演習Ⅲ 最終発表
〈RTOSを自作する〉

演習Ⅲ 2026年度 第3期 品川研 — 05251014 杉山 衣吹

① RTOSとは

Real Time Operating Systemの略
コンテキストスイッチにおける時間を減らすことに特化している

特に組み込み環境では計算資源が乏しいため、**タスクの並列実行**(≒計算資源の分配)に力を入れ、他の機能はほぼない
例えば、メモリ管理や保護機能、ファイルシステムのサポートがない。

興味は以前からあって、自作して理解を深めようと思った

② 作ったもの

〈マイコンで動作する、プリエンプションができる最低限のRTOS〉
github.com/ibuki2003/ex3_rtos

対象CPU: RP2040（Cortex M0+（ARMv6-M）を2コア搭載したマイコン）→
　　選定理由: 自室にたくさんあって、使い慣れている

使用言語: Rust
　　選定理由: 低レベルな挙動が記述でき、使い慣れている

割り込み: タイマー、GPIO

カーネル部のコードサイズ: 964 Bytes

③ 構成

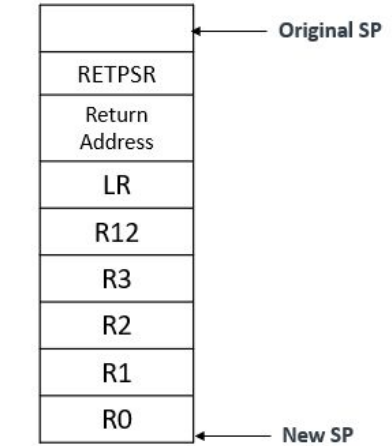
- （初期化部）
- TCB, user stack
- **割込ハンドラ**
  - save/restore
  - スケジューラ
- **システムコール**(subscribe for event)
- （ディスプレイ描画タスク）

③ 構成 — 割込ハンドラ

ARMv6-MにはMSP/PSPの2つのstack pointer regが存在しており、
task用とハンドラ(カーネルタスク)用で自然に使い分けられる

ARMコアの割り込みの挙動:

1. HWによって一部レジスタがstackに退避される
2. CONTROLレジスタの情報をEXC_RETURNとしてLR(リンクレジスタ)に設定
   (EXC_RETURNの値は0xFFFF_FFFDまたは0xFFFF_FFF9)
3. ベクタテーブルを参照しジャンプ
4. ハンドラからBX LRでreturnすると、PCがEXC_RETURNに設定される
5. コアがこれを検知し、HWによるstackingをrestoreする
6. 割込直前の位置から実行を再開



Exception frame saved by hardware on to stack

③ 構成 — 割込ハンドラ

context switchのためのsave/restoreには、
HW stackingだけでは足りないため、自力でasmを書く必要がある

R4-R11をpush/popするだけ

最後にEXC_RETURNとして0xFFFF_FFFDをLRに設定する

```
// save context: save regs(r4-r11) to stack¬
mrs r0, PSP¬
subs r0, r0, #16 // allocate for r4-r7¬
stmia r0!, {r4-r7}¬
subs r0, r0, #32 // allocate for r8-r11¬
mov r4, r8¬
mov r5, r9¬
mov r6, r10¬
mov r7, r11¬
stmia r0!, {r4-r7} // store r8-r11¬
subs r0, r0, #16 // move back to original PSP¬

// mov r0, r0 // arg1: sp¬
mov r1, LR  // arg2: exc_return¬

// call inner handler¬
bl {inner_handler}¬

// and then, restore the next task's context¬
ldmia r0!, {r4-r7}¬
mov r8, r4¬
mov r9, r5¬
mov r10, r6¬
mov r11, r7¬
ldmia r0!, {r4-r7}¬
msr PSP, r0¬

movs r0, #2 // ~0xFFFFFFFD¬
mvns r0, r0¬
mov LR, r0¬
bx LR¬
```

③ 構成 — タスク作成

タスクの実行は、開始時も割込ハンドラから行っている

HW unstackingで読まれるデータを事前にメモリ上に作成し、SPとしてそのアドレスを記憶する

PCにタスクの関数の先頭アドレス、
xPSRをThumbモードに設定している

```
pub fn setup_user_task(idx: usize, task: fn() -> !) {
    if idx >= TASK_COUNT {
        panic!("task index out of bounds");
    }
    unsafe {
        if !TASKS[idx].sp.is_null() {
            panic!("task stack already allocated");
        }

        let sp: *mut u32 = USER_STACK[idx].as_mut_ptr().add(count: USER_STACK[idx].len()); // bottom
        let sp: *mut u32 = sp.offset(count: -8); // allocate space for r4-r11

        // R0, R1, R2, R3, R12, LR, PC, xPSR
        *sp.offset(count: 5) = task as usize as u32; // LR
        *sp.offset(count: 6) = task as usize as u32; // PC
        *sp.offset(count: 7) = 0x01000000; // xPSR (Thumb bit set)

        let sp: *mut u32 = sp.offset(count: -8); // allocate space for exception stack frame
        TASKS[idx].sp = sp;
        TASKS[idx].state = TaskState::Ready;
    }
}
```

