Version : **2020.01**

Dernière mise-à-jour : 2020/01/30 03:28

SO205 - Gestion des Processus et des Tâches

Gestion des Processus

Un processus est un fichier binaire (binary file) qui est chargé en mémoire centrale. Une fois chargé la mémoire exécute le programme en langage machine. Quand le programme est chargé, il a besoin du système d'exploitation qui lui fournit des informations pour qu'il puisse s'exécuter correctement. Ces informations sont appelées des **données d'identification**.

L'ensemble des **données d'identification** est appelé l'**environnement de processus** :

- Un numéro de processus unique (PID),
- Un numéro de processus parent (PPID),
- Un numéro d'utilisateur (UID),
- Un numéro de groupe (GID),
- La durée de traitement,
- La priorité du processus,
- Le répertoire de travail actif,
- Les fichiers ouverts.

Ces informations sont stockés dans le répertoire **/proc**.

Système de fichiers /proc

Le répertoire /proc contient des fichiers et des répertoires virtuels. Le contenu de ces fichiers est créé dynamiquement lors de la consultation. Seul root peut consulter la totalité des informations dans le répertoire /root.

Pour commencer l'étude de /proc, saisissez la commande suivante :

```
# cd /proc; ls -d [0-9]*
0    142  279  372  390  424  471  569  611  669  698  751  816
1    144  3    373  4    434  473  587  653  67   719  771  822
11   149  352  374  420  442  5    594  654  677  726  813  9
113  2    367  382  423  469  568  609  657  691  728  815
```

Chaque répertoire fait référence à un PID d'un processus.

Commencez par regarder le contenu du répertoire **1**.

```
# cd 1; ls -l
total 5211
-rw-------   1 root     root     2617344 Jan 14 12:44 as
-r--------   1 root     root         168 Jan 14 12:44 auxv
dr-x------   2 root     root          48 Jan 14 12:44 contracts
-r--------   1 root     root          32 Jan 14 12:44 cred
--w-------   1 root     root           0 Jan 14 12:44 ctl
lr-x------   1 root     root           0 Jan 14 12:44 cwd ->
dr-x------   2 root     root        8208 Jan 14 12:44 fd
-r--------   1 root     root           0 Jan 14 12:44 ldt
-r--r--r--   1 root     root         120 Jan 14 12:44 lpsinfo
-r--------   1 root     root         816 Jan 14 12:44 lstatus
-r--r--r--   1 root     root         536 Jan 14 12:44 lusage
dr-xr-xr-x   3 root     root          64 Jan 14 12:44 lwp
-r--------   1 root     root        4608 Jan 14 12:44 map
dr-x------   2 root     root         800 Jan 14 12:44 object
-r--------   1 root     root        5504 Jan 14 12:44 pagedata
dr-x------   2 root     root        9008 Jan 14 12:44 path
-r--------   1 root     root          72 Jan 14 12:44 priv
-r--r--r--   1 root     root         336 Jan 14 12:44 psinfo
-r--------   1 root     root        4608 Jan 14 12:44 rmap
lr-x------   1 root     root           0 Jan 14 12:44 root ->
```

```
-r--------   1 root     root        1536 Jan 14 12:44 sigact
-r--------   1 root     root        1136 Jan 14 12:44 status
-r--r--r--   1 root     root         256 Jan 14 12:44 usage
-r--------   1 root     root           0 Jan 14 12:44 watch
-r--------   1 root     root        7296 Jan 14 12:44 xmap
```

Les Types de Processus

Il existe trois types de processus :

- **interactif** qui est lancé par le shell dans une console en premier plan ou en tâche de fond
- **batch** qui est lancé par le système au moment propice
- **daemon** qui est lancé au démarrage par le système (lpd, dns etc)

Un processus peut être dans un de neuf états ou *process states* :

- *user mode* - le processus s'exécute en mode utilisateur,
- *kernel mode*- le processus s'exécute en mode noyau,
- *new* - le processus est nouveau,
- *waiting* – le processus est en attente pour une ressource autre que le processeur,
- *sleeping* - le processus est endormi,
- *swap* - le processus est endormi dans la mémoire virtuelle,
- *runnable* – le processus dispose de toutes le ressources nécessaire à son exécution sauf le processeur,
- *elected* – le processus a le contrôle du processeur,
- *zombie* – le processus a terminé son exécution et est prêt à mourir.

