The image part with relationship ID rId2 was not found in the file.

Department of Computer Science

Lecture #26 – part 1

Non-local Jumps

(Ch. 8 end - Sec. 8.6)

Prof. Soraya Abad-Mota, PhD

The image part with relationship ID rId2 was not found in the file.

Previously

- Covered signals as part of the Exceptional control flow:
  1. What is a signal?
  2. What do signals have to do with exceptional control flow?
  3. How are signals handled? Can you provide an example of a signal and how it is "processed"?
  4. How do signals compare with exceptions? (synchronous or asynchronous?)

Questions about signals?

The image part with relationship ID rId13 was not found in the file.

Remaining for this semester (4 lectures!)

- Today:
  - Finish Ch. 8 ECF with non-local jumps (.25)
  - Start Virtual Memory (1.75 lectures)
- System-level I/O (.75)
- Concurrency (.75)

- Final Exam: Wednesday December 11, 2024
  - 7:45 to 9:45 pm is the schedule, for those answering false or not answering (and without other accommodations)
  - At 5pm pending the Dean of School of Engineering response

The image part with relationship ID rId13 was not found in the file.

Topics

- Signals
  - definition
  - sending and receiving
  - pending and blocked + process groups
  - Signal handlers and Signal Concurrency
- Non-local returns in C
  - nonlocal jumps
- Summary

The image part with relationship ID rId13 was not found in the file.

Nonlocal Jumps: `setjmp/longjmp`

- Powerful (but dangerous) user-level mechanism for transferring control to an arbitrary location
  - Controlled way to break the procedure call / return sequence
  - Useful for error recovery and signal handling

- `int setjmp(jmp_buf j)`
  - Must be called before longjmp
  - Identifies a return site for a subsequent longjmp
  - Called **once**, returns **one or more** times

- Implementation:
  - Saves current calling environment, i.e. current ***register context***, ***stack pointer***, and ***PC value*** in buffer j (a **`jmp_buf)`**
  - Return 0

`setjmp/longjmp` (cont)

- `void longjmp(jmp_buf j, int i)`
  - Meaning:
    - return from the **`setjmp`** remembered by jump buffer **`j`** again ...
    - ... this time returning **`i`** instead of 0
  - Called after **`setjmp`**
  - Called **once**, but **never** returns

- `longjmp` Implementation:
  - Restore register context (stack pointer, base pointer, PC value) from jump buffer **`j`**
  - Set **`%eax`** (the return value) to **`i`**
  - Jump to the location indicated by the PC stored in jump buf **`j`**

`setjmp/longjmp` Example 1

- Goal: return directly to original caller from a deeply-nested function

```c
/* Deeply nested function foo */
void foo(void)
{
    if (error1)
      longjmp(buf, 1);
    bar();
}

void bar(void)
{
    if (error2)
      longjmp(buf, 2);
}
```

The image part with relationship ID rId13 was not found in the file.

setjmp/longjmp Example (cont)

```
jmp_buf buf;

int error1 = 0;
int error2 = 1;

void foo(void), bar(void);

int main()
{
    switch(setjmp(buf)) {
    case 0:
        foo();
        break;
    case 1:
        printf("Detected an error1 condition in foo\n");
        break;
    case 2:
        printf("Detected an error2 condition in foo\n");
        break;
    default:
        printf("Unknown error condition in foo\n");
    }
    exit(0);
}
```

Limitations of Nonlocal Jumps

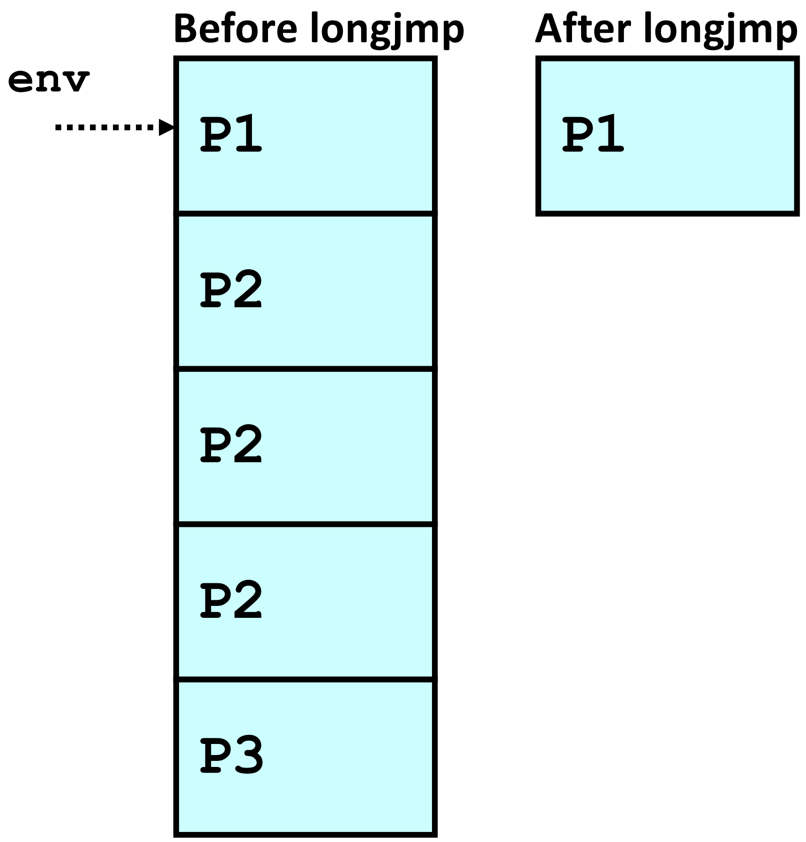
- Works within stack discipline
  - Can only long jump to environment of function that has been called but not yet completed

```
jmp_buf env;

P1()
{
  if (setjmp(env)) {
    /* Long Jump to here */
  } else {
    P2();
  }
}

P2()
{  . . . P2(); . . . P3(); }

P3()
{
  longjmp(env, 1);
}
```



