SecHack365

ぼくがつくったさいきょうのおーえす

学習駆動コース
坂井ゼミ
石垣有逢

さいきょうとは！

- 簡単に使いこなせる！
- 簡単に動作がわかる！
- 簡単に開発ができる！

つくったおーえす

Raspberry Pi pico向けOS

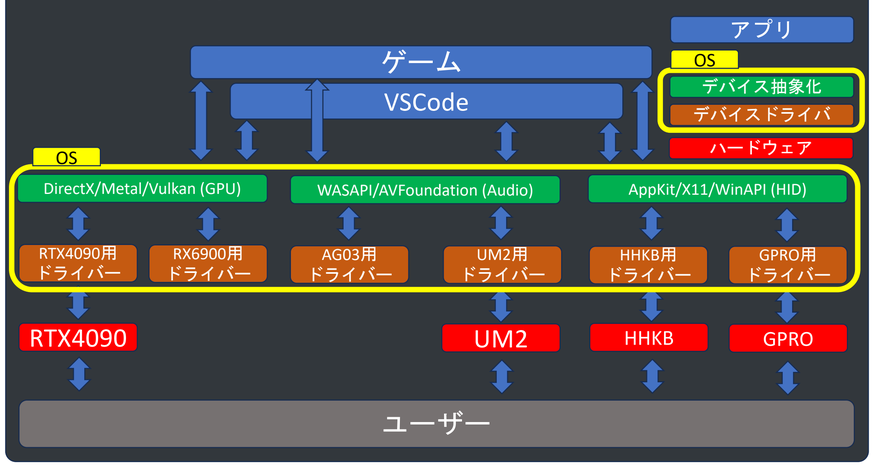
Shizuku

ソースコード
（GitHub）

OS(とその役割)とは何か

- デバイスの抽象化
- ファイルやデバイスの権限管理
- プログラム間の通信

パソコンの模式図



既存のOSはどうしてる？

全てはファイルである！(Linuxの場合)

- ファイルという存在があって、アプリケーションプログラムはカーネルを通して、読み込み操作と書き込み操作、制御命令(ioctlのこと)を出すことができる。
- Linuxというカーネル（実際にデバイスを操作することができるOSの根幹となるプログラム）はファイルへの操作を実現するファイルシステムを搭載している
- 情報を記録するためのファイルだけでなく、デバイスもファイルである。（CPUやGPU、USBメモリも全て特定のファイルとして扱うことができる。OSはUSBメモリが接続されたら、自動的に接続されたUSBメモリを表すファイルを作成する)
- 全てファイルとすることで、アプリケーションは限られたシステムコール(Linuxカーネルへの指示)のみで多種多様なデバイスを操作、管理、使用できる。
- デバイスドライバなどはカーネルモジュールという仕組みで動的に追加できる

Shizukuはどうする？

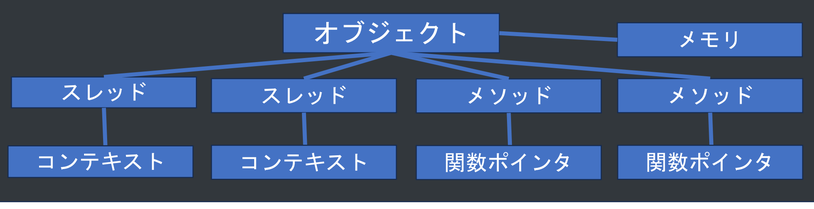
全てはファイルって考えてみると意外と難しい…

- ファイルという仕組みを実現させるのはそこまで難しくない。しかしマウント操作による拡張可能なディレクトリは実現することが難しい（ファイルシステムを規格化して、ファイルに対する操作を統一し、複数のファイルシステムが共存できるようにする必要がある。）
- VFSという複数のファイルシステムを共存させるためのインターフェースがあるが、VFSとして実行すべき処理と各固有のファイルシステムとして実行すべきか、およびその実行タイミングに悩みまくった。
- デバイスを抽象化したいだけだったらファイルじゃなくてもいいんじゃない？(悪魔の囁き)
- 情報の保存よりもデバイスの抽象化の方がやりたかった僕はファイルやファイルシステムに代わる新たな概念とその仕組みであるオブジェクトとオブジェクトシステムの考案へと乗り出したのであった…

さいきょうなしくみ「オブジェクトシステム」

オブジェクトとは～基本概念～

処理力(計算力)と記憶力を持つ
オブジェクトはメソッド(外部から呼び出せるプログラム)を公開できる
公開したメソッドは他のオブジェクトから呼び出すことができる



メソッドを公開して、利用する

ある一定のABI(Shizukuの場合size_t型の引数を4つ持つC言語の関数)に適合するメソッド(関数)を公開できる。
他のオブジェクトから、このメソッドを呼び出すことができる(ただしこのままではセキュリティ的に脆弱)
特定のデバイスなど一つのオブジェクトが独占管理して分配するのが望ましい資源(デバイス)を管理しているオブジェクト(のメソッド)経由で使用する。
メソッドが呼び出されたら、基本的には呼び出し先オブジェクト所属のスレッドとして実行される。

