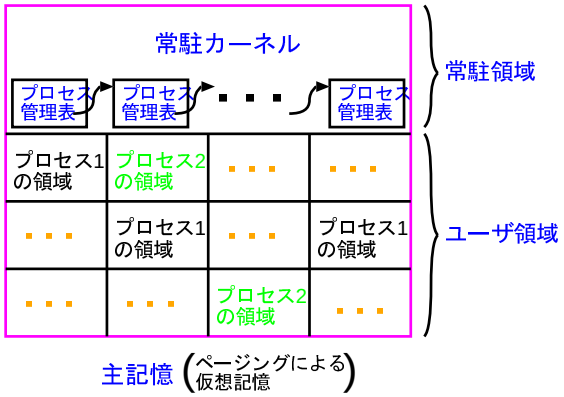
11 自習 プロセス情報獲得のシステム

プロセス管理表： ··· (p.88からの引用)

プロセスの管理／seamlessな切替えのためにOSが必要とする情報は全てプロセス管理表に保持され、主記憶内の常駐領域に置かれる。プロセス管理表には次の様な情報が入っている。

- 名前 ··· 人間がプロセスを識別するために付けた名前
- 識別子 ··· プロセスの識別子(PID), ...
- 所有者情報 ··· プロセスの所有者のID, グループID
- 動作状態 ··· プロセスの動作状態,...
- 時間情報 ··· これまでに消費したCPU時間,...
- 走行情報 ··· プロセス停止時点での...
- プログラム情報 ··· プロセスがどのロードモジュールを使って実行しているかを示す。
- 空間情報 ··· プロセスが利用している実メモリ空間や仮想メモリ空間に関する情報



プロセスのID, 実ユーザID, 実効ユーザID, ... を調べる

プロセスIDを得るためのシステムコール pid_t getpid(void) :
プロセスグループIDを得るためのシステム... pid_t getpgrp(void) :
親プロセスのIDを得るためのシステムコール pid_t getppid(void) :

プロセスの実ユーザIDを得るためのシステム... uid_t getuid(void) :
プロセスの実効ユーザIDを得るための..... uid_t geteuid(void) :
プロセスの実グループIDを得るための..... gid_t getgid(void) :
プロセスの実効グループIDを得るための..... gid_t getegid(void) :

指定したid番号のユーザ情報を得るための.....
struct passwd *getpwuid(uid_t uid) :
login名を得るためのライブラリ関数 char *getlogin(void) :
指定したユーザ名のユーザ情報を得るための.....
struct passwd *getpwnam(char *name) :

プロセスの消費時間を調べる

消費したCPU時間を得るためのシステムコール

clock_t times(struct tms *tp) :

時刻を知るためのシステムコール time_t time(time_t *tp) :

時刻を設定するためのシステムコール int stime(time_t *tp) :

指定時間経過後にソフトウェア割り込みを発生させるシステムコール

unsigned int alarm(unsigned int seconds) :

プロセスの優先度,資源利用制限,資源利用情報を調べる

プロセスの優先度を得るためのシステムコール

int getpriority(int which, int who) :

プロセスの資源制限情報を得るためのシステムコール

int getrlimit(int resource, struct rlimit *rlimits) :

プロセスの資源利用情報を得るためのシステムコール

int getrusage(int who, struct rusage *usage) :

自習

11–1 プロセスのID, 実ユーザID, 実効ユーザID, ... を調べる

まず最初に、プロセスのID, グループID, 親プロセスのID, 実ユーザID, 実効ユーザID, 実グループユーザID, 実効グループユーザID を調べるためのシステムコールを紹介する。

プロセスIDを得るためのシステムコール pid_t getpid(void) :

- ヘッダファイル <unistd.h> を必要とする。
- getpid( ) が呼ばれると、(呼んだ) プロセスのID番号が返される。
- プロセスIDのデータ型 pid_t の実体は int である。

例えば VineLinux2.1.4の場合 、
/usr/include/ bits/types.h で

```
typedef int __pid_t;
```

と定義され、/usr/include/unistd.h で

```
typedef __pid_t pid_t;
```

と定義されている。

プロセスグループIDを得るためのシステムコール

pid_t getpgrp(void) :

- ヘッダファイル <unistd.h> を必要とする。
- getpgrp( ) が呼ばれると、(呼んだ) プロセスのグループIDが返される。

親プロセスのIDを得るためのシステムコールpid_t getppid(void) :

