

## Linux Process and Services

Linux မှာ run နေတဲ့ software, program တွေကို process လိုပေါ်ပါတယ်။ ဒီ process တွေဟာ software တွေ program တွေမှာ ရှိတဲ့ services တွေရဲ့ ခိုင်းစေခြင်းကိုခံပြီးမှ အလုပ်လုပ်ကြရပါတယ်။

**Service** တွေကို တနည်းအားဖြင့် **Service Daemon** တွေလိုသတ်မှတ် ပေါ်ကြပါတယ်။

service နဲ့ Daemon တွေကို ထိန်းချုပ်မယ်ဆိုရင် သူတို့တွေကို မောင်းနှင်ပေးတဲ့ကောင်က systemd (system service daemon) ဖြစ်ပါတယ်။ systemd ဆိုတာက တကယ်တော့ system service တစ်ခုပါပဲ။ သူရဲ့အလုပ်က system မှာရှိနေတဲ့ service တွေကို manage လုပ်တာဖြစ်ပါတယ်။ ဘယ် service ကို running time မှာ အလုပ်ပေးလုပ်မှာလဲ ဘယ်ကောင်ကို boottime မှာပေးလုပ်မှာလဲဆိုတာတွေကို manage လုပ်မှာဖြစ်ပါတယ်။

ဒီလို service တွေကို စီမံပြုးတာနဲ့ service တွေက သက်ဆိုင်ရာ process တွေကို မွေးထုတ်ပေးတဲ့အလုပ်ကိုလုပ်ပါတယ်။ System တစ်ခု စတင် power on လိုက်တာနဲ့ systemd ဆိုတဲ့ system service daemon က အရင်ဆုံး အလုပ်လုပ်ပါတယ်။ ဒါကြောင့် ဘယ် process, service မှ မ run ခင် systemd ကို run ရတာဖြစ်ပါတယ်။ systemd စတင်တဲ့ အခါ systemd ရဲ့ boottime မှာ စတင်ချင်တဲ့ service lists ရှိပါတယ်။

အားဖြစ်ရင်းထဲမှာပါတဲ့ service တွေကို အကုန် စတင်ပေးပြီးတော့ သူစတင်ပေးလိုက်တဲ့ သက်ဆိုင်ရာ service တွေက သူတို့နဲ့ သက်ဆိုင်တဲ့ process တွေကို မွေးထုတ်ပေးပြီးအလုပ်စတင်လုပ်ဆောင်ပါတယ်။



အထက်ပါ ပုံမှာ ဆိုရင် **systemd** က ဦးဆောင်ပြီးတော့ service တွေကို run သွားပေးတယ်ဆိုတာကို tree ပုံစံလေးနဲ့မြင်တွေ့နိုင်ပါတယ်။

Linux ရဲ့ အရင် Version တွေမှာ systemd မပေါ်ခင်က initd နဲ့အလုပ်လုပ်ကြပါတယ်။ initd ရဲ့ အားနဲ့ချက်တွေကို ဖယ်ထုတ်ရင်းနဲ့ systemd ကို စတင် release လုပ်လိုက်တဲ့အချင်မှာ အရမ်းကောင်းမွန်တဲ့ systemd ဖြစ်လာပါတယ်။

systemd ရဲ့ အားသာချက်တွေမှာ service တွေကို အလွယ်တကူ manage လုပ်လာနိုင်တယ်။ Service တွေကို separate လုပ်ပြီးတော့ manage လုပ်စရာမလိုတော့ဘဲ systemd တစ်ခုတည်းကနေပါး အကုန်လုံးကို manage လုပ်လိုရလာပါတယ်။

နောက်တစ်ခုက ဒီလို service တွေကို run ဖို့အတွက် systemd ကလုပ်ပေးသလို ကျွန်ုပ်တော်တို့ user တွေ အနေနဲ့လည်း အားလုံးကို ထိန်းချုပ်ပေးလို့ရပါတယ်။ ကျွန်ုပ်တော်တို့ ထိန်းချုပ်လို့မရတဲ့ service ကတော့ static service အမျိုးအစားပါပဲ။ ဒီ service တွေက system service လိုပေါ်တဲ့ systemd နဲ့ auto အလုပ်လုပ်ပေးတဲ့ကောင်တွေဖြစ်ပါတယ်။

