続・誰にでも使える csh 講座
第 2 回
「眠れるプロセス」

安岡孝一

root ： yasuoka くん、 yasuoka くん。
yasuoka ： 何ですか？
root ： ginkaku で何か変なプロセス、走らせてない？
yasuoka ： 変なプロセスって

```
~/bin% alarm 15:22
```

これのことかな。

```
~/bin% alarm 15:22
Tue Mar 27 15:22:00 JST 1990 （ぴっ）
~/bin%
```

あ、終わった。
root ： どういうコマンドなんだい？
yasuoka ： alarm 15:22 しておくと、 15 時 22 分になったら、ぴっ、て鳴ります。

```
~/bin% ascii 007 （ぽこ）
007 （ぴっ）
~/bin%
```

っていうのを、使ってみようと思って。
root ： ちょっと見せてくれるかい？
yasuoka ： はい。

```
~/bin% cat alarm （ぽこ）
#! /bin/sh
# "alarm" Version 1.0

case "$1" in
??:??) : ;;
*) echo Usage: alarm hh:mm >&2
   exit 1 ;;
esac

ALARM=$1:00
set ‘date‘
until [ $4 = $ALARM ]
do set ‘date‘
done
date | tr ’\012’ ’\007’
echo
exit 0
~/bin%
```

root ： うーん、 007 はベルだとは限らないんだけど…。

tput bel 　　System V のみ。ベルを鳴らすような文字列を標準出力に出力する。

root ： まあ、 BSD だったらこうでもするしかないか。それよりもこのスクリプト、中で date を何千回も実行することになるね。
yasuoka ： はい。
root ： それよりは、 sleep を使って書いた方がいいと思うんだけど。

sleep 数 　　「数」秒間、実行を休止する。

yasuoka ： sleepって？
root ： 文字通り、何も実行せずに眠ってしまうコマンドだよ。例えば

```
~/bin% sleep 5 （ぽこ）
```

を実行すると…、まだかな…

```
~/bin% sleep 5
~/bin%
```

終わった。っていう風に、 5 秒間、何もしないでじっとしてるんだ。
yasuoka ： この alarm でそれを使うんですか？

root ： そうした方がいいと思う。

yasuoka ： でも今の時刻から秒数を計算しなきゃいけないわけでしょう？ むしろ、めんどくさいと思うんですけど。

root ： うん。けど、 sleep は CPU を消費しないからね。

yasuoka ： え？ どういうことですか？

root ： ちょっとやってみせようか。

```
~/bin% date（ぽこ）
Tue Mar 27 15:39:18 JST 1990
~/bin% /bin/time alarm 15:40（ぽこ）
```

yasuoka ： その /bin/timeって何ですか？

time コマンド
コマンドを実行した後、実行中に経過した時間、コマンドが CPU を使用した時間、システムが CPU を使用した時間を、標準エラーに出力する。 C シェル内蔵の time は、出力が標準出力である上にリダイレクトが難しいので、 /bin/time を用いる方がよい。

root ： コマンドを実行するのにかかった時間を表示するんだよ。そろそろかな。

```
~/bin% /bin/time alarm 15:40
Tue Mar 27 15:40:00 JST 1990（ぴっ）
       35.1 real         2.1 user        10.4 sys
~/bin%
```

終わった、終わった。実行時間は 35.1 秒間。そのうち、このスクリプトが CPU を実際に使用した時間が 2.1 秒、内部のコマンドが出す特殊命令が CPU を使用した時間が 10.4 秒。

yasuoka ： どういうことでしょう？

root ： ま、あわてずに。これを sleep 36 の実行時間と較べてみよう。

```
~/bin% /bin/time sleep 36（ぽこ）
```

（間）

root ： 待ってると 36 秒でも長いな。

```
~/bin% /bin/time sleep 36
       36.1 real         0.0 user         0.1 sys
~/bin%
```

よし、終了。ほら、実行時間は 36.1 秒だけど、 sleep は 0.1 秒しか CPU を使ってないだろ。

yasuoka ： CPU を使ってないって、どういう意味なんですか？

root ： うーん、 yasuoka くんが実行したコマンドとかは、 ginkaku の CPUっていうのが実際に処理するんだけど、 CPU は yasuoka くんのスクリプトにかかりっきりになってるわけじゃないんだ。

yasuoka ： と言うと？

root ： Unixってのは皆んなで使うシステムだからね、複数の人が同時にコマンドを実行してることもあるんだ。

yasuoka ： はい。

root ： でも、それを処理する CPU は 1 つしかないから、例えば yasuoka くんと誰かがコマンドを同時に実行したとしたら、 CPU はまず最初の 1/10 秒間は yasuoka くんのコマンドを処理して、次の 1/10 秒間はもう一方のコマンドを処理して、また次の 1/10 秒間はまた yasuoka くんのコマンドを処理して、っていう風に、交互に 2 つのコマンドをちょっとずつ進めていく。

