リストA Cargo.tomlへの追加部分

```
toml
[dependencies]
esp-idf-sys = { version = "=0.32", features = ["binstart"] }
esp-idf-svc = "=0.45"
log = "0.4"
```

図A コンソールにHello, worldがESP-IDFのログ・システムと同じ形式で出力される

```
I (310) cpu_start: Starting scheduler.
I (315) log: Hello, world!
E (315) log: oops! 1
```

リストB エラーの変換処理を書かないとコンパイルできないコード

```
fn main() -> /* ここはどうすれば良い？ */ {
    // std::num::ParseIntErrorのエラー型を返す関数
    let num = i32::from_str_radix("15", 10)?;

    let mut buf: [u8; 64] = [0u8; 64];
    let mut buffer = std::io::BufWriter::new(&mut buf[..]);
    // std::io::Errorのエラー型を返す関数
    buffer.write(&[1u8])?;

    Ok(())
}
```

リストC Cargo.tomlにanyhowの依存を追加する

```
toml
anyhow = "1"
```

リストD エラーを発生させてみる

```
diff
- let num = u8::from_str_radix("15", 10)?;
+ let num = u8::from_str_radix("A5", 10)?;
```

リストE コレクションを使うexamples/alloc.rs

```
use std::collections::BTreeMap;
use log::*;

fn main() -> anyhow::Result<()> {
    esp_idf_sys::link_patches();
    esp_idf_svc::log::EspLogger::initialize_default();

    let mut xs = Vec::new();
    for i in 0..5 {
        xs.push(i);
    }
    info!("{:?}", xs);

    let s = String::from("Hello!");
    info!("{}", s);
```

```
    let mut m = BTreeMap::new();
    m.insert("one", 1);
    m.insert("two", 2);
    m.insert("three", 3);
    info!("{:?}", m);
    info!("{:?}", m.get("one"));

    Ok(())
}
```

図B examples/thread.rsの実行結果

```
I (316) thread: main thread: Thread { id: ThreadId(1), name: Some("main"), .. }
I (317) thread: child 0 thread: Thread { id: ThreadId(2), name: None, .. }
I (317) thread: child 1 thread: Thread { id: ThreadId(3), name: None, .. }
I (327) thread: child 2 thread: Thread { id: ThreadId(4), name: None, .. }
I (337) thread: child 3 thread: Thread { id: ThreadId(5), name: None, .. }
I (347) thread: child 4 thread: Thread { id: ThreadId(6), name: None, .. }
```

リストF sdkconfig.defaultsへの追加

```
CONFIG_FREERTOS_USE_TRACE_FACILITY=y
CONFIG_FREERTOS_USE_STATS_FORMATTING_FUNCTIONS=y
```

リストG vTaskListを呼び出す

```
fn print_freertos_tasks() {
    let mut buf = [0u8; 1024];
    unsafe { esp_idf_sys::vTaskList(buf.as_mut_ptr() as *mut i8) };
    // stack hwm: stack high water mark
    // The minimum amount of stack space that has remained for the task since
    // the task was created.  The closer this value is to zero the closer the task
    // has come to overflowing its stack.
    info!("tasks:\n
name          state  priority stack hwm id\n
{}",
        String::from_utf8_lossy(&buf)
    );
}
```

図C 動いているタスク

```
I (459) thread: tasks:
name          state  priority stack hwm id
main            X       1        4720    2
IDLE            R       0        1236    3
esp_timer       S       22       3700    1
Tmr Svc         B       1        1648    4
```

リストH Resultを返す

```
fn threads_return_result() {
    let mut threads = Vec::new();
    for i in 0..5 {
        let t = thread::spawn(move || -> anyhow::Result<u32> {
            if i == 4 {
                return Err(anyhow::anyhow!("thread 4 returns error!"));
            }
```

```
            Ok(i)
        });
        threads.push(t);
    }

    for t in threads {
        let ret = t.join().expect("thread failed");
        info!("thread returns: {:?}", ret);
    }
}
```

図D　スレッドの戻り値

```
I (1459) thread: thread returns: Ok(0)
I (1459) thread: thread returns: Ok(1)
I (1459) thread: thread returns: Ok(2)
I (1469) thread: thread returns: Ok(3)
I (1469) thread: thread returns: Err(thread 4 returns error!)
```

リストI　スタック・サイズを確認する側のコード

```
    print_freertos_tasks();

    let t = thread::spawn(move || {
            thread::sleep(Duration::from_secs(1));
        });

    print_freertos_tasks();

    t.join().expect("thread railed");
```

