第1回 strace/ltraceゼミ

ゼミ担当者 ： 澤田 淳二，森 隆史，永松 秀人
指導院生 ： 福永 隆宏
開催日 ： 2003年3月31日

ゼミ内容: このゼミでは，strace/ltraceの使い方を説明する．straceは，システムコールとシグナルを追跡するプログラムであり，ltraceは，ダイナミックリンクライブラリの呼び出しを追跡するプログラムである．これらのツールを利用することで，予期せぬシステムコールやシグナル，ダイナミックリンクライブラリコールを発見でき，バグの修正につなげることが可能となる．

1 システムコール，シグナルとダイナミックリンクライブラリコール

1.1 システムコール

システムコールとは，OSの提供するAPI[1]のことである．例えば，現在の時刻を取得するには，time()関数を利用するが，これもシステムコールの1つである．また，普段は意識しないが，printf()などのC言語の関数を利用する場合も，printf()関数の内部でシステムコールが使われて，画面出力がされている．

1.2 シグナル

シグナルとは，プログラムにOSから送られてくる各種信号のことである．例えば，SIGINTというシグナルは，ユーザがCtrl-Cを入力した時に送られてくる．どのシグナルがどんな意味を持つかは，

```
$ man 7 signal
```

とすれば調べられる．なお，killコマンドを使うことで，ユーザがシグナルを送信することも可能である．

1.3 ダイナミックリンクライブラリコール

C言語では，多くのプログラムでprintf()やscanf()といった関数が用いられる．これらの関数を使うプログラムが多くあった場合，関数のコードをプログラムごとに持っていたのでは，ディスクに無駄が生じてしまう．また，関数のコードにバグがあった場合，すべてのプログラムをリンクしなおさないといけなくなる．これらの問題点を解決したものが，**ダイナミックリンクライブラリ**である[2]．この方式は，UNIXのほか，Windowsでも用いられている．

1.4 バグの発見

プログラムにバグがあった場合，プログラムは異常終了することがある．このような時に，システムコールやシグナル，ダイナミックライブラリコールを追跡すると，渡された引数や送られてきたシグナルがわかり，何故，異常終了してしまうかがわかり，問題の解決につながる．

2 straceの利用方法

2.1 システムコールの追跡

straceでプログラムの呼んでいるシステムコールを追跡するには，

```
$ strace 実行ファイル名
```

とすればよい．別途，専用のライブラリを組み込むという処理は必要ない．

Fig. 1で示す，“Hello, world!”と表示するだけの簡単なプログラムを作成し，straceを用いてプログラムがどのようなシステムコールを呼び出しているかを調べる．このプログラムをコンパイルし，straceを使ってシステムコールを追跡すると，Fig. 2のような出力がされる．“Hello, world”と表示するだけでも裏で多くのシステムコールが用いられていることがわかる[3]．C言語で，printf()で出力した“Hello, world!”は，システムコールのレベルでは，write()というシステムコールが用いられていることがわかる．

[1] Application Programming Interface
[2] リンク時に必要なライブラリをリンクする方式をスタティックリンクと呼ぶ
[3] 主にプログラムの起動に関するものである

```
#include <stdio.h>

int main()
{
  printf("Hello, world!\n");
  return 0;
}
```

Fig. 1　“Hello, world!” と表示するプログラム

```
execve("./hello", ["./hello"], [/* 36 vars */]) = 0
uname({sys="Linux", node="eslin", ...}) = 0
brk(0)                                  = 0x80495a8
open("/etc/ld.so.preload", O_RDONLY)    = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=28669, ...}) = 0
(中略)
munmap(0x40014000, 28669)               = 0
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 2), ...}) = 0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40014000
write(1, "Hello, world!\n", 14)         = 14
munmap(0x40014000, 4096)                = 0
_exit(0)                                = ?
```

