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

static FILE *openFile(int argc, char *argv[]);
static void writeToFile(FILE *f);

jmp_buf save;

/* Program should be invoked as:
 *     demo1 filename
 */
int main(int argc, char *argv[])
{
  int retcode;
  FILE *f=0;

  retcode = setjmp(save);
  if (retcode != 0) {
    retcode--;

    printf("Exiting with error code %d\n", retcode);
    if (f != 0) {
      fclose(f);
    }
    exit(retcode);
  }

  f = openFile(argc, argv);
  writeToFile(f);

  longjmp(save, 1);
}

static FILE *openFile(int argc, char *argv[])
{
  FILE *f;

  if (argc != 2) longjmp(save, 2);

  f = fopen(argv[1], "wt");
  if (f == 0) longjmp(save, 3);

  return f;
}

static void writeToFile(FILE *f)
{
  if (fprintf(f, "Hello!\n") < 0) longjmp(save, 4);
}
```

```
>./demo1 aok.txt
Exiting with error code 0

>cat aok.txt
Hello!

>./demo1 too many arguments
Exiting with error code 1

>./demo1 /boot/nowritepermission
Exiting with error code 2
```

```
1  /* setjmp.S from Newlib, simplified for clarity */
2
3  /* ----------------------------------------------------------------------
4                  int setjmp (jmp_buf);
5     ---------------------------------------------------------------------- */
6
7    .syntax unified
8    .text
9    .align 2
10   .thumb
11   .thumb_func
12   .globl setjmp
13   .type setjmp,function
14
15 setjmp:
16   /* Save all the callee-preserved registers into the jump buffer.  */
17   mov    r12, sp
18   stmea  r0!, { r4-r10, r11, r12, lr }
19
20   /* When setting up the jump buffer return 0.  */
21   mov    r0, #0
22   bx     lr
23
24 /* ----------------------------------------------------------------------
25     volatile void longjmp (jmp_buf, int);
26     ---------------------------------------------------------------------- */
27
28   .text
29   .align 2
30   .thumb
31   .thumb_func
32   .globl longjmp
33   .type longjmp,function
34
35 longjmp:
36   ldmfd  r0!, { r4-r10, r11, r12, lr }
37   mov    sp, r12
38
39   /* Put the return value into the integer result register.
40      But if it is zero then return 1 instead.  */
41   movs   r0, r1
42   it     eq
43   moveq  r0, #1
44   bx     lr
```

```
What We Have Seen So Far (Program Structures)
=============================

A. Big while loop with polling (EGR326)

   Advantages:
     * Simple
     * Perfectly fine for simple systems

   Disadvantages:
     * Latency can be large as system gets larger
     * Everything is intertwined --> hard to modify as system
       gets larger
     * Must break code into Funny Little Pieces to minimize latency and
       provide the appearance of concurrency

     Example 1:
         while (1) {
           if (isEvent1()) handleEvent1(); // perhaps UART?

           if (isEvent2()) handleEvent2(); // perhaps USB?

           if (isEvent3()) {
             // Long handlers can make latency between
             // checking for events unacceptable: events and
             // data are missed!
             handleEvent3();            // "long" handler
           }
         }

     Example 2:
         while (1) {
           if (isEvent3()) handleEvent1(); // perhaps UART?

           if (isEvent2()) handleEvent2(); // perhaps USB?

           if (gEvent3) {
             // Break up long handlers into FLP's, check for other
             // events in between
             handleEvent3_partI();               // "short" handler

             if (isEvent1()) handleEvent1();

             handleEvent3_theSequel();           // "short" handler

             if (isEvent2()) handleEvent2();

             handleEvent3_theNotSoGoodSequel();  // "short" handler

             if (isEvent1()) handleEvent1();

             handleEvent3_thePrequel();          // "short" handler

             if (isEvent2()) handleEvent2();

             handleEvent3_doCloning();           // "short" handler
           }
         }
```

B. Big while loop with interrupts and global variables (for events) or circular buffers (for data) used to communicate from ISR's to main code

Advantages:
* Reasonably simple, once initialization issues are worked out
* Much easier to keep latency small −−> buffer data at ISR level

Disadvantages:
* Debugging is more difficult
* Can have "concurrency issues" if both ISR and non−ISR code write to the same memory location (more on this later)
* Code is still a big while loop −−> instead of polling for hardware we are now polling for global variables and everything is still intertwined
* We still need to have FLP code if we want the appearance of concurrency

Example 3:

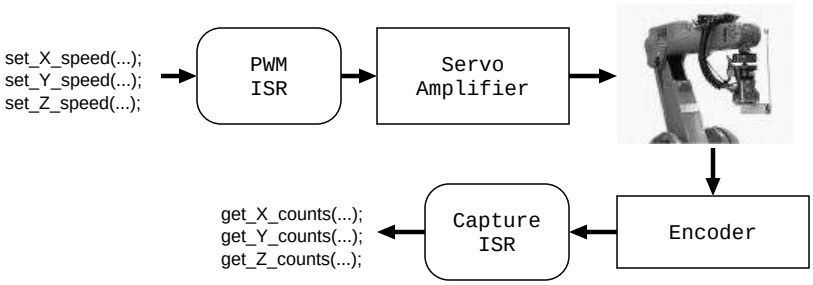
```
ISR(EVENT1) {
   gEvent1 = 1;
   bufferData(HW_REG_READ);
}

while (1) {
  if (gEvent1) handleEvent1(); // perhaps UART?

