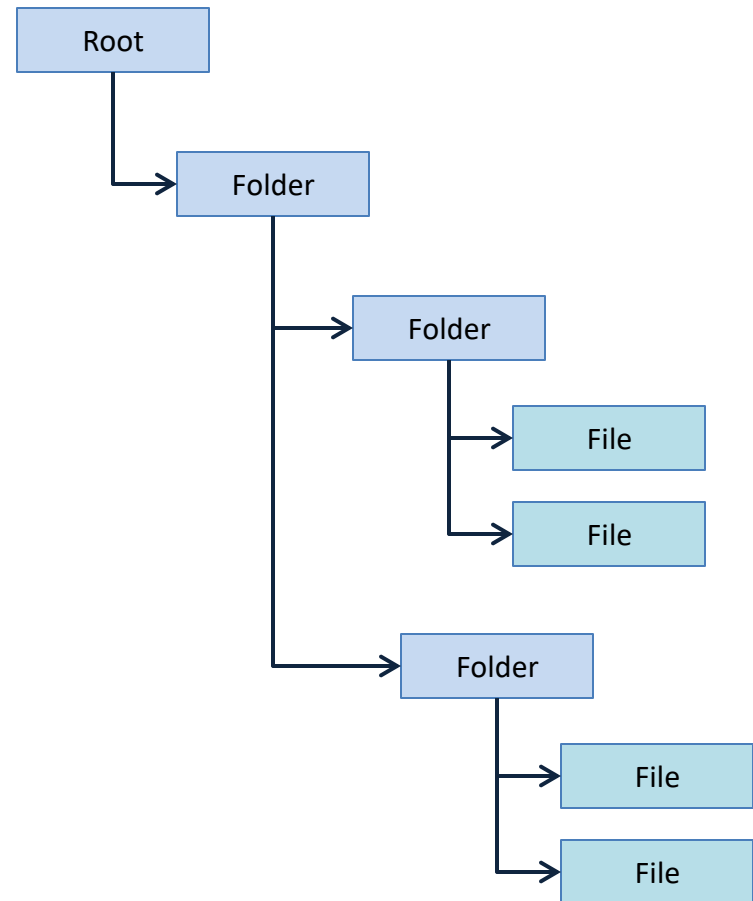
- ကျွန်တော်တို့ File ဆိုတာဘာလဲ ဆိုသည့်မေးခွန်းနှင့် စပါမည်။
- အမှန်တော့ File ဆိုသည်က Logical Organizational Unit of Data ဖြစ်ပါသည်။ တစ်နည်းအားဖြင့် Data များကို စနစ်တကျစုစည်းထားသော Unit တစ်ခု ဖြစ်ပါသည်။
- ထို့ကြောင့် File များတွင် Organizational Structure တစ်ခုရှိပြီး ထို Structure များပေါ်မူတည်ပြီး File Types များကွဲပြားပါသည်။
- File များကို Data Type ပေါ်မူတည်ပြီး Text File နှင့် Binary File ဆိုပြီး ခွဲခြားပါသည်။ ထို့အပြင် File များ၏ Structure ပေါ်မူတည်ပြီး Sequential Access File နှင့် Random Access File ဆိုပြီး ခွဲခြားပါသည်။
- File Attributes များသည် File များအကြောင်းကို ဖော်ပြသော Information များဖြစ်ပါသည်။

# File Operation and Access

- File Operation ဆိုတာသည် File များကို လုပ်ဆောင်နိုင်သော Operations များဖြစ်ပါသည်။
  - Read
  - Write
  - Append
  - Truncate
  - Delete
- ထို့အပြင် File Type ပေါ်မူတည်ပြီး File Access Method များကွဲပြားပါသည်။
- အများအားဖြင့် Text File များသည် Sequential Access File များဖြစ်ပြီး Character by Character သို့မဟုတ် Line by Line Access လုပ်ရပါသည်။ Sequential Access File များသည် အများအားဖြင့် Variable Length Record Structure ရှိသော File များ ဖြစ်ပါသည်။
- အများအားဖြင့် Binary File များသည် Random Access File များဖြစ်ပြီး Binary Record များကို Randomly Access လုပ်နိုင်ပါသည်။ Random Access File များသည် အများအားဖြင့် Fixed Length Record Structure ရှိသော File များ ဖြစ်ပါသည်။

