

11 - Folyamatkezelés

A folyamatokról

A programról „program” néven beszélünk amíg az egy háttértárolón van. Ha elindítottuk a programot, akkor bekerül a memóriába, ekkor folyamatról beszélünk. A folyamat tehát egy futó program a memóriában, másként végrehajtás alatt lévő program. Angolul task vagy process néven találkozunk vele. Az átlagfelhasználó persze nem szokta ezeket neveket használni, egyszerűen csak programként beszél róla.

A folyamatok újabb folyamatokat indíthatnak. Amikor elindítunk egy parancsot, azt eleve egy parancsértelmező programban tesszük. A parancsértelmezőből indított folyamat szülője maga a parancsértelmező. Valójában minden folyamat visszavezethető a hierarchiában egy közös sfolyamatra, ennek neve: „init”.

Egyszerre azonban csak egy folyamattal tudunk kommunikálni (az egy darab billentyűzet miatt), ezt „eltér” (foreground) folyamatnak nevezzük, a többi futó folyamatunkat pedig „háttér” (background) processzeknek. Pl. a Windows-tól eltérően, az eltérben futó folyamat semmilyen szempontból nincs kitüntetett helyzetben, nem élvez nagyobb prioritást - tulajdonképpen a futtató kernel nem is tudja, melyik folyamat van eltérben, hiszen az nézőpontjából mindegyik folyamat ugyanolyan file-mveleteket végez. Folyamatot a program neve után írt & jellel indíthatunk a háttérben: az így indított folyamatnak nem lesz standard inputja (megáll és várakozik, ha a program bemenetrel akar olvasni), a standard outputja pedig a shell standard outputja (vagyis a képernyő) lesz. Próbáljuk is ki:

```
claudius:~$ cat &
[1] 1103
```

Folyamatok elállítása

A folyamat elállítása, tulajdonképpen egy program futtatása. Két eset lehetséges, a rendszer vagy tudja hol van az elindítani kívánt program, vagy mi magunk mondjuk meg hol található az az. A következő lista azon könyvtárakat sorolja fel, ahol nagy valószínűséggel találunk indítható programokat:

- /bin
- /sbin
- /usr/bin
- /usr/sbin

Egyéb helyek:

- /usr/games
- /usr/local/bin
- /usr/local/sbin

Szeretnénk például futtatni az ifconfig parancsot. Írjuk be:

```
/sbin/ifconfig
```

Vagy a dmesg parancs:

```
/bin/dmesg
```

Ha egy parancs **útvonalban** van, akkor nem kell megadnunk a teljes útvonalat. Elég az ifconfig, vagy a dmesg parancs kiadása.

Ha egy parancs az aktuális könyvtárban van, végigírhatjuk a teljes útvonalat, vagy a „./” karakterpárossal jelzem, hogy helyben keresend a parancs.

A felhasználóknak szokás a saját könyvtárukban egy „bin” nevű könyvtár létrehozása, amelyben elhelyezhetik saját programjaikat, scriptjeiket. A joska felhasználó például ide rakja saját programjait:

```
/home/joska/bin/
```

A folyamatok száma a rendszerben korlátozott. A maximálisan létrehozható folyamatok száma alapértelmezésben: 32768. A rendszeredben a maximális értéket így derítheted ki:

```
cat /proc/sys/kernel/pid_max
```

Egy felhasználó azonban nem hozhat létre 32768 számot. A felhasználói limit lekérdezése:

```
ulimit -u
```

Folyamatok listázása

A folyamatok vagy angolosan processzek a ps parancs segítségével listázhatók. Paraméter nélkül használva csak az aktuális terminálról indított folyamatokról informál minket:

```
ps
```

```
PID TTY          TIME CMD
8205 pts/2      00:00:00 bash
8215 pts/2      00:00:00 ps
```

Indítsunk három folyamatot, amelyet rögtön a háttérbe is teszünk:

```
nano &
mcedit &
yes > /dev/null &
```

A folyamataink megtekintése:

```
ps
```

```
PID TTY          TIME CMD
2975 pts/0      00:00:00 bash
22403 pts/0      00:00:00 nano
22407 pts/0      00:00:00 mcedit
22409 pts/0      00:02:59 yes
22411 pts/0      00:00:00 ps
```

Nézzük meg milyen állapotban vannak a háttérbe tett folyamatok (ps a):

```
...
22403 pts/2    T      9:58 nano
22407 pts/2    T      0:00 mcedit
22409 pts/2    R      0:00 yes
...
```

A harmadik oszlop mutatja a státuszokat.