動作(活用)イメージ

簡単に言うとプロセス間通信で呼び出し元と呼び出し先がわかるマイクロカーネル

デバイスマネージャオブジェクト(ターミナルデバイスやUSBバスなどデータを中継するオブジェクトを想定)や、デバイスドライバオブジェクト(UARTやUSBデバイスドライバなど実際にデバイスを操作するためのオブジェクト)経由でデバイスを操作する。

オブジェクトが主に権限管理を行い、カーネルはオブジェクトが行う権限管理を支援する。

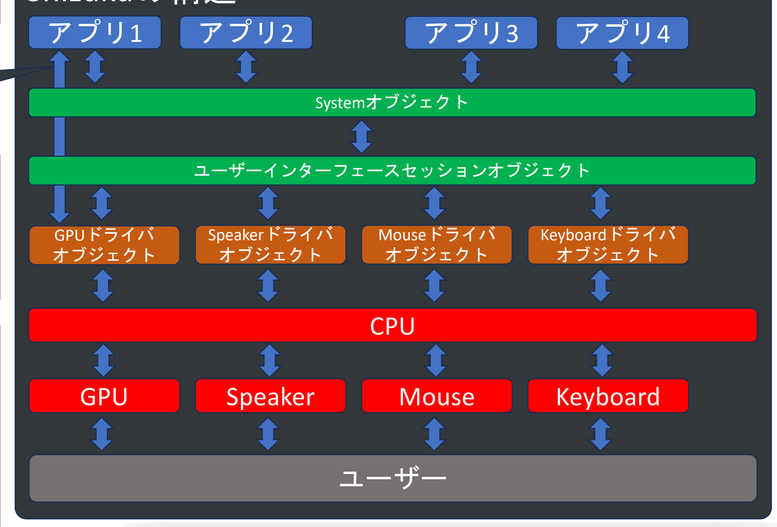
↓デモコード

```c
#include <pico/stdlib.h>
#include <shizuku/kernel.hpp>
#include <stdio.h>
void test_object_main(size_t callee_object_id, size_t creator_object_id,
                      int argc, char **argv);
int test_method(size_t callee_object_id, size_t caller_object_id, size_t arg1,
                size_t arg2);
void object_system_test_main() {
  //デモコードのエントリーポイント
  shizuku::kernel.create_object("test_object",
                                (shizuku::types::method)test_object_main, 1,
                                1); // オブジェクトを作成

  shizuku::kernel.abort_current_task();
}

void test_object_main(size_t callee_object_id, size_t creator_object_id,
                      int argc, char **argv) {
  // "test_object"のエントリーポイント
  shizuku::kernel.export_method(test_method,
                                "test_method"); // メソッドを公開
  shizuku::kernel.call_method(callee_object_id, "test_method", 1,
                              1); // メソッドを呼び出し
}
int test_method(size_t callee_object_id, size_t caller_object_id, size_t arg1,
                size_t arg2) {
  // "test_method"のエントリーポイント
  while (1) {
    printf("callee_object_id:%d\n", callee_object_id);
    printf("caller_object_id:%d\n", caller_object_id);
    printf("arg1:%d\n", arg1);
    printf("arg2:%d\n", arg2);
    sleep_ms(1000);
  }
}
```

↓実行結果

```
New Open Save Connect
callee_object_id:1
caller_object_id:1
arg1:1
arg2:1
callee_object_id:1
caller_object_id:1
arg1:1
arg2:1
callee_object_id:1
caller_object_id:1
arg1:1
arg2:1
callee_object_id:1
caller_object_id:1
arg1:1
arg2:1
callee_object_id:1
caller_object_id:1
arg1:1
arg2:1
usbmodem14101 / 9600 8-N-1
Connected 00:01:32, 5,208 / 0 bytes
```

Shizukuの構造



Systemオブジェクト経由で直接接続処理を行い高速化

妥協は許されない!