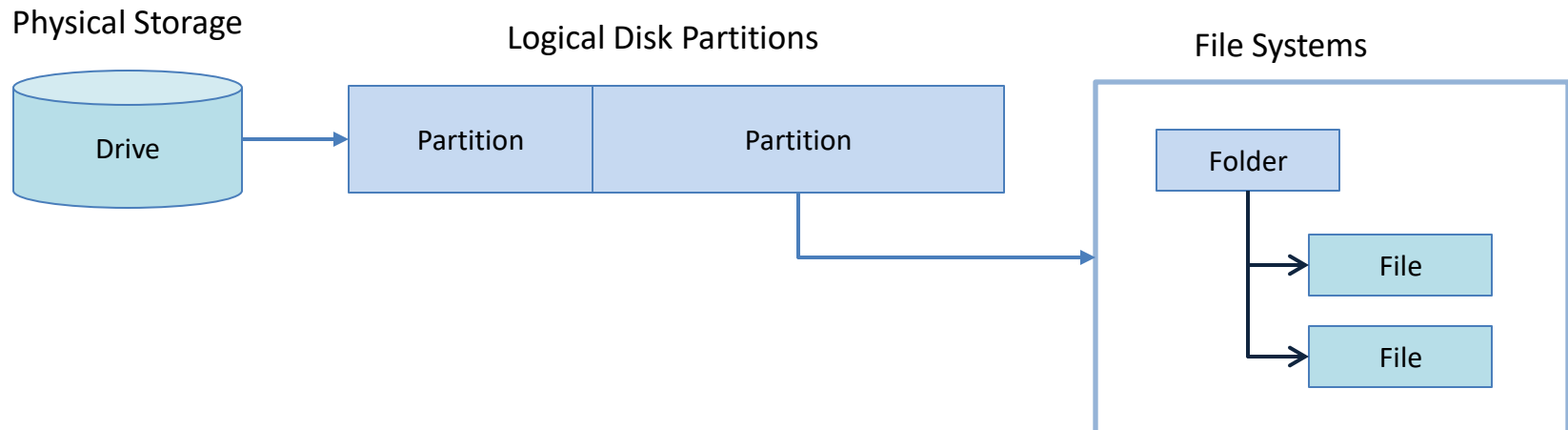
# File Organizations and Directories

- File များကို တခါထပ်ပြီး Organize လုပ်ထားခြင်းကို Folder ဟုခေါ်ပါသည်။
- Folder များ၏ Hierarchical Organizational Structure ကိုတော့ Directory ဟုခေါ်ပါသည်။
- အမှန်တော့ File များနှင့် Directories များကို File Table များဖြင့် Manage လုပ်ပြီး ဒါကို File Systems ဟုခေါ်ပါသည်။
- Operating Systems များပေါ်မူတည်ပြီး File Systems များ ကွာခြားမည် ဖြစ်ပါသည်။
- File Systems များသည် အကြမ်းအားဖြင့် Directory Index (ဥပမာ၊ Yellow Pages) များနှင့် အလားတူပြီး File များကို လွယ်ကူစွာ Search, List, Operate and Access လုပ်နိုင်ဖို့ ဖြစ်ပါသည်။
- ထို့အပြင် File Systems သည် Available Disk Space ကိုလည်း တွက်ချက်ရပါသည်။

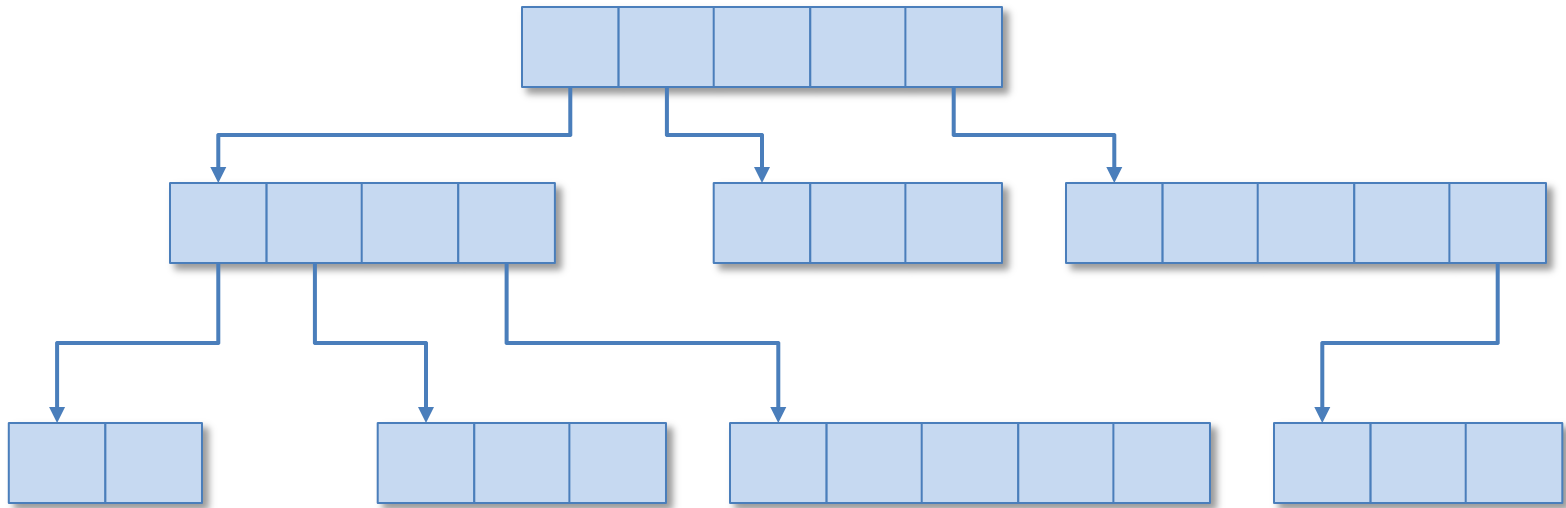


# File Systems and Partitions

- File System များသည် Storage Drive တစ်ခုကို ဘယ်လို Organize လုပ်မလဲကို အဓိကထားပြီး ဖြေရှင်းပါသည်။
- File System များသည် အများအားဖြင့် B-Trees များကို အခြေခံသော File Table များဖြင့် Storage Drive တစ်ခုကို Organize လုပ်ပါသည်။
- ပုံမှန်အားဖြင့် Storage Drive တစ်ခုကို Disk Partitions များခွဲပါသည်။ ထို Disk Partitions များကိုမှ Directory Structure များကို File System များဖြင့် ပြန်ပြီး Organize လုပ်ပါသည်။
- Different Disk Partitions များတွင် Different File Systems များကို အသုံးပြုနိုင်ပါသည်။



