98-023A : Concurrent and Distributed Programming w/ Inferno and Limbo

Phillip Stanley-Marbell
pstanley@ece.cmu.edu

Lecture Outline

- More Limbo data types
- Project discussion

Course Outline : Syllabus

- Week 1: Introduction to Inferno
- Week 2: Overview of the Limbo programming language
- Week 3: Types in Limbo
- Week 4: Inferno Kernel Overview
- Week 5: Inferno Kernel Device Drivers
- Week 6: NO CLASS **February 16th** and **February 18th**
- Week 7: C applications as resource servers: Built-in modules and device drivers

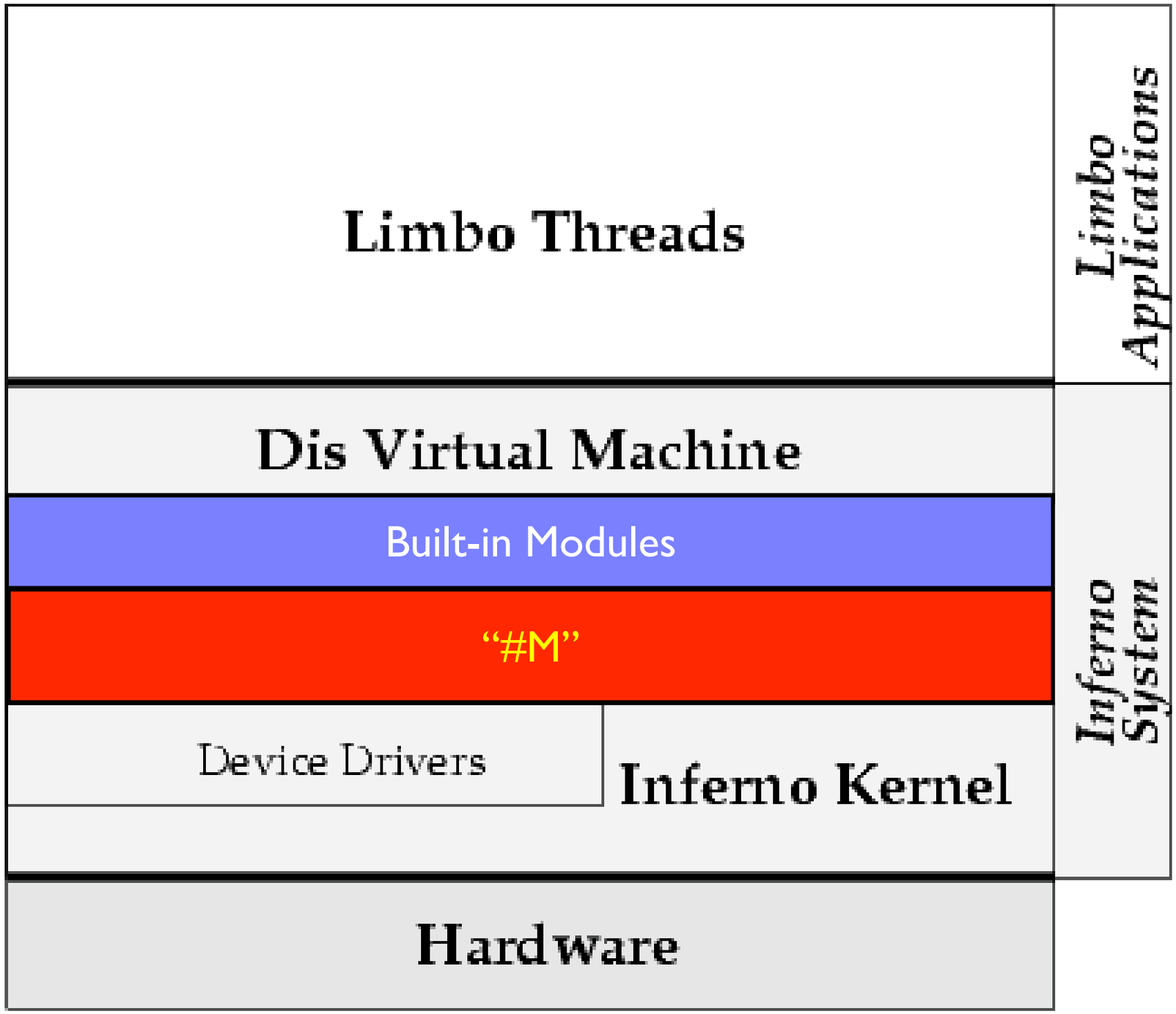
Spring Break

- Week 7: Case study I — building a distributed multi-processor simulator
- Week 8: Platform independent Interfaces: Limbo GUIs; Project Update
- Week 9: Programing with threads, CSP
- Week 10: Debugging concurrent programs; Promela and SPIN
- Week 11: Factotum, Secstore and Inferno's security architecture
- Week 12: Case study II — Edisong, a distributed audio synthesis and sequencing engine

