バックグラウンドタスクの実行と終了

バックグラウンドでタスクを実行させる、という課題を行います。

まず、「やたらと時間のかかるプログラム」を作成します。

vi heavy.c と入力して、以下のプログラムを入力します。

```
#include        <stdio.h>

main( int argc, char **argv )
{
  int i, j;
  float a;
  a = 1.01;
  for( i=0; i<1000; i++ ){
    for( j=0; j<50; j++ ){
        a *= 1.01;
        if( a>1000. ) a = 1.01;
    }
  }
  printf( "a = %g\n", a );
}
```

cc -o heavy heavy.c
と入力して、プログラムを準備します。コンパイルが完了したら、

./heavy
と入力して、動作を確認します。

```
# ./heavy
a = 1.38869
```

などのように、表示が現れた場合は、プログラムが実行を完了してコマンドに制御が戻ったことを意味しますが、大体数秒、反応が戻るまでに時間がかかっているはずです。どれくらい時間がかかるかは、仮想コンピュータの CPU クロックタイムの設定によります。ループ変数(j)の繰り返し回数を 50 回から適当に修正してください。

最初から、ループを「重く」すると、反応が戻ってこない場合にリセットせざるを得なくなるので、小さい値から、徐々に大きくしていくといいと思います。

なぜ、こんなプログラムを作ったか、理由を考えてみてください。今、テスト用に「時間がかかるプログラム」を作ろうとしました。
Float は、C 言語では、浮動小数点です。整数型の計算に比べて、不動小数点型の計算は時間がかかるため、1.01 の階乗を計算させるようなプログラムを作りました。1.01

の階乗でしたら、かなりの回数を反復してもオーバフロウしません。もし、オーバフロウが心配なほど（仮想）コンピュータが高速ならば、1.01 を 1.001 にすれば、さらにオーバフロウしにくくなります。
とにかく、反応が戻ってくるまで時間がかかるプログラムを作成しました。

```
# ./heavy
```

この表示のまま、しばらく「固まって」いて、数秒たってから

```
# ./heavy
a = 1.38869
*
```

という表示画面に切り替わります。これは、フォアグラウンド実行です。
　フォアグラウンドは、フォア・グラウンドと区切ります。
　フォアグラ・ウンドではありません。（オヤジギャグで済みません）

同じプログラムを、バックグラウンド実行させます。バックグラウンド実行では、コマンドの後に&（アンパーサンド；アンドマーク）をつけます。

```
# ./heavy &
#
```

こう入力すると、すぐにコマンドを入力できる状態になり、プロンプト（#）が表示されます。プロンプトは、「コマンドを入力できる」というコンピュータからのメッセージです。

この状態で、色々とコマンドを入力できるのですが、とりあえず、フォアグランド実行の時と同じくらいの時間、待ってみます。
そうすると、突然

```
# ./heavy &
# a = 1.38869
```

のように、「計算結果」が画面に表示されます。ここで、ENTER キーを押してみると

```
# ./heavy &
# a = 1.38869

[1] 64 Exit 12                ./heavy
#
```

のように、Exit というメッセージが表示されます。

ここで、

```
[1] 64 Exit 12                ./heavy
```

と表示されている意味を直接調べてみましょう。

cd /usr/src/commands/ash

と入力し（/usr/src/commands/ash に移動し）

vi jobs.c

と入力して、jobs.c を編集し、Exit（最初のみ大文字）を検索します。(Exit、最初の一文字のみ大文字の名称)は、このモデルでは１回しか使われていません。）

プログラムを見てみると

```
                        continue;
                procno = jp->nprocs;
                for (ps = jp->ps ; ; ps++) {    /* for each process */
                        if (ps == jp->ps)
                                fmtstr(s, 64, "[%d] %d ", jobno, ps->pid);
                        else
                                fmtstr(s, 64, "    %d ", ps->pid);
                        out1str(s);
                        col = strlen(s);
                        s[0] = '\0';
                        if (ps->status == -1) {
                                /* don't print anything */
                        } else if ((ps->status & 0xFF) == 0) {
                                fmtstr(s, 64, "Exit %d", ps->status >> 8);
                        } else {
                                i = ps->status;
#if JOBS
                                if ((i & 0xFF) == 0177)
                                        i >>= 8;
#endif
                                if ((i & 0x7F) <= MAXSIG && sigmesg[i & 0x7F])
                                        scopy(sigmesg[i & 0x7F], s);
                                else
                                        fmtstr(s, 64, "Signal %d", i & 0x7F);
/Exit
```