# B-Trees

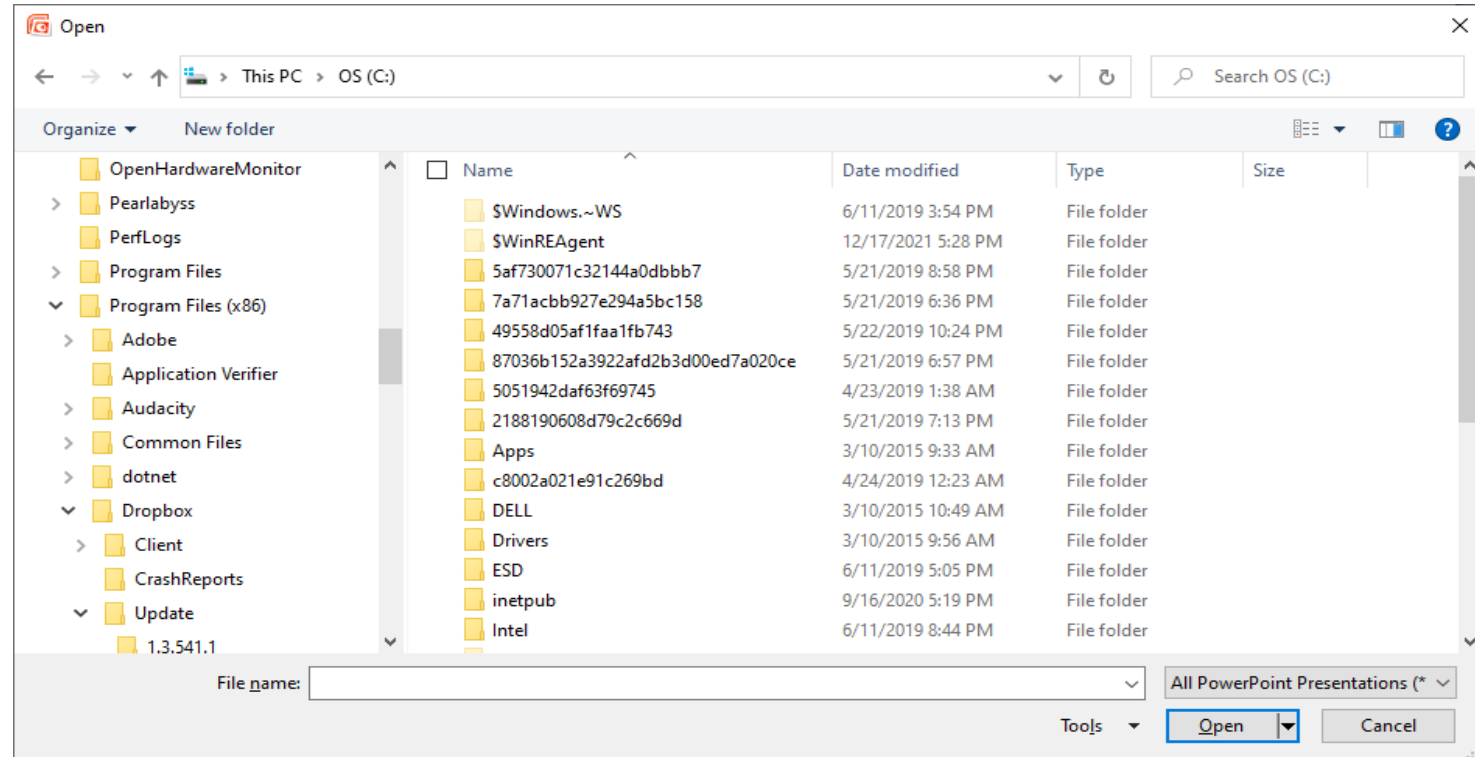


- B-Tree ကို အများအားဖြင့် Index များတည်ဆောက်ရာတွင် အသုံးပြုပြီး File System များနှင့် Database Management System များတွင် အသုံးများပါသည်။
- B-Tree အသေးစိတ်ကို ဒီမှာ လေ့လာကြည့်ပါ။
  - <https://en.wikipedia.org/wiki/B-tree>

# File Systems in Windows

## Disk Partitions, File Systems and Directories

Volume	Layout	Type	File System	Status	Capacity	Free Spa...	% Free
(Disk 0 partition 1)	Simple	Basic		Healthy (EFI System Partition)	500 MB	500 MB	100 %
(Disk 0 partition 2)	Simple	Basic		Healthy (OEM Partition)	40 MB	40 MB	100 %
(Disk 0 partition 4)	Simple	Basic		Healthy (Recovery Partition)	750 MB	750 MB	100 %
(Disk 0 partition 7)	Simple	Basic		Healthy (Recovery Partition)	7.42 GB	7.42 GB	100 %
Data (D:)	Simple	Basic	NTFS	Healthy (Basic Data Partition)	463.21 GB	209.53 GB	45 %
OS (C:)	Simple	Basic	NTFS	Healthy (Boot, Page File, Crash Dump, ...)	459.50 GB	245.70 GB	53 %



Windows Operating Systems တွင် NTFS File System ကို အသုံးပြုပါသည်။

# File Systems in Linux

- အခု ကျွန်တော်တို့ lab setup မှာသုံးတဲ့ Ubuntu 20.04 ရဲ့ default filesystem ကတော့ ext4 ဖြစ်ပါတယ်။
- Linux က open-source OS ဖြစ်တဲ့အတိုင်း filesystem လဲရွှေးချယ်စရာများပါတယ်။ မိမိကြိုက်ရာ filesystem ကိုရွှေးလို့ရပါတယ်။
- Linux OS variance ပေါ်သူတို့ရဲ့ OS version ပေါ်မူတည်ပြီး recommended/default filesystem တွေပြောင်းလေ့ရှိပါတယ်။
- Linux filesystem ကို journaled filesystem အဖြစ် ext3 version မှ support လုပ်ပါတယ်။
- XFS ကတော့ advanced journaling ဖြစ်တဲ့အတွက် ပို stable ဖြစ်ပါတယ်။ နောက်ပိုင်း XFS ကို default filesystem အဖြစ် တွေ့ရမှာပါ။
- ဘယ် filesystem နဲ့ format လုပ်ပြီးသုံးထားတာကိုတော့ mount command နဲ့ကြည့်လို့ရပါတယ်။

```
root@sysadmin-Virtual-Machine:/home/sysadmin# mount |grep sda
/dev/sda5 on / type ext4 (rw,relatime,errors=remount-ro)
/dev/sda1 on /boot/efi type vfat (rw,relatime,fmask=0077,dmask=0077,codepage=437,iocharset=iso8859-1,shortname=mixed,errors=remount-ro)
root@sysadmin-Virtual-Machine:/home/sysadmin#
```

