

システムプログラム

第10回

創域理工学部 情報計算科学科

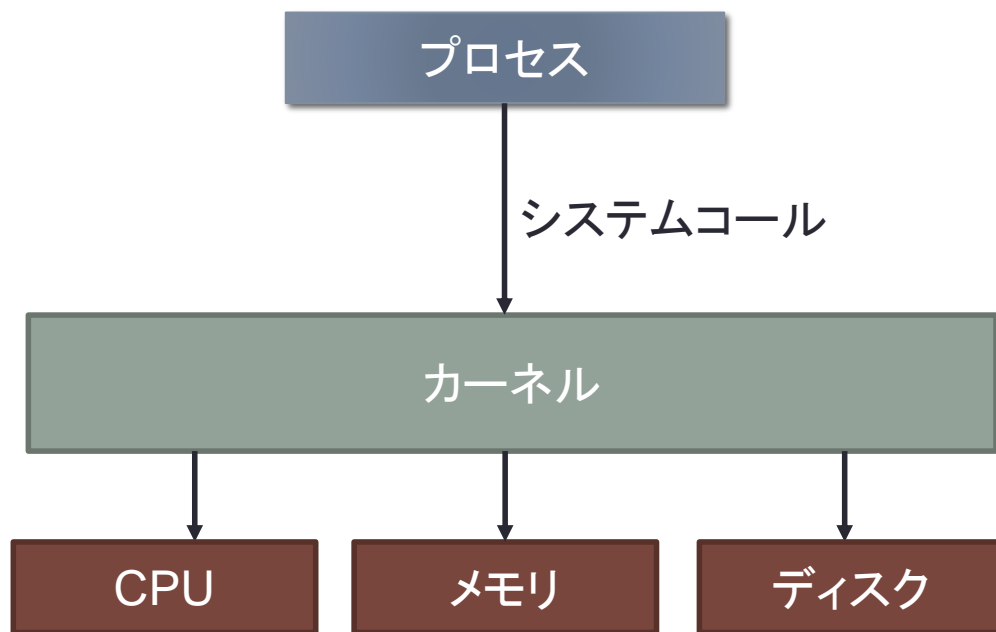
松澤 智史

本日の内容

- システムコール

システムコール

- カーネルはCPUやメモリ, 各種デバイスの管理をする
- カーネルの機能の実行依頼をシステムコールと呼ぶ



システムコール

- アプリケーション開発者が直接システムコールを扱うことは稀
 - ※間接的には必ず使うことになる
- 直接扱う例
 - 処理速度を限界まであげるようなシステム開発
 - デバイスドライバなどのハードウェアを直接扱うシステム開発
- 多くのプログラミング言語やツールは,
システムコールを扱いやすくするための機能やライブラリを提供

