

CHAPITRE 8 : Gestion des processus par un système Linux

Mohammed SABER

Département Électronique, Informatique et Télécommunications
École Nationale des Sciences Appliquées "ENSA"
Université Mohammed Premier OUJDA

Année Universitaire : 2018-2019

Plan de chapitre

- 1 Introduction
- 2 Priorité des processus et ordonnancement
- 3 Caractéristiques des processus Linux(Unix)
- 4 États des processus
- 5 Modes d'exécution des processus Linux(Unix)
- 6 Commandes de gestion de processus Linux(Unix)

- 1 Introduction
- 2 Priorité des processus et ordonnancement
- 3 Caractéristiques des processus Linux(Unix)
- 4 États des processus
- 5 Modes d'exécution des processus Linux(Unix)
- 6 Commandes de gestion de processus Linux(Unix)

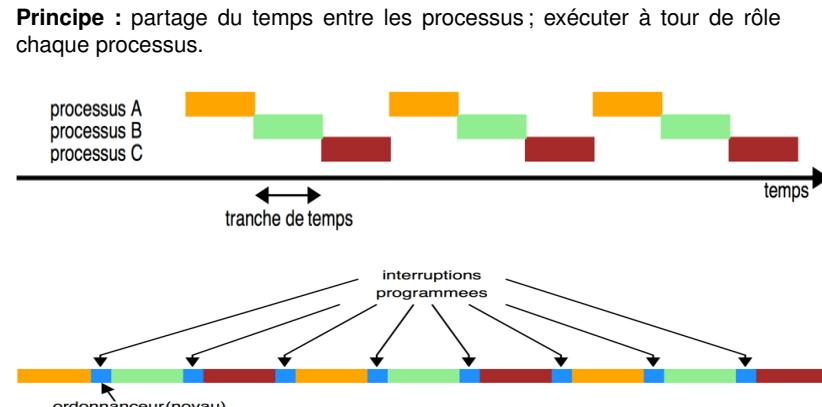
- Un processus est un programme ou une commande en cours d'exécution sur un système.
- Linux peut exécuter plusieurs processus en même temps. C'est pourquoi on qualifie Linux de système **multi-tâche**.
- Ce processus est une entité qui, de sa création à sa mort, est identifiée par une valeur numérique, le **PID** (Process IDentifier). Le noyau utilise une table des processus pour la gestion des tâches.
- Deux primitives du noyau Linux permettent de créer des processus : **fork** et **exec**. La première permet à un processus de créer un clone de lui-même, la seconde servira à ce clone pour exécuter le code d'un autre programme à sa place. Il en résulte une affiliation entre processus ; on parle alors de processus fils et de processus pères.
- Tout processus a donc obligatoirement un père , sauf le premier processus du système : **init**. Celui-ci est donc l'ancêtre de tous les processus du système et son **PID est 1**.
- Un processus démon est un processus qui s'exécute en permanence sur le système ; son code boucle à l'infini afin d'attendre et d'être prêt lorsqu'un utilisateur sollicite le service auquel il est attaché.
- Les noms des démons finissent généralement par un d; ainsi on trouvera généralement les démons **httpd** (serveur Web Apache) et **syslogd** (journalisation des messages) sur un système Linux.

Plan de chapitre

- 1 Introduction
- 2 Priorité des processus et ordonnancement
- 3 Caractéristiques des processus Linux(Unix)
- 4 États des processus
- 5 Modes d'exécution des processus Linux(Unix)
- 6 Commandes de gestion de processus Linux(Unix)

Priorité des processus et ordonnancement

Partage du temps



- L'**ordonnanceur** est le programme du noyau qui gère l'ordre d'exécution des processus.
- Il utilise un **timer** pour interrompre l'exécution de chaque processus à la fin de sa tranche de temps.

Priorité des processus et ordonnancement

- Linux exécute donc plusieurs processus sur la même machine.
- La méthode consiste alors affecter une petite plage de temps (appelée quantum) successivement à chaque processus, pendant laquelle le processus peut utiliser le CPU.
- La partie du module chargée de l'affectation de ces quantum est appelée **ordonnanceur**.
- Il existe plusieurs algorithmes d'ordonnancement des processus :
 - **Nice** : signifie gentil on parle donc aussi de valeurs de gentillesse comme niveau de priorité.
 - **Renice** : cette commande permet toujours de modifier les valeurs de gentillesse, cette fois sur des processus en cours d'exécution.
- Le niveau de propriété par défaut d'un processus utilisateur est **0**. celui-ci peut varier de **-20** (le plus propriétaire) à **19** (le moins prioritaire). Par la commande **nice** en modifiant la valeur de gentillesse par défaut.