နောက်ပါး **systemd** မှာ **unit** တွေနဲ့အလုပ်လုပ်ပါတယ်။ **unit** အမျိုးအစားတွေအများကြီးထဲက အမျိုးအစားနဲ့ ကို ဥပမာလေးကြည့်ရအောင်။

**Service unit** ဆိုတဲ့အမျိုးအစားရယ်။ နောက်ပြီး **service** တွေတစ်ခုနဲ့တစ်ခု ဆက်သွယ်ဖို့အတွက် **communication** လုပ်ဖို့အတွက် **socket unit** ဆိုတဲ့အမျိုးအစားရယ်။ **path** ဆိုတဲ့အမျိုးအစားတွေ လည်းရှိပါတယ်။ ဒီလိုမျိုး **unit** အလိုက်ခွဲထားပေးတာရှိပါတယ်။ ဒီ **unit** တွေက **service** တစ်ခုလုပ်ဖို့အတွက် နောက်ကွယ်က နောက်ကွယ်တဲ့အနေနဲ့ ထောက်ပဲ ပေးထားခြင်းဖြစ်ပါတယ်။ ဒီ **unit dependencies** တွေဟာ **service** တိုင်းမှာ မလိုအပ်ပါဘူး။ တစ်ချို့ **service** တွေက **socket unit** ပဲလိုအပ်တာ ရှိသလို တစ်ချို့က ဒီ **unit** တွေမလိုအပ်ပဲ **service unit** သီးသန် အလုပ်လုပ်တဲ့ **service** တွေလည်းရှိပါတယ်။

အိုကေ ဒီ **service** တွေအားလုံးက **process** တွေကို မွေးထုတ်ပေးတယ်ဆိုတော့ **process** တွေက ဘယ်လို အလုပ်လုပ်ဆောင်ကြသလဲဆိုတာကို အောက်က ပုံလေးမှာကြည့်ရအောင်။

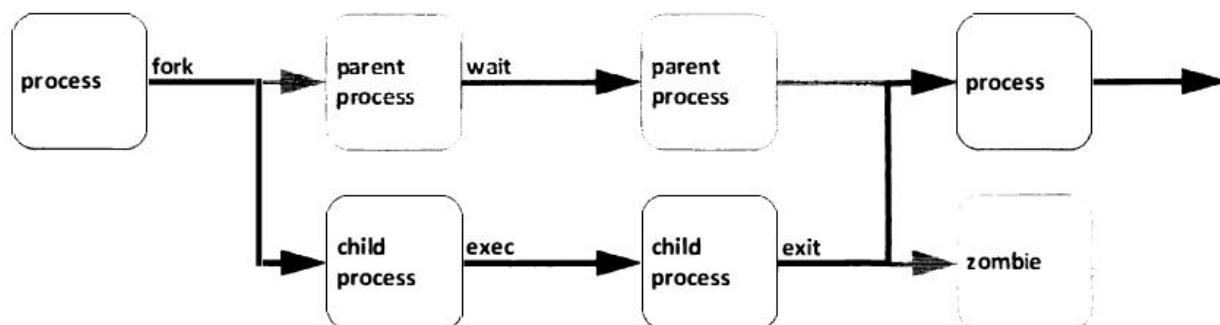


Figure 7.1: Process life cycle

အထက်ပါပုံအရ **service** တစ်ခုသည် **process** တစ်ခုကို မွေးထုတ်လိုက်ပြီဆိုရင် ဒီ **process** က **parent process** အဖြစ်နဲ့ ရပ်တည်ပြီး၊ သူ့ရဲ့ **child process** တစ်ခုကို မွေးထုတ်ပေးလိုက်ပါတယ်။ ဒီ နေရာမှာ **child process** ကို မွေးထုတ်ပေးလိုက်တဲ့ **parent process** သည် **wait state** ဖြစ်နေမှာဖြစ်ပါတယ်။