Les Commandes relatives aux Processus

La commande ps

Cette commande affiche les processus de l'utilisateur attaché au terminal

```
# ps
   PID TTY          TIME CMD
   822 pts/2        0:00 sh
   964 pts/2        0:00 ps
```

Utilisez avec l'option **f**, vous obtiendrez la table des processus :

```
# ps -f
     UID   PID  PPID   C    STIME TTY          TIME CMD
    root   822   816   0 12:45:20 pts/2       0:00 -sh
    root   965   822   0 13:58:00 pts/2       0:00 ps -f
```

Dans la table des processus sont stockés les données des processus en cours. On note :

UID	User ID	Numéro de l'Utilisateur
PID	Process Indentification	Numéro Unique de Processus
PPID	Parent Process ID	PID du processus parent
C	Priority	Priorité instantanée
STIME	Start Time	Heure de démarrage
TTY	Terminal	Terminal parent
TIME	Duration	Durée
CMD	Command	Commande

Pour visualiser la table des processus de tout le monde, utilisez la commande ps avec les options -f et -e :

```
# ps -fe
     UID   PID  PPID   C    STIME TTY          TIME CMD
    root     0     0   0 12:44:40 ?           0:17 sched
    root     4     0   0 12:44:42 ?           0:00 kmem_task
    root     1     0   0 12:44:42 ?           0:00 /sbin/init
    root     2     0   0 12:44:42 ?           0:00 pageout
```

```
    root     3     0   0 12:44:42 ?           0:03 fsflush
    root     5     0   0 12:44:42 ?           0:00 vmtasks
    root   390     1   0 12:44:55 ?           0:00 /lib/svc/method/iscsi-initiator
    root     9     1   0 12:44:43 ?           0:01 /lib/svc/bin/svc.startd
    root    11     1   0 12:44:43 ?           0:03 /lib/svc/bin/svc.configd
    root   113     1   0 12:44:46 ?           0:00 /usr/lib/sysevent/syseventd
    root   352     1   0 12:44:54 ?           0:00 /usr/sbin/cron
    root   420     9   0 12:44:55 ?           0:00 /usr/lib/saf/sac -t 300
    root   568     1   0 12:44:56 ?           0:00 /usr/lib/autofs/automountd
    root    67     1   0 12:44:45 ?           0:00 /sbin/dhcpagent
    root   279     1   0 12:44:53 ?           0:00 /usr/lib/picl/picld
    root   822   816   0 12:45:20 pts/2       0:00 -sh
  daemon   382     1   0 12:44:54 ?           0:00 /usr/lib/nfs/lockd
  daemon   372     1   0 12:44:54 ?           0:00 /usr/lib/nfs/nfs4cbd
  daemon   374     1   0 12:44:54 ?           0:00 /usr/lib/nfs/nfsmapid
    root   149     1   0 12:44:46 ?           0:00 /usr/lib/power/powerd
    root   751   719   0 12:44:58 ?           0:00 /usr/dt/bin/dtlogin -daemon
  daemon   373     1   0 12:44:54 ?           0:00 /usr/lib/nfs/statd
    root   434     9   0 12:44:55 console     0:00 /usr/lib/saf/ttymon -g -d /dev/console -l console -m
ldterm,ttcompat -h -p sola
noaccess   813     1   0 12:45:02 ?           0:11 /usr/java/bin/java -server -Xmx128m -XX:+UseParallelGC -
XX:ParallelGCThreads=4
    root   587     1   0 12:44:56 ?           0:00 /usr/sbin/vold -f /etc/vold.conf
  daemon   144     1   0 12:44:46 ?           0:00 /usr/lib/crypto/kcfd
  daemon   367     1   0 12:44:54 ?           0:00 /usr/sbin/rpcbind
    root   142     1   0 12:44:46 ?           0:00 /usr/sbin/nscd
    root   473   469   0 12:44:55 ?           0:00 /usr/sadm/lib/smc/bin/smcboot
    root   424   420   0 12:44:55 ?           0:00 /usr/lib/saf/ttymon
    root   471   469   0 12:44:55 ?           0:00 /usr/sadm/lib/smc/bin/smcboot
    root   966   822   0 13:58:26 pts/2       0:00 ps -fe
    root   423     1   0 12:44:55 ?           0:00 /usr/lib/inet/inetd start
    root   469     1   0 12:44:55 ?           0:00 /usr/sadm/lib/smc/bin/smcboot
    root   442     1   0 12:44:55 ?           0:00 /usr/lib/utmpd
    root   569   568   0 12:44:56 ?           0:00 /usr/lib/autofs/automountd
```

```
    root   691     1   0 12:44:57 ?           0:00 /usr/lib/dmi/dmispd
    root   698     1   0 12:44:57 ?           0:00 /usr/lib/dmi/snmpXdmid -s solaris.i2tch.loc
    root   816   815   0 12:45:16 ?           0:00 /usr/lib/ssh/sshd
    root   726   719   0 12:44:58 ??          0:00 /usr/openwin/bin/fbconsole -n -d :0
    root   815   594   0 12:45:16 ?           0:00 /usr/lib/ssh/sshd
    root   594     1   0 12:44:56 ?           0:00 /usr/lib/ssh/sshd
    root   657     1   0 12:44:56 ?           0:00 /usr/lib/snmp/snmpdx -y -c /etc/snmp/conf
    root   611     1   0 12:44:56 ?           0:01 /usr/lib/fm/fmd/fmd
    root   609     1   0 12:44:56 ?           0:00 /usr/sbin/syslogd
    root   728   719   0 12:44:58 ?           0:04 /usr/X11/bin/Xorg :0 -depth 24 -nobanner -auth /var/dt/A:0-
aZa4zb
    root   677     1   0 12:44:56 ?           0:00 /usr/sfw/sbin/snmpd
    root   771   751   0 12:45:01 ?           0:01 dtgreet -display :0
   smmsp   653     1   0 12:44:56 ?           0:00 /usr/lib/sendmail -Ac -q15m
    root   654     1   0 12:44:56 ?           0:00 /usr/lib/sendmail -bl -q15m
    root   669     1   0 12:44:56 ?           0:00 devfsadmd
    root   719     1   0 12:44:57 ?           0:00 /usr/dt/bin/dtlogin -daemon
```