Az R futó (runing), a T megállított (terminate, de lehet trace vagyis nyomkövetett. Láthatjuk, ahogy a nano és az mcedit programokat háttérbe tettük, azok megállított állapotba kerültek. A yes parancs viszont futó állapotba került. A yes parancs csak az y karaktert írja a képernyre szünet nélkül. Mi persze ezt a /dev/null-ba irányítottuk. Amikor az "&," karakterrel háttérbe tesszük a folyamat nem áll meg.

Megnézhetjük az össze folyamatot a ps ax parancssal:

```
ps ax
```

A kapcsolókat nem vezettük be kötjellel.

Ezek után ehhez hasonló kimenetet láthatunk (részlet másolata):

```

PID TTY      STAT   TIME COMMAND
...
4244 ?        S      0:01 /usr/sbin/apache2 -k start
4245 ?        S      0:00 /usr/sbin/apache2 -k start
...
8474 ?        S      0:01 [kworker/1:2]
8509 ?        R      0:01 [kworker/1:1]
8572 pts/2    R+    0:00 ps ax

```

A PID oszlop tartalmazza a folyamatazonosítókat (process identity) az els oszlopban.

A TTY oszlop azt mutatja, hogy a folyamat TeleTYpe-hoz van-e kötve. A szolgáltatást nyújtó démon programok nincsenek.

A STAT oszlop a folyamat állapotáról tájékoztat.

A TIME oszlop mutatja, hogy a processzor idejébl mennyit használ.

Az utolsó oszlopban (COMMAND) a futtatott parancs nevét látjuk kapcsolókkal együtt.

A -f kapcsoló a full szóból ered, használata teljes kiemenet ad:

```
ps ax -f
```

Az „f” alparancs viszont megmutatja a gyermek szül kapcsolatokat:

```
ps axf
```

Például egy részlet:

```

1366 ?        Ss      0:00 /usr/sbin/apache2 -k start
2522 ?        S      0:00 \_ /usr/sbin/apache2 -k start
2523 ?        S      0:00 \_ /usr/sbin/apache2 -k start
2524 ?        S      0:00 \_ /usr/sbin/apache2 -k start
2525 ?        S      0:00 \_ /usr/sbin/apache2 -k start
2526 ?        S      0:00 \_ /usr/sbin/apache2 -k start

```

A folyamatot indító felhasználók mutatása:

```
ps axu
```

STAT mez

R	futó (runing) vagy futáskész (runable)
S	alvó (sleeping) de megszakítható
D	nem megszakíthatóan alvó (uninterruptible sleep) rendszerint IO
T	megállított (job kontroll; stoped) vagy nyomkövetett (traced)
Z	zombi (zombie) megállított, de elvesztette a szüljét
W	nincs rezidens lapja (érvénytelen a 2.6.xx kernelek óta)
X	halott (valószínűleg sosem látható)

A STAT mezhöz járulékos jelzések adhatók a BSD szerinti formátumban. A man -L en ps kézikönyvben találunk róla infot:

<	magas prioritás (más felhasználó nem tudja felülrni)
N	alacsony prioritás (más felhasználó felülírhatja)

L	a lapok a memóriába vannak zárva (valós id eléréséhez és IO tevékenységhez)
s	vezet munkamenet
I	többszálás (CLONE_THREAD használata, az NPTL pthreads-hez hasonló tevékenység)
+	folyamat csoport az eltérben

Néhány példa:

```
ps -f
```

```

UID      PID  PPID  C  STIME TTY      TIME  CMD
andras   22290 22151  0  07:32 pts/5    00:00:00 bash
andras   23071 22290  0  11:40 pts/5    00:00:00 ps -f