今回はこのシステムコールについて学ぶ

システムコールのメリット

- デバイス操作コード(デバイスドライバ)の共通化
- デバイスの排他制御
- API の提供
- 実行可能なルーチンの制限
- 不正な要求のチェック

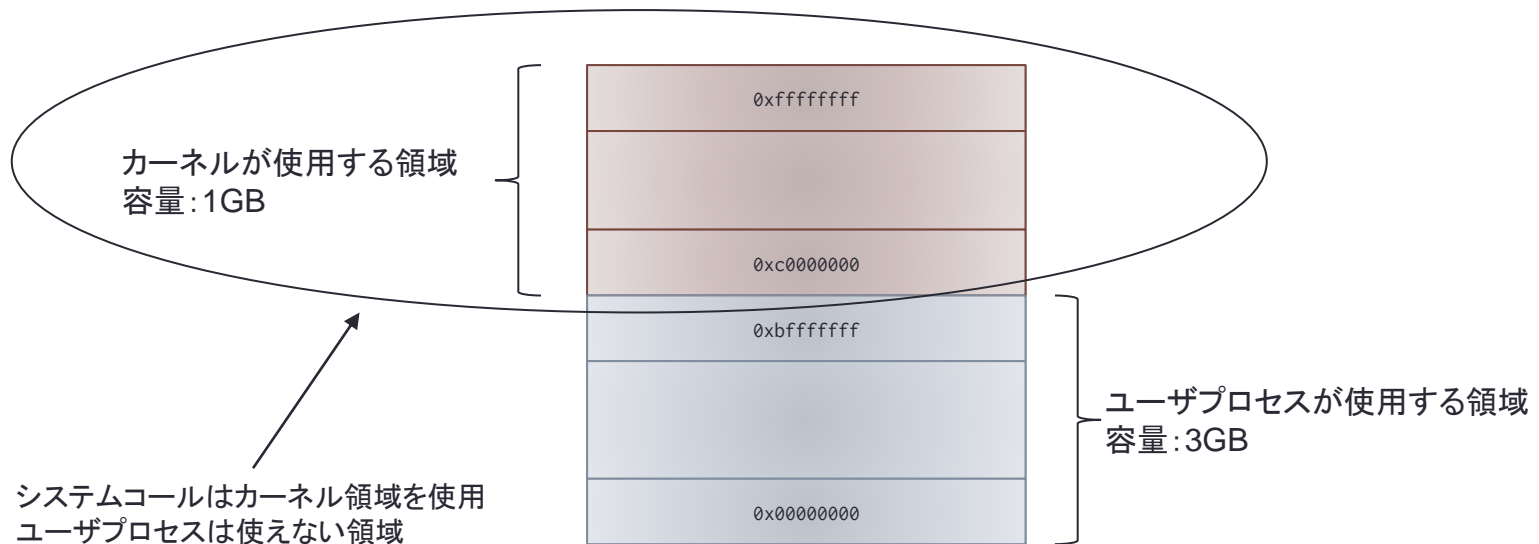
システムコールの呼び出し

- 通常関数呼び出しと同じ方法ではない
 - ユーザ空間とカーネル空間はメモリ空間が異なる
 - ユーザアプリケーションがカーネルのメモリ領域にアクセスできない
- 割り込みやsyscall命令を利用した呼び出しで実現する

復習:仮想アドレス空間の内訳

32ビット OSの場合

- 仮想メモリはプロセスごとに異なるが
カーネルが使用する領域は全仮想メモリで共通となる

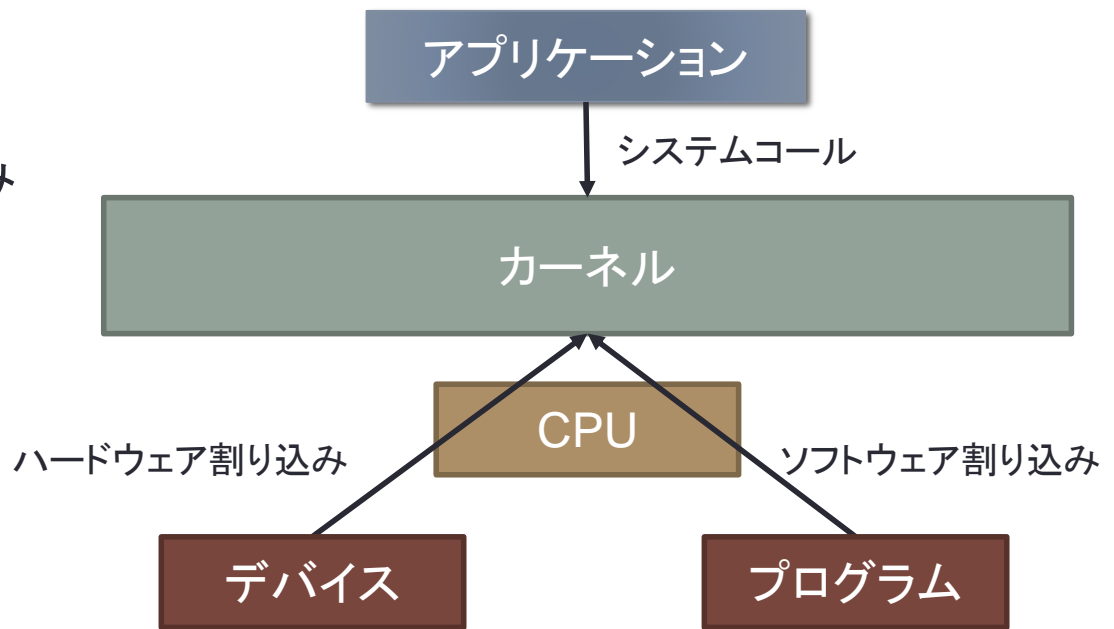


割り込み

- CPUは現在実行中のコードを停止(後に再開可能)し、特定のコードを実行する処理
 - irq(インターラプトリクエスト)
- ソフトウェア割り込み
 - システムコール
 - 例外処理
- ハードウェア割り込み
 - キーボードからの入力
 - ネットワークの受信

