

OSSライセンスを理解するための Linuxが動く仕組み

2021.2.8

富士通コンピュータテクノロジーズ

浅羽 鉄平



CC-BY-SA 4.0

\$ whoami

- 名前: 浅羽 鉄平 (あさば てっぺい)
- 所属: 富士通コンピュータテクノロジーズ
- 仕事: 組込みシステム向けLinuxディストリビューションのメンテナンス。最近はLinuxであれば、「組込み」と付けなくてもよいのではと思う今日この頃。
- 好きなライセンス: GPL-3.0



The names of products are the product names, trademarks or registered trademarks of the respective companies. Trademark notices ((R),TM) are not necessarily displayed on system names and product names in this material.

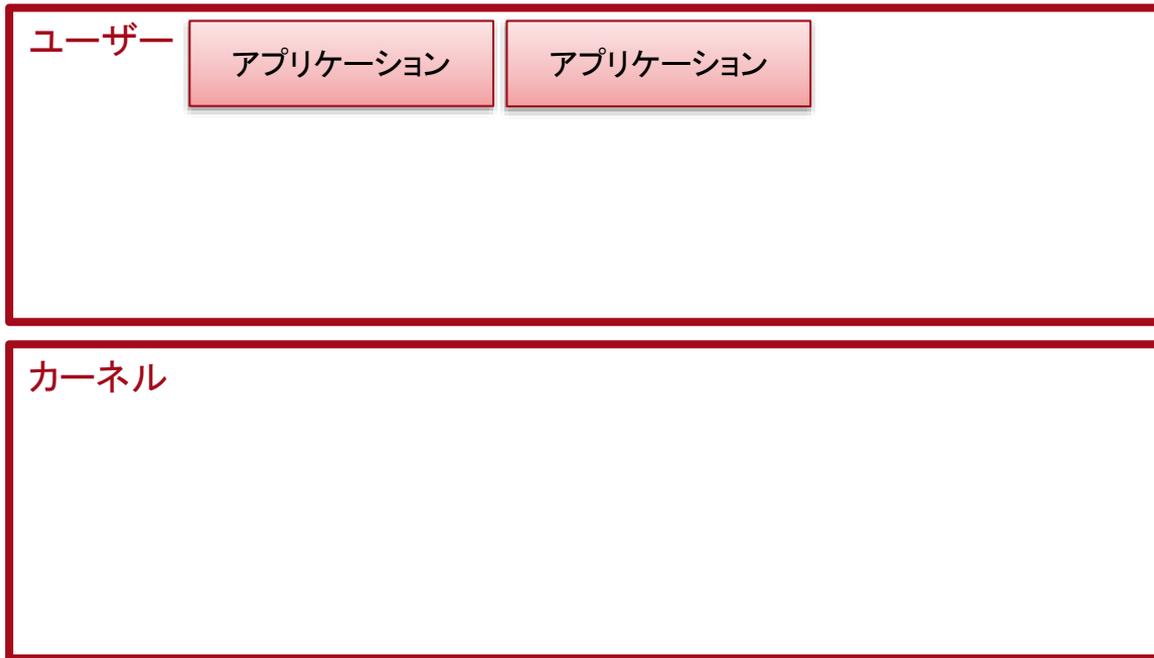
Linuxのシステム構成

よく見るLinuxのシステム構成



今日お話しするライセンスの伝播を理解するためには、

よく見るLinuxのシステム構成



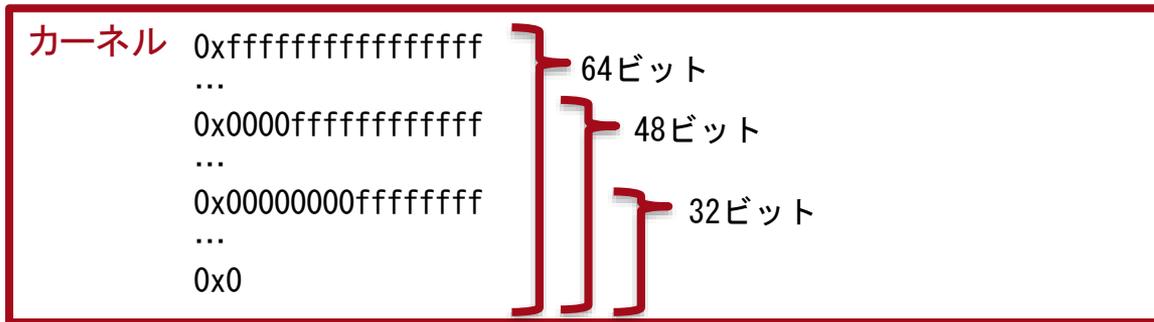
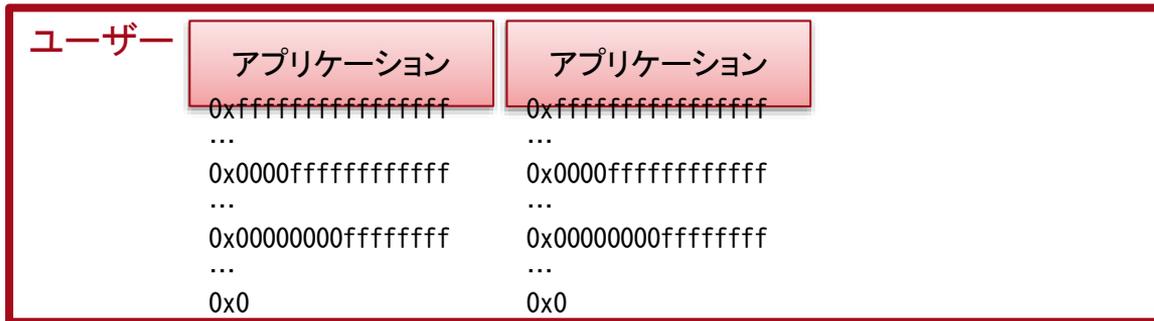
これだけ意識すればOK