Les commandes fg et bg

Normalement les commandes s'exécutent en avant plan. Vous pouvez également lancer des processus en arrière plan ou en tâche de fond. La gestion des tâches de fond n'est pas possible en sh. Vous devez donc passez en ksh ou bash. Si vous lancez une commande en tâche de fond, il faut rajouter (espace)& à la fin de la commande

```
# /usr/bin/ksh
# sleep 9999 &
[1]     969
```

Important - Notez qu'un processus en arrière plan est dit asynchrone car il se poursuit indépendamment de son parent qui est le shell. En avant plan le processus est dit synchrone.

Solaris numérote tous les processus qui sont placés en tâches de fond. On parle donc d'un numéro de tâche.

La commande jobs permet de se renseigner sur les processus en arrière plan.

```
# jobs -l
[1] + 969        Running                 sleep 9999 &
```

Important - Le numéro de tâche est indiqué entre [crochets] tandis que le PID ne l'est pas.

Si on souhaite envoyer un processus en arrière plan de façon à libérer le shell pour d'autres commandes, il faut d'abord suspendre le processus en question.

Lancez la commande **sleep 1234** en avant plan puis ensuite suspendez ce processus à l'aide de ^Z :

```
# sleep 1234
^Z[2] + Stopped (SIGTSTP)        sleep 1234
# jobs -l
[2] + 972       Stopped (SIGTSTP)        sleep 1234
[1] - 969        Running                 sleep 9999 &
```

Un fois suspendu, on utilise la commande bg (background) pour envoyer le processus en arrière plan. Une fois en arrière plan, le processus continue :

```
# bg %2
[2]     sleep 1234&
# jobs -l
[2] + 972        Running                 sleep 1234
[1] - 969        Running                 sleep 9999 &
```

Pour ramener le processus en avant plan, il faut de nouveau interrompre le processus concerné. Or cette fois-ci, nous ne pouvons pas utiliser la commande ^Z. Il faut utiliser la commande kill avec l'opérateur –s stop :

```
# kill -s stop %2
# jobs -l
[2] + 972       Stopped (SIGSTOP)        sleep 1234
[1] - 969        Running                 sleep 9999 &
```

Important - Notez bien l'utilisation de la commande jobs -l pour se renseigner sur l'état du processus concerné.

Pour ramener le processus en avant plan, on utilise la commande fg :

```
# fg %2
sleep 1234
^C#
```

La commande wait

Cette commande permet de transformer une commande asynchrone en synchrone. Elle est utilisée pour attendre jusqu'à ce que les processus en tâches de fond soient terminés :

```
# jobs -l
[1] + 969        Running                 sleep 9999 &
# wait %1
^C#
# jobs -l
[1] + 969        Running                 sleep 9999 &
```

La commande nice

Cette commande modifie la priorité d'un processus. La priorité par défaut de nice est 20. La plage des valeurs de NOMBRE est de 0 à 40 où 0 est la plus prioritaire. La syntaxe de cette commande est :

```
# nice -n +/-NOMBRE COMMANDE
```

La commande renice

Cette commande modifie la priorité d'un processus déjà en cours. La valeur de la priorité ne peut être modifiée que par le propriétaire du processus ou par root. La syntaxe de cette commande est :

```
# renice -n +/-NOMBRE -p PID
```

La plage des valeurs de NOMBRE est de 0 à 39. La valeur de NOMBRE est ajouter ou déduit de la valeur par défaut, soit 20.

Important - Il est à noter que seul root peut décrémenter la valeur de priorité avec la commande renice.

La commande nohup

Cette commande permet à un processus de poursuivre son exécution après la déconnexion. Un processus enfant meurt quand le processus parent meure ou se termine. Comme une connexion et un processus, quand vous vous déconnectez, vos processus se terminent. Pour éviter de rester connecté après avoir lancé un processus long, vous utiliserez la commande nohup :

```
# nohup sort ventes & [Entrée]
```

La commande kill

Cette commande est utilisée pour arrêter un processus. Elle est à utiliser avec le PID.

```
# kill PID1 PID2 [Entrée]
```

La commande kill envoie des signaux aux processus. La liste des signaux possibles peut être afficher avec la commande :

```
# kill -l [Entrée]
```

Vous constaterez une liste des signaux possibles :

