Linux – Seconda Parte

Abbiamo visto cos'è Linux, la shell, la gestione dei FileSystem e la compilazione di un programma scritto in C.

Ora ci occuperemo dei processi e della memoria.

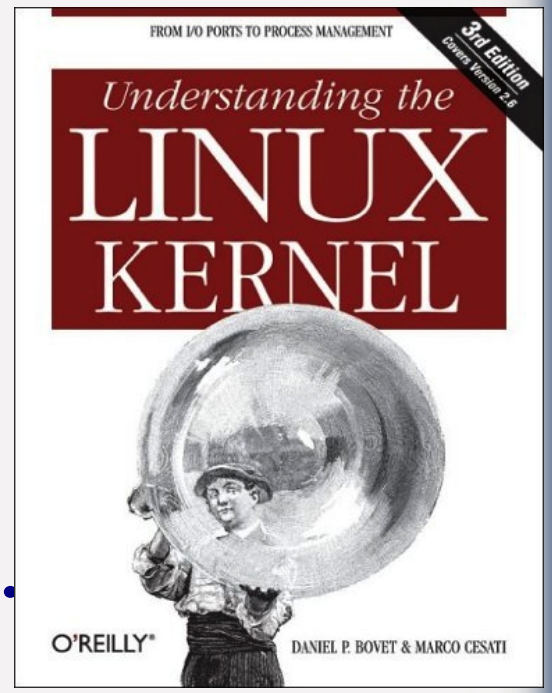
Linux – Seconda Parte

Linux - ripasso

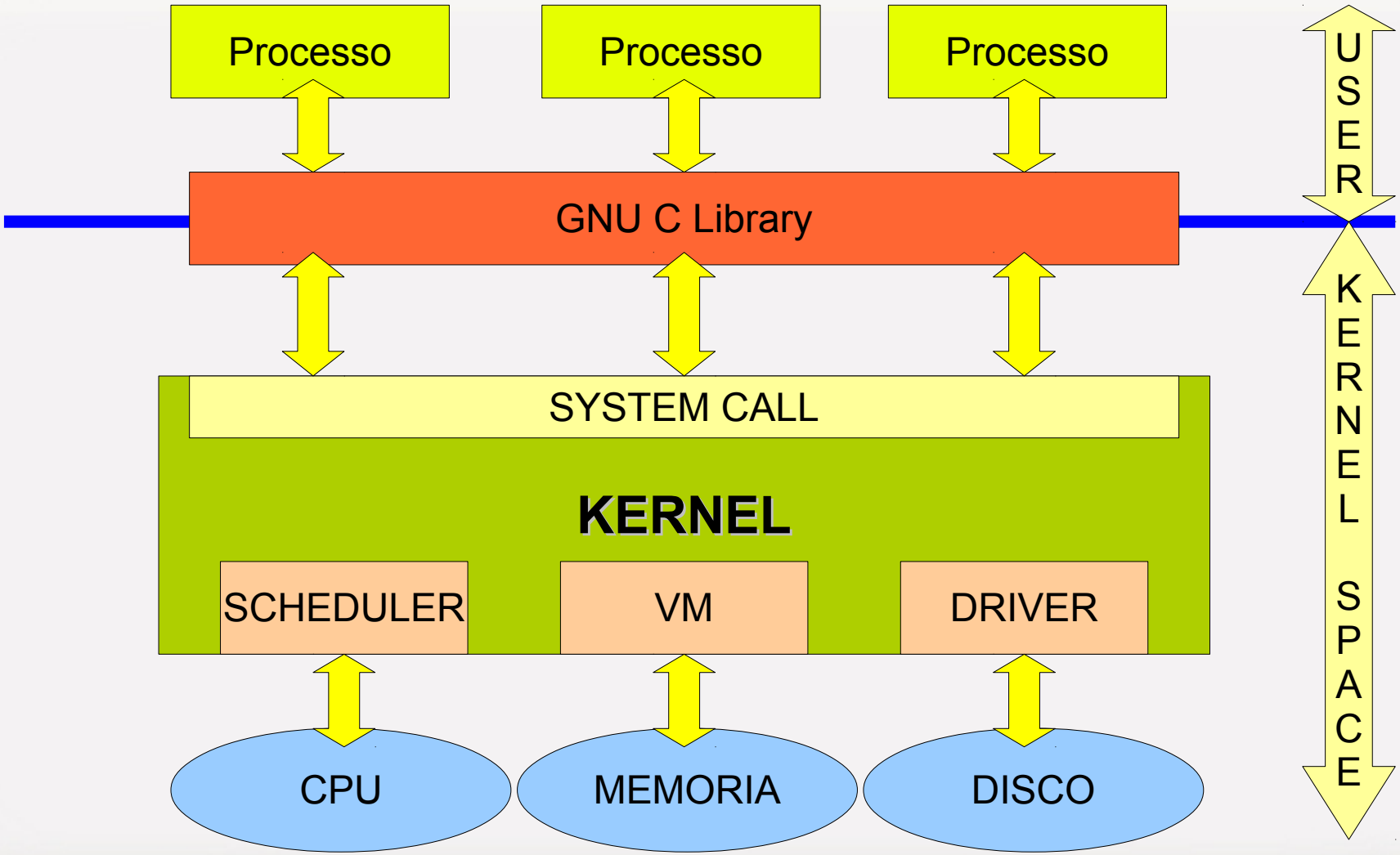
GNU/Linux è un sistema operativo multiutente e multitasking composto da un kernel LINUX e un sistema GNU (interfaccia grafica, riga di comando, compilatori, editor di testo...).

