11. プロセス管理

11.1 プロセスとは

- **プロセス**
  - 実行中のプログラム (アプリケーション) を管理する単位
  - コマンドインタープリタであるシェル自身もプロセスである.

シェルからコマンドを実行した場合を例に，プロセスの親子関係とプロセス生成から消滅までの流れを説明する.

1. ユーザがシェルからコマンドを実行すると，シェルは子プロセスとして自分の分身を作る (これを fork と呼ぶ).
2. シェルは子プロセスにコマンドの実行 (exec と呼ぶ) を任せ，子プロセスの終了を待つ.
3. 子プロセスはコマンドの実行を終えると，親プロセスに終了を伝え，消滅する.
4. 親プロセスは子プロセスの終了を受け取り，シェルプロンプトを表示し，ユーザの次のコマンドに備える.

11.2 スケジューリング

Linux の特徴: マルチユーザ，マルチタスク

- 1つの CPU でも複数のユーザが同時にログインでき，複数のプロセスを同時に実行できる．
  - 厳密には瞬間瞬間では，プロセスは1つしか実行されず，それぞれのプロセスの実行順序は Linux のスケジューラによって管理されている．
- 各プロセスは Run Queue と呼ばれる待ち行列で待機し，自分が CPU を利用できる順番を待つ．
- 自分の順番が来ると CPU を使用することができ，一定時間 (タイムスライス) だけ処理を前に進める．
- 一定の時間が超過すると，またキューで次の CPU 使用機会を待つことになる (ラウンドロビン方式)．
  - その他に FIFO 方式もある．
- 各プロセスはキューで待機するが，プロセスによっては優先度の高いもの，それほど高くないものがある．
  - e.g. Nice 値
    - -20 から 19 までの値を取る．
    - -20 が最も実行優先度が高く，19 が最も高い
  - nice コマンドにより実行優先度をプロセスに指定できる．
  - renice コマンドにより優先度を変更することができる．

11.3 フォアグラウンドジョブとバックグラウンドジョブ

- ジョブ
  - シェルが管理するプログラムの単位
- c.f. プロセス
  - 実行中のプログラムを管理する単位

シェルからコマンドを実行する場合，ジョブをフォアグラウンドとバックグラウンドに切り替える機能がある．

e.g. 処理に時間のかかるジョブを実行したとき，何もせずジョブの完了を待つのではなく，その間に別の作業をしたり，進捗を確認したい場合．
→ ジョブをバックグラウンドで実行すれば，同じ端末から別のコマンドを実行することができる．

e.g. プログラムの編集作業を中断して，プログラムを試行し，また編集作業に戻る場合．
← 編集を中断した場合，編集の undo や redo が中断前後で継続できる．

- コマンドをバックグラウンドで起動するには，コマンドの後ろに `&` を付けて実行する．
- 実行中のコマンドをバックグラウンドに切り替えるには，ctrl + z でサスペンドしてから， `bg` コマンドでバックグラウンドで実行を継続させる．
  - `fg` コマンドでフォアグラウンドに戻すことも可能．
- ジョブの状態は， `jobs` コマンドで確認することが可能．

実行例:

```
ai@ai-VirtualBox:~$ sleep 3600& <- 1時間スリープする
[1] 1383
ai@ai-VirtualBox:~$ sleep 3601&
[2] 1384
ai@ai-VirtualBox:~$ sleep 3602&
[3] 1385
ai@ai-VirtualBox:~$ jobs
[1]   Running                 sleep 3600 &
[2]-  Running                 sleep 3601 &
[3]+  Running                 sleep 3602 &
ai@ai-VirtualBox:~$ %1  <- [1] のジョブをフォアグラウンドに
sleep 3600
^Z  <- ctrl + z でフォアグラウンドジョブをサスペンドする
[1]+  Stopped                 sleep 3600
ai@ai-VirtualBox:~$ jobs
[1]+  Stopped                 sleep 3600
[2]   Running                 sleep 3601 &
[3]-  Running                 sleep 3602 &
ai@ai-VirtualBox:~$ %-  <- "-" の付いたジョブをフォアグラウンドに
sleep 3602
^C  <- フォアグラウンドにあるジョブを停止
ai@ai-VirtualBox:~$ %%  <- "+" のついたジョブをフォアグラウンドジョブに
sleep 3600
^C  <- フォアグラウンドにあるジョブを停止
ai@ai-VirtualBox:~$ fg  <- この場合, fg は fg %+, %;, %% と同じ動作になる
sleep 3601
^C  <- フォアグラウンドにあるジョブを停止
```

11.4 プロセス ID

Linux のプロセスには，一意の ID であるプロセスID (PID) が付与される．自身の PID は `$$` で取得できる．

実行例:

```
ai@ai-VirtualBox:~$ echo $$
1328
```

c.f. `echo` コマンド
引数で与えた文字列を標準出力する．

書式:

```
echo [option] 文字列
```

option:

```
-n
改行の抑制．通常の出力は改行されるが，このオプションがあると改行されない．
```

実行例:

```
ai@ai-VirtualBox:~$ echo One of the Linux distribution
One of the Linux distribution
```

`ps` コマンド

現在実行されているプロセスのスナップショットを表示する．

書式:

```
ps [option]
```

option:

-A
全てのプロセスを選択する．-e と等しい．

e
コマンドの後に環境を表示する．

l, -l
長いフォーマット．- の有無で表示が異なる．

w, -w
出力幅を広げる．このオプションを2つ付けると幅の制限がなくなる．

実行例:

```
ai@ai-VirtualBox:~$ ps l
F   UID     PID    PPID PRI  NI    VSZ   RSS WCHAN  STAT TTY        TIME COMMAND
0  1000    1328    1327  20   0  20436  5200 do_wai Ss   pts/0      0:00 -bash
0  1000    3471    1328  20   0  21084  3408 -      R+   pts/0      0:00 ps l
```

11.5 シグナル

- シグナル
  - イベントを送信してプロセスを制御する昨日
  - シグナル番号およびシグナル名が割り振られている.

