Pen 言語の進捗報告

@raviqqe

目次

- 進捗報告
  - TCP/UDP インターフェース
  - 参照カウント GC の変更
- 今後の予定
  - 非同期ランタイムの実装

進捗報告

TCP･UDP インターフェース

- `os` 標準パッケージの拡張
- TCP･UDP が使えるようになった
  - サーバ･クライアントの両方
- WASI のソケット扱い周りが何も決まってないことを知る
  - 起動時にもらったソケットにしかアクセスできない
  - 元になった Cloud ABI が capability-based security を基礎にしている
    - ソケット勝手に開けられるのは困る
  - QUIC みたいなマルチストリームなプロトコルでは問題ない?

参照カウント GC の変更

- クローン関数を Rust と互換性のある型に変更
  - 旧：`fn (payload: u64)`
  - 新：`fn (payload: u64) -> u64`
- `Box` 等より多くの Rust の型が Pen 側でも使えるように
- サンクの実装のメモリーリークの修正
  - Conway's game of life の例でのメモリーリークが直った

今後の予定

非同期ランタイムの実装

- Rust の非同期ランタイムをそのまま持ってくる
  - 基本的な型やインターフェースは Rust の標準ライブラリが提供
    - e.g. `Future`, `Task`, ...
  - それらを実行するランタイムはライブラリが実装しないといけない
    - `tokio` と `async-std` がメジャー
    - シングルスレッドとマルチスレッド（work stealing scheduler）の両方ある

Rust が Pen 言語の非同期関数を呼ぶ場合

- `poll_fn()` を使う
  - `Future` っぽい関数を `Future` にする高階関数
  - まだ、nightly でしか使えない
- 基本的に main 関数と一部の並列計算関数の実装にのみ使う

```
pub fn poll_fn<T, F>(f: F) -> PollFn<F> where
    F: FnMut(&mut Context<'_>) -> Poll<T>,
```

Pen が Rust 言語の非同期関数を呼ぶ場合

- `Future::poll()` を使う
- I/O、ネットワーク処理等色々使う

```
pub trait Future {
    type Output;
    fn poll(self: Pin<&mut Self>, cx: &mut Context<'_>) -> Poll<Self::Output>;
}
```

例

```
unsafe extern "C" fn _pen_foo(
    stack: *mut ffi::cps::Stack,
    continuation: extern "C" fn(*mut ffi::cps::Stack, f64) -> ffi::cps::Result,
) -> ffi::cps::Result {
    let context = stack.async_context();

    let future = if let Some(future) = restore() {
      future
    } else {
      foo()
    };

    let value = match future.poll(context) {
        Poll::Ready(value) => value,
        Poll::Pending => {
            save(future);

            return ffi::cps::Result::new(...);
        },
    }

    continuation(stack, value)
}
```

まとめ

- TCP・UDP が使えるようになった
- 参照カウント GC が熟れてきた
- 非同期ランタイムは大変