Interview Template

Logger-Editor

Rust and Wyvern
Step 1 (~30 mins)

Requirements for the logger library:

1. The name of the log file should be `log.txt`
2. Location for the log file
    a. Rust - should be in `$DATA_DIR` (capability library provides better support for that)
    b. Wyvern - should be in the same folder as the program
3. The logger should contain the functionalities for the following (note that one can change the function names:
    a. `create_logger(logFile : String)` - A constructor which returns a new logger object with the name `logFile`
    b. `append_to_log(entry : String)` - Append a new `entry` to the `logFile`

Rust
Given a template `extension.rs, main.rs` file [1] - design the corresponding Logger module with capability library in Rust ([2], [3])

A potential template is given as follows:

```rust
/* Some imports the user may / may not need */
// use cap_directories::{ambient_authority, ProjectDirs};
// use cap_std::fs::{Dir, OpenOptions};
// use std::ffi::OsString;
// use std::io::Write;
// use std::path::PathBuf;

pub struct Logger {
    // TODO: Define the fields for logger structure
}

impl Logger {
    pub fn create_logger(rel_file_name: &str) -> anyhow::Result<Self> {
        Ok(Logger {})
    }
    pub fn append_to_log(&self, entry: String) -> anyhow::Result<()> {
        Ok(())
    }
}
```

The following documentation may be useful

[1] https://docs.rs/cap-std/0.26.1/cap_std/
[2] https://docs.rs/cap-directories/0.26.1/cap_directories/
[3] https://doc.rust-lang.org/std/

Wyvern

For Wyvern, the extension library is `wordCloud`, and you have to design the `logger` library. Since capability security is inbuilt you are given more freedom as to how to call the logger library from the main function.

```
import fileSystem
import logger
import wordFactory
import wordCloud

val fs = fileSystem(java)
val logFile = fs.fileFor("log.txt")

// val logger = ??
val word = wordFactory.create("temp")
val wordCloud = wordCloud(logger, wordFactory, word)
wordCloud.updateCloud()
```

Step 2 (~20 mins)

Upon completing the corresponding functions, now try to break the security of the filesystem in the corresponding programs only by modifying `extension.rs` (for Rust) and `wordCloud.wyv` (for Wyvern).

Step 3 (~10 mins)

Please provide your ratings out of 5 on the following:

1. How useful do you think capabilities are?
2. How much did you like working on Wyvern?
3. How much did you like working on Rust?
4. How much do you think you understand the concept of capabilities?

Subjective questions:

1. Is there a part of the language / task design which the participant would want to be improved?
   Shuffle the order of languages