**child process** ကို မွေးထုတ်ပြီးနောက် **child process** သည် သူ့ရဲ့ အလုပ်တွေကို လုပ်ဆောင်သွားပြီး တာနဲ့ တစ်ပြိုင်တည်း သေဆုံးသွားမှာ ဖြစ်ပါတယ်။ သေဆုံးသွားပြီးနောက်

zombie ဖြစ်သွားတဲ့အခါ အလုပ်မပြီးသေးရင် parent process က နောက်ထပ် child process တစ်ခု ထပ်မံပိုလွှတ်ပါတယ်။

Child process အလုပ်တွေပြီးသွားမှသာ parent process နောင်ဆက်လက်ကျနှုနိဖော်တဲ့ process များကိုဆက်လက်လုပ်သွားတာဖြစ်ပါတယ်။

Parent Process , child process များကို PPID (parent process Id) , PID (process Id) နဲ့ ID နံပတ်များဖြင့် သတ်မှတ်ကြပါတယ်။

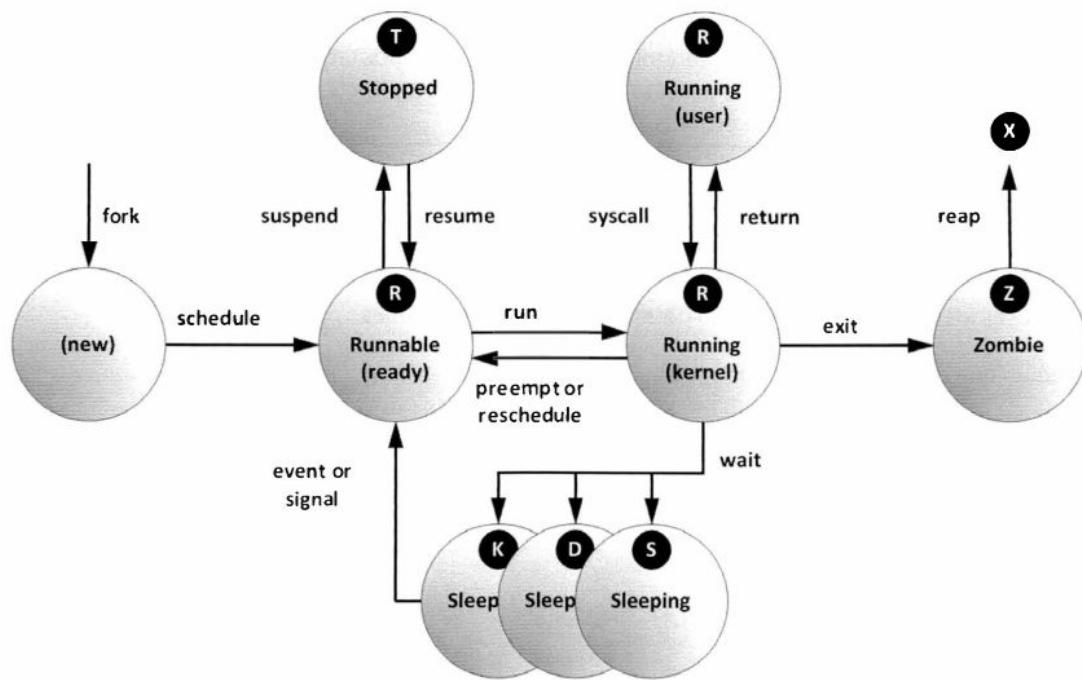


Figure 7.2: Linux process states

ဒီ process တွေဟာ service တွေနဲ့ တိုက်ရှိက်သက်ဆိုင်တဲ့အတွက် ဒီ service တွေက လိုသလို stop, start, restart, အစရှိသဖြင့် service ရဲ့ လိုအပ်ချက်အလိုက် process တွေကို ထိန်းချုပ်ကြပါတယ်။