Sous sh :

```
# echo $SHELL
/sbin/sh
# kill -l
HUP     INT     QUIT    ILL     TRAP    ABRT    EMT     FPE     KILL    BUS
SEGV    SYS     PIPE    ALRM    TERM    USR1    USR2    CLD     PWR     WINCH
URG     POLL    STOP    TSTP    CONT    TTIN    TTOU    VTALRM  PROF    XCPU
XFSZ    WAITING LWP     FREEZE  THAW    CANCEL  LOST    XRES    JVM1    JVM2
RTMIN   RTMIN+1 RTMIN+2 RTMIN+3 RTMAX-3 RTMAX-2 RTMAX-1 RTMAX
```

Sous ksh :

```
# which ksh
/usr/bin/ksh
# /usr/bin/ksh
# kill -l
EXIT HUP INT QUIT ILL TRAP ABRT EMT FPE KILL BUS SEGV SYS PIPE ALRM TERM USR1 USR2 CLD PWR WINCH URG POLL STOP
TSTP CONT TTIN TTOU VTALRM PROF XCPU XFSZ WAITING LWP FREEZE THAW CANCEL LOST XRES JVM1 JVM2 RTMIN RTMIN+1
RTMIN+2 RTMIN+3 RTMAX-3 RTMAX-2 RTMAX-1 RTMAX
```

Sous bash :

```
# exit
```

```
# echo $SHELL
/sbin/sh
# which bash
/usr/bin/bash
# /usr/bin/bash
bash-3.2# kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGEMT       8) SIGFPE
 9) SIGKILL     10) SIGBUS      11) SIGSEGV     12) SIGSYS
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGUSR1
17) SIGUSR2     18) SIGCHLD     19) SIGPWR      20) SIGWINCH
21) SIGURG      22) SIGIO       23) SIGSTOP     24) SIGTSTP
25) SIGCONT     26) SIGTTIN     27) SIGTTOU     28) SIGVTALRM
29) SIGPROF     30) SIGXCPU     31) SIGXFSZ     32) SIGWAITING
33) SIGLWP      34) SIGFREEZE   35) SIGTHAW     36) SIGCANCEL
37) SIGLOST     38) SIGXRES     41) SIGRTMIN    42) SIGRTMIN+1
43) SIGRTMIN+2  44) SIGRTMIN+3  45) SIGRTMAX-3  46) SIGRTMAX-2
47) SIGRTMAX-1  48) SIGRTMAX
bash-3.2# exit
exit
```

Les commandes procTools

pcred

Cette commande affiche le propriétaire d'un processus :

```
# pcred 1
1:      e/r/suid=0  e/r/sgid=0
```

pfiles

Cette commande indique les fichiers ouverts :

```
# pfiles 1
1:      /sbin/init
  Current rlimit: 256 file descriptors
   0: S_IFIFO mode:0600 dev:102,0 ino:44720 uid:0 gid:0 size:0
      O_RDWR|O_NDELAY
      /etc/initpipe
 253: S_IFREG mode:0444 dev:275,1 ino:65538 uid:0 gid:0 size:0
      O_RDONLY|O_LARGEFILE FD_CLOEXEC
      /system/contract/process/pbundle
 254: S_IFREG mode:0666 dev:275,1 ino:65539 uid:0 gid:0 size:0
      O_RDWR|O_LARGEFILE FD_CLOEXEC
      /system/contract/process/template
 255: S_IFREG mode:0666 dev:275,1 ino:65539 uid:0 gid:0 size:0
      O_RDWR|O_LARGEFILE FD_CLOEXEC
      /system/contract/process/template
```

pflags

Cette commande donne des informations d'état :

```
# pflags 1
1:      /sbin/init
        data model = _ILP32  flags = ORPHAN|MSACCT|MSFORK
 /1:    flags = ASLEEP  pollsys(0x806a458,0x1,0x8047610,0x0)
```

pldd

Cette commande liste les libraires dynamiques utilisées :

```
# pldd 1
1:      /sbin/init
/lib/libpam.so.1
/lib/libbsm.so.1
/lib/libcontract.so.1
/lib/libc.so.1
/lib/libcmd.so.1
/lib/libsocket.so.1
/lib/libnsl.so.1
/lib/libmd.so.1
/lib/libsecdb.so.1
/lib/libnvpair.so.1
/lib/libscf.so.1
/lib/libdoor.so.1
/lib/libuutil.so.1
/lib/libgen.so.1
```

