Unix et Programmation Shell

Philippe Langevin

département d'informatique
UFR sciences et technique
université du sud Toulon Var

Automne 2013

document

brouillon en révision

- site du cours :
  http://langevin.univ-tln.fr/cours/UPS/upsh.html
- localisation du fichier :
  http://langevin.univ-tln.fr/cours/UPS/doc/proc.pdf

dernières modifications

```
    upsh.tex 2024-01-28 21:05:32.653614398 +0100
   tools.tex 2024-01-28 21:05:32.653614398 +0100
    term.tex 2024-01-28 21:05:32.651614387 +0100
 syntaxe.tex 2024-01-28 21:05:32.650614381 +0100
   shell.tex 2024-01-28 21:05:32.649614376 +0100
prologue.tex 2024-01-28 21:05:32.648614371 +0100
    proc.tex 2024-01-28 21:05:32.646614360 +0100
    pipe.tex 2024-01-28 21:05:32.645614355 +0100
    perm.tex 2024-01-28 21:05:32.645614355 +0100
     man.tex 2024-01-28 21:05:32.643614344 +0100
    part.tex 2024-01-28 21:05:32.644614349 +0100
    file.tex 2024-01-28 21:05:32.642614338 +0100
  direct.tex 2024-01-28 21:05:32.641614333 +0100
    bash.tex 2024-01-28 21:05:32.639614322 +0100
```

6 - processus

- exécutable vs processus
- environnement
- variable du shell
- contexte d'exécution
- commande processus
- gentillesse
- ressource
- sous-processus
- job
- processus léger

processus ?

Il s'agit d'un fichier binaire en cours d'exécution ! Il y a un processus actif par processeur, le noyau partage et ordonne leur exécution.

processus ?

Il s'agit d'un fichier binaire en cours d'exécution ! Il y a un processus actif par processeur, le noyau partage et ordonne leur exécution.

- `bg`, `jobs`, `time`, `kill`
- `ps`, `pstree`, `/usr/bin/time`, `top`

ps

La commande `ps` renseigne sur les attributs

PID, PPID, TID (SPID), UID, EUID ...

et l'état d'un processus

	état
D	Uninterruptible sleep (usually IO)
R	Running or runnable (on run queue)
S	Interruptible sleep
T	Stopped, traced
X	dead (should never be seen)
Z	Defunct ("zombie") process

format BSD : $<$ haute priorité, N basse, L pages vérouillées, s session leader, l multithreadé, $+$ en avant plan.

exemple

```
⇝ ps -e -ostate,pid,cmd | grep -v S
R 12447 ps -e -ostate,pid,cmd
```

exemple

```
⇝ ps -e -ostate,pid,cmd | grep -v S
R 12447 ps -e -ostate,pid,cmd
```

```
⇝ ps --ppid 1 -oppid,pid,user,cmd
1   338 root /usr/lib/udev/udevd
1   339 root /usr/lib/systemd/systemd-journald
1   531 root /usr/lib/systemd/systemd-logind
1   538 root /usr/sbin/atd -f
1   540 root /usr/sbin/lxdm-binary -nodaemon
1   577 root /usr/sbin/crond -n
1   867 root /sbin/dhclient -H dellpl -1 ...
1   934 root /usr/sbin/sshd -D
1  1187 pl   xterm
1  1256 root sendmail: accepting connections
1  1272 smmsp sendmail: Queue runner@01:00:00 ...
1  1439 pl ssh -Nf pl@nil.fr -L2222:10.2.73.86:22
```

Environnement

Un processus hérite des variables d'environnement. Elles sont passées par la pile, tout comme le résultat de la commande.

```
1 ...
2 int main( int argc, char *argv[], char* env[] )
3 {
4 ...
5 return SUCCES;
6 }
```