Kernel

Il cuore di un sistema operativo è il kernel. Quello di Linux si occupa solamente di una cosa: eseguire processi sfruttando le risorse hardware della macchina.



Struttura Linux



I processi

Ogni comando che si lancia, ogni servizio attivo sul sistema dà origine a uno o più processi.

Ogni processo ha un proprio PID (process ID) che lo identifica.

Ogni processo, tranne init, è generato da un altro processo di cui si definisce il PPID (Parent PID).

Si parla quindi di processo padre (parent) e processo figlio (child).

Pstree

Vediamo il risultato del comando:

$ pstree

Tutti i processi partono da init

```
giz@giz-HP-Pavilion:~$ pstree
init─┬─NetworkManager─┬─dhclient
     │                ├─dnsmasq
     │                └─2*[{NetworkManager}]
     ├─accounts-daemon───{accounts-daemon}
     ├─acpid
     ├─apache2───5*[apache2]
     ├─at-spi-bus-laun─┬─dbus-daemon
     │                 └─3*[{at-spi-bus-laun}]
     ├─at-spi2-registr───{at-spi2-registr}
     ├─atd
     ├─avahi-daemon───avahi-daemon
     ├─bamfdaemon───2*[{bamfdaemon}]
     ├─bluetoothd
     ├─chrome─┬─chrome
     │        ├─chrome-sandbox───chrome─┬─chrome─┬─2*[chrome───3*[{chrome}]]
     │        │                         │        └─chrome───6*[{chrome}]
     │        │                         └─nacl_helper_boo
     │        └─26*[{chrome}]
     ├─colord───2*[{colord}]
     ├─console-kit-dae───64*[{console-kit-dae}]
     ├─cron
     ├─cupsd───dbus
     ├─2*[dbus-daemon]
     ├─dbus-launch
     ├─dconf-service───2*[{dconf-service}]
     ├─dropbox───21*[{dropbox}]
     ├─evolution-sourc───2*[{evolution-sourc}]
     ├─gconfd-2
     ├─geoclue-master───2*[{geoclue-master}]
     ├─6*[getty]
     ├─gnome-keyring-d───6*[{gnome-keyring-d}]
     ├─gnome-terminal─┬─bash───pstree
     │                ├─gnome-pty-helpe
     │                └─4*[{gnome-terminal}]
```