# File Systems in Linux

- Linux filesystem ကို tree structure လို့ မြင်ကြည့်လို့ရပါတယ်။ / (root) folder ပါ။
- tree utility ကရှိရင်းစေ့ပါ။ apt install tree နဲ့ tree utility ကို install လုပ်ပါ။

```
root@sysadmin-Virtual-Machine:/home/sysadmin# tree -L 1 /
/
├── bin -> usr/bin
├── boot
├── cdrom
├── dev
├── etc
├── home
├── lib -> usr/lib
├── lib32 -> usr/lib32
├── lib64 -> usr/lib64
├── libx32 -> usr/libx32
├── lost+found
├── media
├── mnt
├── opt
├── proc
├── root
├── run
├── sbin -> usr/sbin
├── snap
├── srv
├── swapfile
├── swapfile-2G
├── sys
├── tmp
├── usr
└── var

24 directories, 2 files
root@sysadmin-Virtual-Machine:/home/sysadmin# tree -L 1 /home/
/home/
└── sysadmin

1 directory, 0 files
root@sysadmin-Virtual-Machine:/home/sysadmin# tree -L 1 /home/sysadmin/
/home/sysadmin/
├── Desktop
├── Documents
├── Downloads
├── fork.py
├── Music
├── Pictures
├── Public
├── Templates
├── thread.py
└── Videos

8 directories, 2 files
root@sysadmin-Virtual-Machine:/home/sysadmin#
```

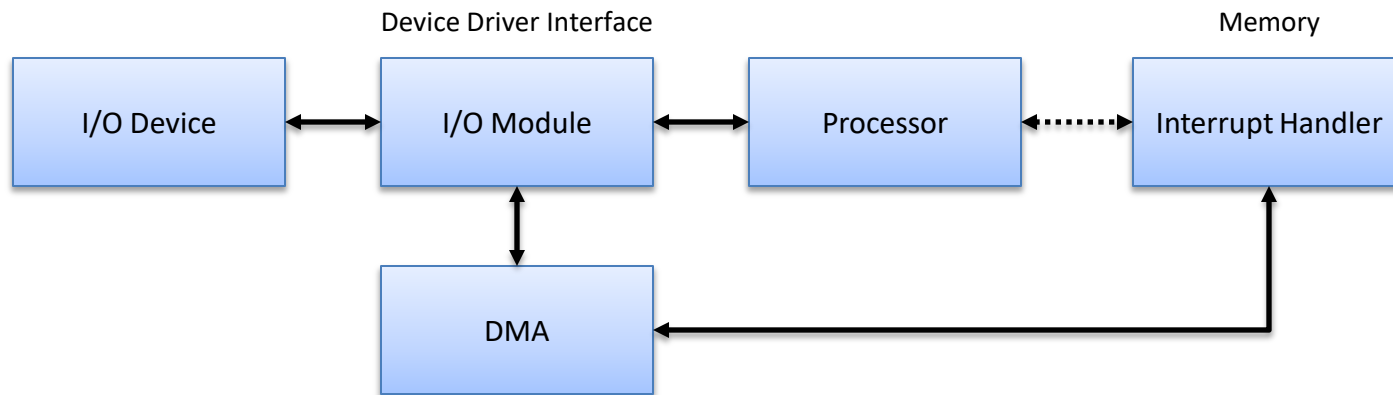


# I/O Management

- Disk Drive များအပါအဝင် အခြား I/O Device များကို Device Drivers များမှတစ်ဆင့် Manage လုပ်ပါသည်။
- အမှန်တော့ I/O Device များကို
  - Programmed I/O Management (ဥပမာ၊ Printer များ)
  - Interrupt Driven I/O Management (ဥပမာ၊ Keyboard များ၊ Mouse များ)
  - Direct Memory Access Management (ဥပမာ၊ Hard Drive များ၊ USB External Drive များ၊ Graphics Card များ)
- Programmed I/O Management တွင် Operating Systems က I/O Module ကို Command ပေးလိုက်ပြီး ကျန်တာကို I/O Module သူ့ဟာသူ ဆက်လုပ်သွားမည် ဖြစ်သည်။
- Interrupt Driven I/O Management တွင် Operating Systems က I/O Module ကို Interrupt Handler ဖြင့် Manage လုပ်ပါသည်။ I/O Buffer မှ IRQ (Interrupt Request) များဖြင့် Interrupt Handler များ အလုပ်လုပ်ပုံကို ပြောခဲ့ဖူးပါသည်။
- Direct Memory Access Management တွင်တော့ DMA Module က Memory နှင့် တိုက်ရိုက်ချိတ်ဆက်ပြီး အလုပ်လုပ်နိုင်ပါသည်။