- code de retour = 0 : SUCCES

capture de l'environnement

```
1  #include <stdio.h>
2  int main( int arg, char** argv , char** envr )
3  {
4  int res = 0;
5  while ( *argv )
6      printf ("\narg:%s", *argv++);
7  while ( *envr ) {
8      printf ("\nenv:%s", *envr++);
9      res ++;
10     }
11 return res;
12 }
```

code de retour

```
1  ::  gcc -Wall env.c
2  ::  ./a.out hello
3  arg:./ a.out
4  arg: hello
5  env:HOSTNAME=dellpl
6  env:TERM=xterm
7  env:SHELL=/bin/bash
8  env:HISTSIZE=1000
9  env:USER=pl
10 ...
11 ::  echo $?
12 41
13 ::  env  -i ./a.out hello
14 arg:./ a.out
15 arg: hello
```

variable usuelle

- std : LOGNAME, HOME PATH, SHELL, TERM, USER, UID, GID,...
- `bash` : PS1, PS2, HISTFILE, SHELLOPT, BASH_VERSION,...
- paramètres de `bash` : $, 0–9, −, ♯, ∗, @, !, ?, _

```
↝ help variable | grep -E '(PS|SH)'
    BASH_VERSION Version information for this
   Bash.
    PS1  The primary prompt string.
    PS2  The secondary prompt string.
    SHELLOPTS A colon-separated list of enabled
   shell options.
```

```
↝ env | grep -e '^P'
PATH=/usr/kerberos/sbin:/usr/kerberos/bin:/usr/
   local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/
```

transmission

- les variables du shell sont transmises aux sous-shell.
- `export` permet la transmission aux processus fils.

```
1 XYZ='hello world'
2 env | grep XYZ
3 export XYZ
4 env | grep XYZ
5 XYZ=hello world
6 export -p XYZ
7 export -n XYZ
8 env | grep XYZ
```

commandes

- `help variable` : variables usuelles.
- `env` : environnement d'exécution
- `export` : transmission automatique
- `unset` : démonte une variable du shell.
- `which`
- `declare`

```
↝ grep PATH ~/.bash_profile
PATH=$PATH:$HOME/bin:$HOME/script
export MANPATH=$MANPATH:$HOME/texinput
```

- `set` : configuration de `bash`

declare

```
⇝ X=(1 2 3)
⇝ declare -p X
declare -a X='([0]="1" [1]="2" [2]="3")'
declare z=15
⇝ declare -p z
declare -i z="15"
⇝ declare str=hello
⇝ declare -p str
declare -- str="hello"
⇝ declare -A t
⇝ t['un']=1 ; t['deux']=2; t['trois']=3
declare -p t
declare -A t='([un]="1" [deux]="2" [trois]="3"
)'
```

option du shell

```
1 #!/bin/bash
2 PS4=::
3 help set | grep -E '^[ ]*-(e|x)'
4 set -x
5 echo foo > /etc/shadow
6 set -e
7 echo bar > /etc/shadow
8 echo hello world
```

option du shell

```
1 #!/bin/bash
2 PS4=::
3 help set | grep -E '^[ ]*-(e|x)'
4 set -x
5 echo foo > /etc/shadow
6 set -e
7 echo bar > /etc/shadow
8 echo hello world
```

```
1       -e  Exit immediately if a command exits with a non-
    zero status.
2       -x  Print commands and their arguments as they are
    executed.
3 :: echo foo
4 ./set.sho: line 5: /etc/shadow: Permission non accordee
```

contexte d'exécution

Le contexte d'exécution,

- registre
- mémoire
- descripteur

et l'ensemble des données concernant un processus sont disponibles dans le répertoire `/proc/self` ou `/proc/pid`.

proc.sho

```
1 #!/bin/bash -v
2 ls   -d /proc/self/* |  sed 's/\/proc.self \///g'\
3       |   tr  '\n' ' ' |   fold  -w 54
4 #
5 cat /proc/self /environ  \
6   | tr  '\000' '\n' | grep -i name
7 cat /proc/self /status
```

proc.out

```
1  #!/bin/bash -v
2  ls  -d /proc/self/* | sed 's/\/proc.self \///g'\
3       | tr '\n' ' ' |  fold -w 54
4  attr  auxv cgroup clear_refs  cmdline comm coredump_filt
5  er  cpuset  cwd environ exe fd  fdinfo  io  latency  limits
6  loginuid  maps mem mountinfo mounts mountstats net oom_
7  adj  oom_score pagemap personality root  sched  schedstat
8   sessionid  smaps stack stat  statm status  syscall  task
9  wchan #
10 cat /proc/self /environ  \
11   | tr '\000' '\n' | grep -i name
12 HOSTNAME=microbe.rts.fr
13 USERNAME=pl
14 LOGNAME=pl
15 G_BROKEN_FILENAMES=1
```