余談:カーネルの駆動トリガ

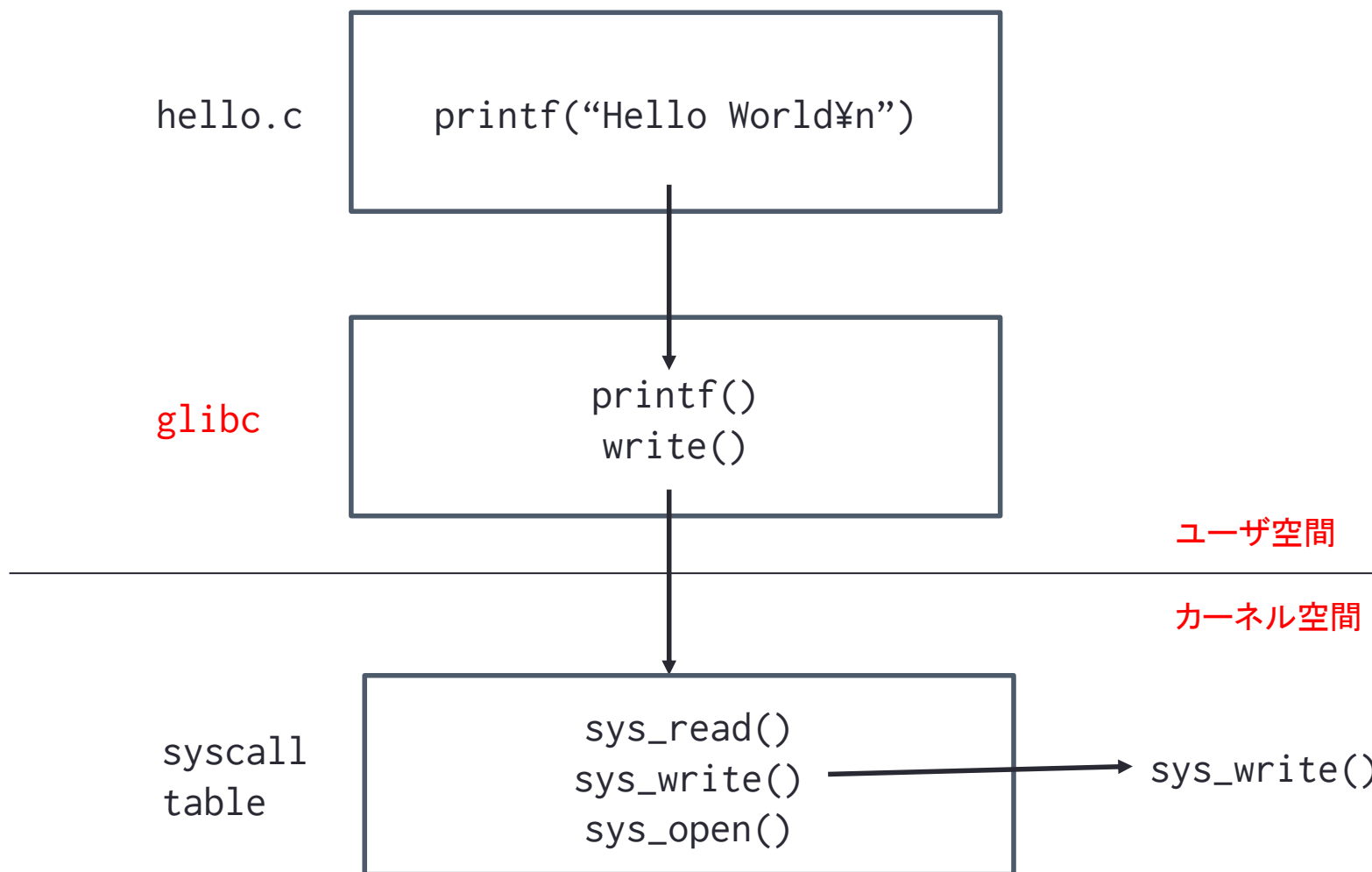
- カーネルは基本的にイベント駆動型
 - 何かイベントが発生した際に仕事を開始する
- トリガとなるもの
 - システムコール
 - ハードウェア割り込み
 - ソフトウェア割り込み



具体的なシステムコール呼び出し

- 特定のレジスタ(raxレジスタ)にシステムコールの番号を格納
- 結果などを格納するポインタなどをレジスタに格納
 -
 -
- syscall命令実行
 - 割り込み処理
- 結構重い