③ 構成 — システムコール

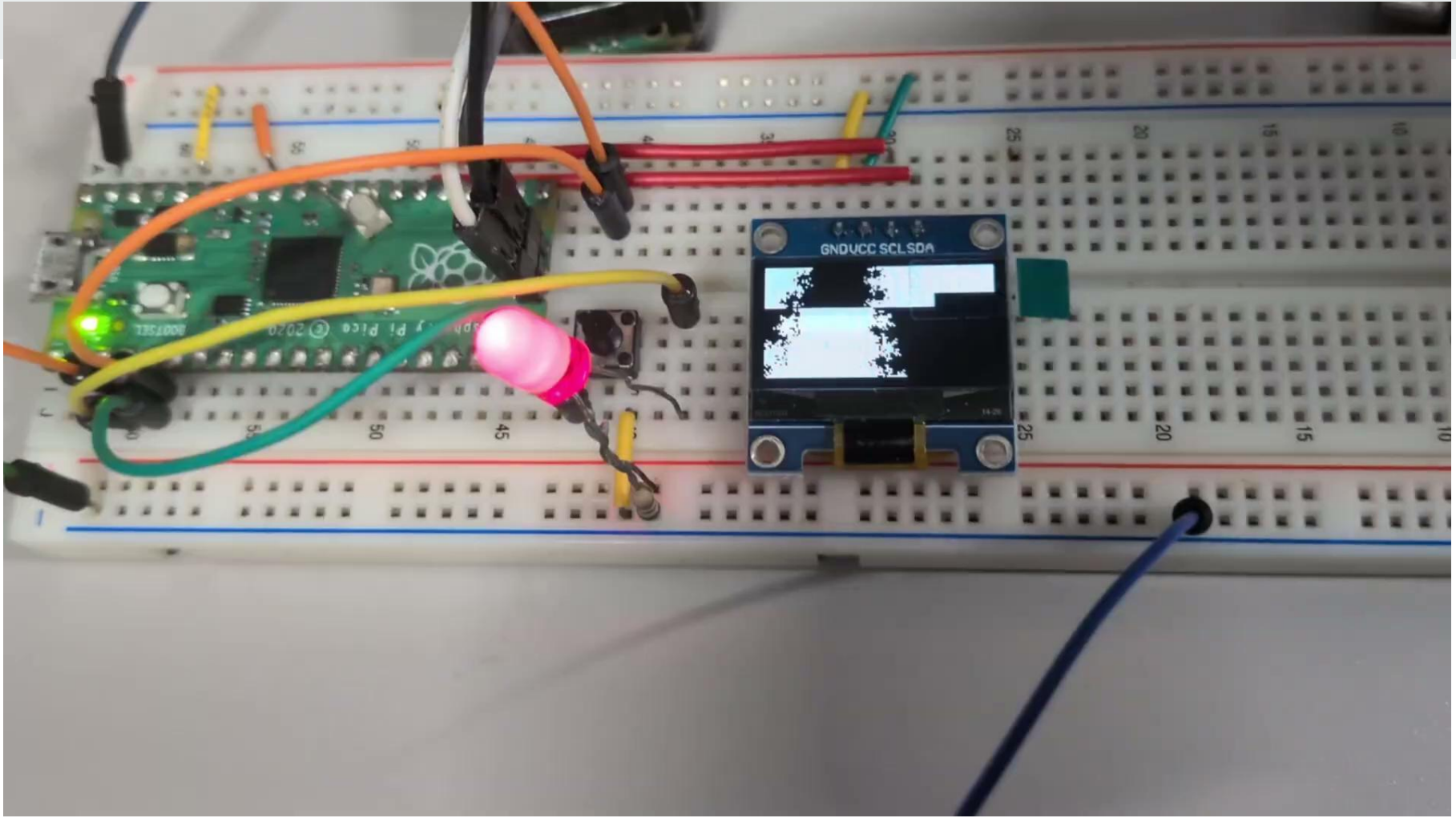
命令で呼び出せるSVCall（SuperVisor Call)例外が用意されている。これをtaskが休眠するタイミングで実行すればよい

```rust
/// kernel call to sleep the current task for a given number of microseconds
pub fn sleep(us: u32) {
    unsafe {
        setup_timer_wakeup(tid: RUNNING_TASK, us);

        // and then pause the current task by svcall
        // NOTE: no noreturn here, because the task will be resumed later
        asm!("svc 0", options(nomem, nostack));
    }
}
```