```

- UID felhasználó azonosító
- PID folyamatazonosító
- PPID szülőfolyamat azonosítója
- C processzor kihasználtság
- STIME indulási id
- TTY melyik teletájpához van kötve
- TIME felhalmozott processzorid
- CMD indító parancs

Felhasználói oszlopok

```

ps -o pid,stat,cmd
  PID STAT  CMD
 3823 Ss    bash
 4582 R+    ps -o pid,stat,cmd

```

Szül folyamatok PID száma

A -ef kapcsolópárossal a folyamatok szüleinek PID-t is láthatjuk, ez a PPID.

Az oszlopok ekkor így néznek ki:

```

UID      PID  PPID  C  STIME TTY      TIME  CMD
root      1      0      0  10:24 ?        00:00:01 /sbin/init
root      2      0      0  10:24 ?        00:00:00 [kthreadd]
root      3      2      0  10:24 ?        00:00:00 [ksoftirqd/0]
root      4      2      0  10:24 ?        00:00:00 [kworker/0:0]
root      5      2      0  10:24 ?        00:00:00 [kworker/0:0H]
...

```

Folyamatok leállítása

Folyamat leállítása mindenképpen

```
kill -9 PIDSZÁM
```

A PIDSZÁMOT a „ps ax” parancs kimenetéből nézhetjük ki. Ha leállítandó program azonosítója 4852, akkor így állítjuk le:

```
kill -9 4852
```

A Linuxos rendszerben a folyamatok jelzésekkel kommunikálnak egymással. A használható jelzéseket kilistázhatod a következő paranccsal:

```
kill -1
```

További információkat olvashatsz a jelzésekről a man7 signal kézikönyvből.

Egy folyamat egyes jelzéseket figyelmen kívül hagyhat, másokat kötelezően figyelembe kell vennie. Egyes jelzéshez tartozik alapértelmezett művelet, egyesekhez nem.

Az alábbi táblázatban 31 jelzést látunk. Ezek az első Bell Laboratórium létrehozott Unixból származnak. A POSIX szabvány alapján, további jelzésekkel egészült ki a jelzések listája, 34-től 64-ig.

man 7 signal kézikönyv alapján:

Jel	Azonosító	Tevékenység	Leírás
HUP	1	Term	A kontroll terminál megszt (A felhasználó kilépett) vagy démon esetén újra kell olvasni a konfigurációs fájlokat
INT	2	Term	Megszakítás a billentyzetről (Ctrl + C)
QUIT	3	Core	Kilépés a billentyzetről
ILL	4	Core	Illegális utasítás
ABRT	6	Core	Abort signal from abort(3)
FPE	8	Core	Lebegőpontos kivétel
KILL	9	Term	Semlegesítő jel
SEGV	11	Core	Érvénytelen memóiahivatkozás
PIPE	13	Term	Eltört cs (pájp): írás vagy olvasás nem létezik csbe
ALRM	14	Term	Idzítőjel az alarm(2)-től
TERM	15	Term	Megszakításjel
USR1	30,10,16	Term	Felhasználó által definiált jelzés 1
USR2	31,12,17	Term	Felhasználó által definiált jelzés 2
CHLD	20,17,18	Ign	A gyermek leállt vagy megszakadt
CONT	19,18,25	Cont	Folytatás, ha meg lett állítva
STOP	17,19,23	Stop	A folyamat megállítása
TSTP	18,20,24	Stop	Megálljt gépeltek egy tty eszközön
TTIN	21,21,26	Stop	tty bevitel egy háttérfolyamat számára
TTOU	22,22,27	Stop	tty kivitel egy háttérfolyamat számára

- Termi - megállítja a folyamatot
- Core - megállítja a folyamatot, majd a kidumpolja a core adatokat
- Stop - megállítja a folyamatot
- Cont - folytatja a folyamatot, ha az meg volt állítva
- Ign - elutasítja a folyamatot

A kill paranccsal lekérdezhethetjük egy jelzés száma alapján a nevét. Például:

```
kill -1 1
```

Démon folyamat újraindítása:

```
killall -HUP pidszám
```

Egy 0 számú szignállal azt is megvizsgálhatjuk, hogy egy folyamat fut vagy nem fut.