status.out

```
1  #!/bin/bash -v
2  cat /proc/self /status
3  Name: cat
4  State:    R (running)
5  Tgid: 2286
6  Pid:  2286
7  PPid: 2284
8  TracerPid:   0
9  Uid:  501   501   501   501
10 Gid:  501   501   501   501
11 Utrace:   0
12 FDSize:   256
13 Groups:   10 501
14 VmPeak:      4084 kB
15 VmSize:      4084 kB
```

/proc/pid

```
1  #include <sys/types.h>
2  #include <stdio.h>
3  #include <unistd.h>
4  #include <stdlib.h>
5
6  int main( int argc, char *argv[] )
7  {
8  char cmd[64];
9  if ( argc > 1 )
10   sprintf (cmd, "cat /proc/%d/%s", getpid() , argv[1]) ;
11 else
12   sprintf (cmd, "ls -al /proc/%d", getpid() );
13 system( cmd );
14 putchar( '\n');
15 return 0;
```

```
↝ gcc -Wall proc.c -o proc.exe ;  ./proc.exe
dr-xr-xr-x.   2 pl pl0   3 sept. 12:19 attr
-rw-r--r--.   1 pl pl 0   3 sept. 12:19 autogroup
-r--------.   1 pl pl 0   3 sept. 12:19 auxv
-r--r--r--.   1 pl pl 0   3 sept. 12:19 cgroup
--w-------.   1 pl pl 0   3 sept. 12:19
   clear_refs
-r--r--r--.   1 pl pl 0   3 sept. 12:19 cmdline
-rw-r--r--.   1 pl pl 0   3 sept. 12:19 comm
-rw-r--r--.   1 pl pl 0   3 sept. 12:19
   coredump_filter
-r--r--r--.   1 pl pl 0   3 sept. 12:19 cpuset
lrwxrwxrwx.   1 pl pl 0   3 sept. 12:19 cwd -> /
   home/pl/UPS/doc
-r--------.   1 pl pl 0   3 sept. 12:19 environ
lrwxrwxrwx.   1 pl pl 0   3 sept. 12:19 exe -> /
   home/pl/UPS/doc/proc.exe
```

Carte mémoire

```
↝ ./proc.exe maps
08048000-08049000 r-xp 00000000 fd:01 793772 /
    home/pl/UPS/doc/a.out
08049000-0804a000 rw-p 00000000 fd:01 793772 /
    home/pl/UPS/doc/a.out
42c92000-42cb1000 r-xp 00000000 fd:01 163870 /
    usr/lib/ld-2.15.so
42cb1000-42cb2000 r--p 0001e000 fd:01 163870 /
    usr/lib/ld-2.15.so
42cb2000-42cb3000 rw-p 0001f000 fd:01 163870 /
    usr/lib/ld-2.15.so
42cb5000-42e5d000 r-xp 00000000 fd:01 163871 /
    usr/lib/libc-2.15.so
42e5d000-42e5f000 r--p 001a8000 fd:01 163871 /
    usr/lib/libc-2.15.so
```

commande processus

- ps, pstree, top, uptime
- time, time
- nice, ulimit
- gdb, ltrace, strace
- kill, killall
- wait
- jobs, bg, fg
- at, batch
- nohup
- disown

calcul

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main( int argc, char*argv[] )
4  {
5  int x = atoi( argv[1] );
6  int y = 1;
7  while ( --x )
8      y = (x * y) % 23;
9  return y;
10 }
```

calc.exe

```
#!/bin/bash -v
gcc -Wall -O2 calc.c -o calc.exe
./calc.exe 100000000 &
sleep 1
ps   -o cmd,pcpu
CMD                             %CPU
evince proc.pdf                  2.7
bash                             0.0
make -B pcpu.out                 0.0
/bin/sh -c ./pcpu.sh >& pcp      0.0
/bin/bash -v ./pcpu.sh           0.0
./calc.exe 100000000             102
ps -o cmd,pcpu                   0.0
```

charge/wait