Stato di un processo

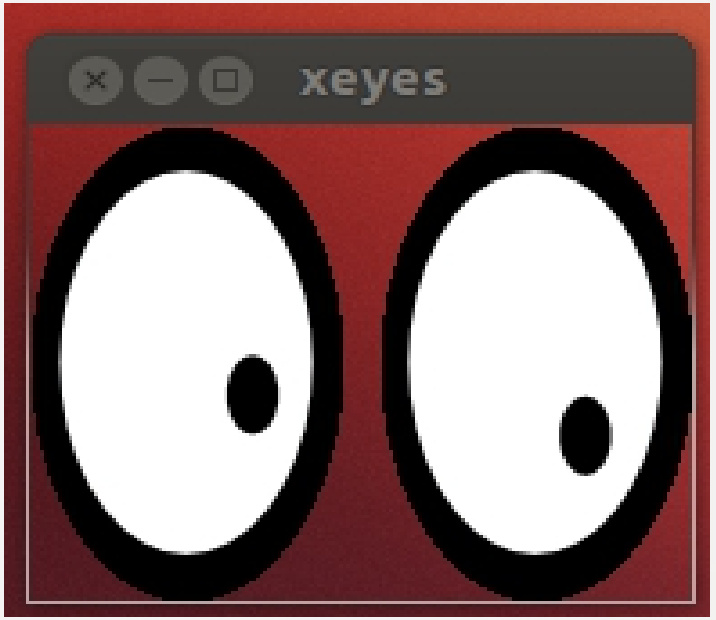
STATO	DESCRIZIONE
R - RUNNING	il processo è in esecuzione
S - SLEEPING	il processo è in attesa (input dell'utente, conclusione di altri processi ecc..)
T - SOSPESO	il processo è stato fermato da un segnale o è fermo in fase di debug
Z - ZOMBIE	il processo è morto ed aspetta che il parent chieda un codice d'uscita
D - PAUSA	il processo è in pausa non interrompibile
W - SWAPPED	il processo è temporaneamente trasferito in memoria secondaria

Esempio

Da shell lanciamo il comando:

```
$ xeyes &
```

Appariranno due occhietti che seguono i movimenti del nostro mouse.



Verificare lo stato del processo:

```
$ ps -l
```

Process Status

```
giz@giz-HP-Pavilion:~$ xeyes&
[1] 10693
giz@giz-HP-Pavilion:~$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000 10422 10413  0  80   0 -  1526 wait   pts/2    00:00:00 bash
0 S  1000 10693 10422  2  80   0 -  1690 poll_s pts/2    00:00:00 xeyes
0 R  1000 10694 10422  0  80   0 -  1242 -      pts/2    00:00:00 ps
giz@giz-HP-Pavilion:~$
```

Shell

xeyes

Comando ps

Stato

PID

PID del processo Padre

Priorità

Mem occupata

Terminale

UserID

Nice

Evento di attesa

Tempo impegno CPU

Utilizzo CPU in %

System Call

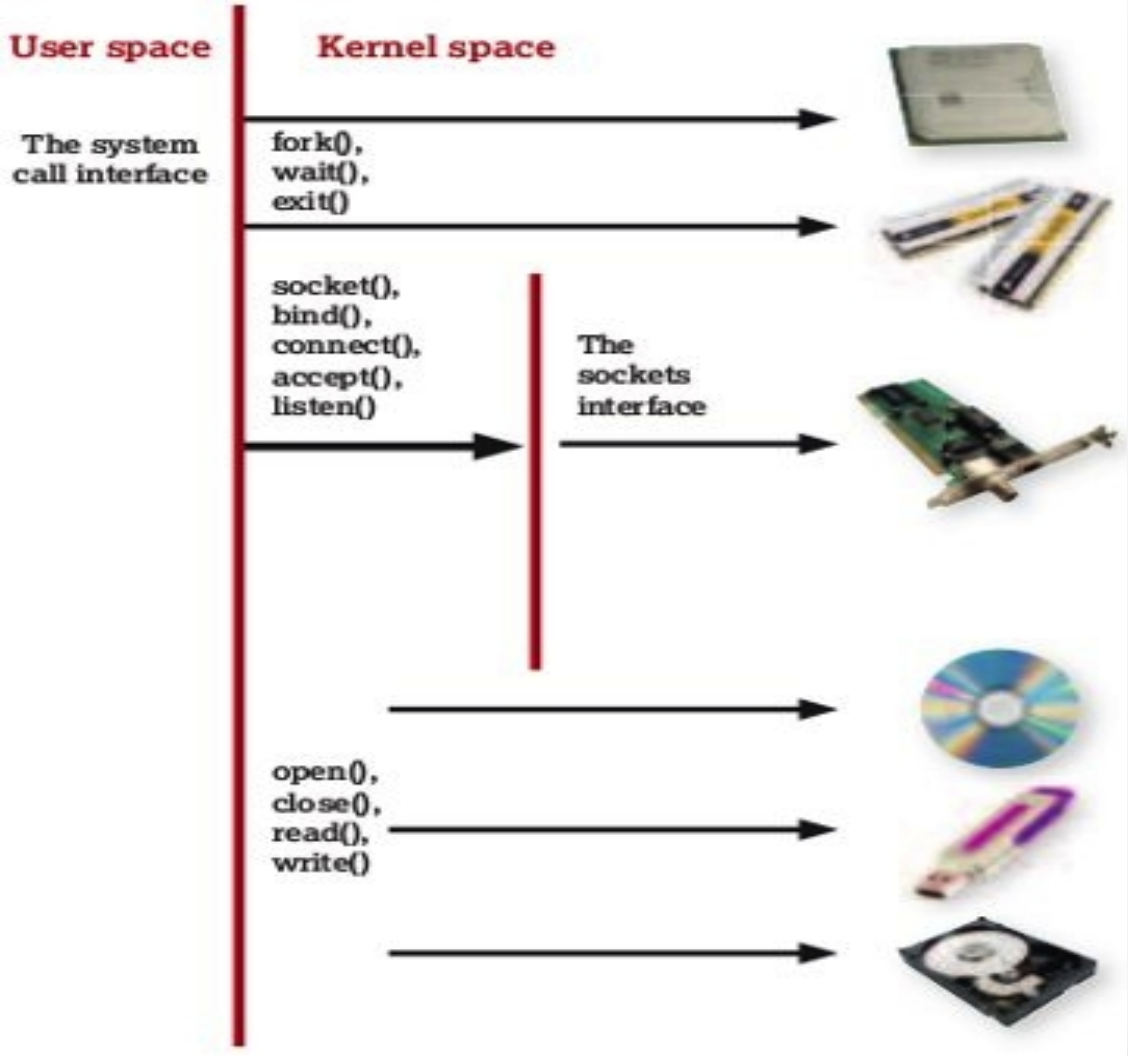
Un processo attivo può ottenere dei servizi dal kernel attraverso le System Call.

