Subject: Practice on Operating System Lecture Practice Process
From: IKENOYA Katsutoshi <j05002@ie.u-ryukyu.ac.jp>
Date: Wed, 07 Feb 2007 16:11:23 +0900
To: Shinji KONO <kono@ie.u-ryukyu.ac.jp>

学籍番号 : 055702B

・修正点

課題11の考察(Java API documentの引用)を追加しました。

・開発環境
OS : Mac OS X 10.4.8
コンパイラ :gcc version 4.0.1 (Apple Computer, Inc. build 5250)
java version "1.5.0_06"

・実行環境
同上

課題1 コマンドによるプロセスの観察

・実行結果
開いた２つのウィンドウをw1,w2とする.

まずw1でps uコマンドを実行してみた
```
[j05002@~]% ps u
USER PID %CPU %MEM VSZ RSS TT STAT STARTED TIME COMMAND
j05002 16479 0.0 0.1 31836 932 p1 S+ 11:41PM 0:00.11 -tcsh
j05002 16493 0.0 0.1 31836 936 p2 S 11:46PM 0:00.10 -tcsh
```

次にw2でcatコマンドを実行
```
[j05002@~]% cat
```

さらにw1でps uコマンドを実行
```
[j05002@~]% ps u
USER PID %CPU %MEM VSZ RSS TT STAT STARTED TIME COMMAND
j05002 16479 0.0 0.1 31836 932 p1 S 11:41PM 0:00.11 -tcsh
j05002 16493 0.0 0.1 31836 936 p2 S 11:46PM 0:00.10 -tcsh
j05002 16495 0.0 0.0 27368 344 p1 S+ 11:47PM 0:00.00 cat
```

プロセスの状態:"S"はスリープ状態,"+"はフォアグラウンドに属するプロセスである
ことを示している.

メモリの使用量:-tcshは0.1%,catは0.0%,メモリを使用している.

プロセスが実行しているプログラム:tcshのログインと,catを実行している.

実行結果で,-tcshが2つ存在しており,その2つは違った処理状態にあることから
プロセスとはプログラムの処理状態を表すものということが分かる.

プログラム:命令とデータの集合
プロセス:実行状態にあるプログラムの１つ１つの処理状態

課題2 ps aux コマンドによるプロセスの観察

・実行結果(一部分だけ抜粋)

```
[j05002@~]% ps aux
```

```
USER PID %CPU %MEM VSZ RSS TT STAT STARTED TIME COMMAND
```

省略

```
j05002 16476 11.7 1.2 137868 9760 ?? S 11:41PM 0:42.29
/Applications/Utilities/Terminal.
```

省略

```
j05002 16471 0.9 6.4 294104 50616 ?? S 11:39PM 2:42.24
/Applications/Opera.app/Contents/
j05002 16497 0.6 6.5 211316 51508 ?? S 11:47PM 2:13.98
/Applications/Thunderbird.app/Con
```

省略

```
root 16492 0.0 0.1 27528 468 p2 Ss 11:46PM 0:00.03 login -pf j05002
```

省略

レポート作成時に,Opera,mThunderbird,Terminalのアプリケーションを起動させていたので
そのプロセスが表示されている.
また,j05002というユーザーにログインしているというプロセスも表示されている.

topコマンドでプロセスの状態をリアルタイムで見てみた.

・実行結果

```
Processes: 59 total, 2 running, 57 sleeping... 183 threads 01:22:51
Load Avg: 1.07, 0.58, 0.46 CPU usage: 11.2% user, 24.1% sys, 64.7% idle
SharedLibs: num = 16, resident = 5.62M code, 552K data, 1.49M LinkEdit
MemRegions: num = 8332, resident = 256M + 12.0M private, 153M shared
PhysMem: 93.4M wired, 353M active, 288M inactive, 735M used, 32.5M free
VM: 4.70G + 10.1M 400099(0) pageins, 65518(0) pageouts

PID COMMAND %CPU TIME #TH #PRTS #MREGS RPRVT RSHRD RSIZE VSIZE
16720 lookupd 0.0% 0:00.05 2 18 35 348K 620K 932K 28.1M
16709 top 12.3% 0:24.21 1 19 22 580K 440K 1016K 26.9M
16708 tcsh 0.0% 0:00.11 1 15 20 416K 696K 940K 31.1M
16707 login 0.0% 0:00.03 1 16 37 176K 456K 568K 26.9M
16666 TextEdit 0.0% 0:02.37 1 93 162 2.75M 13.8M 7.63M 125M
16613 mdimport 0.0% 0:00.39 3 61 49 768K 3.62M 2.12M 38.8M
16608 mdimport 0.0% 0:06.98 4 66 139 6.60M 3.76M 8.58M 53.6M
16497 thunderbir 0.5% 3:03.67 3 105 361 26.3M 50.4M 51.7M 217M
16493 tcsh 0.0% 0:00.15 1 15 21 420K 696K 940K 31.1M
16492 login 0.0% 0:00.03 1 16 38 168K 456K 468K 26.9M
16476 Terminal 4.8% 0:56.25 7 183 202 3.36M 18.7M 11.1M 137M
16471 Opera 0.9% 3:24.44 9 119 382 33.3M 36.9M 40.2M 261M
13747 mount_nfs 0.0% 0:01.11 1 14 17 116K 372K 540K 26.6M
```