Status

- We've learnt a bit about Inferno in general
- We've seen an introduction to Limbo
  - Today we'll learn more about data types in Limbo
- Next, Dis VM, Inferno Kernel
- *Once we've gotten a good feel for both Inferno and Limbo, we'll start looking specifically at concurrency and building distributed applications*

Inferno System Structure



Unicode vs. ASCII

- ASCII and Unicode are encodings or code sets for representing characters — they assign unique numbers to characters
- ASCII is a 7 bit encoding, with 128 member characters, ranging from 0 (NUL) to 127 (DEL) with everything in between (e.g., p is 112)
- Unicode is a 16 bit encoding (well, the full ISO 10646 is 32 bits)
- UTF-8 is an encoding for representing a Unicode characters
  - A 16 bit Unicode character maps to 1, 2 or 3 UTF-8 bytes

Strings

- Strings are sequences of Unicode characters
  - The length of a string in terms of characters is therefore not always the same os length in bytes. Why ?

  - Anything wrong with the following statements ?

```
c : string;
c = ‘p’;
d : string;
d = “Hello!”;
d[5] = ‘?’;
d[5] = 33;
d[7] = ‘?’;
d[6] = ‘?’;
```

Lists

- Can create lists of any (single) data type

```
menu0 : list of string;
menu1 := list of {”Quinoa”, “Soy”};
menu0 = “Soy”::menu0;
menu0 = “Quinoa”::menu0;
```

```
p = hd menu0;
q = hd menu1;

x := tl menu0;
y := “Soy”;
```

Arrays

- Arrays of any (single) data type
  ```
  jim : array of int;
  ```
- Declaration (above) does not allocate storage for array
  - After the above, you cannot do
    ```
    jim[4] = 2;
    ```
- Allocation must explicitly be performed, either
  ```
  jim = array [32] of int;
  ```
- Or, at the same time as declaration
  ```
  jim := array [32] of int;
  ```
- Can statically initialize array elements at declaration
  ```
  jim := array [] of {"James", "J.", "Mikusi"};
  ```

Array and String Slices

- Can take "slices" or subset ranges of arrays and strings

```
jim := array [] of {"James", "J.", "Mikusi"};

lastname := jim[2:3];
```

  - Can leave out top or bottom index if array/string is the source

```
lastname := jim[2:];

firstname := jim[:1];

nickname := jim[0][:1]+"imbox";
```

Arrays, Strings and UTF 8

- Recall, strings are sequences of Unicode characters, each of which may need more than one UTF-8 byte for its representation

```
english : = “ants”;

greek  := “μυρμιγκια”;

englishlen := len english;

greeklen := len greek;
```

- Cast to array of byte converts a Unicode string to ... an array of bytes

```
englishbytes := array of byte english;

greekbytes := array of byte greek;
```

Demo / Example

Tuples

- Tuples are collections of data items of any number of types

```
info := (”Jane”, “Doe”, 22, 3.8);
```

- Most useful as the return types of functions

```
myfun(args : list of string) : (string, array of byte)
{
  ...
  if (error)
  {
    return (”error”, nil);
  }
  return (””, array [] of {byte 22, byte 33});
}
```

englishbytes = {byte "a", byte "n", ...} strlen = len s;

:=

a: list of String

a := ("hello", 7); s: String;

s = "hello!";

array of byte "H" →

s = 'P';

s[5] = '?';

jm := array[] of {"J", "i", "m"};

jm[0]

lastname := jm[2:3]

jm[0][:1] == "J"

Example: CacheLib

More...

- Pick ADTs
- Parametric polymorphism
- Fixed point types
- Type definitions
  - e.g., `long : type int;`

Homework 2 *(due Feb. 11th)*

- Question 1
  a. Install the Inferno emulator
  b. Change the string defined in include/version.h to one of your own choice
  c. Compile the emulator
  d. Submit an executable emulator binary for your platform

- Question 2
  a. Implement a Limbo program that reads in a text file and shifts each letter one step in the alphabet, i.e., a -> b, b -> c, z -> a, Z -> A etc., and prints the resulting transformed text.

Possible Project Ideas

- Extend the emulator or native kernel to make process creation possible through the name space

- Implement a load balancer that works through the name space, and manages the process creation interface of multiple hosts, add / delete hosts, etc

- Implement a Server that speaks an extended version of Styx, that can:
  - Associate a Limbo data type with each name space entry
  - New Styx messages T_NAMETYPE and R_NAMETYPE

- Extend the emulator or native kernel to add the ability to associate any valid Limbo data type with a name in the namespace
  - Will involve extending Styx ([T/R]_NAME2TYPE
  - All device driver interfaces will have to change

- Any other project of your own design / desire


/proj/c++
/projects
/proj/c++
/projects
/de

Course Outline : Grading

- First 4 homeworks are mandatory, the remainder are optional (5% each)
- 1 mini project (20%)
- 1 final project (60%)
- You should not be worried about your grade

Reading

- Relevant chapter in textbook : Chapter 3

Next Lecture

- The Dis VM internals pertaining to Limbo data types

Fin.