高速化機構・モノリシックカーネル化

- オブジェクトに他のオブジェクトのメソッドのエイリアス(ファイルディスクリプタもどき)を持たせる(持っているメソッドをArmedメソッドと定義し、こうなっていないメソッドはDisarmedメソッドと定義する。)
- ArmedメソッドはO(1)で呼び出せる(常に最速処理ができる)。
  Disarmedメソッドの呼び出し時間は保証されない
  (メソッドによっては時間がかかるかもしれない)
- ユーザー空間には一般オブジェクト、カーネル空間には特権オブジェクトを置き特権オブジェクトはIO操作ができるようにする

不満:
・同じメソッド繰り返し呼び出すたびに余計な検索処理
・ドライバオブジェクトに権限管理をさせたくない

欠点:
既存の仕組みに対応し難い
(ファイルディスクリプタがないので再現しようとするとSDKの実装が大変)

セキュリティ機構の導入～配下オブジェクトの導入～

- オブジェクトに配下オブジェクトを持たせる
- オブジェクトは、配下オブジェクトを自由にコントロールできる(作成や削除、スレッドの強制停止、メソッドの上書きまでなんでもできる)
- 特権オブジェクトはすべてのオブジェクトに対してすべての操作を行うことができる。

最終的なオブジェクトの図


```c
#include "shizuku/object.hpp"
using namespace shizuku::types;

void object::export_method(method &entry, shizuku::string const &name) {
  method_map.insert_or_assign(
      name, shizuku::method_shared_ptr{new shizuku::types::method{entry}});
}
```

pico_sdkに乗っかってるのでコード行数の大幅な削減に成功(ロジックはたった400行!)

```c
#include "shizuku/kernel.hpp"

using namespace shizuku::types;

int kernel::call_method(size_t callee_object_num,
                        shizuku::string const &method_name, size_t arg1,
                        size_t arg2) {
  if (auto caller_thread = this->get_current_thread().lock()) {
    if (auto caller_object = caller_thread->parent_object.lock()) {
      return this->call_method(callee_object_num, caller_object->object_id,
                               method_name, arg1, arg2);
    }
  }
  return 0;
}

int kernel::call_method(size_t callee_object_num, size_t caller_object_num,
                        shizuku::string const &method_name, size_t arg1,
                        size_t arg2) {
  return (*(object_tree[callee_object_num]->method_map[method_name].get()))(
      callee_object_num, caller_object_num, arg1, arg2);
}
```

本当はここはスレッドを作成する(一応これでもRP2040では動くけど満足できない動作)

```c
#include "shizuku/processors/rp2040.hpp"
namespace shizuku {
namespace processor = shizuku::types::processors::SHIZUKU_PROCESSOR;
using context = processor::context;
using cpu_driver = processor::cpu_driver;
constexpr unsigned int processor_count = SHIZUKU_PROCESSOR_COUNT;
} // namespace shizuku
#endif
```

Cmakeを使ってCPUのアーキテクチャを選択可能(つまり移植しやすい)

Objectクラスの中身

```c
#ifndef SHIZUKU_OBJECT_HPP
#define SHIZUKU_OBJECT_HPP
#include "cstdint"
#include "shizuku/config.hpp"
#include "shizuku/method.hpp"
#include "shizuku/thread.hpp"
namespace shizuku {
namespace types {

class kernel;
}
extern types::kernel kernel;
namespace types {
enum object_attribute { super_object, normal_object };
struct permission {
  bool writable;
  bool readable;
  bool executable;
  bool kernel_only;
};
struct method_descriptor {
  shizuku::object_weak_ptr callee_object;
  shizuku::method_weak_ptr method;
};
struct memory {
  void *real_start;
  void *virtual_start;
  size_t size;
  shizuku::types::permission permission;
  int operator<=>(const memory &right) const {
    return (int)this->virtual_start - (int)right.virtual_start;
  }
};

inline bool operator<(memory x, memory y) {
  return x.virtual_start < y.virtual_start;
}
class object {
  friend shizuku::types::kernel;
  shizuku::platform::std::string name;
  shizuku::platform::std::set<memory> memory_map;
  shizuku::map<shizuku::string, shizuku::method_shared_ptr> method_map;

  size_t total_thread_count;
  shizuku::map<size_t, shizuku::thread_shared_ptr> thread_map;
  shizuku::map<size_t, shizuku::object_weak_ptr> child_object_map;

public:
  const size_t object_id;
  const shizuku::types::object_attribute attribute;
  object(size_t this_object_id, size_t creator_object_id, method entry,
         uint32_t arg1, uint32_t arg2,
         shizuku::types::object_attribute attribute = normal_object)
      : object_id{this_object_id}, attribute{attribute} {
    total_thread_count = 1;
    thread_map[total_thread_count] = shizuku::thread_shared_ptr(
        new shizuku::types::thread(entry, this_object_id, creator_object_id,
                                   arg1, arg2, total_thread_count));
  }
  void export_method(method &entry, shizuku::string const &name);
  shizuku::thread_shared_ptr create_thread(method entry,
                                           size_t caller_object_id,
                                           size_t callee_object_id, size_t arg1,
                                           size_t arg2);
  void remove_thread(size_t thread_id);
};

} // namespace types

} // namespace shizuku

#endif
```

NICT National Institute of Information and Communications Technology