Priorité des processus et ordonnancement

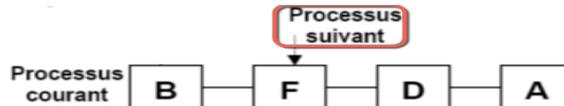
Ordonnancement des processus

- Plusieurs processus sont prêts à être exécutés.
- Le noyau de système d'exploitation doit faire un choix (algorithme d'ordonnancement) :
 - **Équité** : chaque processus doit avoir du temps processeur ;
 - **Efficacité** : le processeur doit être utilisé à 100% ;
 - **Temps de réponse** : l'utilisateur devant sa machine ne doit pas trop attendre ;
 - **Temps d'exécution** : une séquence d'instructions ne doit pas trop durer ;
 - **Rendement** : il faut faire le plus de choses en une heure ;
- Ordonnancement **sans réquisition** : un processus est exécuté jusqu'à la fin (Inefficace et dangereux (ex : exécution d'une boucle sans fin...)) ;
- Ordonnancement **avec réquisition** :
 - À chaque signal d'horloge, le noyau de système d'exploitation reprend la main, décide si le processus courant a consommé son quota de temps machine et alloue éventuellement le processeur à un autre processus ;
 - Il existe de nombreux algorithmes d'ordonnancement avec réquisition ;

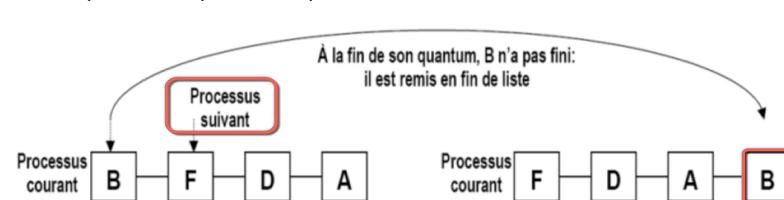
- Chaque processus possède un quantum d'exécution.



- Si le processus a fini dans cet intervalle : **au suivant !**.



- S'il n'a pas fini : le processus passe en **fin de liste et au suivant !**.



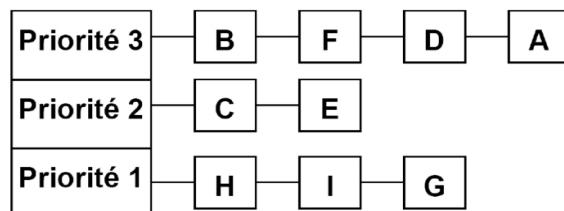
Problème = réglage du quantum

- Quantum trop petit** / commutation (= temps de passage d'un processus à l'autre) : le processeur passe son temps à commuter ;
- Quantum trop grand** : augmentation du temps de réponse d'une commande (même simple) ;
- Réglage correct** : Quantum / commutation = 5 ;

Inconvénient du tourniquet

Processus de même priorité.

- Plusieurs files d'attente plus ou moins prioritaires ;
- La priorité d'un processus décroît au cours du temps pour ne pas bloquer les autres files d'attente ;



- 1 Introduction
- 2 Priorité des processus et ordonnancement
- 3 Caractéristiques des processus Linux(Unix)
- 4 États des processus
- 5 Modes d'exécution des processus Linux(Unix)
- 6 Commandes de gestion de processus Linux(Unix)

- Un processus comprend :
 - Une mémoire qui lui est propre (mémoire de stockage) ;
 - Un contexte d'exécution (état instantané) :
 - Pile (en mémoire) ;
 - Registres du processeur ;
- Processus fils / père (Exemple : Le shell est un processus comme les autres) ;
- Chaque commande exécutée correspond à la création d'un processus « fils » par rapport au shell (« père ») ;
- Sous Linux (Unix), chaque processus est identifié par :
 - PID (Processus Identifier) ;
 - PPID (Parent Processus Identifier) ;
- Deux types de processus :
 - Processus systèmes (**daemons**) : exécution de tâches générales, souvent contrôlées par root ;
 - Processus utilisateurs ;