အထက်ပါပုံမှာ လေ့လာကြည့်မယ်ဆိုရင် process တွေရဲ့ state တွေကို တွေ့ရမှာဖြစ်ပါတယ်။

Process တစ်ခု ဖြစ်ပေါ်လာတာနဲ့ R ဆိုတဲ့ runnable state ကိုရောက်ရှိပါတယ်။ memory ပေါ်ရောက်နေပြီ run ဖို့အဆင်သင့်ဆိုတဲ့ အခြေအနေဖြစ်ပါတယ်။ ဒီ အခြေအနေမှာ process ကို

အပေါက T လေးကိုတွေ့ပါ လိမ့်မယ်။ Process ကို စန ရပ်တန္ထားလို့ရပါတယ်။ suspend, ဖြစ်တယ်။ ပြီးရင် resume လုပ်လို့ရတယ်ဆိုတာတွေ့ရပါလိမ့်မယ်။

ဒီ အခြေအနေမှာ process က ready state ကနေ running (kernel) ဆိုတဲ့ state ကို သွားလို့ရပါတယ်။ ဒါဆိုရင် CPU ပေါ်မှာ processing လုပ်တာဖြစ်ပါတယ်။ Running ကနေ process တစ်ခုရဲ့၊ အလုပ်မပြီးသေးလို့ Runnable state ကို ပြန်သွားလို့ရသလို process ရဲ့ အလုပ်ပြီးသွားလို့ Z ဆိုတဲ့ zombie state ကိုသွားနိုင်ပါတယ်။ ဒါဆိုရင် process က died ဖြစ်သွားပါပြီ။ dead ဖြစ်သွားပေမယ့် zombie ရဲ့ အခြေအနေ ကိုပြန်ကြည့်နိုင်ပါသေးတယ်။ ဒါပေမယ့် reap ဖြစ်တဲ့ X အခြေအနေကို ရောက်သွားရင်တော့ ဘယ်လို့ မှ သုံးစားလို့မရတော့ပါဘူး။ အိုက process မပြီးသေးလို့ running ဖြစ်နေတဲ့ အချိန်ကနေ wait state ကို တန်းဂျောက်သွားနိုင်ပါတယ်။ S,D,K လို့သတ်မှတ်ထားတဲ့ process wait state ကို တွေ့ရပါမယ်။

S ဆိုတာ sleep ဖြစ်ပြီး။ D ဆိုတာ device တွေကပြန်လာမယ့် အခြေအနေကို စောင့်နေတဲ့ wait state ဖြစ်ပါတယ်။ K ကတော့ killable ဖြစ်ပါတယ်။ ဒီ process ကို kill နိုင်တဲ့အခြေအနေပဲဖြစ်ပါတယ်။

Process တွေမှာ background process နဲ့ foreground process တွေရှိပါတယ်။ ကျွန်တော်တို့ ခုလက်ရှိ သုံးနေတဲ့ google chrome ဆိုပါစို့ ကျွန်တော်တို့ ကိုယ်တိုင်မြင်ရတယ်။ run နေကြောင်းလဲသိတဲ့ process ကို foreground process လို့ခေါ်ပြီး ၁ ကျွန်တော်တို့မမြင်နိုင်တဲ့ နေရာမှာ ဒါမှမဟုတ် command တွေအသုံးပြုပြီး မှ ထွက်ပေါ်လာမယ့် process တွေကို background process လို့သတ်မှတ်ပါတယ်။

## Killing Process

ကျွန်တော်တို့ မလိုချင်တဲ့ process တွေကို kill ချင်တဲ့အခါ သူကို ခိုင်းစေတဲ့ service တွေကို stop မလုပ်ပဲ kill နိုင်ပါတယ်။ ဘာလိုလဲတစ်ချို့ process တွေရဲ့ service အမျိုးအစားက static service တွေဖြစ်တဲ့ အခါ ကျွန်တော်တို့ service ကို control မလုပ်နိုင်ပါဘူး။ ဒါကြောင့် process တွေကို kill လို့ရအောင် ပြုလုပ်ထားခြင်းဖြစ်ပါတယ်။