Limitations of Long Jumps (cont.)

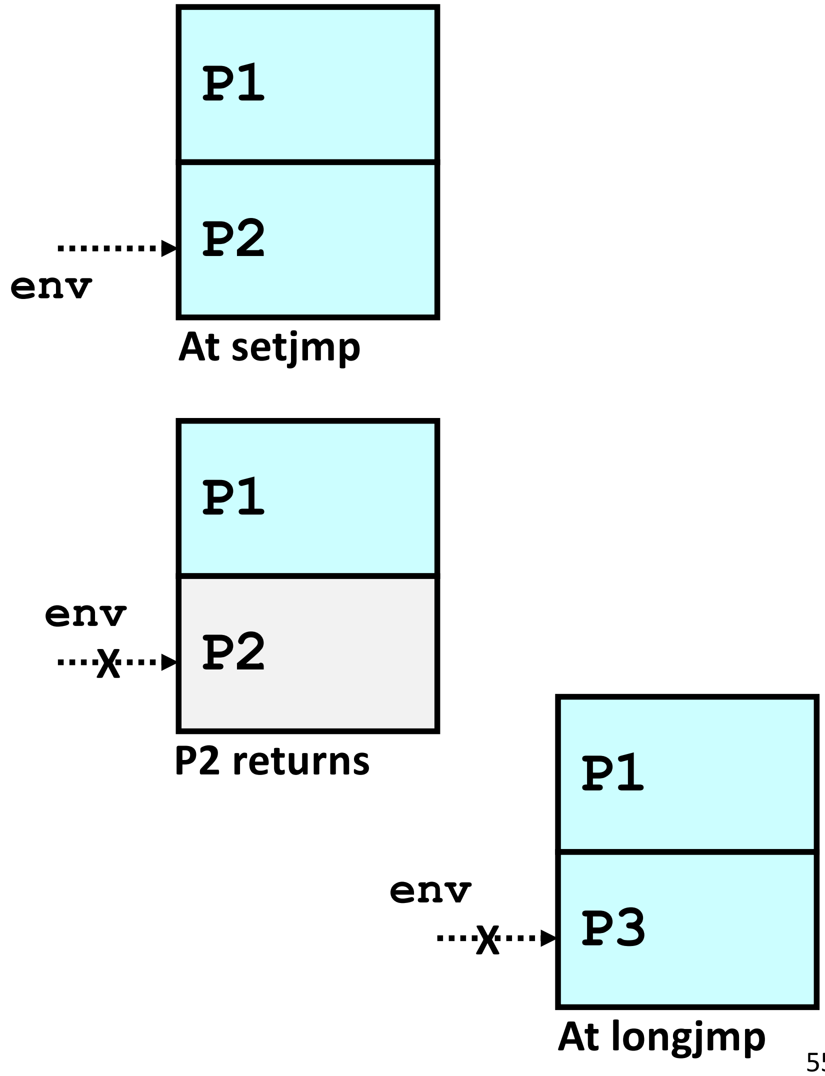
- Works within stack discipline
  - Can only long jump to environment of function that has been called but not yet completed

```
jmp_buf env;

P1()
{
  P2(); P3();
}

P2()
{
   if (setjmp(env)) {
    /* Long Jump to here */
  }
}

P3()
{
  longjmp(env, 1);
}
```



The image p
the file.

Putting It All Together: (Example 2) A Program That Restarts Itself when ^-c'd

```c
#include "csapp.h"

sigjmp_buf buf;

void handler(int sig)
{
    siglongjmp(buf, 1);
}

int main()
{
    if (!sigsetjmp(buf, 1)) {
        Signal(SIGINT, handler);
        Sio_puts("starting\n");
    }
    else
        Sio_puts("restarting\n");

    while(1) {
     Sleep(1);
     Sio_puts("processing...\n");
    }
    exit(0); /* Control never reaches here */
}
```

restart.c

```
greatwhite> ./restart
starting
processing...
processing...
processing...
restarting          Ctrl-c
processing...
processing...
restarting          Ctrl-c
processing...
processing...
processing...
```

Summary

- Signals provide process-level exception handling
  - Can generate from user programs
  - Can define effect by declaring signal handler
  - Be very careful when writing signal handlers

- Nonlocal jumps provide exceptional control flow within a process
  - Within the constraints of the *stack discipline*

The image part with relationship ID rId13 was not found in the file.