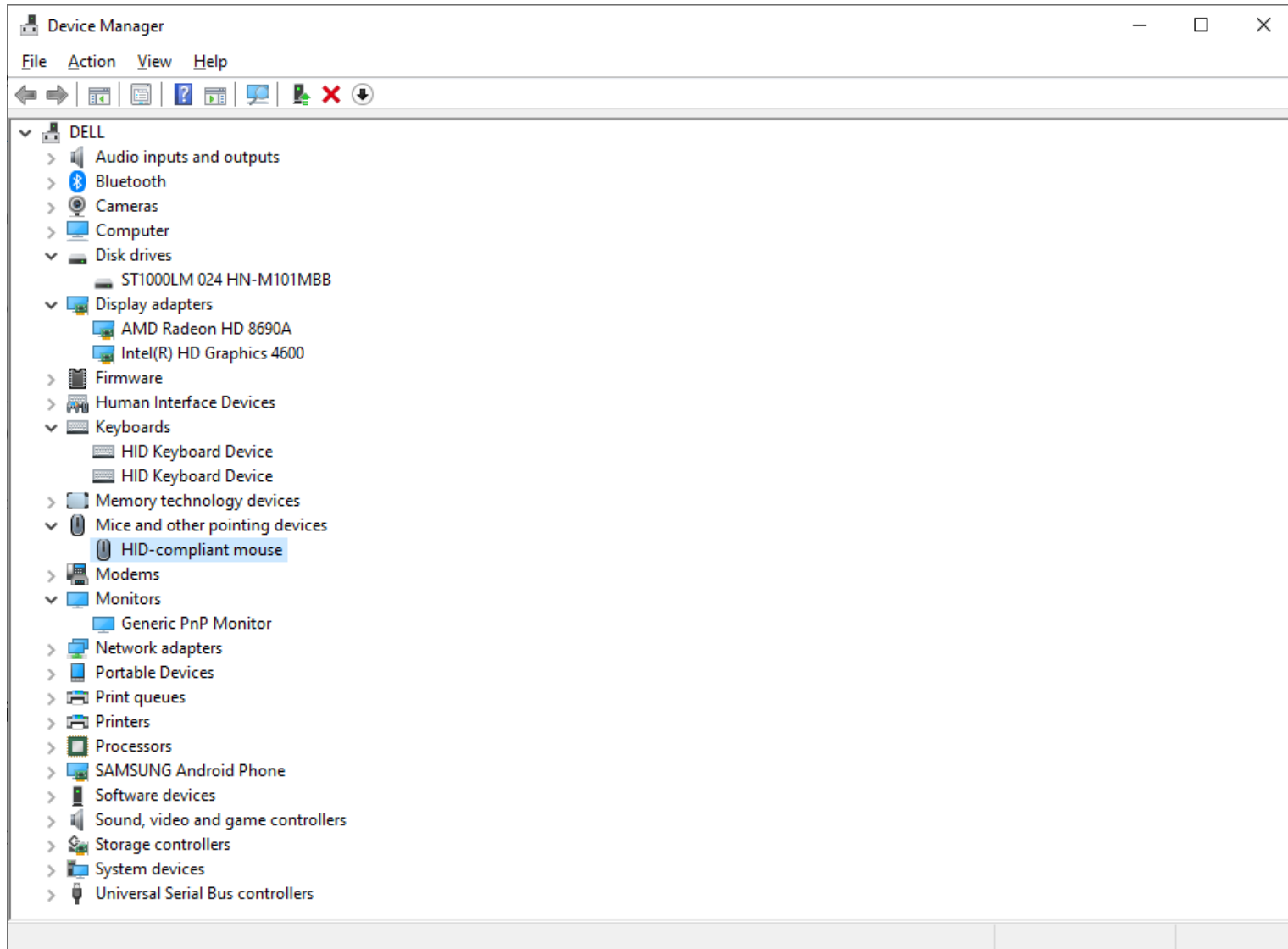
# I/O Management

	No Interrupts	Interrupts
I/O to Memory via Processor	Programmed I/O Management	Interrupt Driven Management
Direct I/O to Memory		DMA Management



- Disk Drive များအပါအဝင် အခြား I/O Device များကို Device Drivers များမှတစ်ဆင့် Manage လုပ်ပါသည်။

# I/O Management in Windows



# I/O Management in Linux

- I/O stack ကို OS ရဲ့ အရေးပါတဲ့ လုပ်ဆောင်မှု အဖြစ်ယူဆလိုရပါတယ်။ Disk access time က HDD နှေးရင်နှေးသလောက် performance impact သိသာပါတယ်။
- I/O Scheduler အမျိုးမျိုးကိုလဲ မိမိ program ရဲ့ လိုအပ်မှုပေါ်မူတည်ပြီး disk seek time ကိုလျော့ချဖို့ tune လုပ်ကျပါတယ်။
- ကျွန်တော်တို့ lab system ရဲ့ IO scheduler ကိုခုလို review လုပ်လိုရပါတယ်။

```
root@sysadmin-Virtual-Machine:/home/sysadmin# cat /sys/block/sda/queue/scheduler  
[mq-deadline] none
```

- Modern OS ဖြစ်တဲ့ Ubuntu လိုမျိုးမှာ SSD disk ဖြစ်နေတာကို auto detect ဖြစ်ပြီး mq-deadline scheduler ကို သုံးနေတာတွေ့ရပါတယ်။

# User and Security Management

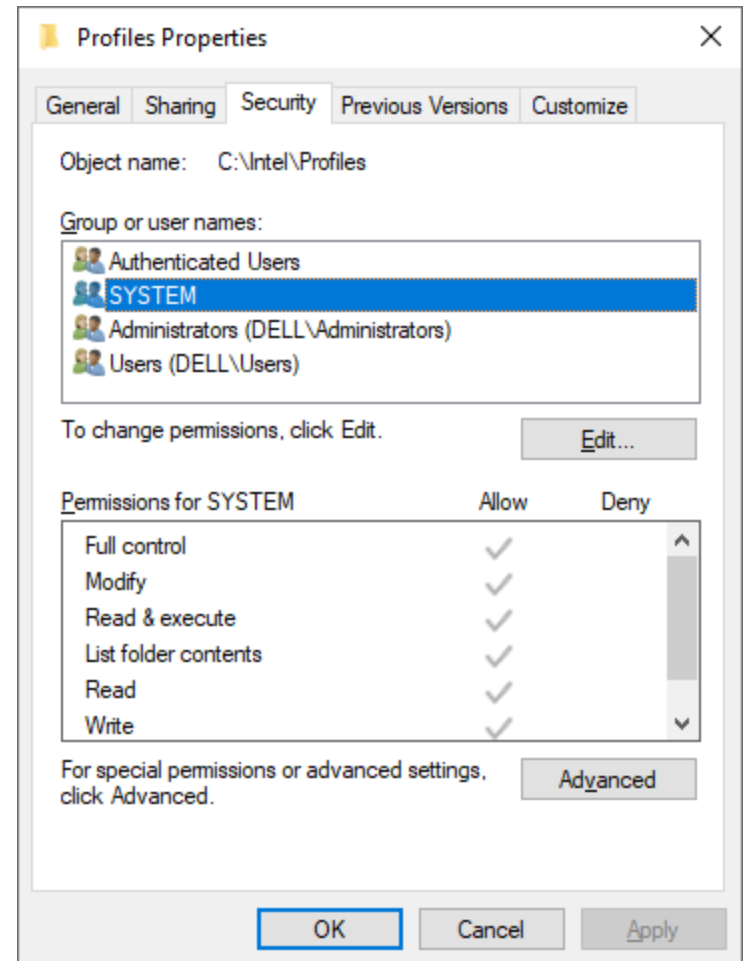
- Operating Systems များသည် အမှန်တော့ Multiple Users များအတွက် လုပ်ဆောင်ရာမှ ပေါ်ပေါက်လာခြင်း ဖြစ်၍ User Management သည် Operating Systems များအတွက် အရေးကြီးသည်ကို အထူးပြောစရာ မလိုပါ။
- Operating Systems များကို Multiple Users များ တချိန်တည်း အသုံးပြုနိုင်သည့် အတွက် User များအကြား Computer Resources များ (Files, Memory, I/O) ကို Share လုပ်ဖို့၊ Protect လုပ်ဖို့ လိုအပ်ပါသည်။
- ထိုမျှသာမက Internet နှင့် ချိတ်ဆက် အသုံးပြုလာသည့်အခါ External Threats (Virus, Spyware, Trojans) စသည်တို့မှလည်း ကာကွယ်ပေးဖို့ လိုအပ်လာပါသည်။
- ထို့ကြောင့် User and Security Management သည် အင်မတန် ကျယ်ပြန့်သော နယ်ပယ်ဖြစ်ပြီး ကျွန်တော်တို့က အကြမ်းဖျဉ်းသာ ဆွေးနွေးပါမည်။