- ヘッダファイル <unistd.h> を必要とする。
- getppid( ) が呼ばれると、親プロセスのID番号が返される。

プロセスの実ユーザIDを得るためのシステムコール

uid_t getuid(void) :

- ヘッダファイル <unistd.h> を必要とする。
- getuid() が呼ばれると、プロセスの実ユーザIDが返される。

実ユーザID(real user ID) :
プロセスを起動したユーザのIDのこと。

プロセスの実効ユーザIDを得るためのシステムコール

uid_t geteuid(void) :

- ヘッダファイル <unistd.h> を必要とする。
- geteuid() が呼ばれると、プロセスの実効ユーザIDが返される。

実効ユーザID(effective user ID) :
アクセス許可の判定に使用されるユーザIDのこと。通常は実ユーザIDと同じであるが、setuid()システムコールによって変更されることもある。

プロセスの実グループIDを得るための システムコール gid_t getgid(void) :

- ヘッダファイル <unistd.h> を必要とする。
- getgid() が呼ばれると、プロセスの実グループIDが返される。

実グループID(real user ID) :
プロセスを起動したユーザのグループIDのこと。

プロセスの実効グループIDを得るための システムコール gid_t getegid(void) :

- ヘッダファイル <unistd.h> を必要とする。
- getegid() が呼ばれると、プロセスの実効グループIDが返される。

実効グループID(effective user ID) :
アクセス許可の判定に使用されるグループIDのこと。通常は実グループIDと同じであるが、setgid() システムコールによって変更されることもある。

例11. 1 (プロセスに関連する各種idを表示する)　起動したプロセスに関する各種id情報を表示するプログラムを次に示す。

```
[motoki@x205a]$  nl show-various-ids-on-process.c
 1 /*****************************************************
 2 /* Operating-Systems/C-Programs/show-various-ids-on-process
 3 /*-----------------------------------------------------
 4 /* getpid(),getpgrp(),getppid(),
 5 /* getuid(),geteuid(),getgid(),getegid()システムコール使...*
 6 /* C.オール「例題で学ぶLinuxプログラミング」ピアソン,4.2.1-2...
 7 /*****************************************************
 8 #include <stdio.h>
 9 #include <unistd.h>      /* for getpid(),getpgrp(),getppid()
10                          /*  getuid(),geteuid(),getgid(),
11                          /*  and getegid() system calls

12 int main(void)
13 {
```

```
14   printf("Process ID         = %d\n", getpid());
15   printf("Process Group ID   = %d\n", getpgrp());
16   printf("Parent Process ID = %d\n", getppid());

17   printf("Real User ID        = %d\n", getuid());
18   printf("Effective User ID   = %d\n", geteuid());
19   printf("Real Group ID       = %d\n", getgid());
20   printf("Effective Group ID = %d\n", getegid());
21
22   return 0;
23 }
[motoki@x205a]$  gcc show-various-ids-on-process.c
[motoki@x205a]$  ./a.out
Process ID         = 12480
Process Group ID   = 12480
Parent Process ID = 10307
Real User ID        = 500
```

```
Effective User ID  = 500
Real Group ID      = 500
Effective Group ID = 500
[motoki@x205a]$
```

ユーザのid番号が分かれば、ユーザ名を知ることが出来る。

指定したid番号のユーザ情報を得るためのライブラリ関数

struct passwd *getpwuid(uid_t uid) :

- ヘッダファイル <pwd.h> と <sys/types.h> を必要とする。
- パスワードファイル(/etc/passwd)の中から引数で指定されたユーザidを要素とするエントリを探し、そのエントリ(struct passwd型構造体)へのポインタを返す。 検索できない場合はNULLを返す。
- 間数値の指すデータ型である struct passwd は /usr/include/pwd.h の中で次の様に定義される。

```
struct passwd {
  char *pw_name;     /* Username. */
  char *pw_passwd;   /* Password. */
  uid_t pw_uid;      /* User ID.  */
  gid_t pw_gid;      /* Group ID. */
  char *pw_gecos;    /* Real name.*/
     ............
};
```