Linuxでのシステムコール

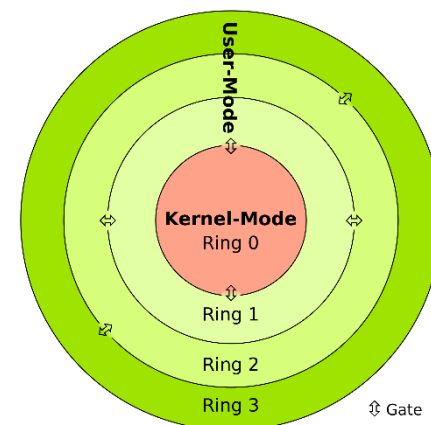


glibc

- 正式名称:GNU Cライブラリ
 - GNU:GNUプロジェクト
 - ユーザが自由にソフトウェアを実行
 - ユーザが自由にコピーや配布により共有・研究
 - ユーザが自由に修正可能
- 様々なカーネルやハードウェア上で使用可能なライブラリ
- 他のライブラリより比較的重い・遅い

CPUモード

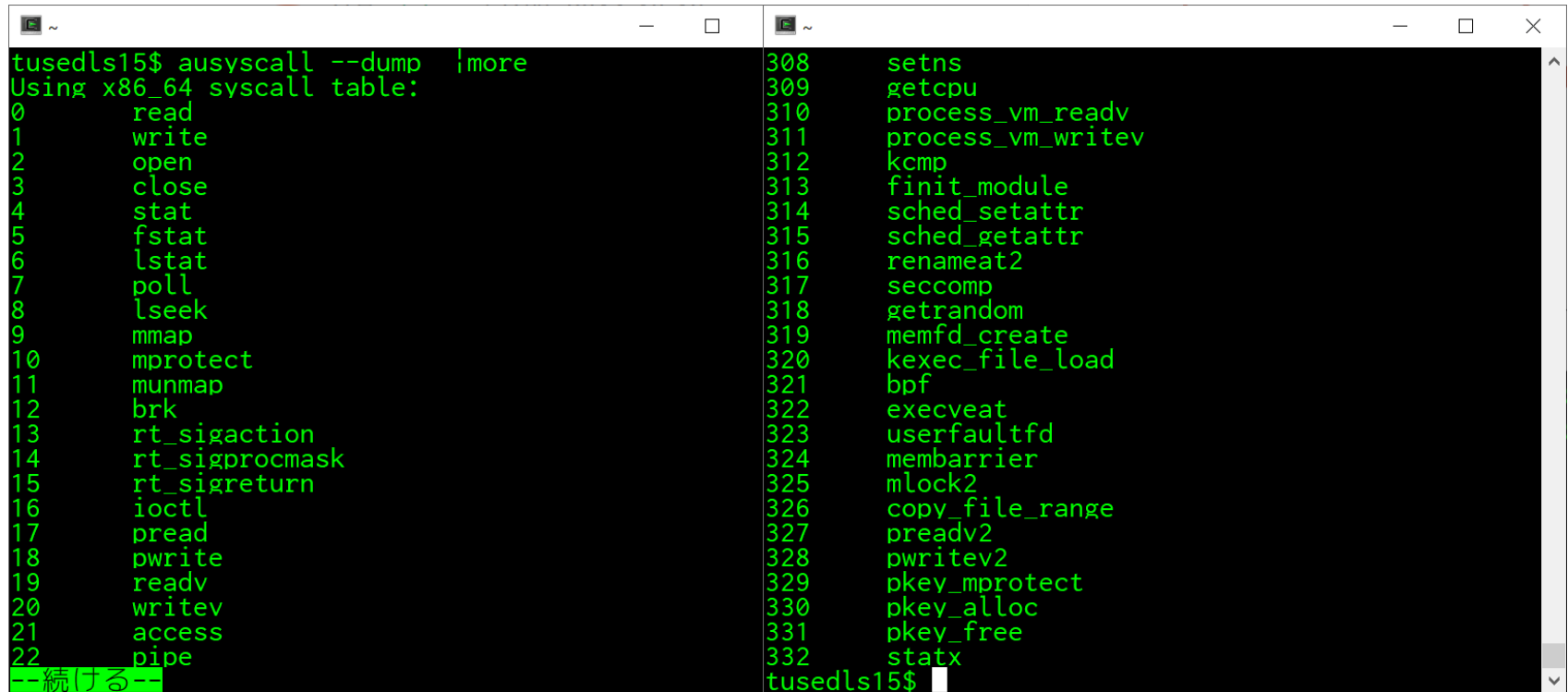
- すべてのCPUは必ず2つ以上のCPUモードを持つ
 - CPUのアーキテクチャにより種類は異なる
 - ※ Intel 64bit CPUは4つ
- 特権モード(特権レベル0)
 - カーネルモード, カーネル空間, スーパバイザモードなどと呼ぶ
 - すべてのハードウェア制御が可能
- ユーザモード(特権レベル1~3)
 - ユーザ空間
 - ハードウェアによってCPUの動作に制限がかけられる
- CPUモードの変更
 - CPUモードの変更は高い特権レベルから低い特権レベルの場合は自由に行える
 - 低いレベルから高いレベルへの変更はハードウェアが制御する「ゲート」を特殊な命令を使って通過する → システムコール