つまり、

■ Linuxシステムは、

- カーネル
 - ユーザー空間のアプリケーション達
- で構成されています。

ということかというと、



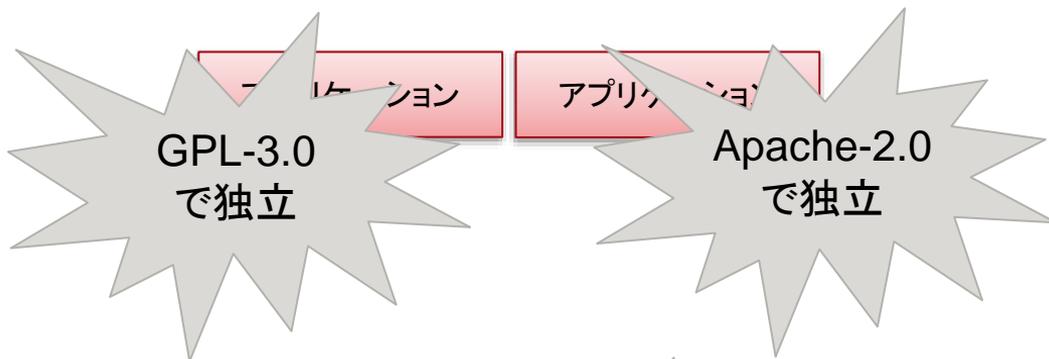
カーネルとアプリケーションの数だけ空間 (アドレス、番地)があるということ

■ Linuxシステムは、

- カーネル
- ユーザー空間のアプリケーション達

でそれぞれ空間があり、**独立している**と覚えてください

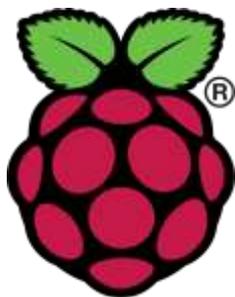
つまり、



こんなイメージです

Linux上でアプリケーションを動かしてみよう

➤ 実行環境：
Raspberry Pi 4

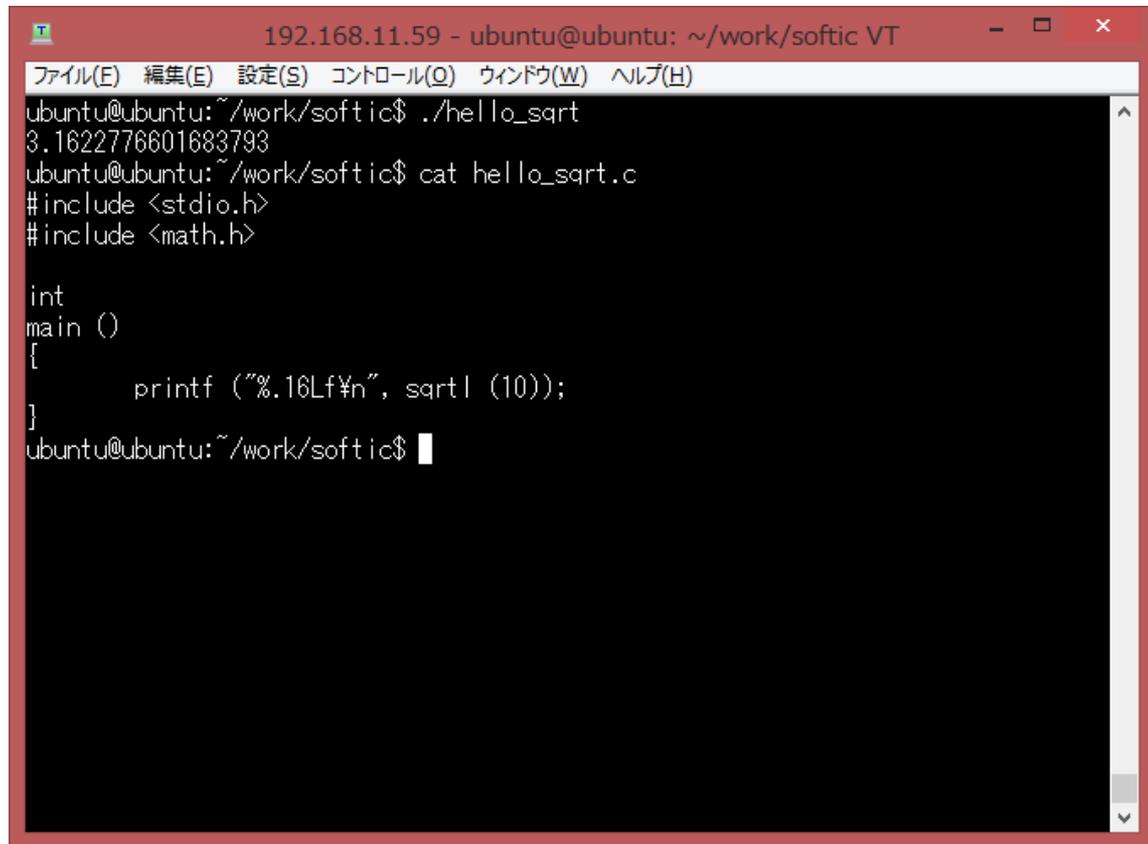


Linux (Ubuntu)上で
アプリケーション ($\sqrt{10}$ を計算する)を実行

```
192.168.11.59 - ubuntu@ubuntu: ~/work/softic VT
ファイル(E) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