pmap

Cette commande indique l'espace d'adressage :

```
# pmap 1
1:      /sbin/init
08046000       8K rw---    [ stack ]
08050000      36K r-x--  /sbin/init
08069000       8K rw---  /sbin/init
0806B000      24K rw---    [ heap ]
D0470000      24K r-x--  /lib/libgen.so.1
```

```
D0486000       4K rw---  /lib/libgen.so.1
D0490000      24K r-x--  /lib/libuutil.so.1
D04A6000       4K rw---  /lib/libuutil.so.1
D04B0000       4K r-x--  /lib/libdoor.so.1
D04C1000       4K rw---  /lib/libdoor.so.1
D04D0000      88K r-x--  /lib/libscf.so.1
D04F6000       4K rw---  /lib/libscf.so.1
D0500000      24K rwx--    [ anon ]
D0510000      24K r-x--  /lib/libnvpair.so.1
D0526000       4K rw---  /lib/libnvpair.so.1
D0530000      12K r-x--  /lib/libsecdb.so.1
D0543000       4K rw---  /lib/libsecdb.so.1
D0550000      56K r-x--  /lib/libmd.so.1
D056E000       4K rw---  /lib/libmd.so.1
D0570000     516K r-x--  /lib/libnsl.so.1
D0601000      20K rw---  /lib/libnsl.so.1
D0606000      32K rw---  /lib/libnsl.so.1
D0610000      44K r-x--  /lib/libsocket.so.1
D062B000       4K rw---  /lib/libsocket.so.1
D0630000      12K r-x--  /lib/libcmd.so.1
D0643000       4K rw---  /lib/libcmd.so.1
D0650000     756K r-x--  /lib/libc.so.1
D071D000      28K rw---  /lib/libc.so.1
D0724000       8K rw---  /lib/libc.so.1
D0730000      16K r-x--  /lib/libcontract.so.1
D0740000       4K rwx--    [ anon ]
D0744000       4K rw---  /lib/libcontract.so.1
D0750000     116K r-x--  /lib/libbsm.so.1
D077D000      16K rw---  /lib/libbsm.so.1
D0781000       4K rw---  /lib/libbsm.so.1
D0790000      24K r-x--  /lib/libpam.so.1
D07A0000       4K rwx--    [ anon ]
D07A6000       4K rw---  /lib/libpam.so.1
D07B0000       4K rwx--    [ anon ]
```

```
D07B2000       4K rwxs-    [ anon ]
D07C0000       4K rwx--    [ anon ]
D07C5000     156K r-x--  /lib/ld.so.1
D07F0000       4K rwx--    [ anon ]
D07FC000       8K rwx--  /lib/ld.so.1
D07FE000       4K rwx--  /lib/ld.so.1
 total      2160K
```

pstack

Cette commande indique la pile d'exécution :

```
# pstack 1
1:      /sbin/init
 d06f45a7 pollsys  (806a458, 1, 8047610, 0)
 d069e79e poll     (806a458, 1, 493e0) + 52
 08053815 main     (1, 8047f6c, 8047f74) + 3eb
 08053346 ???????? (1, 8047fe0, 0, 0, 7d8, 8047feb)
```

ptree

Cette commande démontre un arbre de parents et de fils du processus :

```
# ptree
9     /lib/svc/bin/svc.startd
  413   /usr/lib/saf/sac -t 300
    419   /usr/lib/saf/ttymon
  435   /usr/lib/saf/ttymon -g -d /dev/console -l console -m ldterm,ttcompat -h
11    /lib/svc/bin/svc.configd
67    /sbin/dhcpagent
113   /usr/lib/sysevent/syseventd
```

```
137   /usr/lib/power/powerd
144   /usr/lib/crypto/kcfd
269   /usr/lib/picl/picld
371   /usr/sbin/nscd
380   /lib/svc/method/iscsi-initiator
383   /usr/sbin/rpcbind
387   /usr/lib/nfs/statd
388   /usr/lib/nfs/nfsmapid
390   /usr/lib/nfs/nfs4cbd
407   /usr/lib/nfs/lockd
424   /usr/lib/inet/inetd start
429   /usr/sbin/cron
432   /usr/lib/utmpd
495   /usr/sadm/lib/smc/bin/smcboot
  497   /usr/sadm/lib/smc/bin/smcboot
  504   /usr/sadm/lib/smc/bin/smcboot
609   /usr/sbin/vold -f /etc/vold.conf
611   /usr/lib/autofs/automountd
  612   /usr/lib/autofs/automountd
629   devfsadmd
644   /usr/sbin/syslogd
680   /usr/lib/ssh/sshd
  875   /usr/lib/ssh/sshd
    876   /usr/lib/ssh/sshd
      882   -sh
        886   sleep 9999
        917   ptree
695   /usr/lib/fm/fmd/fmd
701   /usr/lib/snmp/snmpdx -y -c /etc/snmp/conf
707   /usr/lib/sendmail -Ac -q15m
708   /usr/lib/sendmail -bl -q15m
736   /usr/lib/dmi/dmispd
738   /usr/sfw/sbin/snmpd
739   /usr/lib/dmi/snmpXdmid -s solaris.i2tch.loc
```

```
767   /usr/dt/bin/dtlogin -daemon
  768   /usr/openwin/bin/fbconsole -n -d :0
  769   /usr/X11/bin/Xorg :0 -depth 24 -nobanner -auth /var/dt/A:0-Dca4Fb
  791   /usr/dt/bin/dtlogin -daemon
    811   dtgreet -display :0
856   /usr/java/bin/java -server -Xmx128m -XX:+UseParallelGC -XX:ParallelGCThre
```

pwdx

Cette commande indique le répertoire courant du processus :

```
# pwdx 1
1:      /
```

Gestion des crash dump

Le termination anormale d'un processus génére un **crash dump** avec la création de fichiers **process core dump**.