အိုက process ကို kill တဲ့အခါ ကျွန်တော်တို့ အပြင်မှာ လူသတ်သလိုပဲဖျာ စတာ စတာ၊ တကယ်ပါ၊ လူတစ်ယောက်ကို သတ်တဲ့အခါပျာ၊ လူသတ်နည်းအမျိုးမျိုးရှိတယ်။

မသေမရှင်သတ်မလား တစ်ခါတည်းသေအောင် သတ်မလား၊ င့် ၅ ရက်လောက်မှ သေအောင် သတ်မှာလား၊ စသဖြင့် သတ်နည်းတွေရှိတယ်ဗျာ ဒီလိုပဲ။

Process တွေမှာ လည်း kill တဲ့ ပုံစံရှိပါတယ်။ ဒီ လို kill တာကို signal နဲ့ kill တယ်လိုသတ်မှတ်ပါတယ်။

အောက်ပုံမှာ signal နံပတ်တွေ signal တွေကို အသုံးပြုစိအတွက် short form တွေရှိပါတယ်။

#### Fundamental process management signals

Signal number	Short name	Definition	Purpose
1	HUP	Hangup	Used to report termination of the controlling process of a terminal. Also used to request process reinitialization (configuration reload) without termination.
2	INT	Keyboard interrupt	Causes program termination. Can be blocked or handled. Sent by typing <b>INTR</b> character ( <b>Ctrl-c</b> ).
3	QUIT	Keyboard quit	Similar to <b>SIGINT</b> , but also produces a process dump at termination. Sent by typing <b>QUIT</b> character ( <b>Ctrl-\</b> ).
9	KILL	Kill, unblockable	Causes abrupt program termination. Cannot be blocked, ignored, or handled; always fatal.
15 <i>default</i>	TERM	Terminate	Causes program termination. Unlike <b>SIGKILL</b> , can be blocked, ignored, or handled. The polite way to ask a program to terminate; allows self-cleanup.
18	CONT	Continue	Sent to a process to resume if stopped. Cannot be blocked. Even if handled, always resumes the process.
19	STOP	Stop, unblockable	Suspends the process. Cannot be blocked or handled.
20	TSTP	Keyboard stop	Unlike <b>SIGSTOP</b> , can be blocked, ignored, or handled. Sent by typing <b>SUSP</b> character ( <b>Ctrl-z</b> ).

- ID 1 << ဒီ ကောင်ကတော့ reload လုပ်တာမျိုးတွေ process ကို restart ချုတာမျိုးတွေကို လုပ်နိုင်ပါတယ်။