ubuntu@ubuntu: ~/work/softic$ ./hello_sart
3.1622776601683793
ubuntu@ubuntu: ~/work/softic$
```


これを√の計算をする度に
書くのは大変、、

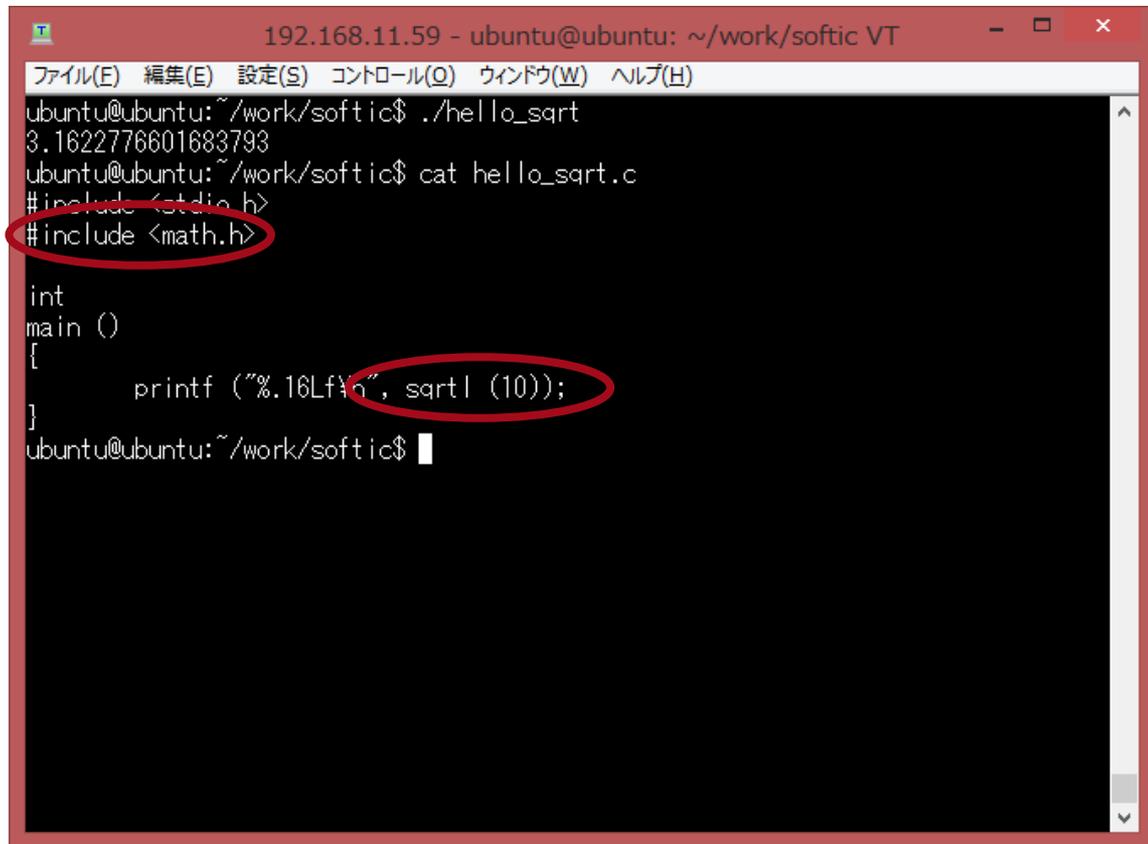
でも実際に書いたのは、



```
192.168.11.59 - ubuntu@ubuntu: ~/work/softic VT
ファイル(E) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
ubuntu@ubuntu:~/work/softic$ ./hello_sqrt
3.1622776601683793
ubuntu@ubuntu:~/work/softic$ cat hello_sqrt.c
#include <stdio.h>
#include <math.h>

int
main ()
{
    printf ("%16Lf\n", sqrt(10));
}
ubuntu@ubuntu:~/work/softic$ █
```