例11. 2 (プロセスの実ユーザをlogin名で表示する)　起動したプロセスの実ユーザのid番号とlogin名を表示するプログラムを次に示す。

```
[motoki@x205a]$  nl show-real-username-of-process.c
 1 /*****************************************************
 2 /* Operating-Systems/C-Programs/show-real-username-of-proce
 3 /*-----------------------------------------------------
 4 /* getuid()システムコール,getpwduid()ライブラリ関数の使用例　*
 5 /* C.オール「例題で学ぶLinuxプログラミング」ピアソン,4.2.4節 */
 6 /*****************************************************
 7 #include <stdio.h>
 8 #include <unistd.h>     /* for getuid() system calls*/
 9 #include <pwd.h>        /* for getpwuid() library function
10 #include <sys/types.h>  /* for getpwuid() library function

11 int main(void)
12 {
13   uid_t uid;
```

```
14    struct passwd *pwd;
15    char *username;

16    uid = getuid();
17    if ((pwd=getpwuid(uid)) == NULL)
18      username = pwd->pw_name;
19    else
20      username = "Unknown";
21    printf("Real User ID = %d (name: %s)\n",
             uid, username);

22    return 0;
23 }
[motoki@x205a]$  gcc show-real-username-of-process.c
[motoki@x205a]$  ./a.out
Real User ID = 500 (name: motoki)
[motoki@x205a]$
```

login名を知るだけなら、getuid() と getpwuid() を組み合わせて使う必要はなく、次のライブラリ関数を利用することが出来る。

login名を得るためのライブラリ関数 char *getlogin(void) :

- ヘッダファイル <unistd.h> を必要とする。
- getlogin() が呼ばれると、login名へのポインタが返される。 失敗した場合はNULLが返される。

指定したユーザ名のユーザ情報を得るためのライブラリ関数 struct passwd *getpwnam(char *name) :

- ヘッダファイル <pwd.h> と <sys/types.h> を必要とする。
- パスワードファイル(/etc/passwd)の中から引数で指定されたユーザ名を要素とするエントリを探し、そのエントリ(struct passwd型構造体)へのポインタを返す。 検索できない場合はNULLを返す。
- 間数値の指すデータ型である struct passwd は/usr/include/pwd.hの中でp.151の様に定義された構造体である。

例11. 3 (プロセスのユーザ名を表示する)　起動したプロセスのユーザ名をlogin名とフルネームで表示するプログラムを次に示す。

```
[motoki@x205a]$  nl show-loginname-of-process.c
 1 /*****************************************************
 2 /* Operating-Systems/C-Programs/show-loginname-of-process.c
 3 /*-----------------------------------------------------
 4 /* getlogin(),getpwdnam()ライブラリ関数の使用例 */
 5 /* C.オール「例題で学ぶLinuxプログラミング」ピアソン,4.2.4節 */
 6 /*****************************************************
 7 #include <stdio.h>
 8 #include <stdlib.h>     /* for exit() library function */
 9 #include <unistd.h>     /* for getlogin() library function
10 #include <pwd.h>        /* for getpwuid() library function
11 #include <sys/types.h>  /* for getpwuid() library function

12 int main(void)
13 {
```

```
14    char *loginname;
15    struct passwd *pwd;

16    if ((loginname=getlogin()) == NULL) {
17      perror("getlogin");
18      exit(EXIT_FAILURE);
19    }

20    if ((pwd=getpwnam(loginname)) == NULL) {
21      perror("getpwnam");
22      exit(EXIT_FAILURE);
23    }
24    printf("Login name = %s  (Real name : %s)\n",
25           loginname, pwd->pw_gecos);

26    return 0;
27 }
```

```
[motoki@x205a]$  gcc show-loginname-of-process.c
[motoki@x205a]$  ./a.out
Login name = motoki  (Real name : MOTOKI Tatsuya)
[motoki@x205a]$
```

11–2 プロセスの消費時間を調べる

OSの持っているタイマ機能を、我々は次の様なシステムコールを通して利用することが出来る。

タイマ機能を使って、
OSは各プロセスが消費したCPU時間を把握しているし、またプロセスにCPUを割り当てる際にCPUを連続使用できる時間の上限を設定している。

消費したCPU時間を得るためのシステムコール

`clock_t times(struct tms *tp)`：

- ヘッダファイル `<sys/types.h>` と`<sys/times.h>` を必要とする。
- システム起動時からの経過時間(実時間)をシステム内のクロック刻みの回数で表したものが関数値として返される。 秒単位に直すには、この値をどこかで定義されたマクロ `CLK_TCK` で割る。