CPUモードの階層構造
通称: リングプロテクション

システムコール番号

```
$ ausyscall --dump
```



```
tusedls15$ ausyscall --dump |more
Using x86_64 syscall table:
0      read
1      write
2      open
3      close
4      stat
5      fstat
6      lstat
7      poll
8      lseek
9      mmap
10     mprotect
11     munmap
12     brk
13     rt_sigaction
14     rt_sigprocmask
15     rt_sigreturn
16     ioctl
17     pread
18     pwrite
19     readv
20     writev
21     access
22     pipe
--続ける--

308    setns
309    getcpu
310    process_vm_readv
311    process_vm_writev
312    kcmp
313    finit_module
314    sched_setattr
315    sched_getattr
316    renameat2
317    seccomp
318    getrandom
319    memfd_create
320    kexec_file_load
321    bpf
322    execveat
323    userfaultfd
324    membarrier
325    mlock2
326    copy_file_range
327    preadv2
328    pwritev2
329    pkey_mprotect
330    pkey_alloc
331    pkey_free
332    statx
tusedls15$
```

syscall関数

- glibcで用意されているシステムコールを間接的に呼び出すための関数
- 関数内部でシステムコールに必要な手続き(レジスタセット, syscall呼び出し等)を呼び出す
- アーキテクチャ特有の引数の渡し方をするので、一般的なC言語のプログラムでは使わない方が良い
 - ハードウェアやカーネルが異なれば当然動かないプログラムとなる

syscall関数を使ってみる

```
tusedls15$ cat sys_write.c
#include <unistd.h>
#include <sys/syscall.h>

int main(){
    long n;
    char text[] = "abc\n";
    n = syscall(1, 1, text, sizeof(text)-1);
    return 0;
}
tusedls15$ gcc sys_write.c -o sys_write
tusedls15$ ./sys_write
abc
tusedls15$
```

→ 標準出力 (1)

→ システムコール番号(1)

```
tusedls15$ cat sys_read.c
#include <unistd.h>
#include <sys/syscall.h>

int main(){
    long n;
    char text[256];
    n = syscall(0, 0, text, sizeof(text)-1);
    return 0;
}
tusedls15$ gcc sys_read.c -o sys_read
tusedls15$ ./sys_read
```


straceで確認

```
tusedls15$ strace ./sys_write
execve("./sys_write", ["/sys_write"], [/* 35 vars */]) = 0
brk(NULL) = 0x1f79000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f51abf0a000
access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or direc
ory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=157557, ...}) = 0
mmap(NULL, 157557, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f51abee3000
close(3) = 0
open("/lib64/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\340$\2\0\0\0\0\0".
.. 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=2151672, ...}) = 0
mmap(NULL, 3981792, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0)
0x7f51ab91d000
mprotect(0x7f51abadf000, 2097152, PROT_NONE) = 0
mmap(0x7f51abcdf000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MA
_DENYWRITE, 3, 0x1c2000) = 0x7f51abcdf000
mmap(0x7f51abce5000, 16864, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MA
_ANONYMOUS, -1, 0) = 0x7f51abce5000
close(3) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f51abee2000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f51abee0000
arch_prctl(ARCH_SET_FS, 0x7f51abee0740) = 0
mprotect(0x7f51abcdf000, 16384, PROT_READ) = 0
mprotect(0x600000, 4096, PROT_READ) = 0
mprotect(0x7f51abf0b000, 4096, PROT_READ) = 0
munmap(0x7f51abee3000, 157557) = 0
write(1, "abc\n", 4abc
) = 4
exit_group(0) = ?
+++ exited with 0 +++
tusedls15$
```