ポイントは、



```
192.168.11.59 - ubuntu@ubuntu: ~/work/softic VT
ファイル(E) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
ubuntu@ubuntu:~/work/softic$ ./hello_sqrt
3.1622776601683793
ubuntu@ubuntu:~/work/softic$ cat hello_sqrt.c
#include <stdio.h>
#include <math.h>

int
main ()
{
    printf("%.16Lf% $\sqrt{}$ ", sqrt(10));
}
ubuntu@ubuntu:~/work/softic$
```

■ よく使う機能を棚に並べて必要な時に取り出せるようにしておく。

これがライブラリ

- $\sqrt{\quad}$, X^2 , sin, cos, log等の数学関数
- アルファベットを小文字から大文字に変換する等のテキスト処理
- Webアクセスする
- 暗号復号する
- 画面に●や▲等を描画する

など

■ 例えるなら、

- 料理で毎回だしを取るのは大変だから、
- だしを粉末や液体にして溶くだけにしたら、
- 楽ですね。
- 一般的なだしのレシピは公知であっても、おいしいだしは商品として売れる。
- だしの商品がたくさんあるようにデータベースの商品はたくさんある。
- データベースのライセンスは、、

アプリケーションとライブラリの関係は、、

アプリケーション

0xffffffffffffffff

...

0x0000ffffffffffff

...

0x0000ffff9480f000 ライブラリ

...

0x00000000ffffffff プログラム

...

0x0

同じ空間で動く

```
192.168.11.59 - ubuntu@ubuntu: ~/work/softic VT
ファイル(E) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
ubuntu@ubuntu:~/work/softic$ ./hello_sart
3.1622776601683793
ubuntu@ubuntu:~/work/softic$ cat hello_sart.c
#include <stdio.h>
#include <math.h>

int
main ()
{
    printf ("%16Lf\n", sqrtl (10));
}
ubuntu@ubuntu:~/work/softic$ ldd hello_sart
linux-vdso.so.1 (0x0000ffff040c4000)
libc.so.6 => /lib/aarch64-linux-gnu/libc.so.6 (0x0000ffff9480f000)
/lib/ld-linux-aarch64.so.1 (0x0000ffff040c4000)
ubuntu@ubuntu:~/work/softic$
```

動的リンク vs. 静的リンク

■ 動的リンク

- 実行時にライブラリを呼び出し
- プログラム+ライブラリの2ファイル
- プログラムのバイナリのサイズが小さい

■ 静的リンク

- ビルド時にライブラリを組み込み
- プログラム+ライブラリで1ファイル
- ライブラリすべての機能を使わないなら、サイズが小さい

```
192.168.11.59 - ubuntu@ubuntu: ~/work/softic/size VT
ファイル(E) 編集(E) 設定(S) コントロール(Q) ウィンドウ(W) ヘルプ(H)
ubuntu@ubuntu:~/work/softic/size$ ls -lh
total 2.0M
-rwxrwxr-x 1 ubuntu ubuntu 9.1K Feb  6 09:40 hello_sqrt-dynamic
-rwxrwxr-x 1 ubuntu ubuntu 589K Feb  6 07:57 hello_sqrt-static
-rwxr-xr-x 1 ubuntu ubuntu 1.4M Dec 16 11:04 libc-2.31.so
ubuntu@ubuntu:~/work/softic/size$
```