補足：
ライブラリ関数 `clock()` で得られたクロック刻みの回数の場合は、秒単位に直すためには `CLOCKS_PER_SEC` という`<time.h>`で定義されたマクロで割るが、`times()`関数の場合は `CLK_TCK` で割る。

- 失敗した時は -1 が返される。
- struct tms 型変数へのポインタを引数として指定すると、関数終了時にはこの変数の構造体要素に 自プロセスと子プロセスが現在までに消費したCPU時間、およびそれらに関してシステムの消費したCPU時間の情報が設定される。これらの値もクロック刻みの回数なので、秒単位に直すには CLK_TCK で割る必要がある。

```
struct tms {
  clock_t  tms_utime;   /* ユーザCPU消費時間    */
  clock_t  tms_stime;   /* システムCPU消費時間 */
  clock_t  tms_cutime;  /* 子プロセスのユーザCPU消費時間 */
  clock_t  tms_cstime;  /* 子プロセスのシステムCPU消費時間 *
};
```

時刻を知るためのシステムコール time_t time(time_t *tp) :

- ヘッダファイル <sys/types.h> と<time.h> を必要とする。
- time( ) が呼ばれると、**1970年1月1日0時グリニッジ標準時からの経過秒数** (カレンダー時間) を関数値として返される。
- 失敗した時は -1 が返される。
- time_t 型変数へのポインタを引数として指定すると、
  関数終了時にはこの変数にも経過秒数が設定される。

  ローカル時間を表すASCII文字列に変換するには標準ライブラリ関数 ctime( ) を使う。

時刻を設定するためのシステムコール int stime(time_t *tp) :

- ヘッダファイル <sys/types.h> と<time.h> を必要とする。
- スーパユーザだけが使用可。
- stime( ) が呼ばれると、コンピュータの保持する時計の時刻が引数で指定された通り (グリニッジ標準時) に設定される。 この時刻設定に成功すると 0 が返され、失敗すると −1 が返される。 (例えば、スーパユーザ以外が呼び出すと失敗する。)
- 関数引数の tp には、カレンダー時間 (i.e.1970年1月1日00:00::00からの通算秒) を保持したtime_t型変数へのポインタを与える。

  struct tm型で表されたローカル時間をカレンダー時間に変換するには標準ライブラリ関数mktime( )を使う。

指定時間経過後にソフトウェア割り込みを発生させるシステムコール

`unsigned int alarm(unsigned int seconds)`:

- ヘッダファイル `<unistd.h>` を必要とする。
- `alarm()` が呼ばれると、引数で指定された秒数後にSIGALRMシグナル が自プロセスに送られて来る。
- 関数値はSIGALRMシグナルまでの秒数である。
- 端末からの入力打ち切りの時間を設定したい時などに使える。
- 再度この関数を呼び出すと、SIGALRMシグナルが送られるまでの時間はリセットされる。

例11. 4 (プロセスの消費したCPU時間を表示する; 2004年頃のVineLi

別に作られた、

> 100要素の整数をbubblesortする作業を30000回繰り返して
> 1回当たりの処理時間を割り出すプログラム
> (実行プログラム名： clock-bubblesort-100)