Lors d'une anomolie critique, Solaris appelle la routine **panic()**. Cette routine interompe tous les processus et crée les fichiers **system core dump** unix.<X> et vmcore.<X> dans **le périphérique de vidage**. Ensuite Solaris utilise la commande **savecore** pour déplacer les fichiers core dump vers **le répertoire Savecore** lors du démarrage suivant.

Pour administrer les crash dumps, il convient d'utiliser la commande **dumpadm** :

```
# dumpadm
      Dump content: kernel pages
       Dump device: /dev/dsk/c0t0d0s1 (swap)
Savecore directory: /var/crash
  Savecore enabled: yes
   Save compressed: on
```

Dans la sortie de la commande, on peut noter :

- le contenu de vidage est les pages mémoire du noyau,
- le périphérique de vidage est la tranche swap,
- les fichiers core dump seront écrits dans /var/crash/,
- crash dump est activée.

Pour désactiver les crash dump il convient d'utiliser la commande **dumpadm** avec l'option **-n** :

```
# dumpadm -n
      Dump content: kernel pages
       Dump device: /dev/dsk/c0t0d0s1 (swap)
Savecore directory: /var/crash
  Savecore enabled: no
   Save compressed: on
```

Pour activer les crash dump il convient d'utiliser la commande **dumpadm** avec l'option **-y** :

```
# dumpadm -y
      Dump content: kernel pages
       Dump device: /dev/dsk/c0t0d0s1 (swap)
Savecore directory: /var/crash
  Savecore enabled: yes
   Save compressed: on
```

Pour modifier la configuration des crash dump, il convient de stipuler les options adéquates :

```
dumpadm -c contenu -d dump-device -m nnnk | nnnm | nnn% -n -s rép-savecore
```

Dans cette ligne de commande on peut noter les options suivantes :

- **contenu**,
  - kernel - la mémoire du noyau,
  - all - toute la mémoire,

    - curproc - la mémoire du noyau plus celle du processus en exécution au moment du crash,
- **dump-device**,
    - le device qui stocke les données du dump,
    - la valeur par défaut est la partition swap primaire,
- **-m**
    - nnnk - indique le nombre de Ko qui ne peuvent pas être utilisés par les fichiers core dump,
    - nnnm - indique le nombre de Mo qui ne peuvent pas être utilisés par les fichiers core dump,
    - nnn% - indique le pourcentage du *filesystem* qui ne peuvent pas être utilisés par les fichiers core dump,
- **-n**
    - spécifie que la commande savecore ne doit pas s'exécuter lors du re-démarrage du système,
    - cette option est **fortement** déconseillée car l'information du crash dump sera écrasée quand le système commence à utiliser le swap,
- **-s**
    - indique un répertoire alternatif pour le stockage des fichiers core dump,
    - par défaut la valeur de **rép-savecore** est **/var/crash/hostname**.

Exécutez donc la commande suivante pour modifier la configuration des crash dump :

```
# dumpadm -c kernel -d /dev/dsk/c0t0d0s1 -m 10%
      Dump content: kernel pages
       Dump device: /dev/dsk/c0t0d0s1 (swap)
Savecore directory: /var/crash (minfree = 849594KB)
  Savecore enabled: yes
   Save compressed: on
```

Utilisez maintenant la commande **gcore** pour forcer un core dump du processus de votre terminal :

```
# gcore $$
gcore: core.882 dumped
```

Le fichier core.PID est sauvegardé dans le répertoire courant.

Utilisez maintenant la commande **pstack** pour visualiser le stack du processus :

```
# pstack core.882
```

```
core 'core.882' of 882: -sh
 fef1cd85 waitid   (0, 3a5, 8047cd0, 83)
 fef0e644 waitpid  (3a5, 8047d98, 80) + 75
 0805e95d ???????? (8076da8)
 0805f240 postjob  (3a5, 1) + ce
 08059c95 execute  (80771e4, 0, 0) + 71d
 0806072f ???????? (0)
 080605ab main     (1, 8047e5c, 8047e64) + 4af
 0805528a ???????? (1, 8047f08, 0, 8047f0c, 8047f16, 8047f23)
```

La commande **mdb** indique le processus générateur du fichier crash dump ainsi que la commande utilisée pour le générer :

```
# mdb core.1118
Loading modules: [ libc.so.1 ld.so.1 ]
> ::status
debugging core file of sh (32-bit) from unknown
file: /sbin/sh
initial argv: sh
threading model: multi-threaded
status: process core file generated with gcore(1)
```

Durée approximative : 1 heure

Gestion des Tâches sous Solaris

cron

Le démon **cron** est normalement lancé au démarrage de la machine. Ce service est chargé de faire exécuter des tâches (commandes et scripts) à des moments précis.

Le service cron lit toutes les minutes les fichiers propres à chaque utilisateur qui se trouvent dans **/var/spool/cron/crontabs/** et qui sont au nom de

chaque utilisateur.