ここでもう１つのウィンドウからcatコマンドを実行してみた.

```
Processes: 60 total, 2 running, 58 sleeping... 184 threads 01:23:58
Load Avg: 0.76, 0.60, 0.48 CPU usage: 19.8% user, 19.8% sys, 60.3% idle
SharedLibs: num = 16, resident = 5.62M code, 552K data, 1.49M LinkEdit
MemRegions: num = 8342, resident = 256M + 12.1M private, 153M shared
PhysMem: 93.8M wired, 353M active, 288M inactive, 735M used, 32.2M free
VM: 4.72G + 10.1M 400103(0) pageins, 65518(0) pageouts

PID COMMAND %CPU TIME #TH #PRTS #MREGS RPRVT RSHRD RSIZE VSIZE
16723 cat 0.0% 0:00.00 1 13 17 104K 368K 344K 26.7M
```

```
16722 lookupd 0.0% 0:00.02 2 17 28 276K 576K 764K 27.5M
16709 top 12.9% 0:31.59 1 20 22 580K 440K 1016K 26.9M
省略
```

プロセスに新しくcatが増えたのが分かる.新しく生成したプロセスは一番上に表示された.
次にcatのプロセスを削除してみた.

```
Processes: 58 total, 2 running, 56 sleeping... 181 threads 01:25:40
Load Avg: 1.20, 0.77, 0.56 CPU usage: 6.1% user, 17.5% sys, 76.3% idle
SharedLibs: num = 16, resident = 5.62M code, 552K data, 1.49M LinkEdit
MemRegions: num = 8297, resident = 256M + 11.9M private, 153M shared
PhysMem: 92.3M wired, 354M active, 286M inactive, 732M used, 35.5M free
VM: 4.65G + 10.1M 400119(0) pageins, 65518(0) pageouts

PID COMMAND %CPU TIME #TH #PRTS #MREGS RPRVT RSHRD RSIZE VSIZE
16709 top 11.7% 0:42.83 1 18 22 580K 440K 1016K 26.9M
16708 tcsh 0.0% 0:00.11 1 15 20 416K 696K 940K 31.1M
16707 login 0.0% 0:00.03 1 16 37 176K 456K 568K 26.9M
省略
```

catのプロセスが無くなったことが分かる.

課題3 コマンドによるプロセスの消去

・実行結果

```
[j05002@~]% cat
^C
[j05002@~]% cat
Terminated
```

プロセスを^Cで殺すと^Cと表示され,killコマンドで殺すと,Terminatedと表示された.

課題4 コマンドによるプロセスの一時停止

・実行結果

```
[j05002@~]% cat
^Z
Suspended
[j05002@~]% fg
cat
```

課題5 ps コマンドの利用法

・実行結果

```
[j05002@~]% ps
PID TT STAT TIME COMMAND
16771 p1 S 0:00.11 -tcsh
[j05002@~]% ps u
USER PID %CPU %MEM VSZ RSS TT STAT STARTED TIME COMMAND
j05002 16771 0.0 0.1 31836 940 p1 S 2:06AM 0:00.11 -tcsh
[j05002@~]% ps l
UID PID PPID CPU PRI NI VSZ RSS WCHAN STAT TT TIME COMMAND
502 16771 16770 0 31 0 31836 940 - S p1 0:00.12 -tcsh
```

psコマンドでは

PID:プロセスID

TT:プロセスを実行している端末番号
STAT:プロセスの状態
TIME:CPU使用時間
COMMAND:実行コマンド名

が表示される.

ps uコマンドではpsコマンドで表示されたものに加えて,

USER:ユーザー名
%CPU:CPUの使用率
%MEM:メモリの使用率
VSZ:プロセスのメモリ使用量
RSS:実際のメモリ使用量
STARTED:実行開始時刻

が表示される.

ps lコマンドでは,psコマンドの表示に加えて

UID:ユーザーID
PPID:親プロセスID
CPU:CPUの使用率
PRI:プライオリティ.数値が大きい程,低いプライオリティ.
NI:プロセスの nice 数
VSZ:プロセスのメモリ使用量
RSS:実際のメモリ使用量
WCHAN:プロセスが休眠状態の時のカーネル関数名

が表示される.

