CSE461: Discussion section, January 9, 2002

Upcalls

In a conventional structured program, we have an explicit outline of program flow:

```
main {
  do_first();
  do_second();
  if ( …) {
    do_third();
  } else {
    do_fourth();
  }
  exit(0);
}
```

In a networking program, we don't have a well-defined sequence of events: we have to handle things in the order they happen in the "real world".

```
  Handle packet from Alice
  Handle packet from Bob
  Handle a string input by the User
  Handle packet from Alice
```

How do we structure such a program? One answer: upcalls.

An "upcall" occurs when the system calls one of the program's functions.

Instead of explicitly telling the system the sequence of things to do, we instead tell it what to do whenever some event occurs:

```
main {
  fish_recvhook(receive_packet);
  fish_keybhook(read_string);
  fish_main();
}
```

This tells the system that when it receives a packet, it should call receive_packet and when it gets a line of keyboard input, it should call read_string.

Function pointers

But how do we tell the system what functions to call? Answer: Function pointers.

Function pointers give us flexibility

- We CAN store a pointer to a function in a variable.
- We CAN call the function pointed to by a function pointer.
- We CANNOT modify the functions themselves using the pointers.
- This allows us to specialize generic code: we give the system a pointer to the function we want it to call at runtime.

Function pointer sample program:

```
#include <stdio.h>
#include <stdlib.h>

//Create a typedef named greeting_type for the type
//void (*) (char *,char *)
typedef void (*greeting_type) (char *,char *);

void print_greeting(char *first_name,char *last_name) {
  printf("Hello: %s %s\n",first_name,last_name);
}

//Take a function pointer as a parameter and call this function
void use_func(void (*ptr) (char *,char *)) {
  //Call the function pointed at by ptr()
  ptr("Sushant","Jain");
}

int main(void) {
  //Directly make a function pointer variable;
  void (*function_ptr) (char *,char *);
  //Use a typedef to make a function pointer variable
  greeting_type function_ptr2;

  //Assign and use the pointer
  function_ptr = &print_greeting;
  function_ptr("Eric","Lemar");

  //Assign and use the pointer
  function_ptr2 = &print_greeting;
  function_ptr2("Janet","Davis");

  //Pass a function pointer to use_func
  use_func(print_greeting);
}
```

Output:

```
Hello: Eric Lemar
Hello: Janet Davis
Hello: Sushant Jain
```