代表的なシグナル:

シグナル番号	シグナル名	意味
1	`HUP`	ハングアップ (Hang Up)
2	`INT`	割り込み (Interrupt)
3	`QUIT`	強制停止 (Quit). コアダンプ生成.
9	`KILL`	強制終了 (Kill)
15	`TERM`	終了 (Terminate). `kill` コマンドのデフォルトシグナル.

ユーザがプロセスにシグナルを送信する方法:

- シェルに割り当てられたキーの入力 (e.g. ctrl + c, ctrl + z, ctrl + \)
- `kill` コマンド (e.g. `kill -HUP PID`, `kill -SIGINT PID`, `kill -9 PID`)
- プログラムで `kill()` 関数を呼ぶ.

e.g. ctrl + c
シェルは ctrl + c を解釈して，実行中のプロセスに `SIGINT` (Interrupt Signal: シグナル番号 2) を送信し，受け取ったプロセスは終了する.

シグナルの種類:

```
ai@ai-VirtualBox:~$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

実行例 (無限ループするシェルスクリプトの作成):

```
ai@ai-VirtualBox:~/Documents/11_process_workspace$ vi loop
ai@ai-VirtualBox:~/Documents/11_process_workspace$ cat loop
#!/bin/bash
while /bin/true
do
        sleep 3600
done
:w
^Z
```

実行:

```
ai@ai-VirtualBox:~/Documents/11_process_workspace$ chmod 755 loop
ai@ai-VirtualBox:~/Documents/11_process_workspace$ ./loop <- フォアグラウンドで実行するとプロンプトが戻ってこない
ctrl + c で INT が送信され，loopを終了することができる.
```

次に，作成したシェルスクリプトに trap 処理を挿入する.

```
ai@ai-VirtualBox:~/Documents/11_process_workspace$ vi loop
ai@ai-VirtualBox:~/Documents/11_process_workspace$ cat loop
#!/bin/bash
trap 'echo "...CTRL+C is pressed."'     2       # trap
while /bin/true
do
        sleep 3600
done
:w
^Z
```

これを実行すると，ctrl + c を押してもプロンプトが戻ってこない.

```
ai@ai-VirtualBox:~/Documents/11_process_workspace$ ./loop
^C...CTRL+C is pressed.
^C...CTRL+C is pressed.
^Z  <- ctrl + z でサスペンドして kill コマンドで TERM シグナルを送信
[1]+  Stopped                 ./loop
ai@ai-VirtualBox:~/Documents/11_process_workspace$ kill %%
[1]+  Terminated              ./loop
```

実行例 (パイプを使ったときのプロセスID):

```
ai@ai-VirtualBox:~/Documents/11_process_workspace$ vi pipe
ai@ai-VirtualBox:~/Documents/11_process_workspace$ cat pipe
tty=`tty | sed -e 's|/dev/||'`
ps alx | grep $tty | cat -n


ai@ai-VirtualBox:~/Documents/11_process_workspace$ sh pipe
     1  5  1000    1327    1161  20   0  14016  5960 -      S    ?          0:00 sshd: ai@pts/0
     2  0  1000    1328    1327  20   0  20568  5404 do_wai Ss   pts/0      0:00 -bash
     3  0  1000    3653    1328  20   0   2616   600 do_wai S+   pts/0      0:00 sh pipe
     4  0  1000    3657    3653  20   0  21084  3316 -      R+   pts/0      0:00 ps alx
     5  0  1000    3658    3653  20   0  18708   660 pipe_r S+   pts/0      0:00 grep pts/0
     6  0  1000    3659    3653  20   0  18540   588 pipe_r S+   pts/0      0:00 cat -n
```

ここから，以下のことが分かる.

- `bash` が sshd の子プロセスである.
- `sh pipe` が `bash` の子プロセスである.
- `ps alx`，`grep pts /0`，`cat -n` が `sh pipe` の子プロセスである.

11.6 top コマンドと pstree コマンド

- top コマンド
  - 実行中のプロセスの状態をダイナミックに表示.
  - CPU やメモリの使用率などでソートしたり， top から指定したプロセスにシグナルを送信したりできる.
- pstree コマンド
  - プロセスの親子関係をツリー状に表示する.

```
ai@ai-VirtualBox:~/Documents/11_process_workspace$ pstree $$
bash──pstree
```

11.7 プロセス間通信

複数のプロセスがコミュニケーションを取りながら同期したり，動作を変更して処理を達成すること．

プロセス間通信の手法:

- パイプ
  - 1つのプロセスの標準入出力をつなぎ替える．パイプ (|) の左側のコマンド (プロセス) の標準出力を右側のコマンド (プロセス) の標準入力につなぐことで，通信を実現する．
- シグナル
  - 特定のシグナルの受信をもって特定の処理を開始するなど，同期を実現する．
- System V IPC (Inter Process Communication)
  - 共有メモリ (shared memory)
    - 特定のメモリ領域を複数のプロセスで共有することでメッセージの受け渡し等が行える．
  - セマフォ (semaphore)
    - 資源のロックを行い，複数のプロセス間で同時に書きこみ等しないよう，排他制御を実現する．
  - メッセージキュー (message queue)
    - キューにメッセージを格納しておき，複数のプロセス間での非同期の通信を実現する．
- ソケット
  - ネットワーク経由のホスト間のプロセスでの通信を実現する．