# User Management

- User ဟူသည် Computer ကို အသုံးပြုခွင့် ရထားသူများကို ဆိုလိုပါသည်။ User များသည် Operating Systems မှတစ်ဆင့် Computer Resources များကို သုံးပြီး လိုအပ်သော Application Code များကို အသုံးပြုမည် သို့မဟုတ် Execute လုပ်မည် ဖြစ်ပါသည်။
- ထို့ကြောင့် User များကို Manage လုပ်ဖို့ အခြေခံ 2 ပိုင်းပါဝင်ပါသည်။
  - User Authentication
  - User Authorization
- User Authentication က Computer ကို အသုံးပြုသူသည် အသုံးပြုခွင့် ရထားသူ ဟုတ်မဟုတ်ကို Verify လုပ်ဖို့ ဖြစ်ပါသည်။ User Authentication ကို Password များ၊ PIN Code များ ဖြင့် Authenticate လုပ်ပါသည်။
- User Authorization က Computer ကို အသုံးပြုခွင့်ရသူသည် ဘယ် Resources များကို Access လုပ်ခွင့်ရှိသည် မရှိသည်ကို Manage လုပ်ပါသည်။
- ထုံးစံအားဖြင့် Operating System Kernels သည် Privileges အမြင့်ဆုံးဖြစ်သည်။
- သို့သော် User များကို User Role များဖြင့် ပြန်ပြီး Authorize လုပ်ပါသည်။ User များထဲတွင် System Administrator (Root) သည် Privileges အမြင့်ဆုံးဖြစ်သည်။

# User Management in Windows

- Windows တွင် Windows Authentication ကို Kerberos Protocol ဖြင့် လုပ်ဆောင်ပြီး နောက်ပိုင်းတွင် Single Sign-In Authentication ကိုလုပ်ဆောင်လာပါသည်။ တစ်နည်းအားဖြင့် Windows (Microsoft) Account တစ်ခုတည်းဖြင့် App Store, Windows, Phones များကို Access လုပ်နိုင်ခြင်း ဖြစ်သည်။
- User Roles များအတွက် အခြေခံ 2 မျိုးရှိပါသည်။ Administrators နှင့် Users များ ဖြစ်ပါသည်။
- Files များနှင့် Folders များကို ဘယ် User များကို ပေးသုံးမည်ကို ဆုံးဖြတ်နိုင်ပါသည်။



# User Management in Linux

- Linux OS မှာလဲ user management ရဲ့ role ကအတော်အရေးပါပါတယ်။
- OS ရဲ့ process တိုင်းကို user account တစ်ခုခုကပိုင်ပါတယ်။
- OS ရဲ့ processes တွေကို root account ကပိုင်ပါတယ်။

```
root@sysadmin-Virtual-Machine:/home/sysadmin# ps aux |grep udevd
root      285  0.0  0.1 24032 7392 ?        Ss   01:22   0:00 /lib/systemd/systemd-udev
root      4524  0.0  0.0 17676  732 pts/1    S+   14:27   0:00 grep --color=auto udev
```

- Network service တစ်ခုဖြစ်တဲ့ Apache web server ရဲ့ process ကို www-data user ကပိုင်ပါတယ်။

```
root@sysadmin-Virtual-Machine:/home/sysadmin# ps aux |grep apache
root      649  0.0  0.1  6540 4824 ?        Ss   01:22   0:01 /usr/sbin/apache2 -k start
www-data  2829  0.0  0.0 1997852 4376 ?        Sl   09:20   0:00 /usr/sbin/apache2 -k start
www-data  2830  0.0  0.0 1932316 4376 ?        Sl   09:20   0:00 /usr/sbin/apache2 -k start
```

- Service user ဖြစ်တဲ့ www-data ကို /etc/passwd မှာတွေ့ရပါမယ်။

```
root@sysadmin-Virtual-Machine:/home/sysadmin# cat /etc/passwd |grep www
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
```



# User Management in Linux

- User account တစ်ခုမှာ user ID group ID assignment ရှိတာကိုတွေ့ရပါတယ်။  
ဒီ sysadmin accountကို 1000 assign လုပ်ထားတာတွေ့ရပါတယ်။

```
root@sysadmin-Virtual-Machine:/home/sysadmin#  
root@sysadmin-Virtual-Machine:/home/sysadmin# cat /etc/passwd|grep sysadmin  
sysadmin:x:1000:1000:admin,,,:/home/sysadmin:/bin/bash  
root@sysadmin-Virtual-Machine:/home/sysadmin#  
root@sysadmin-Virtual-Machine:/home/sysadmin#
```

- Home folder path ကိုလဲ တွေ့ရပါမယ်။ Windows မှာ Users folder အောက်ကမိမိ account folder လိုမျိုး folder ပါ။
- File နဲ့ folder permission တွေကို UID GID တွေနဲ့ ခုလိုကြည့်လို့ရပါတယ်။ sysadmin user ရဲ့ home folder ဆိုတော့ သူ့ UID ကပိုင်တာတွေ့ရမှာပါ။

```
root@sysadmin-Virtual-Machine:/home/sysadmin# ls -ln /home/sysadmin/  
total 40  
drwxr-xr-x 2 1000 1000 4096 Dec 27 12:04 Desktop  
drwxr-xr-x 2 1000 1000 4096 Dec 27 12:04 Documents  
drwxr-xr-x 2 1000 1000 4096 Dec 27 12:04 Downloads
```