```
1  #/bin/bash -v
2  cpt=$1
3  uptime | sed 's/load/\nload/'
4  while [ $cpt -gt 0 ]
5  do
6      /usr/bin/time --format="r=%U c=%c t=%E" \
7          ./calc.exe 500000000 &
8      let cpt--
9  done
10 wait
11 date
```

charge/wait

```
 14:11:40 up 7 days, 21:23,  5 users,
load average: 0,15, 0,53, 0,34
r=12.54 c=265 t=0:12.80
r=12.52 c=162 t=0:12.90
r=12.52 c=644 t=0:13.08
r=12.47 c=797 t=0:13.68
ven. sept. 20 14:11:54 CEST 2013
 14:11:54 up 7 days, 21:23,  5 users,
load average: 0,80, 0,66, 0,39
r=12.54 c=958 t=0:19.10
r=12.53 c=928 t=0:19.27
r=12.54 c=1080 t=0:19.30
r=12.53 c=1061 t=0:19.38
r=12.51 c=833 t=0:19.55
r=12.48 c=866 t=0:19.72
```

gentillesse

```
1  #/bin/bash -v
2  cpt=$1
3  gtle =0
4  while [ $cpt -gt 0 ]
5  do
6      /usr/bin/time --format="nice=$gtle cpu=%P ctx=%c
       total=%E" \
7          nice -n $gtle ./calc.exe 500000000 &
8      let cpt--
9      let gtle +=5
10 done
11 lscpu | grep '(s)' | iconv -f UTF-8 -t ASCII -c
12 ps -opid,nice,psr,cmd -C calc.exe
```

gentillesse

```
Mode(s) opratoire(s) des processeurs: 32-bit,
   64-bit
Processeur(s):          4
Liste de processeur(s) en ligne: 0-3
Thread(s) par cur: 2
Cur(s) par socket: 2
Socket(s):              1
  PID  NI PSR CMD
20555   5   3 ./calc.exe 500000000
20557   0   1 ./calc.exe 500000000
20560  10   3 ./calc.exe 500000000
20561  15   3 ./calc.exe 500000000
20562  19   0 ./calc.exe 500000000
nice=0 cpu=97% ctx=107 total=0:12.71
nice=5 cpu=91% ctx=1039 total=0:13.63
```

limites

```
↝ ./proc.exe limits| sed -E 's/unlimited/*/g;s/[
    \t]+/ /g'
Limit                     Soft       Hard Units
Max cpu time              *          *    seconds
Max file size             *          *    bytes
Max data size             *          *    bytes
Max stack size            10485760   *    bytes
Max core file size        0          *    bytes
Max resident set          *          *    bytes
Max processes             1024       7911 processes
Max open files            1024       1024 files
Max locked memory         65536      65536 bytes
Max address space         *          *    bytes
Max file locks            *          *    locks
Max pending signals       7911       7911 signals
```

limit.c

```
1  #include <sys/time.h>
2  #include <sys/resource.h>
3  #include <stdio.h>
4  int  main( void )
5  {
6  struct rlimit   z;
7  printf ("limites :      souple      maximum\n");
8  getrlimit  ( RLIMIT_AS, &z );
9  printf ("mem    :%10lu %10lu\n", z.rlim_cur,  z. rlim_max);
10 getrlimit  ( RLIMIT_NOFILE, &z );
11 printf ("fic      :%10lu %10lu\n", z.rlim_cur ,  z. rlim_max);
12 getrlimit  ( RLIMIT_NPROC, &z );
13 printf ("proc    :%10lu %10lu\n", z.rlim_cur ,  z. rlim_max);
14 getrlimit  ( RLIMIT_STACK, &z );
15 printf ("pile    :%10lu %10lu\n", z.rlim_cur ,  z. rlim_max);
```

```
./limit.exe
limites:     souple     maximum
mem     :4294967295 4294967295
fic     :       1024        1024
proc    :       1024        7911
pile    :   10485760 4294967295
cpu     :4294967295 4294967295

Temps UCT limite expire
```

stack.c

```
1  #include <signal.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  int    n = 0;
5  void  go( void  )
6  {
7   //printf(" %d\n", n );
8   n++;
9   go( );
10 }
11
12 int  main( int  argc,  char*argv[]  )
13 {
14 go( );
15 return 0;
```

stack.out