yasuoka ： そうなんですか。

root ： で、さっきの場合だと、 yasuoka くんの alarm は 35.1 秒の間に CPU を 12.5 秒も使ってるけど、 sleep は 36.1 秒の間に CPU をほとんど使ってない。これがどういうことかわかるかい？

yasuoka ： sleep は CPU を使わないから損。

root ： まあ、 yasuoka くんにとっては損かな。でも他の人にとっては、 yasuoka くんのコマンドが CPU を使ってない分だけ、よけいに CPU を使えるわけだよ。だから、こういうスクリプトは sleep を使って書いた方がいい。

yasuoka ： そうなのかなぁ。

```
#! /bin/sh
# "alarm" Version 2.0

case "$1" in
??:??) : ;;
*) echo Usage: alarm hh:mm >&2
   exit 1 ;;
```

```
esac

ALARM=‘expr $1 : ’\(..\)’ ’*’ 3600 + $1 : ’..:\(..\)’ ’*’ 60‘
set ‘date | tr : ’ ’‘
sleep ‘expr $ALARM - $4 ’*’ 3600 - $5 ’*’ 60 - $6‘
date | tr ’\012’ ’\007’
echo
exit 0
```

（間）

root ： yasuoka くん、 yasuoka くん。

yasuoka ： 何ですか？

root ： ginkaku で変なプロセス、走らせてない？

yasuoka ： え、さっきの alarm ですか？ そういえば、 root さんが来る前に & 付けて走らせたのがあったような…。

root ： それそれ。 alarm を作り直したんなら、一旦 kill して走らせなおしてくれないかい？

yasuoka ： でも、プロセス番号がわかりません。

root ： ps してごらん。

```
ps
      プロセスの一覧表を標準出力に出力する。
ps u
      BSD のみ。プロセスの持ち主を含めた一覧表を標準出力に出力する。
ps -e
      System V のみ。プロセスの持ち主を含めた一覧表を標準出力に出力する。
ps ax
      BSD のみ。全プロセスの一覧表を標準出力に出力する。
ps -f
      System V のみ。全プロセスの一覧表を標準出力に出力する。
```

yasuoka ： ps ですか？

```
~/bin% ps （ぽこ）
  PID TT STAT  TIME COMMAND
 8272 co S    48:02 sh /home/yasuoka/bin/alarm 17:00
19821 co R     0:00 date
19822 co R     0:00 ps
~/bin%
```

root ： alarm 17:00 & で実行したのかい？

yasuoka ： たしかそうだったと思います。

root ： じゃあその 8272ってのが、プロセス番号だよ。 kill 8272 してごらん。

yasuoka ： kill 8272 ですね。

```
~/bin% kill 8272 （ぽこ）
[1]    Terminated            alarm 17:00
~/bin%
```

何か表示されました。これで止まったんですか？

root ： もう一度 ps してごらん。

yasuoka ： はい。

```
~/bin% ps （ぽこ）
  PID TT STAT  TIME COMMAND
20591 co R     0:00 ps
~/bin%
```

8272 はなくなりましたね。この psってどういうコマンドなんですか？

root ： ginkaku で走ってるプロセスのうち、この端末に関係あるものを表示する。

yasuoka ： ふーん、そうなんですか。じゃ、 alarm を立ち上げなおしてっと。

```
~/bin% alarm 17:00 & （ぽこ）
[1] 20593
~/bin%
```

ところで root さん、 & 付けてコマンドを実行すると、いつもこんな風に表示されるんですけど、これ何なんですか？

root ： 今、実行した alarm 17:00 のジョブ番号とプロセス番号だよ。 [1] の方がジョブ番号で、 20593 の方がプロセス番号。

yasuoka ： プロセス番号の方はわかりますけど

```
~/bin% ps （ぽこ）
  PID TT STAT  TIME COMMAND
20593 co I     0:00 sh /home/yasuoka/bin/alarm 17:00
20598 co I     0:00 sleep 2133
20599 co R     0:00 ps
~/bin%
```

ジョブ番号って何ですか？

root ： BSD だけの機能なんだけど、まずは jobs を実行してみてごらん。

jobs C シェル内蔵、 BSD のみ。バックグラウンドで実行中あるいは停止中のジョブの一覧を、標準出力に出力する。

yasuoka ： jobs ですね。

```
~/bin% jobs （ぽこ）
[1]  + Running                 alarm 17:00
~/bin%
```