Olvasnivalók: man signal, man 7 signal

Sokáig futó vagy idzített folyamatok

Lehetségünk van azonban arra is, hogy egy folyamatot immunissá tegyünk kilépésünkre: hosszan, több óráig, több napig futó programokat a "nohup" paranccsal indíthatunk. Például:

```
claudius:~$ nohup program >outputfile <inputfile &
[1] 88
```

Ekkor a program kilépésünk után is tovább fut, és amint láttuk, gondoskodtunk arról, hogy a programfutás eredményei az "outputfile"-ban megrzdenek. Egyes shell-ekben a nohup-pal indított parancsok kimenete automatikusan a "nohup.out" file-ba kerül. Ha fut a gépünkön a "cron" nev program (általában minden Unixon fut) akkor lehetőségünk van idzített programindításra is az "at" paranccsal:

```
claudius:~$ at 3am program &
```

Több folyamat egyetlen terminálban

Folyamat leállítása

A folyamatokat leállíthatjuk a Ctrl+Z billentyűkombinációval.

Indítsuk el a nano nev szövegszerkeszt programot:

```
nano
```

Az indítás után a Ctrl+Z billentyűkombinációval tegyük háttérbe. Most indítsunk egy másik nano nev programot, tegyük ezt is háttérbe a Ctrl+Z billentyűvel. Indítsunk egy vi nev szövegszerkeszt. Tegyük a Ctrl+Z-vel háttérbe. Most indítsunk egy mutt nev levelez programot. Tegyük háttérbe ezt is.

Ezek után nézzük meg a háttérbe tett folyamatokat a „jobs” paranccsal:

```
jobs
```

Az eredmény ehhez hasonló:

```
[1]  Megállítva      nano
[2]  Megállítva      nano
[3]- Megállítva      vi
[4]+ Megállítva      mutt
```

A megállított nano, vi és mutt programot látjuk. Mindegyik kapott egy sorszámot. A sorszámokat használhatjuk annak jelzésére, hogy melyiket szeretnénk eltérbe hozni.

Eltérbe az „fg” paranccsal hozhatjuk a folyamatokat. Ha csak önmagában adom ki az „fg” parancsot, akkor az utolsót, esetünkben a mutt nev folyamatot hozza vissza. Az „fg” parancsnak azonban paraméterként megadhatok egy sorszámot is. Például a második nano program visszahívása:

```
fg 2
```

Egy folyamat persze már indításkor is háttérbe tehet:

```
yes > /dev/null &
```

Ha az fg parancsnak nem adunk meg paramétert, akkor a legkisebb azonosító számú programot fogja eltérbe hozni.

Gyakorlás

A yes parancs folyamatosan a képernyre írja a paraméterként megadott szöveget, és a coreutils csomagban található. Ha nem adunk meg paramétert akkor az y bett írja. Használhatjuk tesztekhez. Például irányítsuk a y betk sorozatát a /dev/null állományba:

```
yes > /dev/null
```

Így kapunk egy programot, amely folyamatosan a fut. A termináltól úgy tudom elszakítani, hogy a tanult módon háttérbe helyezem:

```
Ctrl + Z
```

Vagy már indításkor is a háttérbe helyezhetjük:

```
yes > /dev/null &
```

Az elindított folyamatok számát ellenrizzük, majd hozzuk eltérbe azokat és szakítsuk meg. Megszakítás a következ billentykombinációval:

```
Ctrl + C
```

Folyamatok fastruktúrában

A használható parancs:

```
pstree
```

Csak a jozsi felhasználó folyamatainak megjelenítése:

```
pstree joszi
```

A pstree kimenete átadható a less parancsnak lapozás céljából:

```
pstree | less
```