Un kernel di Linux ha tra 60 e 200 system calls.

Le categorie principali di system call sono:

- controllo dei processi/thread,
- gestione dei file e dei file system,
- gestione dei dispositivi,
- gestione delle informazioni di sistema (tipo la data),
- comunicazione di rete.

System Call


User space
Kernel space
The system call interface
fork(),
wait(),
exit()
socket(),
bind(),
connect(),
accept(),
listen()
The sockets interface
open(),
close(),
read(),
write()

System Call per gestione processi

Esempi di system call in linguaggio C per la gestione dei processi:

Creazione e terminazione: fork(), exec(), wait(), exit()

Signal ai processi: kill() (il comando kill)

Controllo dei processi: ptrace(), nice(), sleep()

Signal

Possiamo modificare lo stato di un processo. Un comando per inviare un signal ad un processo è il comando kill.

Con comando kill -l si possono vedere tutti i SIGNAL che è possibile inviare ad un processo:

```
 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
```

Cambiare stato ad un processo

Di tutti i SIGNAL nella lista, si può specificare il numero, la dicitura SIGKILL o la dicitura omettendo 'SIG'.

Ad esempio: kill -9, kill -KILL, kill -SIGKILL sono perfettamente uguali.

Con il comando ps -l verifichiamo il PID del nostro processo xeyes.

```
giz@giz-NEC-VERSA-M160:~$ xeyes &
[1] 3526
giz@giz-NEC-VERSA-M160:~$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000  2455  2446  0  80   0 -  1645 wait   pts/1    00:00:00 bash
0 S  1000  3526  2455  1  80   0 -  1690 poll_s pts/1    00:00:00 xeyes
0 R  1000  3527  2455  0  80   0 -  1242 -      pts/1    00:00:00 ps
giz@giz-NEC-VERSA-M160:~$
```

Esempio

$ kill -STOP *pid* per stopparlo

Verificare che gli occhietti si sono fermati e che facendo ps -l lo stato del processo ora è T

$ kill -CONT *pid* per riavviarlo

Verificare che gli occhietti hanno ripreso a muoversi.

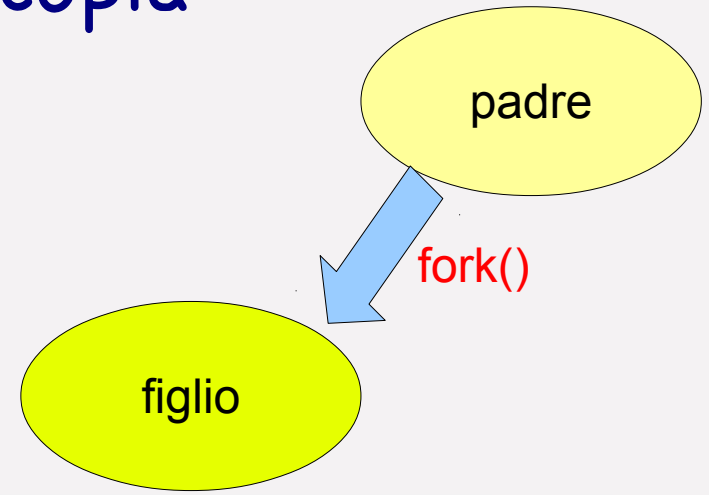
$ kill -KILL *pid* per terminarlo

Termina il processo.