リストJ　スレッド・ビルダでスレッドを生成する

```
fn thread_builder() -> anyhow::Result<()> {
    print_heap_free_size();

    let t = std::thread::Builder::new()
        .name("named thread".to_string())
        .stack_size(8092)
        .spawn(|| {
            info!("thread: {:?}", thread::current());
            thread::sleep(Duration::from_secs(1));
        })?;

    print_heap_free_size();
    print_freertos_tasks();

    t.join().expect("thread railed");
    Ok(())
}
```

図E　スレッド・ビルダでスレッドを生成した実行結果

```
I (1469) thread: heap: 319284
I (1469) thread: thread: Thread { id: ThreadId(12), name: Some("named thread"), .. }
I (1479) thread: heap: 310392
I (1489) thread: tasks:
name          state  priority stack hwm id
main            X       1       4672    2
IDLE            R       0       1204    3
```

```
pthread         B       5       6572    25
Tmr Svc         B       1       1648    4
esp_timer       S       22      3700    1
```

リストK　タスク・プライオリティの変更

```
fn change_thread_priority() {
    let mut threads = Vec::new();
    for i in 0..5 {
        let t = thread::spawn(move || {
            unsafe {
                esp_idf_sys::vTaskPrioritySet(std::ptr::null_mut(), 5 + i);
            }
            thread::sleep(Duration::from_secs(1));
        });
        threads.push(t);
    }

    print_freertos_tasks();

    for t in threads {
        t.join().expect("thread failed");
    }
}
```

図F　プライオリティが変化している

```
I (2479) thread: tasks:
name            state   priority stack hwm id
main              X       1       4672    2
IDLE              R       0       1204    3
pthread           B       6       2436    28
pthread           B       7       2428    29
pthread           B       8       2436    30
pthread           B       9       2428    31
pthread           B       5       2436    27
esp_timer         S       22      3700    1
Tmr Svc           B       1       1648    4
```

リストL　moveを使うコード

```
    // スレッドを5つ作成する
    for i in 0..5 {
        // i を親スレッドからムーブするために move をつける
        let t = thread::spawn(move || {
            // 作成されたスレッドは，何番目に作成されたか，と，自分自身のスレッド情報を出力する
            info!("child {} thread: {:?}", i, thread::current());
        });
```

図G　moveを取り除くとコンパイル・エラーが発生する

```
error: could not compile play-std due to 2 previous errors
error[E0373]: closure may outlive the current function, but it borrows i, which is owned by the current
function
  --> examples/thread.rs:46:31
   |
46 |         let t = thread::spawn(|| {
   |                               ^^ may outlive borrowed value i
47 |             info!("child {} thread: {:?}", i, thread::current());
   |                                            - i is borrowed here
```

```
   |
note: function requires argument type to outlive 'static
  --> examples/thread.rs:46:17
   |
46 |           let t = thread::spawn(|| {
   |  _________________^
47 | |             info!("child {} thread: {:?}", i, thread::current());
48 | |         });
   | |__________^
help: to force the closure to take ownership of i (and any other referenced variables), use the move keyword
   |
46 |         let t = thread::spawn(move || {
   |                               ++++

error: aborting due to previous error

For more information about this error, try rustc --explain E0373.
```

リストM　イミュータブルなスタティック変数を共有する

```
    static I: usize = 1;
    let t = thread::spawn(|| {
        info!("{}", I);
    });
    t.join().expect("thread railed");
```

図H　生成されたスレッド数を数えた結果

```
I (317) sync: created threads: 1
I (317) sync: created threads: 2
I (317) sync: created threads: 3
I (327) sync: created threads: 4
I (327) sync: created threads: 5
```

図I　構造体Dataの共有の実行結果

```
I (337) sync: Data { a: 0, b: 1, c: 2 }
I (337) sync: Data { a: 0, b: 1, c: 2 }
I (337) sync: Data { a: 0, b: 1, c: 2 }
I (347) sync: Data { a: 0, b: 1, c: 2 }
I (347) sync: Data { a: 0, b: 1, c: 2 }
```

図J　Mutexで共有している値の排他制御をする実行結果

```
I (357) sync: Data { a: 1, b: 1, c: 2 }
I (357) sync: Data { a: 2, b: 1, c: 2 }
I (367) sync: Data { a: 3, b: 1, c: 2 }
I (367) sync: Data { a: 4, b: 1, c: 2 }
I (377) sync: Data { a: 5, b: 1, c: 2 }
```