- 1 Introduction
- 2 Priorité des processus et ordonnancement
- 3 Caractéristiques des processus Linux(Unix)
- 4 États des processus
- 5 Modes d'exécution des processus Linux(Unix)
- 6 Commandes de gestion de processus Linux(Unix)

- Le mécanisme de création des processus est le **fork()** :
 - Un processus nouvellement créé est une copie exacte du processus qui l'a créé excepté la valeur renournée par **fork()** ;
- Organisation arborescente des processus ;
- Un processus interagit avec l'extérieur à l'aide des E/S standards ;
- Un processus peut communiquer avec un ou plusieurs processus à l'aide de tubes de communication ;
- Un processus peut émettre ou recevoir des signaux ;

Un processus devient tour à tour actif/inactif tout au long de sa vie ; il passe donc par différents états, notamment :

En exécution (élu)

Le processus est exécuté par le processeur.

Prêt

Le processus pourrait être exécuté mais un autre processus est actuellement en cours d'exécution.

Suspendu

Le processus est en attente d'une ressource, par exemple il attend la fin d'une E/S.

Un processus devient tour à tour actif/inactif tout au long de sa vie ; il passe donc par différents états, notamment (suite) :

Stoppé

Le processus a été suspendu par une intervention extérieure.

Zombie

Le processus a terminé son exécution mais il est toujours référencé dans le système.

System Monitor								
Processes								
Process Name	ID	User	Status	% CPU	CPU Time	Started	Memory	Virtual Memory
gnome-system-monitor	11873	pa	Running	4	1:20.17	Today 06:00 PM	7.7 MIB	361.0 MIB
compiz	1803	pa	Sleeping	4	2:58:15	Sat 11:49 AM	372.5 MIB	1.0 Gib
nautilus	3598	pa	Sleeping	0	1:04:58	Sat 08:30 AM	68.8 MIB	1.5 Gib
glibber-service	2054	pa	Sleeping	0	2:50.75	Sat 11:50 AM	1.65 MIB	385.7 MIB
glibber-service	2055	pa	Sleeping	0	2:52.19	Sat 11:50 AM	10.8 MIB	385.7 MIB
glibber-service	2054	pa	Sleeping	0	2:49.69	Sat 11:50 AM	10.4 MIB	385.2 MIB
glibber-service	2053	pa	Sleeping	0	2:51.40	Sat 11:50 AM	10.6 MIB	385.3 MIB
empathy	10812	pa	Sleeping	0	0:24.31	Today 11:26 AM	25.5 MIB	672.5 MIB
thunderbird-bin	6065	pa	Sleeping	0	21:06.27	Yesterday 11:01 A	154.8 MIB	814.2 MIB
nautilus	1816	pa	Sleeping	0	3:24.80	Sat 11:49 AM	82.2 MIB	866.7 MIB
dropbox	1860	pa	Sleeping	0	1:04.21	Sat 11:49 AM	36.0 MIB	662.8 MIB
evince	10949	pa	Sleeping	0	0:17.49	Today 05:48 PM	27.0 MIB	520.6 MIB
evince	10173	pa	Sleeping	0	0:06.50	Today 04:00 PM	56.1 MIB	520.4 MIB
glibber-service	1930	pa	Sleeping	0	3:24.17	Sat 11:50 AM	16.8 MIB	486.7 MIB

Un processus peut être dans 3 états possibles (modèle simplifié) :

En exécution (élu)

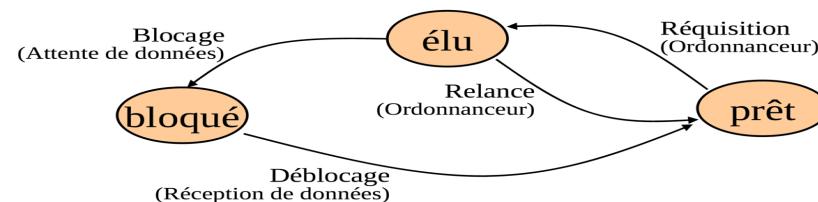
processus OK, processeur OK.

Prêt

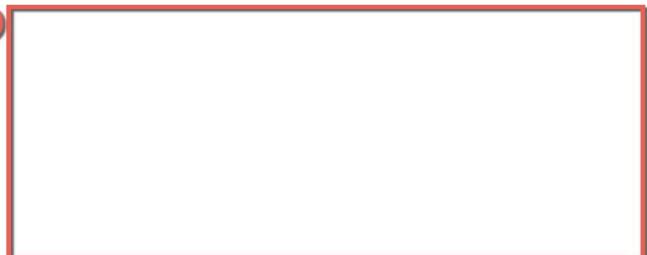
processus OK, processeur non OK(occupé).

bloqué (attendant un événement extérieur pour continuer) (Suspendu)

processus non OK, même si processeur OK.