動的リンク vs. 静的リンク

```
192.168.11.59 - ubuntu@ubuntu: ~/work/softic VT
ファイル(E) 編集(E) 設定(S) コントロール(Q) ウィンドウ(W) ヘルプ(H)
ubuntu@ubuntu:~/work/softic$ nm hello_sart
000000000010d90 a __DYNAMIC
000000000010fc0 a __GLOBAL_OFFSET_TABLE__
000000000000830 R __IO_stdin_used
w __ITM_deregisterTMCloneTable
w __ITM_registerTMCloneTable
000000000000988 r __FRAME_END__
000000000000860 r __GNU_EH_FRAME_HDR
0000000000011010 D __TMC_END__
0000000000011018 B __bss_end__
0000000000011010 B __bss_start__
0000000000011010 B __bss_start__
w __cxa_finalize@@GLIBC_2.17
0000000000011000 D __data_start
0000000000000720 t __do_global_dtors_aux
0000000000010d88 d __do_global_dtors_aux_fini_array_entry
0000000000011008 D __dso_handle
0000000000011018 B __end__
0000000000010d80 d __frame_dummy_init_array_entry
w __gmon_start__
0000000000010d88 d __init_array_end
0000000000010d80 d __init_array_start
0000000000000818 T __libc_csu_fini
0000000000000798 T __libc_csu_init
```

```
192.168.11.59 - ubuntu@ubuntu: ~/work/softic VT
ファイル(E) 編集(E) 設定(S) コントロール(Q) ウィンドウ(W) ヘルプ(H)
ubuntu@ubuntu:~/work/softic$ nm hello_sart-s | head -30
0000000000489060 V DW.ref.__gcc_personality_v0
000000000048bb58 W __stapsdt.base
000000000041e238 W __Exit
0000000000488b50 d __GLOBAL_OFFSET_TABLE__
0000000000489070 D __IO_2_1_stderr_
0000000000489400 D __IO_2_1_stdin_
0000000000489238 D __IO_2_1_stdout_
0000000000412d38 T __IO_adjust_column
0000000000439dd0 T __IO_adjust_wcolumn
0000000000413020 T __IO_cleanup
00000000004129b0 T __IO_default_doallocate
0000000000412b80 T __IO_default_finish
0000000000413940 T __IO_default_imbuf
00000000004137a8 T __IO_default_pbackfail
0000000000413928 T __IO_default_read
0000000000413918 T __IO_default_seek
0000000000412bf8 T __IO_default_seekoff
0000000000412938 T __IO_default_seekpos
0000000000412840 T __IO_default_setbuf
0000000000413938 T __IO_default_showmanyc
0000000000413920 T __IO_default_stat
0000000000412b78 T __IO_default_sync
0000000000412568 T __IO_default_uflow
```

動的リンク vs. 静的リンク

動的リンク	静的リンク
0xffffffffffffffff	0xffffffffffffffff
...	...
0x0000ffffffff	0x0000ffffffff
...	...
0x0000fff9480f000	0x0000fff9480f000
...	...
0x00000000ffffffff	0x00000000ffffffff
...	...
0x0	0x0

動的リンクも静的リンクも
同じ空間で動くことに
違いはない

(GPL)の及ぶ作品に対し、静的 vs 動的にリンクされたモジュールについて、GPLには異なる要求がありますか?
(*GPLStaticVsDynamic)

いいえ。GPLの及ぶ作品に静的もしくは動的にほかのモジュールとリンクすることは、GPLの及ぶ作品をもとにして結合著作物を作成することです。ですから、全体としての結合には、GNU一般公衆ライセンスの条項が及びます。
GPLソフトウェアにGPLと固立しないライブラリを用いた場合、どのような法的問題が発生するのでしょうか?もご覧ください。

<https://www.gnu.org/licenses/gpl-faq.ja.html#GPLStaticVsDynamic>



「集積物」とそのほかの種類の「改変されたバージョン」の違いは何ですか? (#MereAggregation)

「集積物」はいくつかの別々のプログラムからなり、それらは同じCD-ROMやその他のメディアにいっしょに入れられて配布されます。GPLは集積物を作成し配布することを認めています。たとえ、ほかのソフトウェアが不自由あるいはGPLと両立しないものである場合でもです。ただ一つの条件は、あなたは、それぞれのプログラムの個別のライセンスが許す権限をユーザが行使用することを妨げるような、あるライセンスのもとで集積物をリリースすることはできないということです。

二つの別々のプログラムと二つの部分の一つのプログラムを分ける線はどこにあるでしょうか? これは法的な問題であり、最終的には裁判官が決めることです。わたしたちは、適切な基準はコミュニケーションのメカニズム(exec, パイプ, rpc, 共有アドレス空間での関数呼び出し、など)とコミュニケーションのセマンティクス(どのような種の情報が相互交換されるか)の両方によると考えています。

モジュールが同じ実行ファイルに含まれている場合、それらは言うまでもなく一つのプログラムに結合されています。もしモジュールが共有アドレス空間でいっしょにリンクされて実行されるよう設計されているならば、それらが一つのプログラムに結合されているのはほぼ間違いありません。