を子プロセスとして起動した上で、その時にプロセスと子プロセスで消費されるCPU時間を表示するプログラムを次に示す。

```
[motoki@x205a]$  nl show-CPU-time-of-process.c
 1 /*****************************************************
 2 /* Operating-Systems/C-Programs/show-CPU-time-of-process.c
 3 /*-----------------------------------------------------
 4 /* times()システムコールを使って、
 5 /*      100要素の整数をbubblesortする作業を30000回繰り返して
 6 /*      1回当たりの処理時間を割り出すプログラム(clock-bubblesor
 7 /* の計算時間を測ってみる。
 8 /* C.オール「例題で学ぶLinuxプログラミング」ピアソン,4.2.5節 */
 9 /*****************************************************
```

```
10 #include <stdio.h>
11 #include <stdlib.h>      /* for exit() and system() library
12 #include <sys/types.h>   /* for times() system call */
13 #include <sys/times.h>   /* for times() system call */
14 #include <time.h>        /* CLOCKS_PER_SEC macro    */

15 #define  CLK_TCK  (CLOCKS_PER_SEC/10000)
16     // 古いOSでは CLK_TCK が定義されていたが、CLK_TCK 自体が

17     // 古く消える運命にあるマクロらしいので、VineLinux5.2の
18     // 下でも実習室のDebianの下でも CLK_TCK を定義している
19     // 標準のヘッダファイルは見当たらない。
20     // ==>ここでは、#defineで実行結果に合わせてCLK_TCKを設定
21     //     (VineLinux5.2の下でも実習室のDebianでもこれでOK)

22 int main(void)
23 {
```

```
24    clock_t    start_clock, end_clock;
25    struct tms start, end;

26    start_clock = times(&start);
27    system("./clock-bubblesort-100");
28    end_clock = times(&end);

29    printf("\n-------------------------\n");
30    printf("elapsed real time: %6.2f sec\n",
31         (double)(end_clock-start_clock)/CLK_TCK);
32    printf("In this process,\n");
33    printf("   user CPU time: %6.2f sec\n",
34         (double)(end.tms_utime-start.tms_utime)/CLK_TCK);
35    printf("    sys CPU time: %6.2f sec\n",
36         (double)(end.tms_stime-start.tms_stime)/CLK_TCK);
37    printf("In the child process (clock-bubblesort-100),\n");
38    printf("   user CPU time: %6.2f sec\n",
```

```
39          (double)(end.tms_cutime-start.tms_cutime)/CLK_TCK);
40    printf("     sys CPU time: %6.2f sec\n",
41          (double)(end.tms_cstime-start.tms_cstime)/CLK_TCK);

42    return 0;
43 }
[motoki@x205a]$  gcc show-CPU-time-of-process.c
[motoki@x205a]$  ./a.out
Clocking the execution time of the program that sorts 100 elem
    (Bubblesort)
Input a random seed (0 - 2147483647):  333
Process time =  0.044 m sec
Real    time =  0.033 m sec

------------------------
elapsed real time:   3.56 sec
In this process,
```

1回当りの処理時間, by clock-bubbles

```
   user CPU time:   0.00 sec
    sys CPU time:   0.00 sec
In the child process (clock-bubblesort-100),
   user CPU time:   1.31 sec
    sys CPU time:   0.00 sec
[motoki@x205a]$
```

≒ 0.044*10^{-3}*30000

11–3 プロセスの優先度,資源利用制限,資源利用

プロセスの優先度を得るためのシステムコール

`int getpriority(int which, int who)`:

- ヘッダファイル `<sys/resource.h>` を必要とする。
- 引数で指定されたプロセスの優先度が返される。.....

  補足：
  Linuxの場合、優先度は -20~ $+20$ の整数値で...

- エラーが発生した場合は -1 が返され、errno にエラー番号が設定される。
- 関数引数の `which` は第2引数whoの扱い方を表す定数で、ここには PRIO_PROCESS, PRIO_PGRP または PRIO_USER のいずれかを指定する。
- 関数引数の `who` には

  whichが PRIO_PROCESS の時はプロセスIDを、
  whichが PRIO_PGRP の時はプロセスグループIDを、
  whichが PRIO_USER の時はユーザIDを

  指定する。

プロセスの資源制限情報を得るためのシステムコール

`int getrlimit(int resource, struct rlimit *rlimits) :`

- ヘッダファイル <sys/resource.h> と<sys/types.h>, <sys/ time.h> を必要とする。

- getrlimit( ) が呼ばれると、第1引数で指定された資源項目についての制限情報が調べられ、第2引数の指す構造体領域にセットされる。成功すると 0 が返され、失敗すると −1 が返される。

- 関数引数の resource は資源項目を表す定数で、次のいずれかを指定する。

  RLIMIT_CORE ···コアダンプファイルサイズの上限(バイト単位)
  RLIMIT_CPU ···CPU時間の上限(秒単位)
  RLIMIT_DATA ···データ(malloc()/sbrk())セグメントの上限(バイ
  RLIMIT_FSIZE ···ファイルサイズの上限(バイト単位)
  RLIMIT_NOFILE···オープンするファイル数の上限(個数)
  RLIMIT_STACK ···スタックサイズの上限(バイト単位)
  RLIMIT_AS ···アドレス空間(スタックおよびデータ) の上限(バイ