余談: strace -T

```
tusedls15$ strace -T ./sys_write
execve("./sys_write", ["/sys_write"], [/* 35 vars */]) = 0 <0.000801>
brk(NULL) = 0x1743000 <0.000051>
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
  0x7fe930088000 <0.000053>
access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or direc
tory) <0.000055>
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3 <0.000058>
fstat(3, {st_mode=S_IFREG|0644, st_size=157557, ...}) = 0 <0.000051>
mmap(NULL, 157557, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fe930061000 <0.00005
5>
close(3) = 0 <0.000051>
open("/lib64/libc.so.6", O_RDONLY|O_CLOEXEC) = 3 <0.000063>
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\340$\2\0\0\0\0".
..., 832) = 832 <0.000051>
fstat(3, {st_mode=S_IFREG|0755, st_size=2151672, ...}) = 0 <0.000052>
mmap(NULL, 3981792, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0)
= 0x7fe92fa9b000 <0.000056>
mprotect(0x7fe92fc5d000, 2097152, PROT_NONE) = 0 <0.000061>
mmap(0x7fe92fe5d000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MA
P_DENYWRITE, 3, 0x1c2000) = 0x7fe92fe5d000 <0.000063>
mmap(0x7fe92fe63000, 16864, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MA
P_ANONYMOUS, -1, 0) = 0x7fe92fe63000 <0.000057>
close(3) = 0 <0.000049>
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
  0x7fe930060000 <0.000064>
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
  0x7fe93005e000 <0.000053>
arch_prctl(ARCH_SET_FS, 0x7fe93005e740) = 0 <0.000050>
mprotect(0x7fe92fe5d000, 16384, PROT_READ) = 0 <0.000058>
mprotect(0x600000, 4096, PROT_READ) = 0 <0.000056>
mprotect(0x7fe930089000, 4096, PROT_READ) = 0 <0.000057>
munmap(0x7fe930061000, 157557) = 0 <0.000085>
write(1, "abc\n", 4abc
) = 4 <0.000078>
exit_group(0) = ?
+++ exited with 0 +++
tusedls15$
```

マイクロ秒単位で
実行にかかった時間が表示される

他の言語でも確認

```
tusedls02$ cat hello.py  
print("Hello world")  
tusedls02$ strace -o hello.py.log python3 hello.py
```

```
brk(0x1c94000) = 0x1c94000  
brk(NULL) = 0x1c94000  
close(3) = 0  
munmap(0x7f655cdab000, 8192) = 0  
write(1, "Hello world\n", 12) = 12  
brk(NULL) = 0x1c94000  
brk(NULL) = 0x1c94000  
brk(0x1c92000) = 0x1c92000  
brk(NULL) = 0x1c92000  
rt_sigaction(SIGINT, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x7f655c45f630}, {sa_handler=0x7f655c85d120, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x7f655c45f630}, 8) = 0  
:
```

代表的なシステムコール

- read write open close
 - 省略
- access
 - ユーザのファイルに対するアクセス権をチェック
- execve
 - 新しいプログラムを実行する
 - プログラムを実行するための唯一の方法
- fstat
 - ファイルの状態を取得する
 - ファイルの存在するデバイスID, アクセス権限, ブロックサイズなど

代表的なシステムコール

- brk
 - ヒープ(第6回参照)を読み書きできるようにカーネルに要求
- mmap munmap
 - ファイルやデバイスをメモリにマップ/アンマップする
- mprotect
 - メモリ領域のアクセス許可を制御する

システムコール(例え)

- 人間でいえば, 親指を曲げる, 関節を曲げるなどの原始的な処理(筋肉の収縮など)がシステムコール
- 物を持つ, 歩くなどの複数のシステムコールをまとめた動作の記述を提供しているのがAPIやライブラリ
- ○○駅へ移動するなどのライブラリの提供する機能をまとめて記述したプログラムがアプリケーション

余談: Windowsでは

- Windows APIと呼ばれるシステムコール用APIを用意
- Windows APIの種類
 - Win32
 - Win64
- Windows APIの機能分類
 - Kernel
 - User
 - GDI

まとめ

システムコール

- カーネルに実行依頼するための処理
- CPUの特権モードを変更して処理の依頼
- システムコールの種類はカーネルに依存
 - 現在も増えている
- 比較的手続きが面倒なため、一連の処理をまとめた「ライブラリ」をプログラマには提供
 - 一般のアプリケーション開発者が直接システムコール呼び出しの記述を書くことはほとんどない
 - すべてのプログラム実行は必ずシステムコールを使うことになる

質問あればどうぞ