# Security Management

- User များကို Protect လုပ်ဖို့ အပြင် External Threats များကိုပါ Protect လုပ်ဖို့ Security Management များကို လိုအပ်ပါသည်။
- ကျွန်တော်တို့ Operating Systems Services နှင့် Application Services များကို ပြောခဲ့ပါသည်။ ထို Services များသည် Local Services များ ဖြစ်နိုင်သလို Remote (Network) Services များ ဖြစ်နိုင်ပါသည်။
- Remote Services များကို Socket များဖြင့် Communicate လုပ်ကြပါသည်။ External Threat အများဆုံးဝင်ရောက်တာသည် Remote Services များကြောင့် ဖြစ်ပါသည်။
- မိမိ Computer မှာ Hacker တစ်ယောက်၏ Remote Service တစ်ခုကို Install လုပ်မိပါက ထို Service သည် Hacker သို့ မိမိ Data များကို ပေါက်ကြားစေမည်ဖြစ်သည်။
- ထို့ကြောင့် ထိုဖြစ်ရပ်ကို ကာကွယ်ဖို့ Firewall System တစ်ခုရှိပြီး Remote Services များကို Communicate လုပ်လို့ မရအောင် ပိတ်လို့ ရပါသည်။

# Security Management in Windows

- Firewall ကို ကျွန်တော်တို့ User, Local Port, Remote Port, Network Protocol (TCP/IP, UDP, FTP), Remote IP Address တို့ပေါ်မူတည်ပြီး အပိတ်အဖွင့် လုပ်လို့ရပါသည်။
- Inbound က Incoming Access ဖြစ်ပြီး Outbound က Outgoing Access ဖြစ်ပါသည်။
- ထို့အပြင် အခြား Malware များကို ကာကွယ်ဖို့ အခြား Operating Systems Services များလိုအပ်ပါသည်။
- သို့သော် အသေးစိတ်တော့ ကျွန်တော်တို့ မဆွေးနွေးတော့ပါ။

Name	Profile	Action	Override	Direction	Program	Local
Dropbox	All	Allow	No	Inbound	C:\Progr...	Any
EdgeDevtoolsPlugin	All	Allow	No	Inbound	Any	Any
File and Printer Sharing (Echo Request - ICMPv4-In)	Private	Allow	No	Inbound	System	Any
File and Printer Sharing (Echo Request - ICMPv6-In)	Private	Allow	No	Inbound	System	Any
File and Printer Sharing (LLMNR-UDP-In)	Private	Allow	No	Inbound	C:\WIND...	Any
File and Printer Sharing (NB-Datagram-In)	Private	Allow	No	Inbound	System	Any
File and Printer Sharing (NB-Name-In)	Private	Allow	No	Inbound	System	Any
File and Printer Sharing (NB-Session-In)	Private	Allow	No	Inbound	System	Any
File and Printer Sharing (SMB-In)	Private	Allow	No	Inbound	System	Any
File and Printer Sharing (Spooler Service - RPC)	Private	Allow	No	Inbound	C:\WIND...	Any
File and Printer Sharing (Spooler Service - RPC-EPMAP)	Private	Allow	No	Inbound	C:\WIND...	Any
Firefox (C:\Program Files\Mozilla Firefox)	Private	Allow	No	Inbound	C:\Progr...	Any
Firefox (C:\Program Files\Mozilla Firefox)	Private	Allow	No	Inbound	C:\Progr...	Any
genshinimpact	Private...	Allow	No	Inbound	C:\progr...	Any
genshinimpact	Private...	Allow	No	Inbound	C:\progr...	Any
Google Chrome (mDNS-In)	All	Allow	No	Inbound	C:\Progr...	Any
Groove Music	Domai...	Allow	No	Inbound	Any	Any
Java(TM) Platform SE binary	Private	Allow	No	Inbound	C:\progr...	Any
Java(TM) Platform SE binary	Private	Allow	No	Inbound	C:\progr...	Any
Java(TM) Platform SE binary	Private	Allow	No	Inbound	D:\applic...	Any
Java(TM) Platform SE binary	Private	Allow	No	Inbound	D:\applic...	Any
Java(TM) Platform SE binary	Private	Allow	No	Inbound	C:\progr...	Any
Java(TM) Platform SE binary	Private	Allow	No	Inbound	C:\progr...	Any
Mail and Calendar	All	Allow	No	Inbound	Any	Any
mDNS (UDP-In)	Private	Allow	No	Inbound	C:\WIND...	Any
Message Queuing TCP Inbound	All	Allow	No	Inbound	C:\WIND...	Any
Message Queuing UDP Inbound	All	Allow	No	Inbound	C:\WIND...	Any
Microsoft Edge	Domai...	Allow	No	Inbound	Any	Any
Microsoft Edge (mDNS-In)	All	Allow	No	Inbound	C:\Progr...	Any
Microsoft Edge (mDNS-In)	All	Allow	No	Inbound	C:\Progr...	Any
Microsoft Office Groove	Private	Allow	No	Inbound	C:\Progr...	Any
Microsoft Office Groove	Private	Allow	No	Inbound	C:\Progr...	Any