Mémoire RAM

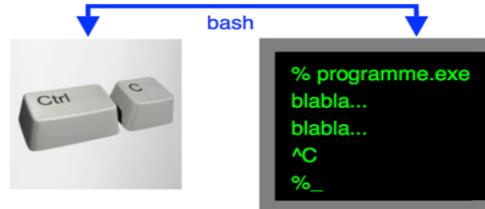


Mémoire SWAP



- 1 Introduction
- 2 Priorité des processus et ordonnancement
- 3 Caractéristiques des processus Linux(Unix)
- 4 États des processus
- 5 Modes d'exécution des processus Linux(Unix)
- 6 Commandes de gestion de processus Linux(Unix)

- Lorsqu'une commande est en train de s'exécuter, le **shell** ne rend pas la main et attend que la commande se termine (correctement ou incorrectement).
- On parle ainsi d'**avant plan** (en anglais **foreground**).
- Pour interrompre prématûrement une commande : taper sur la touche **Ctrl** et aussi sur la touche **C** du clavier. Cela tue la commande qui tourne.
- On notera l'appui sur ces 2 touches par « **Ctrl-C** ».



- Si l'on veut lancer une commande et récupérer la main tout de suite, avant même que la commande ait fini de s'exécuter,
- Il faut lancer la commande par : **commande &**
- On parle ainsi d'**arrière plan** (en anglais **background**).
- Le signe « & » signifie de lancer en tâche de fond, en background la commande. Sans ce signe, la commande est lancée en premier plan, en **foreground**.
- Pas d'interaction avec l'utilisateur.

Pour passer en **background** une commande lancée en **foreground** :

- Figer la commande en cours : Taper sur la touche « **Control** » et aussi sur la touche « **Z** », soit « **Ctrl-Z** ».

```
% commande
^Z.
[1]+ Stopped commande
```

- Indiquer de l'exécuter dorénavant en **background** : Taper la commande « **bg** » (en anglais **background**).

```
% bg
[1]+ commande &
```

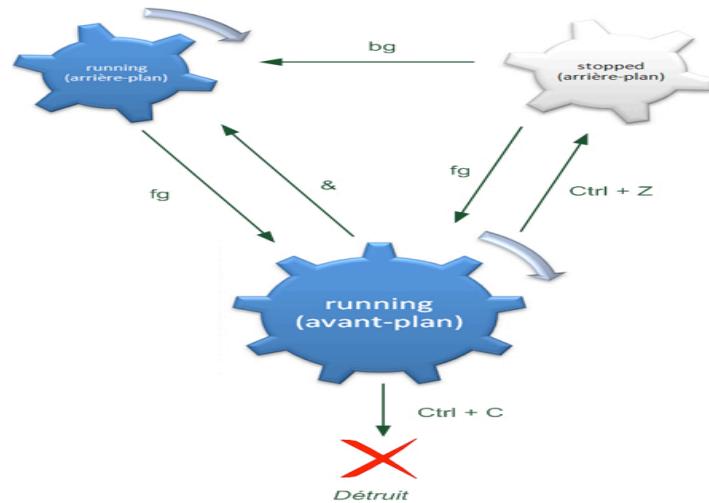
Pour passer en **foreground** une commande lancée en **background** :

- La commande est lancée. On a la main :

```
root@Nom_machine :#commande &
```

- Indiquer de l'exécuter dorénavant en **foreground**. Taper la commande « **fg** » (en anglais **foreground**) :

```
root@Nom_machine :#fg
```



- 1 Introduction
- 2 Priorité des processus et ordonnancement
- 3 Caractéristiques des processus Linux(Unix)
- 4 États des processus
- 5 Modes d'exécution des processus Linux(Unix)
- 6 Commandes de gestion de processus Linux(Unix)

Différé (at)

- Le fichier de commandes est exécuté à une date fixée ;
- Pas d'interaction avec l'utilisateur ;
- Les résultats peuvent être envoyés à l'utilisateur par e-mail ;

Cyclique (crontab)