課題6 ps コマンドによるプロセスの優先順度の観察

・実行結果

```
[j05002@~]% ps alx | sort -nk5
502 13746 459 0 0 0 0 0 - Z ?? 0:00.00 (mount)
502 20122 459 0 0 0 0 0 - Z ?? 0:00.00 (mount)
UID PID PPID CPU PRI NI VSZ RSS WCHAN STAT TT TIME COMMAND
0 12425 1 0 1 0 27244 200 - Rs ?? 288:39.82 (server)
0 12738 1 0 3 0 27244 200 - Rs ?? 271:36.47 (server)
0 11368 1 0 7 0 27244 200 - Rs ?? 429:00.12 (server)
502 13058 1 0 13 18 40280 2592 - SNs ?? 0:00.51
/System/Library/Frameworks/CoreS
4294967294 13060 1 0 13 18 39892 2340 - SNs ?? 0:00.26
/System/Library/Frameworks/CoreS
0 53 1 0 26 0 27244 480 - Rs ?? 12:56.99 /usr/sbin/update
0 13092 13086 0 30 0 27312 420 - R+ p2 0:00.00 ps alx
502 13086 13085 0 30 0 31836 940 - S p2 0:00.12 -tcsh
0 31 1 0 31 0 28212 508 - Ss ?? 0:05.19 kextd
0 36 1 0 31 0 27844 380 - Ss ?? 0:00.16 /usr/sbin/KernelEventAgent
502 13087 64 0 57 0 270192 48676 - S ?? 0:18.38
/Applications/Thunderbird.app/Co
502 459 64 0 62 0 263736 24380 - S ?? 39:25.11
/System/Library/CoreServices/Fin
0 27 1 0 63 0 27260 60 - Ss ?? 0:01.73 /sbin/dynamic_pager -F /private/
88 64 1 0 63 0 180396 35188 - Ss ?? 475:19.10
/System/Library/Frameworks/Appli
502 208 64 0 63 0 149972 12568 - S ?? 14:12.51
/System/Library/CoreServices/Sys
502 300 207 0 63 0 115020 9968 - S ?? 0:35.53
/System/Library/CoreServices/Doc
502 301 207 0 63 0 382364 189112 - S ?? 27:29.23
/System/Library/CoreServices/Doc
```

```
502 307 207 0 63 0 126816 2592 - S ?? 0:45.73
/System/Library/CoreServices/Doc
502 24036 207 0 63 0 155844 7072 - S ?? 0:35.92
/System/Library/CoreServices/Doc
502 24143 207 0 63 0 128772 5856 - S ?? 0:24.39
/System/Library/CoreServices/Doc
502 25542 207 0 63 0 150780 3276 - S ?? 3:05.01
/System/Library/CoreServices/Doc
502 191 1 0 97 0 144168 8036 - Ss ?? 4:55.39
/System/Library/CoreServices/log
```

sortコマンドのオプション説明
-n: ソート対象文字列を数値としてみる
-k: 行頭から指定数のフィールドに従ってソートする。

今回の場合、オプションは"-nk5"なので、プロセスの優先順位を表す"PRI"
フィールドを
数値としてソートするという処理を行った。

PRIは、数値が低ければ低いほど優先順位は高くなる。
ソートした結果、一番上にあるプロセスが一番優先順位の高いものとなる。
今回の結果では,(mount),(server),/System/Library/Frameworks/CoreS,などが
優先度の
高いものである。

課題7 他人のプロセス

・実行結果

```
[j05002@pw002 ~]% ps a
PID TTY STAT TIME COMMAND
1395 tty1 Ss+ 0:00 /sbin/mingetty tty1
1396 tty2 Ss+ 0:00 /sbin/mingetty tty2
1397 tty3 Ss+ 0:00 /sbin/mingetty tty3
1398 tty4 Ss+ 0:00 /sbin/mingetty tty4
1399 tty5 Ss+ 0:00 /sbin/mingetty tty5
1400 tty6 Ss+ 0:00 /sbin/mingetty tty6
10228 pts/0 Ss 0:00 -tcsh
10307 pts/0 R+ 0:00 ps a
[j05002@pw002 ~]% kill 1395
1395: 許可されていない操作です
```

以上の結果より,他人のプロセスは殺すことができない.