- 関数引数の rlimits には、制限情報を受け取るための struct rlimit 型構造体へのポインタを指定する。

- 制限情報を表すデータ型である struct rlimit は /usr/include/sys/resource.h の中で次の様に定義された構造体である。

```
struct rlimit
  {
    rlim_t rlim_cur;  /* The current (soft) limit. */
    rlim_t rlim_max;  /* The hard limit. */
  };
```

ここで、データ型 rlim_t の実体は int である。構造体要素 rlim_cur はそのプロセスに対して効力のある現在の上限を表し、rlim_max は rlim_cur がとることの出来る最大値を表す。

プロセスの資源利用情報を得るためのシステムコール

`int getrusage(int who, struct rusage *usage) :`

- ヘッダファイル `<sys/resource.h>` と`<sys/types.h>`, `<sys/ time.h>` を必要とする。
- 指定されたプロセスがCPUやメモリを始めとする資源をどれだけ使い、またページフォールトやスワップが何回起きたか、などが調べられ、それらの利用情報が第2引数の指す構造体領域にセットされる。成功すると 0 が返され、失敗すると −1 が返される。
- 関数引数の `who` はプロセスを指定するための定数で、次のいずれかを指定する。

  `RUSAGE_SELF` ···自プロセス(i.e.getrusage( ) を呼んだプロセス)
  `RUSAGE_CHILDREN` ···終了済の全ての子プロセス(の消費した資源の総
  `RUSAGE_BOTH` ···自プロセスと
  終了済の全ての子プロセスの消費した資源の総計

- 関数引数の `usage` には、利用情報を受け取るための `struct rusage` 型構造体へのポインタを指定する。

- 制限情報を表すデータ型である struct rusage は /usr/include/sys/resource.h の中で次の様に定義された構造体である。

```
struct rusage
  {
    struct timeval ru_utime;  /* Total amount of user time used.  */
    struct timeval ru_stime;  /* Total amount of system time used.*/
    long ru_maxrss;   /* Maximum resident set size (in kilobytes).*/
    long ru_ixrss;    /* Amount of sharing of text segment memory
                         with other processes (kilobyte-seconds).*/
    long ru_idrss;    /* Amount of data segment memory used
                         (kilobyte-seconds).  */
    long ru_isrss;    /* Amount of stack memory used
                         (kilobyte-seconds).  */
    long ru_minflt;   /* Number of soft page faults (i.e. those
                         serviced by reclaiming a page from the
                         list of pages awaiting reallocation. */
    long ru_majflt;   /* Number of hard page faults
                         (i.e. those that required I/O).  */
    long ru_nswap;    /* Number of times a process was swapped
                         out of physical memory.  */
    long ru_inblock;  /* Number of input operations via the file
                         system. Note: This and `ru_oublock' do
                         not include operations with the cache. */
    long ru_oublock;  /* Number of output operations via the
                         file system.  */
```

```
    long ru_msgsnd;   /* Number of IPC messages sent.  */
    long ru_msgrcv;   /* Number of IPC messages received.  */
    long ru_nsignals; /* Number of signals delivered.  */
    long ru_nvcsw;    /* Number of voluntary context switches,
                         i.e. because the process gave up the
                         process before it had to (usually to
                         wait for some resource to be available).*/
    long ru_nivcsw;   /* Number of involuntary context switches,
                 i.e. a higher priority process became
                         runnable or the current process used up
                         its time slice.  */
  };
```

この中に現れる struct timeval という構造体は、更に、/usr/include/sys/time.h の中で次の様に定義されている。

```
  struct timeval
    {
      time_t tv_sec;     /* Seconds.      */
      time_t tv_usec;    /* Microseconds. */
    };
```

ここで、データ型 time_t の実体はlong intである。

例11. 5 (プロセスの消費したCPU時間を表示する2) 別に作られた、

> 100要素の整数をbubblesortする作業を30000回繰り返して
> 1回当たりの処理時間を割り出すプログラム
> (実行プログラム名： clock-bubblesort-100)