Table B1-3 Exception numbers

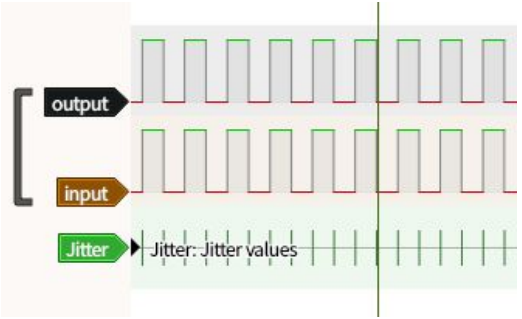
Exception number	Exception
1	Reset
2	NMI
3	HardFault
4-10	Reserved
11	SVCall
12-13	Reserved
14	PendSV
15	SysTick, optional
16	External Interrupt(0)
…	…
16 + N	External Interrupt(N)

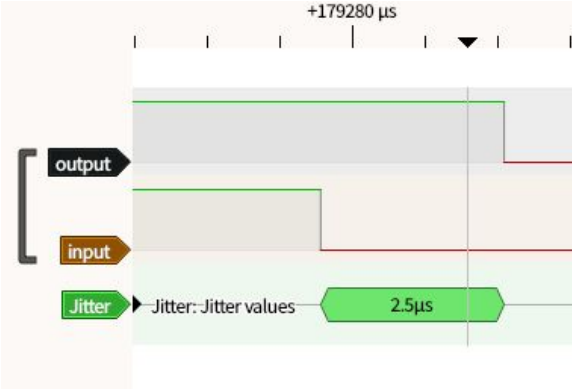

GNDVCC SCLSDA

④ 性能評価

RTOSの性能指標として、イベント発生からタスク切り替えまでの遅延時間を使用する

low-prio taskのCPU負荷を変えて、
high-prio taskの反応時間を測定



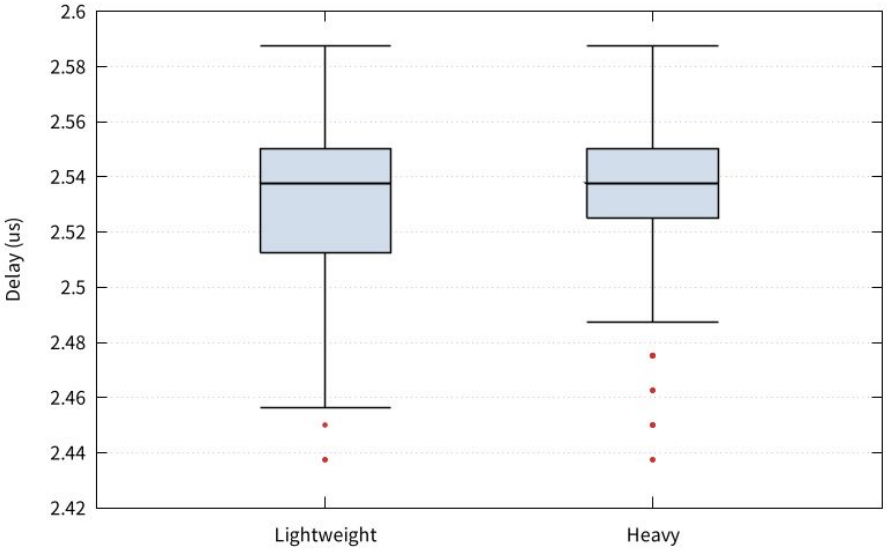


④ 性能評価

CPU負荷の有無で遅延時間の大きな差は見られ

- 最大値・最小値・中央値は**一致**
- 高負荷時 ばらつきがやや大

割り込みハンドラ内で最低限の処理ののち
context switchを行うため、
時間の予測がしやすい
→**RTOSとして望ましい振る舞い!**




```
--- fast2.txt ---
  N=500  median=2.537  Q1=2.513  Q3=2.55  IQR=0.0375
  min=2.438  max=2.587  whiskers=[2.456, 2.587]
--- slow2.txt ---
  N=500  median=2.537  Q1=2.525  Q3=2.55  IQR=0.025
  min=2.438  max=2.587  whiskers=[2.487, 2.587]
```

参考までに、既存実装の例としてFreeRTOSで200cy程度らしい やや劣る？

⑤ まとめ

- Armのデータシートと「にらめっこ」して、理解を深められた
  - 割り込み周辺の振る舞い、呼び出し規約との関連
  - ベクタテーブル
  - あとELFともにらめっこしました
- 今後の展望（というより、開発したRTOSに残る改善点）
  - Mutex、タスク間通信
  - スケジューラの改善 現在は毎回全タスクを舐めている

⑥ 参考資料

- ARMv6-M Architecture Reference Manual