Vagy:

```
pstree joszi | less
```

A példa kedvéért indítsunk két folyamatot, amit leállítunk. Feltételezzük, hogy a felhasználónevünk „tibi”. A teendk a következ:

```
nano <Ctrl>+<Z>
nano <Ctrl>+<Z>
pstree -c tibi
```

Lehetséges kimenet (ha tibi csak egyszer lépett, és csak ezeket a programokat futtatja egy teletájról):

```
bashnano
  nano
```

top

Folyamatok valós idej figyelése

Kilépés:

```
q
```

A h billentyvel lekérdezhetjük a használható billentyket:

h

Z,B	Globálisan: a 'Z' a szintértékek cseréje; 'B' félkövér tiltása/engedése
l,t,m	Összefoglalók: 'l' terhelési átlagok; 't' folyamat/CPU statisztika; 'm' memóriainformációk
1,l	SMP nézet: '1' egyszer/kifejtett; 'l' Irix/Solaris mód
f,o	Mezk/Oszlopok: 'f' hozzáad vagy töröl; 'o' a megjelenés sorrendje
F vagy O	A kiválasztott mez rendezése
<,>	A mez mozgatása: '<' a következő oszlop balra; '>' a következő oszlop jobbra
R,H	Kulcs: 'R' normál/fordított rendezés; 'H' szálak mutatása
c,i,S	Kulcs: 'c' cmd name/line; 'i' üres folyamatok; 'S' növekv id
x,y	Kiemel kulcsok: 'x' rendezett mez; 'y' futó folyamatok
z,b	Kulcs: 'z' színes/mono; 'b' félkövér/fordít (csak 'x' vagy 'y' esetén)
u	Csak a felhasználók folyamatai
n vagy #	Maximálisan mutatott folyamatok száma
k,r	Folyamatok kezelése: 'k' kill; 'r' prioritás (renice)
d vagy s	Frissítési időköz beállítása
W	Beállítások fájlba írása
q	Kilépés

fuser

Használt fájl vagy könyvtár

A folyamatok által használt fájlokat és socketeket az fuser paranccsal tudjuk vizsgálni.

Szeretnénk látni azoknak a folyamatoknak a PID-jét, amelyeket az aktuális felhasználó használ:

```
fuser .
```

A kimenet:

```
. :                21384c
```

A példában szerepl PID szám végén van egy „c” karakter. Ez a hozzáférés típusát mondja meg számunkra. Lássuk milyen típusok vannak még:

c	az aktuális könyvtár
e	végrehajtható állomány
f	nyitott fájl
F	írásra megnyitott fájl
r	root könyvtár











m	memóriába mapolt fájl vagy osztott könyvtár
---	---

A részletek megmutatásához írjuk be a -v kapcsolót:

```
fuser -v .
```

A kimenet:

```
FELHASZNÁLÓ  PID HOZZÁFÉRÉS PARANCS
.:           joska      21384 ..c.. bas
```

- Milyen processzek futnak a rendszerben? 
- Jelenítse meg a futó processzek listáját fa elrendezésben! 
- Milyen processzek futnak most a rendszerben, amelynek a tulajdonosa a root? 
- Mi a processz id-je az init nev processznek? 
- Jelentkezzen be még egy példányban és indítsa el az mc nev programot! 
- A másik Putty ablakban kérdezze le ennek a Commandernek a process id-jét, majd küldjön neki egy TERM szignált! (A megoldásban az Ön bejelentkezési nevét a <username>, a leolvasott processz id-t <PID> szóval jelöltük). 
- Egyetlen paranccsal terminálja az összes processzét! 
- Jelenítse meg a futó processzek listáját terhelés szerint! Monitorozza a terhelést folyamatosan! 
- Keresse meg a rendszer összes .conf kiterjesztés fájlját és azok listáját írja a /tmp/find-<username>.txt fájlba! Mivel ez sokáig fut, a keresést a háttérben végezze! 
- Végezze el az elz feladatot úgy, hogy a hibaüzenetek ne kerüljenek a képernyre! 
- Végezze el újra az elz feladatot úgy, hogy az alacsony prioritással fusson! 