を子プロセスとして実行させ、その後で

プロセスの優先度,
プロセスに設定されたCPU時間の上限,
ファイルサイズの上限,
プロセスの消費したCPU時間,
ページフォールトの回数

を表示させるプログラムを次に示す。

```
[motoki@x205a]$  nl show-resource-info-of-process.c
 1 /*****************************************************
 2 /* Operating-Systems/C-Programs/show-resource-info-of-proce
 3 /*-----------------------------------------------------
 4 /* ライブラリ関数system()を使って、
 5 /*      100要素の整数をbubblesortする作業を30000回繰り返して
```

```
 6 /*      1回当たりの処理時間を割り出すプログラム(clock-bubbleso
 7 /* を子プロセスとして実行させ、その後で                          *
 8 /*      プロセスの優先度，CPU時間の上限，
 9 /*      ファイルサイズの上限，消費したCPU時間，
10 /*      ページフォールトの回数
11 /* を表示させてみる。
12 /* C.オール「例題で学ぶLinuxプログラミング」ピアソン,4.2.5節 */
13 /* N.Matthew&R.Stones「Linuxプログラミング」ソフトバンク4.8...
14 /*****************************************************
15 #include <stdio.h>
16 #include <stdlib.h>     /* for exit() and system() library
17 #include <sys/resource.h> /* for getpriority(), getrlimit(
18                           /* and getrusage() system calls *
19 #include <sys/types.h>  /* for getrlimit() and getrusage()
20 #include <sys/time.h>   /* for getrlimit() and getrusage()

21 int main(void)
```

```
22 {
23   struct rlimit r_limit;
24   struct rusage r_usage;

25   system("./clock-bubblesort-100");

26   printf("\n-------------------------\n");
27   printf("Current priority = %d\n",
           getpriority(PRIO_PROCESS, getpid()));

28   getrlimit(RLIMIT_CPU, &r_limit);
29   printf("Limit of CPU time:   current = %d, hard = %d\n",
30          r_limit.rlim_cur, r_limit.rlim_max);
31   getrlimit(RLIMIT_FSIZE, &r_limit);
32   printf("Limit of file size:  current = %d, hard = %d\n",
33          r_limit.rlim_cur, r_limit.rlim_max);
```

```
34   getrusage(RUSAGE_SELF, &r_usage);
35   printf("In this process,\n");
36   printf("  |user CPU time: %6.2f sec\n",
37          (double)r_usage.ru_utime.tv_sec
38          + (double)r_usage.ru_utime.tv_usec*1e-6);
39   printf("  | sys CPU time: %6.2f sec\n",
40          (double)r_usage.ru_stime.tv_sec
41          + (double)r_usage.ru_stime.tv_usec*1e-6);
42   printf("  |num. of soft page faults: %d\n",
          r_usage.ru_minflt);
43   printf("  |num. of hard page faults: %d\n",
          r_usage.ru_majflt);

44   getrusage(RUSAGE_CHILDREN, &r_usage);
45   printf("In the child process (clock-bubblesort-100),\n");
46   printf("  |user CPU time: %6.2f sec\n",
47          (double)r_usage.ru_utime.tv_sec
```

```
48          + (double)r_usage.ru_utime.tv_usec*1e-6);
49    printf("  | sys CPU time: %6.2f sec\n",
50          (double)r_usage.ru_stime.tv_sec
51          + (double)r_usage.ru_stime.tv_usec*1e-6);
52    printf("  |num. of soft page faults: %d\n",
            r_usage.ru_minflt);
53    printf("  |num. of hard page faults: %d\n",
            r_usage.ru_majflt);

54    return 0;
55 }
[motoki@x205a]$  gcc show-resource-info-of-process.c
[motoki@x205a]$  ./a.out
Clocking the execution time of the program that sorts 100 elem
    (Bubblesort)
Input a random seed (0 - 2147483647):  333
Process time =  0.044 m sec
```

1回当りの処理時間, by clock-bubbles

```
Real    time =  0.033 m sec

------------------------
Current priority = 0
Limit of CPU time:   current = -1, hard = -1
Limit of file size:  current = -1, hard = -1
In this process,
  |user CPU time:   0.00 sec
  | sys CPU time:   0.00 sec
  |num. of soft page faults: 166
  |num. of hard page faults: 0
In the child process (clock-bubblesort-100),
  |user CPU time:   1.32 sec
  | sys CPU time:   0.00 sec
  |num. of soft page faults: 423
  |num. of hard page faults: 0
[motoki@x205a]$
```

= 0.149*10^{-3}*30000