逆に、パイプ、ソケット、コマンドライン引数は通常二つの分離したプログラムの間で使われるコミュニケーションメカニズムです。ですからそれらがコミュニケーションのために使われるときには、モジュールは通常別々のプログラムです。しかしコミュニケーションのセマンティクスが複雑であったり、複雑な内部データ構造を交換したりする場合は、それらも二つの部分により大規模なプログラムに結合されていると考える基準となりうるでしょう。

<https://www.gnu.org/licenses/gpl-faq.ja.html#MereAggregation>

なぜLGPLは静的リンクで要求が変わるのか

静的リンク

```
0xffffffffffffffff  
...  
0x0000ffffffffffff  
...  
0x0000ffff9480f000  
...  
0x00000000ffffffff  
...  
0x0
```

静的リンクされたLGPLのライブラリを差し替える自由として、「アプリケーションのオブジェクト (リンクされる前の状態)の提供」が要求されている。
「リバース エンジニアリングの許諾」も同じ理由。

(LGPLの)及ぶ作品に対し、静的 vs 動的にリンクされたモジュールについて、LGPLには異なる要求がありますか？

(#LGPLStaticVsDynamic)

LGPL (現存のどのバージョンでも: v2, v2.1, or v3)に適合する目的では:

- (1) LGPLのライブラリに対し静的にリンクする場合、ユーザがライブラリを改変してアプリケーションと再リンクできる機会のために、あなたは、あなたのアプリケーションを、オブジェクト(ソースの必要は必ずしもありません)フォーマットでも提供する必要があります。

<https://www.gnu.org/licenses/gpl-faq.ja.html#LGPLStaticVsDynamic>

ヘッダとの関係

```
192.168.11.59 - ubuntu@ubuntu: ~ VT
ファイル(E) 編集(E) 設定(S) コントロール(Q) ウィンドウ(W) ヘルプ(H)
SQRT(3) Linux Programmer's Manual SQRT(3)
NAME
sart, sartf, sartil - square root function
SYNOPSIS
#include <math.h>

double sart(double x);
float sartf(float x);
long double sartil(long double x);

Link with -lm.

Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

sartf(), sartil():
  _ISOC99_SOURCE || _POSIX_C_SOURCE >= 200112L
  || /* Since glibc 2.19: */ _DEFAULT_SOURCE
  || /* Glibc versions <= 2.19: */ _BSD_SOURCE || _SVID_SOURCE
DESCRIPTION
These functions return the nonnegative square root of x.
Manual page sart(3) line 1 (press h for help or q to quit)
```

```
192.168.11.59 - ubuntu@ubuntu: ~/work/softic VT
ファイル(E) 編集(E) 設定(S) コントロール(Q) ウィンドウ(W) ヘルプ(H)
ubuntu@ubuntu:~/work/softic$ gcc -E hello_sart.c | head -30
# 1 "hello_sart.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 31 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 32 "<command-line>" 2
# 1 "hello_sart.c"
# 1 "/usr/include/stdio.h" 1 3 4
# 27 "/usr/include/stdio.h" 3 4
# 1 "/usr/include/aarch64-linux-gnu/bits/libc-header-start.h" 1 3 4
# 33 "/usr/include/aarch64-linux-gnu/bits/libc-header-start.h" 3 4
# 1 "/usr/include/features.h" 1 3 4
# 461 "/usr/include/features.h" 3 4
# 1 "/usr/include/aarch64-linux-gnu/sys/cdefs.h" 1 3 4
# 452 "/usr/include/aarch64-linux-gnu/sys/cdefs.h" 3 4
# 1 "/usr/include/aarch64-linux-gnu/bits/wordsize.h" 1 3 4
# 453 "/usr/include/aarch64-linux-gnu/sys/cdefs.h" 2 3 4
# 1 "/usr/include/aarch64-linux-gnu/bits/long-double.h" 1 3 4
# 454 "/usr/include/aarch64-linux-gnu/sys/cdefs.h" 2 3 4
# 462 "/usr/include/features.h" 2 3 4
# 485 "/usr/include/features.h" 3 4
# 1 "/usr/include/aarch64-linux-gnu/gnu/stubs.h" 1 3 4
```

アプリケーションとアプリケーションの関係

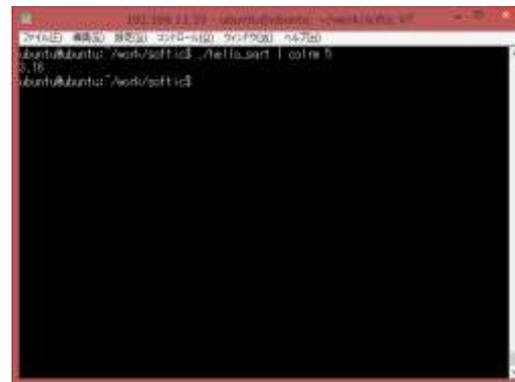
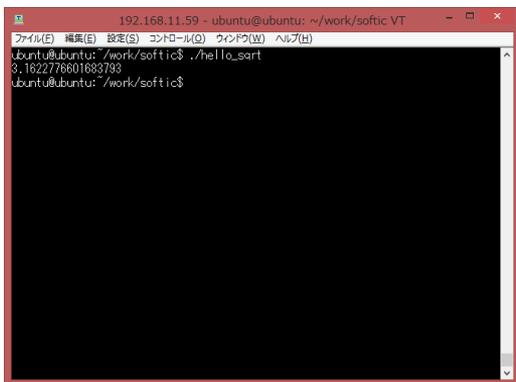
アプリケーション