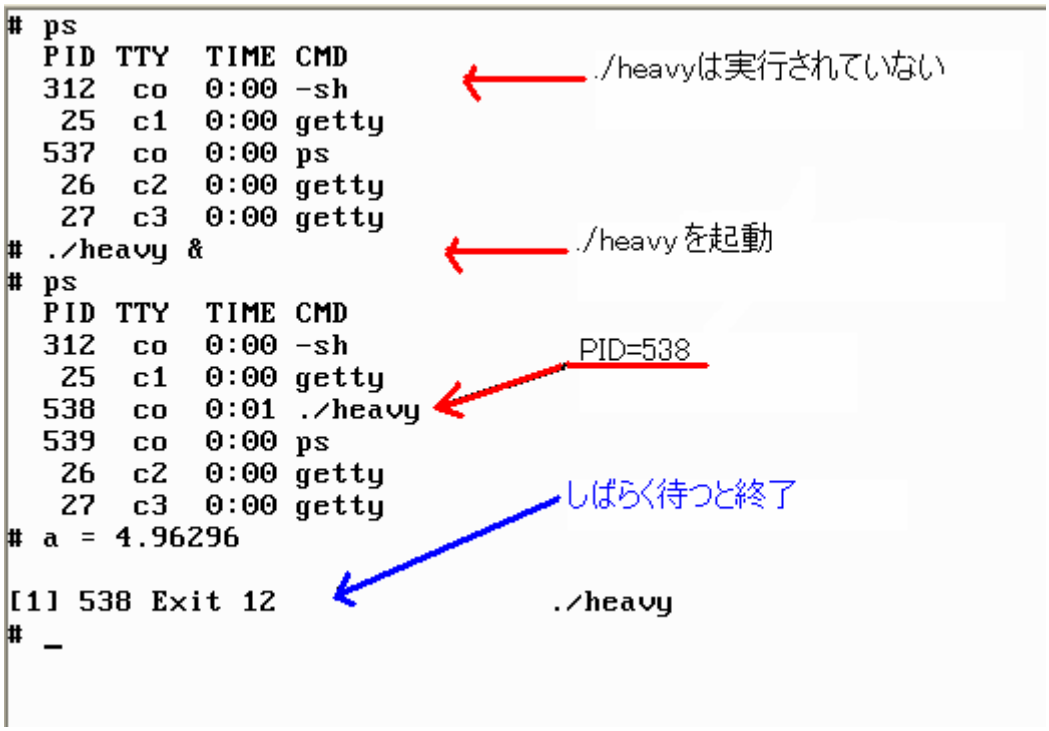
[1]は jobno （ジョブ番号)、次の 64 はプロセス ID であることがわかります。
また、Exit の次に表示されている１２は、ps->status という変数を右に８回シフトした値であることがわかります。

ここで、heavy.c の反応が戻るまでの時間を、もうほんの少し長くします。（だいたい１～２分程度）ループ変数 j の繰り返し回数を 250～300 程度にします。

```
for( i=0; i<1000; i++ ){
  for( j=0; j<250; j++ ){
      a *= 1.01;
      if( a>1000. ) a = 1.01;
  }
```

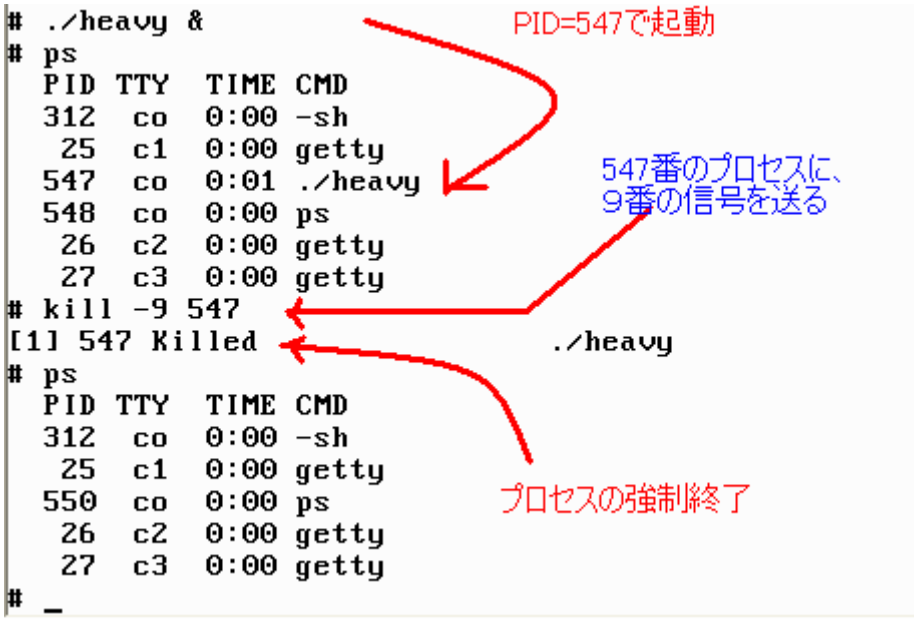
ここで、実行してみます。

ps でプロセス番号を表示すると、./heavy の PID が表示されています。黙ってじっと待っていれば、



そのプロセスは終了します。今度は、バックグラウンドで起動したプロセスに、-9 のシグナルを送ってみます。

Kill というコマンドは、「殺す」ではなく、本来は「信号を送る」という機能があります。



```
kill [ シグナル　または　番号 ]　プロセスID
```

シグナル

HUP	終了後再起動
INT	割り込み、一時停止
KILL	強制終了
TERM	終了（デフォルト）

番号

1	終了後再起動
2	割り込み、一時停止
9	強制終了
15	終了（デフォルト）

ＣＵＩの環境では、フォアグラウンドでプロセスを実行すると制御が戻ってきません。バックグラウンドで実行すれば、続いて他のコマンドを実行したりできますが、その場合には `kill` コマンドでプロセスを終了することができます。