Fig. 2　“Hello, world” と表示する時に使用されているシステムコール

2.2　シグナルの追跡

strace では，システムコール以外にもシグナルを追跡する機能もある．まず，Fig. 3 のようなプログラムを作成する．

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
  int i;
  int *arr;

  arr = (int*)malloc(sizeof(int) * 100);
  for(i = 0; i < 100; i++) arr[i] = i;
  for(i = 0; i < 1000; i++) printf("%d ", arr[i]);
  putchar('\n');
  return 0;
}
```

Fig. 3　不正なメモリを参照しているプログラム

このプログラムを実行すると，Fig. 4 のような出力になる．“Segmentation fault” は，OS の出力でここでプログラムが強制終了されたことを示している．

```
0 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 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56
 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 8
3 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 0 2081 0 0 0 0 0 0 0 0 0 0 0
(中略)
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0Segmentation fau
lt
```

Fig. 4　Segmentation falut の発生

このプログラムを strace にかけると，Fig. 5 のように，SIGSEGV シグナルが送られていることがわかる．

```
(前半部分省略)
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40014000
write(1, "0 1 2 3 4 5 6 7 8 9 10 11 12 13 "..., 1024) = 1024
--- SIGSEGV (Segmentation fault) ---
+++ killed by SIGSEGV +++
```

Fig. 5　SIGSEGV シグナルのキャッチ

1.2 節で示したように，man コマンドを用いて SIGSEGV の意味を調べると，不正なメモリを参照した時に送られてくるシグナルだとわかる．そこで，Fig. 3 を見直すと，表示部分の for ループで，確保した容量の 100 ではなく，1000 となっており，その部分が原因であるとわかる．

2.3　strace のオプション

- -c

  プログラムが終了するまでの各システムコールに要した時間，呼び出した回数，そしてエラーが発生したの回数を表示する．Fig. 1 のプログラムに-c オプションをつけて strace を実行した結果を Fig. 6 に示す．

```
execve("./hello", ["./hello"], [/* 39 vars */]) = 0
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 96.13    0.003081        3081         1           write
  1.50    0.000048          10         5           old_mmap
  0.66    0.000021           7         3         1 open
(中略)
  0.16    0.000005           5         1           read
  0.16    0.000005           5         1           mprotect
  0.12    0.000004           4         1           uname
  0.06    0.000002           2         1           brk
------ ----------- ----------- --------- --------- ----------------
100.00    0.003205                    20         1 total
```

Fig. 6　“Hello, World!” と表示するときの各システムコールの呼び出し統計

- -f

  子プロセスにおけるシステムコールの追跡も行う．子プロセスを持つプログラム例を Fig. 7 に示す．子プロセスは，まず “hello” と出力し，設定された秒数だけ処理を一時停止させる．そして “bye” と出力し終了する．このプログラムに対して，-f オプションをつけて strace を実行した結果を Fig. 8 に示す．

  Fig. 8 において [pid 12672] というのが子プロセスを追跡している様子であり，[pid 12671] は親プロセスにあたる．

```
(前半部分省略)
    int pid;
    scanf("%d", &sec);   /* 秒数を読む */
    pid = fork();        /* 子プロセスを作る */
(省略)
    else if (pid == 0) {
      /* 子プロセスの動作. sec 秒 sleep するだけ */
      fprintf(stderr, "hello\n");
      sleep(sec);
      fprintf(stderr, "bye\n");
      exit(0);         /* 子プロセスを終了する */
    }
    /* 親プロセスの動作           */
    fprintf(stderr,"Good bye see you!!\n");
(後半部分省略)
```

Fig. 7　子プロセスを持つプログラムの例