- ID 2 << ဒီကောင်က process တွေ ကို terminate လုပ်နိုင်ပါတယ်။ CTRI C နှင့်လိုက်ရင် သတ်လိုက်သလိုမျိုးပဲ SIGINT signal ထိုလိုက်တာဖြစ်ပါတယ်။
- ID 3 ကတော့ Ctrl + C နှင့်ပြီးသတ်တာနဲ့တူပါတယ်။ ဒီကောင်က လည်း ID 3 နဲ့တူတူပါပဲ ဒါပေမယ့် ဒီကောင်ကဘာလုပ်ပေးသလဲဆိုတော့ termination မှာ dump လုပ်ပေးပါတယ်။ dump လုပ်ပေးတယ်ဆိုတာ termination အချက်အလက်တွေရမှာပေါ့။
- 9 SIGKILL ကတော့ ဘာပဲဖြစ်ဖြစ် process ကို Kill မှာပါ ဘယ်လိုအခြေအနေပဲ ရောက်နေနေ Kill နိုင်ပါတယ်။ kill signal ထဲမှာ အကြမ်းဆုံးဖြစ်ပါတယ်။
- 18 sigcont တွဲသုံးလိုက်မယ်ဆိုရင် kill command ကပြောင်းပြန်ဖြစ်သွားမယ်။ process ကို continuous လုပ်ပြီး အသက်ပြန်သွင်းတာဖြစ်ပါတယ်။
- Id 15 sigkill နဲ့ terminate လုပ်ပေးတာပါပဲ default ပေါ့။ ရိုရိုး kill နဲ့ pid ကို kill လိုက်မယ်။ signal မထည့်ဘူးဆိုရင် ဒီ signal နံပတ် 15 နဲ့ kill ပေးပါတယ်။ ဒါမယ့် ဒီကောင်က sigkill နဲ့ မတူဘူး process တွေကို kill တဲ့နေရာမှာ block တွေ ignore တွေ handle ဖြစ်နိုင်တယ်။ program ကို အေးအေးလေးပဲပိတ်လိုက်ဖို့ ပြော တာဖြစ်ပါတယ်။ အဥ္တတော့ သူပိတ်လိုက်တဲ့ program က အကြောင်းတစ်ခုရုရှုကြောင့် မပိတ်နိုင်ဘူးဆိုရင် သူဘာမှမတက်နိုင်ပါဘူး။
- ID 19 ကတော့ process ကို ခန့် stop တာဖြစ်ပါတယ်။
- ID 20 ကတော့ 19 ထက် ပိုမိုမြဲပါတယ်။ ဒါပေမယ့် ဒီကောင်ကလည်း Stop လုပ်ပေးတာပဲ။ process တစ်ခု က အခြေအနေတစ်ခု ကြောင့် stop လုပ်လို့မရဘူးဆိုရင် သူ မတက်နိုင်ပါဘူး။ block, handle တို့ ဖြစ်နိုင်ပါတယ်။

Process တွေကို kill ဖို့ signal တွေက အခြေခံအားဖြင့် အများဆုံး အထက်ပါ signal များကို သုံးကြပါတယ်။ ဒီ ထက်များတဲ့ signal တွေကို လည်း **kill -l** ဆိုတဲ့ command နဲ့ ကြည့်ပြီး process များကို စိတ်တိုင်းကျ စီမံနိုင်ပါတယ်။

Process တွေကို Linux ပေါ်မှာ Monitoring လုပ်ဖို့အတွက် **top (table of process)** လည်းရှိပါသေးတယ်။

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
3412	cops	20	0	5397072	710972	456528	S	7.3	8.8	16:16.62	chrome
3176	cops	20	0	1699348	448060	330024	S	1.3	5.6	7:06.17	chrome
3702	cops	20	0	4823488	330104	109924	S	0.0	4.1	26:43.10	chrome
2364	cops	20	0	1345948	329472	46508	S	0.0	4.1	0:16.32	snap-store
4801	cops	20	0	4780336	281832	96128	S	0.0	3.5	1:40.56	chrome
2143	cops	20	0	4902876	262700	109120	S	18.9	3.3	10:58.92	gnome-shell
4831	cops	20	0	4778192	245184	92112	S	0.0	3.0	1:58.21	chrome
11170	cops	20	0	1249588	203868	50412	S	0.0	2.5	0:34.94	evince
3263	cops	20	0	642980	174840	78304	S	0.0	2.2	13:58.93	chrome
1981	cops	20	0	1754796	118532	68412	S	24.6	1.5	8:28.18	Xorg
3268	cops	20	0	374400	102856	67684	S	1.7	1.3	3:26.77	chrome
3770	cops	20	0	4598368	92308	71704	S	0.0	1.1	0:03.45	chrome
11133	cops	20	0	1074356	85256	47632	S	0.0	1.1	0:02.96	nautilus
11542	cops	20	0	827516	64960	40364	S	1.0	0.8	0:00.96	x-terminal-emul
3221	cops	20	0	276408	63680	52228	S	0.0	0.8	0:00.23	chrome
3222	cops	20	0	276408	63604	52120	S	0.0	0.8	0:00.16	chrome
3603	cops	20	0	837652	61056	49896	S	1.3	0.8	1:19.32	chrome
2315	cops	20	0	633244	58924	44832	S	0.0	0.7	0:00.83	evolution-alarm
6017	cops	20	0	853512	54848	43244	S	0.0	0.7	0:00.67	gnome-calendar