Par exemple, le fichier propre à l'utilisateur fenestros est le fichier /var/spool/cron/crontabs/fenestros.

Le service cron exécute des tâches en rajoutant une ligne dans le fichier **/var/cron/log**. Si une commande produit une sortie, celle-ci est dirigée vers la messagerie.

L'utilisation de cron est réservé à root. Cependant, vous pouvez établir une liste d'utilisateurs qui ont la permission d'utiliser cron en créant un fichier nommé cron.allow dans **/etc/cron.d/**. A l'inverse, un fichier cron.deny peut contenir une liste d'utilisateurs qui n'ont pas la permission d'utiliser cron. Par défaut, Solaris a un fichier cron.deny :

```
# cat /etc/cron.d/cron.deny
daemon
bin
nuucp
listen
nobody
noaccess
```

Important - Il est à noter ici que le service cron présume que la machine est allumée en permanence.

Quand le démon cron exécute la commande dans un fichier crontab, il définit un environnement réduit comprenant les variables HOME, LOGNAME, SHELL , qui est définit par défaut en **/bin/sh** et PATH. Le variable PATH est définit en tant que **/bin:/usr/bin** mais peut être modifiée par l'édition du fichier **/etc/default/cron** :

```
# cat /etc/default/cron
CRONLOG=YES
```

Ce fichier peut contenir donc les définitions du PATH pour les utilisateurs et de SUPATH pour root.

Chaque ligne dans un fichier crontab contient 5 champs temporels qui décrivent le périodicité de l'exécution de la tâche concernée.

Les 5 champs sont :

Minutes	Heures	Jour du mois	Mois de l'année	Jour de la sem.
(0-59)	(0-23)	(1-31)	(1-12)	(0-6)*

* le 0 correspond à dimanche.

Les champs temporels peuvent contenir des valeurs différentes :

Exemple	Description
Une valeur absolue telle 10	Dans le champs minutes = 10 minutes après l'heure
Une série de valeurs telle 2,6,8	Dans le champs mois = février, juin et août
Une intervalle telle 1-5	Dans le champs Jour de la Semaine = du lundi au vendredi
Le joker *	Dans le champs minutes = toutes les minutes
Une périodicité telle 0-23/2	Dans le champs heures = toutes les deux heures

Dans notre cas nous souhaitons confier à cron la gestion des mises à jour des définitions de virus ainsi que la vérification du répertoire /export/home au quotidien.

Afin de faire ceci, nous allons éditer le fichier crontab de root. Pour vérifier s'il existe une version de crontab existante pour root, il convient de lancer la commande suivante :

```
# crontab -l
#ident  "@(#)root       1.21    04/03/23 SMI"
#
# The root crontab should be used to perform accounting data collection.
#
#
10 3 * * * /usr/sbin/logadm
15 3 * * 0 /usr/lib/fs/nfs/nfsfind
30 3 * * * [ -x /usr/lib/gss/gsscred_clean ] && /usr/lib/gss/gsscred_clean
#
# The rtc command is run to adjust the real time clock if and when
# daylight savings time changes.
```

```
#
1 2 * * * [ -x /usr/sbin/rtc ] && /usr/sbin/rtc -c > /dev/null 2>&1
#10 3 * * * /usr/lib/krb5/kprop_script ___slave_kdcs___
```

Afin de créer ou éditer un fichier crontab, il convient d'utiliser la commande crontab avec l'option -e. Cette commande lance l'interface de l'éditeur définit par la variable $EDITOR. Actuellement, root ne dispose pas de fichier **.profile**. Commençons donc par créer ce fichier à la racine du système de fichiers :

```
# vi .profile
xterm-256color: Unknown terminal type
I don't know what kind of terminal you are on - all I have is 'xterm-256color'.
[Using open mode]
".profile" [New file]
EDITOR=/usr/bin/vi
export EDITOR
:x
".profile" [New file] 3 lines, 34 characters
# cat .profile
EDITOR=/usr/bin/vi
export EDITOR
TERM=vt100
export TERM
```

Sauvegardez votre fichier puis ré-amorcez votre Solaris. Ouvrez de nouveau une session root et vérifiez que votre éditeur est bien VI :

```
# echo $EDITOR
/usr/bin/vi
# echo $TERM
vt100
```

Nous pouvons maintenant éditer le crontab de root :

```
# crontab -e
#ident  "@(#)root       1.21    04/03/23 SMI"
```

```
#
# The root crontab should be used to perform accounting data collection.
#
#
10 3 * * * /usr/sbin/logadm
15 3 * * 0 /usr/lib/fs/nfs/nfsfind
30 3 * * * [ -x /usr/lib/gss/gsscred_clean ] && /usr/lib/gss/gsscred_clean
#
# The rtc command is run to adjust the real time clock if and when
# daylight savings time changes.
#
1 2 * * * [ -x /usr/sbin/rtc ] && /usr/sbin/rtc -c > /dev/null 2>&1
#10 3 * * * /usr/lib/krb5/kprop_script ___slave_kdcs___
~
~
~
~
~
~
~
~
~
"/tmp/crontab.oaqNb" 14 lines, 482 characters
```

Passez en mode EDITION. Tapez ensuite le texte suivant à la fin de votre fichier :

```
* * * * * /bin/pwd > pwd.txt
```

Sauvegardez et quittez vi.