```
(前半部分省略)
fork()                                  = 12672
[pid 12672] write(2, "hello\n", 6hello
)       = 6
[pid 12672] rt_sigprocmask(SIG_BLOCK, [CHLD], [], 8) = 0
[pid 12672] rt_sigaction(SIGCHLD, NULL, {0x8048664, [CHLD], SA_RESTART|0x4000000}, 8) = 0
[pid 12672] rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
[pid 12672] nanosleep({2, 0},  <unfinished ...>
[pid 12671] write(2, "Good bye see you!!\n", 19Good bye see you!!
) = 19
[pid 12671] read(0,  <unfinished ...>
[pid 12672] <... nanosleep resumed> {2, 0}) = 0
[pid 12672] write(2, "bye\n", 4bye
)         = 4
[pid 12672] _exit(0)                    = ?
<... read resumed> 0x40014000, 1024)    = ? ERESTARTSYS (To be restarted)
--- SIGCHLD (Child exited) ---
```

Fig. 8　子プロセスのシステムコール

- -T

  各システムコールの実行時間計測を行う．Fig. 1 のプログラムを-T オプションをつけて strace を実行した結果を Fig. 9 に示す．何もオプションをつけない場合の結果 (Fig. 2) と比べると，それぞれのシステムコールの最後に<0.000014>といったような時間の項目が追加されているのがわかる．

3　ltrace の利用方法

3.1　ダイナミックリンクライブラリコールの追跡

プログラムが使用しているダイナミックライブラリコールを ltrace を用いて追跡するには，strace 同様,

```
$ ltrace 実行ファイル名
```

のようにすればよい．

Fig. 1 で示したプログラムを ltrace にかけた結果を Fig. 10 に示す．

Fig. 10 のように，コール時にどのような引数が渡されたかがわかるため，プログラムの動作がおかしい時に ltrace を使用すれば，原因の究明につながることがある．

例えば，Fig. 11 のようなプログラムがあったとする．“hoge.txt” というファイルが存在しない場合にこのプログラムを実行すると，Fig. 12 のようなエラーメッセージが表示される．

```
execve("./world", ["./world"], [/* 27 vars */]) = 0
uname({sys="Linux", node="shuto", ...}) = 0 <0.000020>
brk(0)                                  = 0x80495a8 <0.000014>
open("/etc/ld.so.preload", O_RDONLY)    = -1 ENOENT (No such file or directory) <0.000025>
(中略)
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x40014000 <0.000013>
write(1, "Hello,World!!\n", 14Hello,World!!
)         = 14 <0.000014>
munmap(0x40014000, 4096)                = 0 <0.000014>
_exit(0)                                = ?
```

Fig. 9　システムコール上の時間計測

```
__libc_start_main(0x080483f0, 1, 0xbffff954, 0x08048298, 0x08048450 <unfinished ...>
__register_frame_info(0x08049494, 0x08049590, 0xbffff8f8, 0x40046f18, 0x40132e48) = 0x080482d0
printf("Hello, world!\n")                          = 14
__deregister_frame_info(0x08049494, 0x40009608, 0x40013c44, 0x40013d70, 1) = 0
+++ exited (status 0) +++
```

Fig. 10　“Hello, world!” と表示する時に使用されるダイナミックライブラリコール

```
#include <stdio.h>

int main()
{
  FILE *fp;
  char buf[256];

  fopen("hoge.txt", "r");
  fgets(buf, sizeof(buf), fp);
  printf("%s", buf);
  fclose(0);
  return 0;
}
```

Fig. 11　よくないプログラム

```
Segmentation fault
```

Fig. 12　“hoge.txt” がない場合に表示されるエラーメッセージ

ltrace にかけると，Fig. 13 のように，fopen() を実行した結果，0(NULL) が返ってきていることがわかる．

マニュアルを参照すると，fopen() は，開こうとしたファイルが存在しなかった場合，FILE 型のポインタではなく，NULL を返すことがわかる．今回のバグは．fopen() の戻り値を確認しなかったために起こったものであることがわかった．

```
__libc_start_main(0x080484b0, 1, 0xbffff874, 0x08048320, 0x08048550 <unfinished ...>
__register_frame_info(0x08049594, 0x0804969c, 0xbffff818, 0x40047c08, 0x40131dd0) = 0x08048358
fopen("hoge.txt", "r")                          = 0
fgets( <unfinished ...>
--- SIGSEGV (Segmentation fault) ---
+++ killed by SIGSEGV +++
```

Fig. 13　ltrace でダイナミックライブラリコールを追跡した結果

3.2　ltrace のオプション

- -f

  子プロセスにおけるダイナミックリンクライブラリコールの追跡も行う．Fig. 7 のプログラムに-f オプションをつけて ltrace を実行した結果の一部を Fig. 14 に示す.