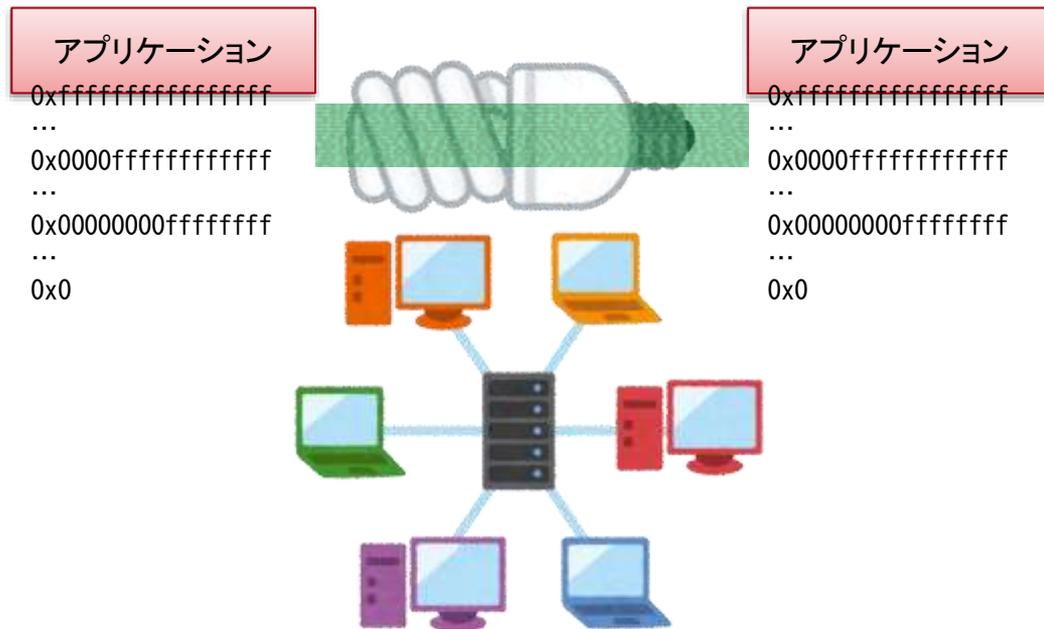
0xffffffffffffffff
...
0x0000ffffffffffff
...
0x00000000ffffffff
...
0x0

出力⇒3.1622776601683793⇒入力

アプリケーション

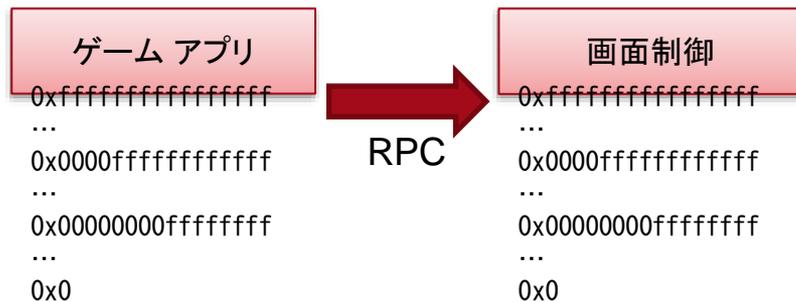
0xffffffffffffffff
...
0x0000ffffffffffff
...
0x00000000ffffffff
...
0x0





プログラムとプログラムでデータをやり取りするネットワークの仕組み。
同じコンピュータ内でも別のコンピュータでも可能。
同じプログラム内でも可能だが、

■ Remote Procedure Call、遠隔操作のこと



ゲーム アプリがグラフィックを描画したいときに画面全体を制御するミドルウェアに依頼を出すとき等に使われる仕組み。
同じコンピュータでもネットワークの向こう側でもよい。