fork()

La funzione fork() consente a un processo di generare un processo figlio:

- padre e figlio condividono lo stesso codice
- il figlio eredita una copia dei dati del padre



fork()

int fork();

La funzione fork non richiede parametri e restituisce un intero che:

- per il processo creato vale 0
- per il processo padre è un valore positivo che rappresenta il PID del processo figlio
- è un valore negativo in caso di errore (la creazione non è andata a buon fine)

Esempio di fork()

File testFork.c

```
#include <stdio.h>

int main() {

        int pid;
        pid=fork();
        if (pid==0)
        { /* codice eseguito dal processo figlio */
                printf("Sono il figlio con PID: %d\n", getpid());
        }
        else if (pid>0)
        { /* codice eseguito dal padre */
                printf("Sono il padre con PID %d e questo è il PID di mio figlio: %d\n",  getpid(),pid);
        }
        else printf("Creazione fallita!\n");

        printf("%d Esecuzione terminata!\n", getpid());

}
```

Esempio di fork()

Compilazione ed esecuzione:

```
giz@giz-HP-Pavilion:~/testC$ gcc testFork.c
giz@giz-HP-Pavilion:~/testC$ ./a.out
Sono il padre con PID 4160 e questo è il PID di mio figlio: 4161
4160 Esecuzione terminata!
Sono il figlio con PID: 4161
4161 Esecuzione terminata!
giz@giz-HP-Pavilion:~/testC$
```

Monitoraggio Processi

Per monitorare i processi in Linux utilizzare il comando:

$ top

Terminarlo con q

Top

giz@giz-NEC-VERSA-M160: ~

```
top - 00:35:36 up  7:11,  2 users,  load average: 0,17, 0,09, 0,06
Tasks: 162 total,   1 running, 161 sleeping,   0 stopped,   0 zombie
%Cpu(s):  9,7 us, 12,9 sy,  0,0 ni, 77,4 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
KiB Mem:   1017388 total,   914020 used,   103368 free,    74220 buffers
KiB Swap:  1037308 total,        0 used,  1037308 free,   500664 cached

  PID USER      PR  NI  VIRT  RES  SHR S  %CPU %MEM    TIME+  COMMAND
 1013 root      20   0 45568  11m 4680 S  21,6  1,1   1:16.15 Xorg
 2008 giz       20   0  238m  60m  25m S   9,6  6,1   1:27.18 compiz
 3642 giz       20   0 41500 9372 7464 S   6,3  0,9   0:00.30 gnome-screensho
 2524 root      20   0     0    0    0 S   1,7  0,0   0:03.90 kworker/0:2
 3064 root      20   0     0    0    0 S   1,0  0,0   0:02.66 kworker/u:35
 1991 giz       20   0  239m  16m  12m S   0,7  1,7   0:03.74 gnome-settings-
 2196 giz       20   0 41692  10m 8404 S   0,3  1,0   0:01.47 gtk-window-deco
 2294 giz       20   0 42688  10m 8180 S   0,3  1,0   0:01.83 gnome-screensav
 2446 giz       20   0 99056  16m  11m S   0,3  1,6   0:11.37 gnome-terminal
 3505 root      20   0     0    0    0 S   0,3  0,0   0:01.02 kworker/u:0
 3630 root      20   0     0    0    0 S   0,3  0,0   0:00.04 kworker/1:1
 3640 giz       20   0  5204 1384 1024 R   0,3  0,1   0:00.05 top
    1 root      20   0  3632 2024 1312 S   0,0  0,2   0:00.67 init
    2 root      20   0     0    0    0 S   0,0  0,0   0:00.00 kthreadd
    3 root      20   0     0    0    0 S   0,0  0,0   0:00.63 ksoftirqd/0
    6 root      rt   0     0    0    0 S   0,0  0,0   0:00.08 migration/0
    7 root      rt   0     0    0    0 S   0,0  0,0   0:00.05 watchdog/0
```