```
fork( <unfinished ...>
[pid 32212] fprintf(0x400ec540, "hello\n"hello
)       = 6
[pid 32212] sleep(1 <unfinished ...>
[pid 32211] <... fork resumed> )                = 32212
[pid 32211] fprintf(0x400ec540, "Good bye see you!!\n"Good bye see you!!
) = 19
[pid 32211] scanf(0x080486f0, 0xbffffcd8, 24, 0x080485a1, 0x08049820 <unfinished ...>
[pid 32212] <... sleep resumed> )               = 0
[pid 32212] fprintf(0x400ec540, "bye\n"bye
)         = 4
[pid 32212] exit(0 <unfinished ...>
[pid 32212] __deregister_frame_info(0x0804972c, 0, 0, 0, 0) = 0x08049820
[pid 32212] +++ exited (status 0) +++
--- SIGCHLD (Child exited) ---
```

Fig. 14　Fig. 7 のプログラムの子プロセスにおけるダイナミックリンクライブラリコール

- -S

  システムコールを追跡する．Fig. 1 のプログラムに-S オプションをつけて ltrace をつけて実行した結果の一部を Fig. 15 に示す.

```
SYS_brk(NULL)                                   = 0x08049838
SYS_open("/etc/ld.so.preload", 0, 00)           = -2
SYS_open("/etc/ld.so.cache", 0, 00)             = 3
SYS_fstat(3, 0xbffff634, 0, 0xbffff6ac, 3)      = 0
SYS_mmap(0xbffff66c, 0, 0x40012a48, 0x400013e0, 3) = 0x40014000
```

Fig. 15　Fig. 1 のプログラムのシステムコール (一部)

- -C

  わかりにくいダイナミックリンクライブラリコールをユーザがわかりやすいように表示する．Fig. 16 に示す C++プログラムをこのオプションをつけずに ltrace にかけた結果を Fig. 17 に，-C オプションをつけて ltrace にかけた結果を Fig. 18 に示す．Fig. 17 において “__ls__7ostreamPCc” となっている部分が，Fig. 18 では “ostream::ls(char const *)” となっている．

```
#include<cstdio>
#include<iostream>

int main()
{
  cout << "Hello,World!!\n";
  return 0;
}
```

Fig. 16　C++のプログラム

```
__libc_start_main(0x08048540, 1, 0xbffffb14, 0x080483e4, 0x080485a0 <unfinished ...>
__register_frame_info(0x080495e4, 0x0804975c, 0xbffffab8, 0x400aff18, 0x4019be48
) = 0x4005f300
__ls__7ostreamPCc(0x08049730, 0x080485c4, 0x400aff18, 0x4019be48,
0x400097c0Hello,World!!) = 0x08049730
__deregister_frame_info(0x080495e4, 0x40009608, 0x40083144, 0x40083370, 1) = 0x0804975c
+++ exited (status 0) +++
```

Fig. 17　Fig. 16 のプログラムを ltrace にかけた結果

```
__libc_start_main(0x08048540, 1, 0xbffffb14, 0x080483e4, 0x080485a0 <unfinished ...>
__register_frame_info(0x080495e4, 0x0804975c, 0xbffffab8, 0x400aff18, 0x4019be48
) = 0x4005f300
ostream::ls(char const *)(0x08049730, 0x080485c4, 0x400aff18, 0x4019be48,
0x400097c0Hello,World!!) = 0x08049730
__deregister_frame_info(0x080495e4, 0x40009608, 0x40083144, 0x40083370, 1) = 0x0804975c
+++ exited (status 0) +++
```

Fig. 18　Fig. 16 のプログラムを-C オプションをつけて ltrace にかけた結果

参考文献

1) *ltrace manual page.*

2) *strace manual page.*