課題8 その他のsignal

kill コマンドで使えるsignalは以下のとおりである.

```
[j05002@~]% kill -l
HUP INT QUIT ILL TRAP ABRT EMT FPE KILL BUS SEGV SYS PIPE ALRM TERM URG
STOP
TSTP CONT CHLD TTIN TTOU IO XCPU XFSZ VTALRM PROF WINCH INFO USR1 USR2
```

ここでは、topコマンドで使用できるsignalの実行結果を記載する.
※以下の実行結果は別端末からkillコマンドでcatを殺した結果である.

HUP:制御端末のハングアップ検出。制御しているプロセスの死。

```
[j05002@~]% cat
Hangup
[j05002@~]%
```

```
-----------------------------------
INT:キーボードからの割り込み。(^C)
   バックグラウンドで動くプロセスは殺せない。
[j05002@~]% cat

[j05002@~]%

-----------------------------------
QUIT:キーボードによる中止
[j05002@~]% cat
Quit
[j05002@~]%

-----------------------------------
ILL:不正な命令
[j05002@~]% cat
Illegal instruction
[j05002@~]%

-----------------------------------
TRAP:トレース/ブレークポイント トラップ
[j05002@~]% cat
Trace/BPT trap
[j05002@~]%

-----------------------------------
ABRT:abort(3) からの中断 (Abort) シグナル
[j05002@~]% cat
Abort
[j05002@~]%

-----------------------------------
BUS:バスエラー (不正なメモリアクセス)
[j05002@~]% cat
Bus error
[j05002@~]%

-----------------------------------
FPE:浮動小数点例外
[j05002@~]% cat
Floating exception
[j05002@~]%

-----------------------------------
KILL:Killシグナル. 完全強制終了
[j05002@~]% cat
Killed
[j05002@~]%

-----------------------------------
USR1:ユーザ定義シグナル 1
[j05002@~]% cat
User signal 1
[j05002@~]%

-----------------------------------
SEGV:不正なメモリ参照
[j05002@~]% cat
Segmentation fault
[j05002@~]%

-----------------------------------
USR2:ユーザ定義シグナル 2
```

```
[j05002@~]% cat
User signal 2
[j05002@~]%

-------------------------------------
PIPE:パイプ破壊．読み手の無いパイプへの書き出し
[j05002@~]% cat
Broken pipe
[j05002@~]%

-------------------------------------
ALRM:alarm(2) からのタイマーシグナル
[j05002@~]% cat
Alarm clock
[j05002@~]%

-------------------------------------
TERM:終了 (termination) シグナル
[j05002@~]% cat
Terminated
[j05002@~]%

-----------------------------------------------------------------------------------------
課題8 実行形式プログラムからのプロセス生成

ソースの場所:/home/y05/j05002/OS/report3/ex8/process-create.c
作成時間:１５分

#include <stdlib.h>

int
main(int argc, char *argv[])
{
process_create_l("/usr/bin/who","who",0 );
process_create_l("/bin/ps","ps","l",0);
return 0;

}
省略


・実行結果
[j05002@report2]% ./process-create
[j05002@report2]% UID PID PPID CPU PRI NI VSZ RSS WCHAN STAT TT TIME COMMAND
502 20773 20772 0 31 0 31836 992 - S+ p1 0:00.21 -tcsh
502 20878 20877 0 31 0 31836 1000 - R p2 0:00.41 -tcsh
502 21220 20878 0 0 0 0 0 - Z+ p2 0:00.00 (process-create)
502 21221 1 0 31 0 27372 360 - R+ p2 0:00.01 who
j05002 console Nov 2 01:42
j05002 ttyp1 Nov 11 18:52
j05002 ttyp2 Nov 11 19:04
j05002 ttyp3 Nov 4 22:59


本来なら、実行結果は
"who の実行結果" -> "ps l の実行結果" -> "プロンプト"
という順番で出力されるはずだが、実際には
"プロンプト" -> "ps l の実行結果" -> "who の実行結果"
となっている。
これは、"process-create1" から作られた子プロセスである"who"と"ps u"のプ
ロセスが
終了する前に親プロセスである"process-create1"が終了してしまっているから
である。
```