```
1  #!/bin/bash -v
2  ulimit -s
3  10240
4  echo -e 'display n\nrun\nquit' > /tmp/gdb.cmd
5  gdb -q -x /tmp/gdb.cmd ../src/stack.exe\
6       | grep SIG -A3
7  Program received signal SIGSEGV, Segmentation fault.
8  go () at stack.c:9
9  9    go( );
10 1: n = 786223
11 ulimit -s 0
12 gdb -q -x /tmp/gdb.cmd ../src/stack.exe\
13      | grep SIG -A3
14 Program received signal SIGSEGV, Segmentation fault.
15 go () at stack.c:9
```

sous-processus

Un processus peut lancer des sous-processus de plusieurs manières :

- `fork` : le sous-processus hérite d'une copie contexte.
- `clone` : partage de variables.
- `execve` : exécute un binaire ou un script.
- `pthread` : processus léger

Dans les cas lourds/léger, un processus parent attend (normalité) la fin des sous-processus, et les sous-processus signalent leur terminaison.

job

Les sous-processus du shell (**jobs**) peuvent être en :

- avant plan
- arrière plan

La commande interne `jobs` donne la liste des sous-processus du shell.

```
1  #!/bin/bash -v
2  sleep 10 &
3  sleep  5 &
4  jobs
5  [1]-  Running                 sleep 10 &
6  [2]+  Running                 sleep 5 &
7  ps -p $!
8    PID TTY          TIME CMD
9   2812 pts/1    00:00:00 sleep
10 jobs -l %1
11 [1]-  2811 Running                 sleep 10 &
```

tâche en arrière plan

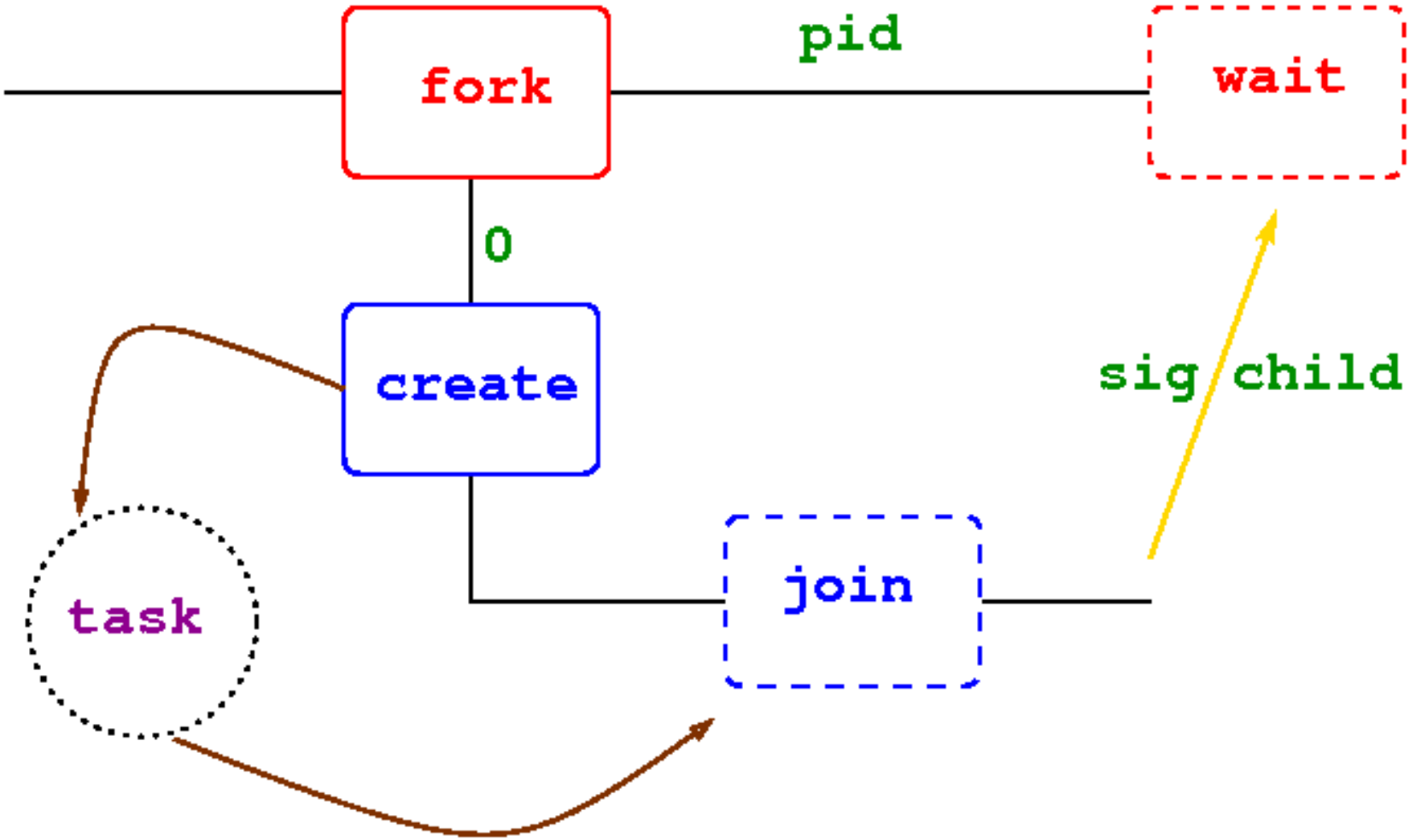
```
⇝ sleep 120 & sleep 60 & sleep 30 &
[1] 3649
[2] 3650
[3] 3651
⇝ kill -s SIGSTOP %2
⇝ jobs
[1]   En cours d'execution    sleep 120 &
[2]+  Stoppe                  sleep 60
[3]-  En cours d'execution    sleep 30  &
⇝ kill -s SIGKILL  %1
[1]   Processus arrete        sleep 120
⇝ jobs
[2]+  Stoppe                  sleep 60
[3]-  En cours d'execution    sleep 30  &
[3]-  Fini                    sleep 30
```

thread POSIX