# Security Management in Linux

- Linux security ကိုလေ့လာကြည့်ရင် AppArmor/SELinux (Mandatory Access Controls) နဲ့ iptables (packet filter firewall) တို့ကိုတွေ့ရပါမယ်။
- MAC module က process and program တွေရဲ့ integrity ကို protect လုပ်ပေးပါတယ်။
- Ubuntu မှာ AppArmor ရဲ့ status ကို systemctl နဲ့ကြည့်လို့ရပါတယ်။ ခုအခြေအနေမှာ Firefox ကို protect မလုပ်ထားပါဖူး။

```
root@sysadmin-Virtual-Machine:/home/sysadmin# systemctl status apparmor
● apparmor.service - Load AppArmor profiles
   Loaded: loaded (/lib/systemd/system/apparmor.service; enabled; vendor preset: enabled)
   Active: active (exited) since Mon 2022-01-03 18:18:15 AWST; 21h ago
     Docs: man:apparmor(7)
           https://gitlab.com/apparmor/apparmor/wikis/home/
   Process: 463 ExecStart=/lib/apparmor/apparmor.systemd reload (code=exited, status=0/SUCCESS)
   Main PID: 463 (code=exited, status=0/SUCCESS)

Jan 03 18:18:15 sysadmin-Virtual-Machine systemd[1]: Starting Load AppArmor profiles...
Jan 03 18:18:15 sysadmin-Virtual-Machine apparmor.systemd[463]: Restarting AppArmor
Jan 03 18:18:15 sysadmin-Virtual-Machine apparmor.systemd[463]: Reloading AppArmor profiles
Jan 03 18:18:15 sysadmin-Virtual-Machine apparmor.systemd[482]: Skipping profile in /etc/apparmor.d/disable: usr.sbin.rsyslogd
Jan 03 18:18:15 sysadmin-Virtual-Machine apparmor.systemd[490]: Skipping profile in /etc/apparmor.d/disable: usr.bin.firefox
Jan 03 18:18:15 sysadmin-Virtual-Machine systemd[1]: Finished Load AppArmor profiles.
```

- ခု enable လုပ်ကြည့်ရအောင်။ disable folder မှ softlink ကို delete လုပ်ပါတယ်။ profile ကို AppArmor parser ကို pass လုပ်ပေးပါတယ်။ နောက်ဆုံးအနေနဲ့ systemd ကို reload လုပ်ပေးရပါမယ်။

```
root@sysadmin-Virtual-Machine:/home/sysadmin# rm /etc/apparmor.d/disable/usr.sbin.rsyslogd
root@sysadmin-Virtual-Machine:/home/sysadmin# cat /etc/apparmor.d/usr.bin.firefox | apparmor_parser -a
root@sysadmin-Virtual-Machine:/home/sysadmin# systemctl reload apparmor
```

# Security Management in Linux

- iptables ကိုတော့ Linux ရဲ့ legacy firewall လို့ပြောရမှာပါ။
- ကျွန်တော်တို့ သိပြီးသားဖြစ်တဲ့အတိုင်း network services တွေမှာ well known port number တွေရှိကြပါတယ်။ အသုံးပြုပေါ်မူတည်ပြီး default port number ကိုသုံးပြီး serviceကို host လုပ်တာရှိသလို တခြား port တွေ change ပြီး host လုပ်တာရှိပါတယ်။
- ခု lab မှာတော့ Web server တစ်ခုရဲ့ default port 80 ကို iptables packet filter ကမိတ်နေအောင် လုပ်ဆောင်မှာဖြစ်ပါတယ်။

- netstat နဲ့ port ကို confirm လုပ်ပါမယ်။

```
root@sysadmin-Virtual-Machine:/home/sysadmin# netstat -anlp |grep apache
tcp6      0      0 :::80          :::*            LISTEN     649/apache2
```

- ufw က iptables ကို manage လုပ်တဲ့ utility ပါ။ ufw enable နဲ့ activate လုပ်ပါတယ်။

```
root@sysadmin-Virtual-Machine:/home/sysadmin# ufw enable
Command may disrupt existing ssh connections. Proceed with operation (y|n)? y
Firewall is active and enabled on system startup
```

- ufw status verbose နဲ့ လက်ရှိ rules တွေကိုကြည့်လို့ရပါတယ်။

```
root@sysadmin-Virtual-Machine:/home/sysadmin# ufw status verbose
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), disabled (routed)
New profiles: skip

To Action From
--
22 ALLOW IN Anywhere
22 (v6) ALLOW IN Anywhere (v6)
```

# Security Management in Linux

- `ufw allow <port number>` နဲ့ port ကို open ပေးရလို့ရပါတယ်။ ခု lab မှာတော့ 80။

```
root@sysadmin-Virtual-Machine:/home/sysadmin# ufw allow 80
Rule added
Rule added (v6)
```

```
root@sysadmin-Virtual-Machine:/home/sysadmin# ufw status verbose
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), disabled (routed)
New profiles: skip

To Action From
--
22 ALLOW IN Anywhere
80 ALLOW IN Anywhere
22 (v6) ALLOW IN Anywhere (v6)
80 (v6) ALLOW IN Anywhere (v6)
```

# Summary

- အခုဆိုရင်တော့ ကျွန်တော်တို့ Operating Systems များအကြောင်းကို နည်းနည်းသိသွားလောက်ပါပြီ။
- အမှန်တော့ Operating Systems များသည် အရှုပ်ထွေးဆုံးသော Software Systems များ ဖြစ်ပါသည်။ Space Shuttle တစ်ခု အတွက် လိုအပ်သော Line of Code သည် 400,000 ဖြစ်ပြီး Android OS အတွက် လိုအပ်သော Line of Code က 12,000,000 ဖြစ်၍ Windows အတွက် လိုအပ်သော Line of Code က 45,000,000 ဖြစ်ပါသည်။
- ထို့ကြောင့် ကျွန်တော့်တို့ အနေဖြင့် Operating Systems တစ်ခုလုံး၏ လုပ်ဆောင်ပုံကို အသေးစိတ်ပြောဖို့ မဖြစ်နိုင်ပါ။
- သို့သော်လည်း အရေးကြီးသော အချက်များကို Overview နားလည်ခြင်းဖြင့် လိုအပ်သလို အသေးစိတ်ထပ်ပြီး လေ့လာနိုင်မည် ဖြစ်ပါသည်။
- Linux နှင့် Android Kernel များ၏ Open Source များကို လေ့လာကြည့်ပါ။ ပြီးလျှင် Open Source Community များကို ဝင်ကြည့်ပြီး ဒီလောက် ရှုပ်ထွေးသော Software Systems များကို အနယ်နယ် အရပ်ရပ်က Software Developers ဘယ်လို ဝိုင်းဝန်းပြီး တည်ဆောက်ကြသည်ကို စိတ်ဝင်စားစရာ တွေ့ရမည် ဖြစ်ပါသည်။