- Un fichier spécial contient les tâches à exécuter régulièrement ;
- Un daemon scrute sans arrêt ce fichier ;

File d'attente (batch)

- La commande est placée dans une file d'attente ;
- La file d'attente est vidée en fonction de la charge du processeur ;
- Les résultats peuvent être envoyés à l'utilisateur par e-mail ;

- Pour connaître la liste des commandes en background : jobs

```
root@Nom_machine :#jobs
```

% jobs		
[1]	Running	commande1 &
[2]	Running	commande2 &
[3]	Running	commande3 &
[4] -	Running	commande4 &
[5] +	Running	commande5 &

- On peut avoir plusieurs commandes en background.
- D'où une numérotation des commandes qui sont affichées.
- Ce numéro peut être repris dans les commandes «fg» et «bg» ainsi que dans la commande suivante, «kill».

Syntaxes générales :

- Syntaxe «`fg %n`»;
- Syntaxe «`bg %n`»;
- Syntaxe «`kill %n`»;
- Avec **n** un des numéros trouvés grâce à «`jobs`».

```
% jobs
[1]  Running      commande1 &
[2]  Running      commande2 &
[3]  Running      commande3 &
[4]- Running      commande4 &
[5]+ Running      commande5 &

% kill %3
[3] Terminated   commande3 &

% jobs
[1]  Running      commande1 &
[2]  Running      commande2 &
[4]- Running      commande4 &
[5]+ Running      commande5 &
```

Commande ps

- Les commandes «`fg`», «`bg`», «`jobs`» ne fonctionnent que sur les processus lancés par le shell courant.
- Les commandes vues précédemment peuvent donc être inutilisables si vous avez quitté votre shell.
- La commande «`ps`» plus générale permet d'avoir des informations sur tous les processus de la machine.
 - Les processus associés à son terminal : «`ps`» ;
 - Tous ses processus : «`ps -x`» ;
 - Tous les processus de la machine : «`ps -ax`» ;
 - Tous les processus de la machine avec les noms de login associés : «`ps url-aux`»
- Affichage de la liste des processus :
 - `-e` : affichage de tous les processus.
 - `-f` : affichage détaillé.

UID	PID	PPID	C	STIME	TTY	TIME	COMMAND
root	1	0	0	Dec 6 ?		1:02	init
jean	319	300	0	10:30:30 ?		0:02	/usr/dt/bin/dtsession
olivier	321	319	0	10:30:34 ttyp1		0:02	csh
olivier	324	321	0	10:32:12 ttyp1		0:00	ps -ef

Commande ps

Interprétation des résultats de la commande `ps` avec les options :

```
* ps aux
USER     PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root      16  0.0  0.0      0   0 ?      S 15:27 0:00 [khungtaskd]
root      17  0.0  0.0      0   0 ?      SW 15:27 0:00 [kswapdo]
root      18  0.0  0.0      0   0 ?      SN 15:27 0:00 [khugepaged]
root      19  0.0  0.0      0   0 ?      SN 15:27 0:00 [fsnotify_mark]
root      20  0.0  0.0      0   0 ?      SW 15:27 0:00 [fsnotify_mark]
root      12  0.0  0.0      0   0 ?      S 15:27 0:00 [bd1-default]
```

- Les états définis par le champ **STAT** contiennent trois caractères. Les valeurs du premier signifiant :

PID	USER	PRI	NI	SIZE	RSS	SHARE	STAT	%CPU	%MEM	TIME	CPU	COMMAND
1	root	15	0	508	508	448	S	0.0	0.0	0:13	0	init
2	root	RT	0	0	0	0	SW	0.0	0.0	0:00	0	migration/0
3	root	RT	0	0	0	0	SW	0.0	0.0	0:00	1	migration/1

- **R(Running)** : processus en cours d'exécution.
- **T(Topped)** : processus stoppé (avec `ctrl Z` par exemple).
- **S(Sleeping)** : processus endormi, en attente de processeur.
- **Z(Zombie)** : processus terminé.

- Le deuxième caractère du champ **STAT** est positionné à **W** si le processus est déplacé en espace de pagination.
- Enfin, un **N** apparaît en troisième caractère si le processus a une valeur «`nice`» positive.