課題9 wait()システム・コールの利用

ソースの場所:/home/y05/j05002/OS/report3/ex9/process-create2.c
作成時間:10分

```
#include <stdlib.h>

main()
{
process_create_l("/usr/bin/cal","cal","5","1995",0 );
wait(0);

}

extern char **environ;


process_create_l( file,a0,a1,a2,a3,a4,a5,a6,a7,a8,a9 )
char *file,*a0,*a1,*a2,*a3,*a4,*a5,*a6,*a7,*a8,*a9 ;
{
int child_pid ;
省略
}
```

・実行結果

```
[j05002@ex9]% ./process-create2
May 1995
S M Tu W Th F S
1 2 3 4 5 6
7 8 9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
```

親プロセスであるprocess-create2が,子供のプロセスであるcalを待たずに
終了し,実行結果の表示が乱れてしまうという問題を解決するため,
calプロセスが終了してから,process-create2プロセスが終了するように,
wait()を用いて改良した.

課題10 process_create_l()関数の改良

ソースの場所:/home/y05/j05002/OS/report3/ex10/process-create3.c
作成時間:40分

```
#include <unistd.h>
#include <stdarg.h>

void exit(int status);
void perror(char *s);

int process_create_l(char *file, int ac, char *a, ...);

int
main(int argc , char *argv[])
{
process_create_l("/bin/ls",2,"ls","-l",0);
wait(0);
process_create_l("/usr/bin/cal",3,"cal","12","2006",0 );
省略
}
extern char **environ;
```

```
int
process_create_l( char *file,int av,char *aq,...)
{
int child_pid ;
if( (child_pid=fork()) == 0 )
{
char *argv[av];
va_list ap;
int i;

va_start(ap,aq);
argv[0] = aq;
for(i=1; i < av; i++) {
argv[i] = va_arg(ap, char *);
}
argv[av] = 0;
va_end(ap);

execve( file, argv, environ );
perror( file );
exit( 1 );
}
省略
}
```

stdargは個数・型が可変な引数リストである.
va_list 型が宣言されており、3 つのマクロが定義されている.これらを用いると,
呼び出された関数側では個数や型を知らない引き数のリストを,順に読み込むこ
とができる。
上のプログラムはstdargを用いて,最大10個までしか引数を取ることができない
という
制限を緩和した.

課題11 Java によるプロセス生成

ソースの場所:/home/y05/j05002/OS/report3/processExample/Create.java
作成時間:30分

```
package processExample;
import java.io.*;

public class Create {
static final int BUFSIZE = 4096;

static public void main(String [] args) throws IOException {
Process p1 = new ProcessBuilder("/bin/ls","-l").start();
Process p2 = new ProcessBuilder("/usr/bin/who").start();
InputStream out1 = p1.getInputStream();
InputStream out2 = p2.getInputStream();
byte[] buf1 = new byte[BUFSIZE];
byte[] buf2 = new byte[BUFSIZE];
int length1,length2;
while((length1=out1.read(buf1))>0 | (length2=out2.read(buf2))>0) {
System.out.write(buf1,0,length1);
System.out.write(buf2,0,length2);
}
}
}
```

・実行結果

```
[j05002@report3]% java processExample/Create
total 0
```

```
drwxr-xr-x 4 j05002 j05002 136 Dec 3 14:45 ex10
drwxr-xr-x 5 j05002 j05002 170 Dec 3 11:20 ex8
drwxr-xr-x 5 j05002 j05002 170 Dec 3 11:18 ex9
drwxr-xr-x 4 j05002 j05002 136 Jan 10 17:52 processExample
j05002 console Jan 9 21:41
j05002 ttyp1 Jan 10 17:38
```

"ls -l","who"の実行結果をout1,out2に代入し、out1,out2の長さが0より大きいならば
System.out.write()によって結果を出力している。

Java API documentの
java.lang.Object
|_java.io.Writer
   |_java.io.BufferedWriter
のwriteメソッドを見ると
通常このメソッドは、指定された配列からこのストリームのバッファへ文字を格納し、必要に応じて基本となるストリームにバッファをフラッシュする。
と書いてあった。
つまり、子プロセスの終了を待たずに、親プロセスが終了するのを防ぐためにwhile文中の操作が必要だと考えられる。