အထက်ပါပို့ဘာ top ဆိုတဲ့ command ကိုအသုံးပြုပြီး process တွေကို အချိန်နဲ့  
တစ်ပြီးညီ real time ဖြစ်စွာ monitor လုပ်နေတာဖြစ်ပါတယ်။

## Managing Linux Priority Process

Linux ပေါ်မှာ process တစ်ခုကို cpu များများသုံးမှာလား နဲ့ သုံးမှာလားဆိုတာကို Adjust လုပ်လိုဂုဏ်ပါတယ်။ software developer တစ်ယောက်ကသူရေးလိုက်တဲ့ software တစ်ခုသည် system ပေါ်မှာ သူ့ software ကို memory ပမာန ဘယ်လောက်စားရမလဲ cpu ပမာန ဘယ်လောက်စားရမလဲဆိုတာကို ရေးထားပြီးသားပါပြီးသား၊ သတ်မှတ်ပြီးသားဖြစ်ပါတယ်။

အားsoftware တွေက ကျွန်တော်တို့စက်ထဲကို install လုပ်လိုက်ပါဆိုရင် သူသတ်မှတ်ထားတဲ့ cpu, mem စားသုံးမှုပမာနကို customize ပြန်ပြင်ပြီး ကျွန်တော်တို့ Linux ပေါ်မှာ အသုံးပြုနိုင်မှာဖြစ်ပါတယ်။ ဒါကို nice လုပ်တယ်၊ renice လုပ်တယ်လို့ခေါ်ပါတယ်။

ဒါ နေရာမှာ cpu များများပေးစားမယ်ဆိုရင် priority တန်ဖိုးကို မြင့်ပေးရမယ်။ နဲ့နဲ့ပဲ စားဆိုရင် priority တန်ဖိုးကို နိမ့်သွားပေးရမှာဖြစ်ပါတယ်။ ဒါဆိုရင် priority တန်ဖိုးတွေကို ဘာနဲ့ control လုပ်သလဲဆိုတော့ ခုနာက ပြောတဲ့ nice value နဲ့ control လုပ်ပါတယ်။ nice value နဲ့ priority တန်ဖိုးနဲ့ ဘယ်လိုအချို့ကျွဲသလဲဆိုတော့ ပြောင်းပြန်အချို့ကျွဲတယ်။ ဒါ

နေရာမှာ priority တန်ဖိုးကို မြင့်ချင်တယ်။ cpu ကို များများစားစေချင်တယ်ဆိုရင် nice value တန်ဖိုးကို လျော့ချပေးရတာဖြစ်ပါတယ်။

Priority တန်ဖိုးနှမ်ချင်တယ်။ cpu နဲ့ ပဲစားစေချင်တယ်ဆိုရင် nice value တန်ဖိုးကို တိုးပေးရမှာဖြစ်ပါတယ်။ nice value တန်ဖိုးတွေက **-20** ကနေ **+19** ထိရှိပါတယ်။ အားတော့ ကြားထဲက **0** ပါထည့်ပါး ရေတွက်မယ်ဆိုရင် nice value တန်ဖိုး သည် priority တန်ဖိုးကိုပေးလို့ရတာ အဆင့်ပေါင်း **40** ရှိပါတယ်။



Figure 5.1: Nice levels and how they are reported by top

ဒီ တန်ဖိုးတွေကို တိုက်ရှိက်သတ်မှတ်ပေးရမှာဖြစ်ပါတယ်။ process အသစ်တစ်ခုစတင်မယ် ဆိုရင် စတင်မယ့် process ကိုတစ်ခါတည်း nice value သတ်မှတ်ပေးချင်တယ်။ ဆိုရင်တော့ command က nice ဆိုတဲ့ command ကိုအသုံးပြုနိုင်ပါတယ်။