```
1  #include <unistd.h>
2  #include <sys/types.h>
3  #include <sys/wait.h>
4  #include <stdio.h>
5  #include <errno.h>
6  #include <stdlib.h>
7  #include <pthread.h>
8
9  void * task (void *argv)
10 {
11 int x = *((int *) argv);
12 sleep (x);
13 pthread_exit (0);
14 }
```

bibliotèque pthread

```
1  int main (int argc, char *argv[] )
2  {
3  pid_t pid;
4  pthread_t th;
5  int status, x = atoi (argv[1]) ;
6  if ((pid = fork ()) < 0)
7      perror ("fork"), exit (1);
8
9  if ( pid )
10     wait (&status);
11 else {
12   if (pthread_create (&th, NULL, task, &x) < 0)
13     perror ("thread"), exit (2);
14     pthread_join (th, NULL);
15 }
```


fork
pid
wait
0
create
sig child
task
join

fork, pthread_create

```
1 #!/bin/bash -v
2 ./fils.exe 30 &
3 sleep 5
4 ps -T -o pid,ppid,spid,cmd
5 kill $!
6 ps -T -o pid,ppid,spid,cmd
```

fils.sh

processus lourds et légers de `make -B fils.out` ?

```
1 fils.exe : fils.c
2         gcc -Wall fils.c -o fils.exe -lpthread
3
4 fils.out : fils.exe fils.sh
5         ./fils.sh >& fils.out
```

```
./fils.exe 30 &
sleep 5
ps -T -o pid,ppid,spid,user,cmd
 1462  1415  1462 pl       /bin/bash
 2119  1462  2119 pl       make -B fils.out
 2125  2119  2125 pl       /bin/sh -c ./fils.sh
   >&  fils.out
 2126  2125  2126 pl       /bin/bash -v ./fils.
   sh
 2127  2126  2127 pl       ./fils.exe 30
 2129  2127  2129 pl       ./fils.exe 30
 2129  2127  2130 pl       ./fils.exe 30
 2131  2126  2131 pl       ps -T -o pid,ppid,
   spid,user,cmd
kill $!
ps -T -o pid,ppid,spid,user,cmd
 1462  1415  1462 pl       /bin/bash
```