A l'examen du fichier **/var/spool/cron/crontabs/root**, vous verrez que votre modification a bien été prise en compte :

```
# cat /var/spool/cron/crontabs/root
#ident  "@(#)root       1.21    04/03/23 SMI"
```

```
#
# The root crontab should be used to perform accounting data collection.
#
#
10 3 * * * /usr/sbin/logadm
15 3 * * 0 /usr/lib/fs/nfs/nfsfind
30 3 * * * [ -x /usr/lib/gss/gsscred_clean ] && /usr/lib/gss/gsscred_clean
#
# The rtc command is run to adjust the real time clock if and when
# daylight savings time changes.
#
1 2 * * * [ -x /usr/sbin/rtc ] && /usr/sbin/rtc -c > /dev/null 2>&1
#10 3 * * * /usr/lib/krb5/kprop_script ___slave_kdcs___
* * * * * /bin/pwd > pwd.txt
#
```

at

Tout comme avec la commande cron, root a la possibilité d'accorder ou d'interdire aux utilisateurs le droit d'exécuter des tâches avec at en utilisant les deux fichiers suivants :

- /etc/cron.d/at.allow
- /etc/cron.d/at.deny

Important - Si le fichier at.allow existe, seuls les utilisateurs dans ce fichier pourront exécuter at.

Pour mettre en place une tâche at, vous devez indiquer au système à quel moment cette tâche doit avoir lieu. Dans l'exemple qui suit, vous mettrez en place une tâche qui s'exécutera 3 minutes après la saisie :

```
# at now +3 minutes
at> pwd > /tmp/test.atd
at> <EOT>
commands will be executed using /sbin/sh
job 1579153920.a at Thu Jan 16 06:52:00 2020
```

Vérifiez ensuite la présence de la tâche :

```
# at -l
user = root     1579153920.a    Thu Jan 16 06:52:00 2020
```

A l'issu des trois minutes, vérifiez le contenu du fichier /tmp/test.atd :

```
# cat /tmp/test.atd
/
```

Mettez en place maintenant deux tâches pour le 31/12 à 13h00 et 14h00 respectivement :

```
at> pwd > /tmp/test13.atd
at> <EOT>
commands will be executed using /sbin/sh
job 1609416000.a at Thu Dec 31 13:00:00 2020
# at 14:00 Dec 31
at> vmstat > /tmp/test14.atd
at> <EOT>
commands will be executed using /sbin/sh
job 1609419600.a at Thu Dec 31 14:00:00 2020
```

Il existent maintenant deux tâches :

```
# at -l
user = root     1609416000.a    Thu Dec 31 13:00:00 2020
user = root     1609419600.a    Thu Dec 31 14:00:00 2020
```

Les fichiers concernant ces deux tâches sont stockés dans le répertoire **/var/spool/cron/atjobs** :

```
# ls /var/spool/cron/atjobs
1609416000.a  1609419600.a
```

A l'examen du deuxième fichier, vous constaterez un résultat similaire à celui-ci :

```
# cat /var/spool/cron/atjobs/1609419600.a
: at job
: jobname: stdin
: notify by mail: no
: project: 1
export EDITOR; EDITOR='/usr/bin/vi'
export HOME; HOME='/'
export LANG; LANG='C'
export LOGNAME; LOGNAME='root'
export MAIL; MAIL='/var/mail//root'
export PATH; PATH='/usr/sbin:/usr/bin'
export SHELL; SHELL='/sbin/sh'
export SSH_CLIENT; SSH_CLIENT='10.0.2.2 55666 22'
export SSH_CONNECTION; SSH_CONNECTION='10.0.2.2 55666 10.0.2.15 22'
export SSH_TTY; SSH_TTY='/dev/pts/2'
export TERM; TERM='vt100'
export TZ; TZ='Europe/Paris'
export USER; USER='root'
$SHELL << '...the rest of this file is shell input'
#ident  "@(#).proto     1.6     00/05/01 SMI"   /* SVr4.0 1.2   */
cd /
umask 22
ulimit unlimited
vmstat > /tmp/test14.atd
```

Pour supprimer cette tâche il convient d'utiliser la commande **at** avec l'option **-r** :

```
# at -l
user = root     1609416000.a    Thu Dec 31 13:00:00 2020
user = root     1609419600.a    Thu Dec 31 14:00:00 2020
# at -r 1609416000.a
# at -l
user = root     1609419600.a    Thu Dec 31 14:00:00 2020
```

Finalement, pour exécuter plusieurs commandes à la même heure d'une manière séquentielle, vous pouvez les insérer dans un fichier :

```
# at 10:00 < todo.txt [Entrée]
```