ရှိပြီသား process ကို ပဲ nice value ပြန်လည်သတ်မှတ်ပေးချင်တယ်ဆိုရင်တော့ renice ဆိုတဲ့ command ကိုအသုံးပြုရမှာဖြစ်ပါတယ်။

ခုနက priority နဲ့ nice value ပြောင်းပြန်အချိုးကျတယ်လို့ပြောခဲ့တယ်။ အားတော့ ဒီနေရာမှာ nice value က လျော့ပြီးပေးမယ် - အနုတ်ဘက်ကို ပေးလာမယ်ဆိုရင် priority တန်ဖိုးမြင့်လာပြီး process က cpu စားနှုန်းတိုးလာမှာဖြစ်ပါတယ်။ nice value ကို တိုးပြီးပေးမယ် + ဘက်ကို တိုးလာပါး ပေးမယ်ဆိုရင် priority တန်ဖိုးက နိမ့်လာပြီး process က cpu စားနှုန်းလျော့သွားမှာဖြစ်ပါတယ်။ အားတော့ process ကို တိုးလိုက်လျော့လိုက် တစ်ဆင့်ချင်းစီလုပ်လို့ရပါတယ်။

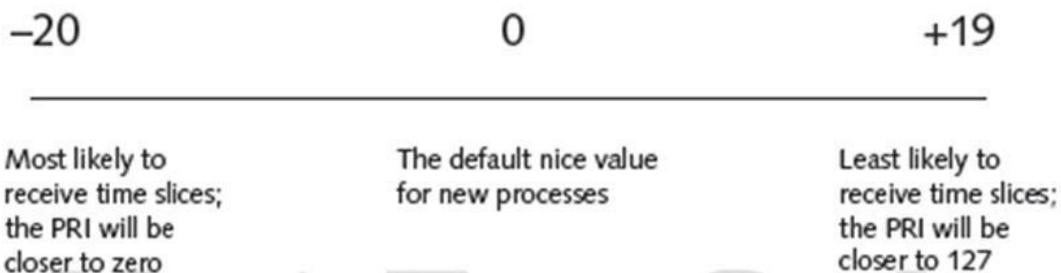


Figure 9-4: The nice value scale

**nice value တန်ဖိုး** - အနုတ်ဘက်ကို တစ်ဆင့်ပြီးတစ်ဆင့် ပေးလာချင်တယ် cpu များများ စားစေချင်တယ်ဆိုရင် normal user အနေနဲ့ သတ်မှတ်ပေးလို့မရပါဘူး။ root user အနေနဲ့ပဲသတ်မှတ်ပေးလို့ရပါတယ်။

တကယ်လို့ nice value တန်ဖိုး အပေါင်းဘက်ကို တစ်ဆင့်ပြီးတစ်ဆင့်တိုးသွားမယ်။ process တစ်ခုကို cpu လျော့စားစေချင်တယ် ဆိုရင်တော့ normal user အနေနဲ့ လုပ်ဆောင်နိုင်ပါတယ်။

root user က nice value ကို တိုးလိုလည်းရတယ်။ လျော့လိုလဲရပါတယ်။ normal user ကတော့ nice value ကို တိုးလိုပဲရတယ်။ nice value တိုးတာသည် priority ကို နိမ့်ပေးတာဖြစ်ပါတယ်။ priority နိမ့်တော့ cpu ကိုလျော့စားတာပေါ့။ ဒါပါပဲ။

The screenshot shows a Facebook group page for "Linux Digger". The group has 175 members and is categorized as a public group. The cover photo features a black background with white text forming the words "DEVNULL". Below the cover, there's a "Create a public post..." button and three action buttons: "Room", "Request Shift Cover", and "Photo/Video". The navigation bar includes links for "About", "Discussion" (which is currently selected), "Rooms", "Members", "Events", and "Media". A search bar and a "..." button are also present. On the right side, there's an "About" section with a description: "Deep And Dive to the Linux Black Hole". It offers two visibility options: "Public" (selected) and "Visible", both with explanatory text about who can see the group.