Commande ps

Les principaux champs disponibles sont (le libellé de la colonne est entre parenthèses) :

- **%cpu (%CPU)** : pourcentage de temps processeur utilisé depuis le lancement de la commande.
- **%mem(%MEM)** : pourcentage d'utilisation de la mémoire disponible.
- **comm(COMMAND)** : nom de la commande à l'origine du processus.
- **command(COMMAND)** : nom de la commande avec ses arguments à l'origine du processus.
- **time(TIME)** : temps total d'utilisation du CPU pour ce processus.
- **etime(ELAPSED)** : temps écoulé depuis le lancement du processus.
- **gid(GID)** : GID sous lequel s'exécute le processus.
- **group(GROUP)** : nom du group sous lequel s'exécute le processus.
- **Istart(STARTED)** : date de lancement du processus.
- **nice(NI)** : valeur de gentillesse ou niveau de priorité du processus.
- **pid(PID)** : identifiant du processus.
- **stat(STAT)** : état du processus.
- **tty(TT)** : nom du terminal auquel est rattaché le processus. «`?`» dans le cas d'un processus non associé à un terminal comme un démon.
- **uid(UUID)** : UID sous lequel s'exécute le processus.
- **user(USER)** : nom d'utilisateur sous lequel s'exécute le processus.

Sert à communiquer avec des processus :

- Envoie un signal à un processus, mais il faut connaître son **PID**.
- Pour tuer un processus, le signal à envoyer est SIGKILL (valeur **-9**) : `kill -9 PID`
- Demande au processus de se reconfigurer
- Passage en mode verbeux du processus
- Un utilisateur ne peut arrêter que ses processus.
- Variante par nom de programme : `killall`.
- Liste des signaux :

```
# kill -1
 1) SIGHUP      2) SIGINT      3) SIGQUIT      4) SIGILL
 5) SIGTRAP     6) SIGABRT     7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGSTKFLT
17) SIGCHLD    18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN    22) SIGTTOU     23) SIGURG      24) SIGXCPU
25) SIGXFSZ    26) SIGVTALRM   27) SIGPROF     28) SIGWINCH
29) SIGIO       30) SIGPWR      31) SIGSYS      34) SIGRTMIN
35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3 38) SIGRTMIN+4
39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8
43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12
47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14
51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10
55) SIGRTMAX-9 56) SIGRTMAX-8 57) SIGRTMAX-7 58) SIGRTMAX-6
59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3 62) SIGRTMAX-2
63) SIGRTMAX-1 64) SIGRTMAX
```

- Inconvénient de «ps» : c'est la liste des processus à un instant **t**.
- On ne pourra jamais sous Linux avoir la liste des processus en cours : le temps de chercher les processus et de faire le rapport, certains processus peuvent avoir disparu.
- Amélioration de «ps» : la commande «top».
- Son intérêt : elle affiche une liste des processus toutes les **n** secondes.

```
top - 00:31:24 up 2 min,  3 users,  load average: 2.39, 0.84, 0.30
Tasks: 118 total,  4 running, 114 sleeping,  0 stopped,  0 zombie
Cpu(s): 33.2%us, 58.1%sy,  1.3%ni,  0.0%id,  0.0%wa,  0.0%hi,  7.3%si,  0.0%st
Mem: 1026436k total, 348740k used, 677696k free, 21856k buffers
Swap: 1646620k total,   0k used, 1646620k free, 157444k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S %CPU	%MEM	TIME+	COMMAND
3861	root	20	0	2016	680	572	R 49.5	0.1	0:16.72	ping
3685	ramesh	20	0	33516	13m	9304	R 45.2	1.4	0:16.81	gnome-terminal
3127	ramesh	20	0	39304	22m	13m	S 3.3	2.2	0:01.61	gnome-panel
3872	ramesh	39	19	35956	31m	2652	R 1.0	3.2	0:00.31	apt-check
2516	root	20	0	118m	17m	8192	S 0.7	1.8	0:03.03	Xorg
3712	ramesh	20	0	2448	1176	912	R 0.3	0.1	0:00.10	top
1	root	20	0	3084	1888	564	S 0.0	0.2	0:01.42	init
2	root	15	-5	0	0	0	S 0.0	0.0	0:00.00	kthreadd
3	root	RT	-5	0	0	0	S 0.0	0.0	0:00.00	migration/0
4	root	15	-5	0	0	0	S 0.0	0.0	0:00.01	ksoftirqd/0
5	root	RT	-5	0	0	0	S 0.0	0.0	0:00.00	watchdog/0

QUESTIONS ?