何か alarm 17:00 が Running とか出ましたけど、これ、どういう意味ですか？

root ： 今、 yasuoka くんが言ったとおり。 alarm 17:00 がバックグラウンドで実行中っていう意味だよ。

yasuoka ： バックグラウンドって？

root ： 大雑把に言えば、 & を付けて実行したっていう意味だよ。で、 & を付けずに実行するのが、フォアグラウンド。ただ BSD の C シェルは賢くて、フォアグラウンドで実行したコマンドをバックグラウンドに回したり、あるいはその逆もできるんだ。その時に必要なのがジョブ番号。

yasuoka ： うーん、もう一つよくわかりません。

root ： じゃあ、実際にやってみせよう。まずは alarm 18:00 を & なしで実行してみてくれるかい？

yasuoka ： はい。

```
~/bin% alarm 18:00 （ぽこ）
```

これでいいですか？

root ： うん。で、この瞬間に「あ、しまった、 & 付けて実行するつもりだったのに」と思ったとする。

yasuoka ： はい、思いました。

root ： そこで、コントロール Z。

yasuoka ： コントロール Z ですね。

```
（コントロール Z）
Stopped
~/bin%
```

root ： もう一度 jobs してごらん。

yasuoka ： はい。

```
~/bin% jobs （ぽこ）
[1]  - Running                 alarm 17:00
[2]  + Stopped                 alarm 18:00
~/bin%
```

alarm 17:00 が Running で、 alarm 18:00 が Stopped です。

root ： それで alarm 18:00 はバックグラウンドに回ったんだ。ただし停止中だけどね。

yasuoka ： 今、 alarm 18:00 は実行されてないんですか？

root ： うん、そう。バックグラウンドで実行を続けたいなら、 bg %2 だよ。

bg %ジョブ番号 C シェル内蔵、 BSD のみ。ジョブをバックグラウンドで実行する。「%ジョブ番号」は複数書いてもよい。

yasuoka ： bg %2って？

root ： 停止中の [2] のジョブを、バックグラウンドで実行する、って意味。

yasuoka ： わかりました。

```
~/bin% bg %2 （ぽこ）
[2]    alarm 18:00 &
~/bin%
```

root ： これで、 alarm 18:00 & を直接実行したのと、同じ状態になった。

yasuoka ： バックグラウンドのジョブは、フォアグラウンドには出せないんですか？

root ： 出せるよ。まずは jobs してごらん。

yasuoka ： はい。

```
~/bin% jobs （ぽこ）
[1]  + Running                 alarm 17:00
[2]    Running                 alarm 18:00
~/bin%
```

root ： alarm 17:00 の方をフォアグラウンドに出すなら fg %1、 alarm 18:00 の方なら fg %2 だ。

fg %ジョブ番号 C シェル内蔵、 BSD のみ。ジョブをフォアグラウンドで実行する。

yasuoka ： じゃあ fg %2 ですね。

```
~/bin% fg %2（ぽこ）
alarm 18:00
```

これで、 alarm 18:00 を & なしで実行したのと同じになったんですか？

root ： うん、そうだよ。

yasuoka ： じゃ、もう一度コントロール Zっと。

```
（コントロール Z）
Stopped
~/bin% jobs（ぽこ）
[1]  - Running              alarm 17:00
[2]  + Stopped              alarm 18:00
~/bin%
```

この時 alarm 18:00 は停止中で

```
~/bin% bg %2（ぽこ）
[2]    alarm 18:00 &
~/bin%
```

これで実行再開っと。バックグラウンドで実行中のジョブを、直接停止させることってできないんですか？

root ： できるよ。 stop だ。

stop %ジョブ番号
　C シェル内蔵、 BSD のみ。ジョブをバックグラウンドで停止する。「%ジョブ番号」は複数書いてもよい。

yasuoka ： すると stop %2 かな。

```
~/bin% stop %2（ぽこ）
[2]  + Stopped (signal)     alarm 18:00
~/bin% jobs（ぽこ）
[1]  - Running              alarm 17:00
[2]  + Stopped (signal)     alarm 18:00
~/bin%
```

ちょっと表示が違うみたいですけど、これで停止したんですか？

root ： うん。実は stop は、 kill -STOP と同じだけどね。

kill -シグナル名 %ジョブ番号
　C シェル内蔵、 BSD のみ。ジョブにシグナルを送る。「%ジョブ番号」は複数書いてもよい。「-シグナル名」が省略された場合は、 -TERM とみなされる。

一般的なシグナル名は以下の通り。

HUP	端末からのブレーク信号
INT	キーボードからの割り込み
QUIT	キーボードからの強制終了
FPE	演算例外
KILL	非常強制終了
BUS	バスエラー
SEGV	セグメンテーションフォールト
PIPE	パイプの行き先がない
TERM	一般強制終了
STOP	停止
TSTP	キーボードからの停止

yasuoka ： killってプロセス番号だけじゃなくって、ジョブ番号も書けるんですね。

root ： C シェル内蔵の kill ならね。 /bin/kill だと無理だ。

yasuoka ： 「誰にでも書ける #! /bin/sh 講座」の時と、シグナルの指定の仕方が違うような気がするんですけど。

root ： まあね。シグナル番号とシグナル名がどう対応してるか、調べてみるのもいいと思うよ。ちなみに STOP と TSTP は BSD だけの機能で、コントロール Z はプロセスに TSTP のシグナルを送る。最近では System V でも、コントロール Z や jobs が使えるのも増えてきてるけどね。さて、もうこんな時間だ。プロセスとかジョブについては理解できたかい？

yasuoka ： はい、よくわかりました。どうもありがとうございました。