■ execute、実行する

- コマンドラインから、アプリケーション内から (fork) などトリガーは様々だが、新たなアプリケーションを実行するという意味



The screenshot shows a terminal window titled '192.168.11.59 - ubuntu@ubuntu: ~/work/softic VT'. The terminal output shows a `strace ./hello_sart` command being executed. The first line of the strace output is `execve("./hello_sart", ["/hello_sart"], 0xffffce9d5e60 /* 24 vars */) = 0`, where the `execve` function name is circled in red. Subsequent lines show various system calls like `brk(NULL)`, `faccessat`, `openat`, `fstat`, `mmap`, `close`, `read`, `mmap`, `mprotect`, `mmap`, `mmap`, `close`, and `mprotect`.

```
192.168.11.59 - ubuntu@ubuntu: ~/work/softic VT
ファイル(E) 編集(E) 設定(S) コントロール(Q) ウィンドウ(W) ヘルプ(H)
ubuntu@ubuntu: ~/work/softic$ cat hello_sart.py
import math

print (math.sqrt (10))
ubuntu@ubuntu: ~/work/softic$ python$ hello_sart.py
3.1622776601683795
ubuntu@ubuntu: ~/work/softic$ █
```

プログラミング言語のインタプリタがGPLのもとでリリースされていた場合、そのインタプリタで解釈されるように書かれたプログラムは、GPLと両立するライセンスでなければならないでしょうか? (≠InterpreterIsGPL)

インタプリタが単に言語を解釈するだけならば、否はノーです。解釈されるプログラムは、インタプリタにとっては単なるデータに過ぎません。GPLのような著作権法にもとづく自由ソフトウェアのライセンスは、あなたがインタプリタ上で利用するデータの種類を限定することはできません。あなたは、好きなデータ(この場合解釈されるプログラム)を使って、好きなようにインタプリタを実行することができますし、そのデータを誰かにライセンスングすることについて必要とされる条件は何もありません。

しかし、インタプリタが他の機能(多くの場合ライブラリですが、ライブラリである必要はありません)への「バインディング」を提供するように拡張されている場合、解釈されるプログラムはバインディングを使うことによって事実上それらの機能とリンクされることになります。ですから、もしそういった機能がGPLのもとでリリースされているならば、機能を利用している解釈されるプログラムはGPLと両立する形でリリースされなければなりません。JNI(Javaネイティブインターフェース)はそのような機能の一例です。JNIによってアクセスされるライブラリは、それらと呼ぶJavaプログラムと動的にリンクされています。これらのライブラリはインタプリタともリンクされています。もしインタプリタがそれらのライブラリと静的にリンクされていれば、あるいは、動的にそれらの特定のライブラリとリンクするように設計されていれば、それもGPLと両立する形でリリースされる必要があります。

<https://www.gnu.org/licenses/gpl-faq.ja.html#IfInterpreterIsGPL>

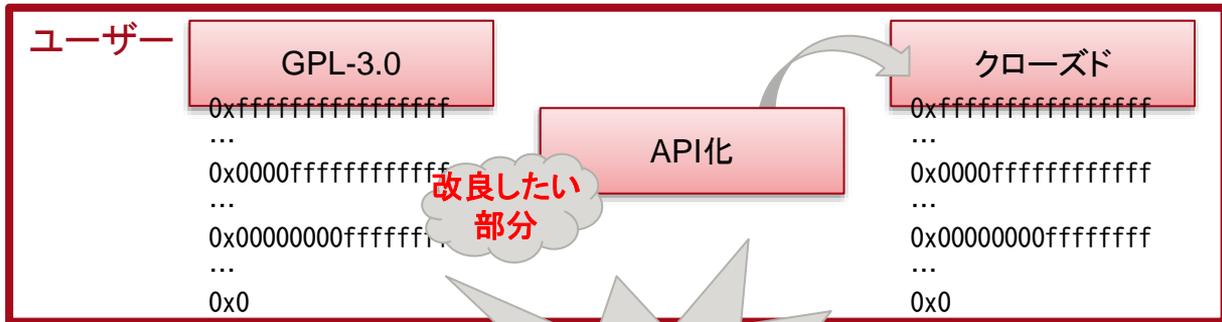
共有アドレス空間での関数呼び出し



空間を共有して
いるので結合

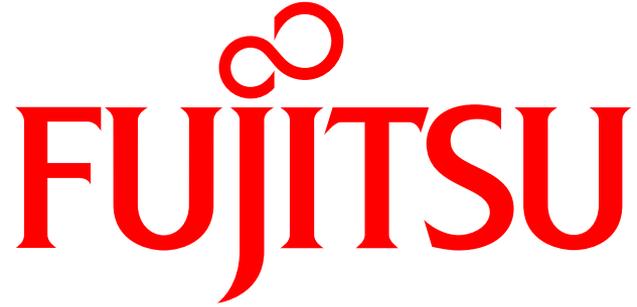
コミュニケーションのセマンティクスが親密

- コミュニケーション=通信の
セマンティクス=データが
親密!?



ライセンスの伝播回避
が目的なら許されない
(処理の分散が目的なら、..)

Q&A?



shaping tomorrow with you