  if (gEvent2) handleEvent2(); // perhaps USB?

  if (gEvent3) {
    // Long handlers are OK because any data arriving
    // is buffered in ISR's. No data is lost (as long as
    // buffers are big enough) but response times may
    // be poor.
    handleEvent3();            // "long" handler
  }
}
```

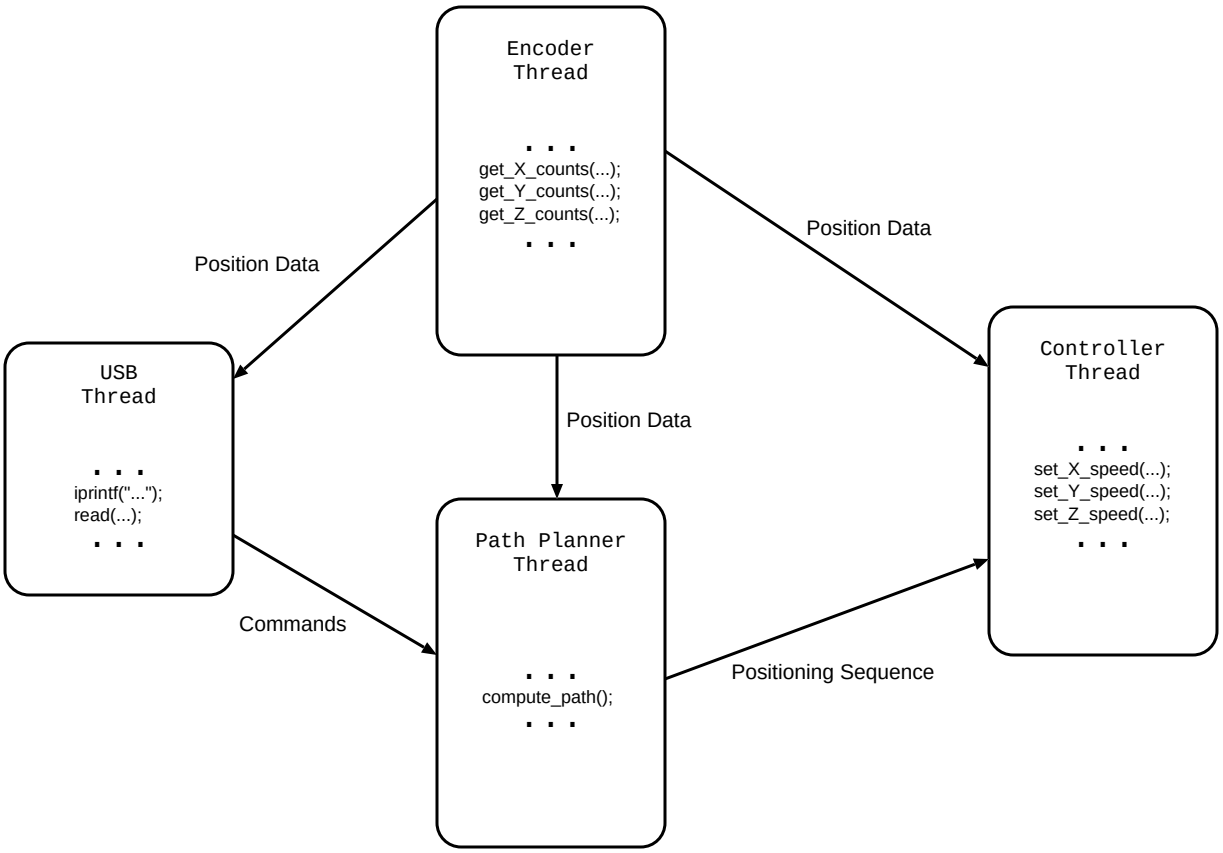
New Ways of Structuring Code

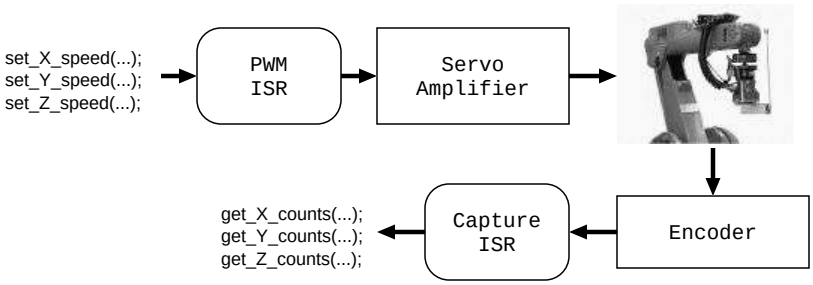
C. Coroutines with interrupts and global variables used to communicate from ISR's to each coroutine

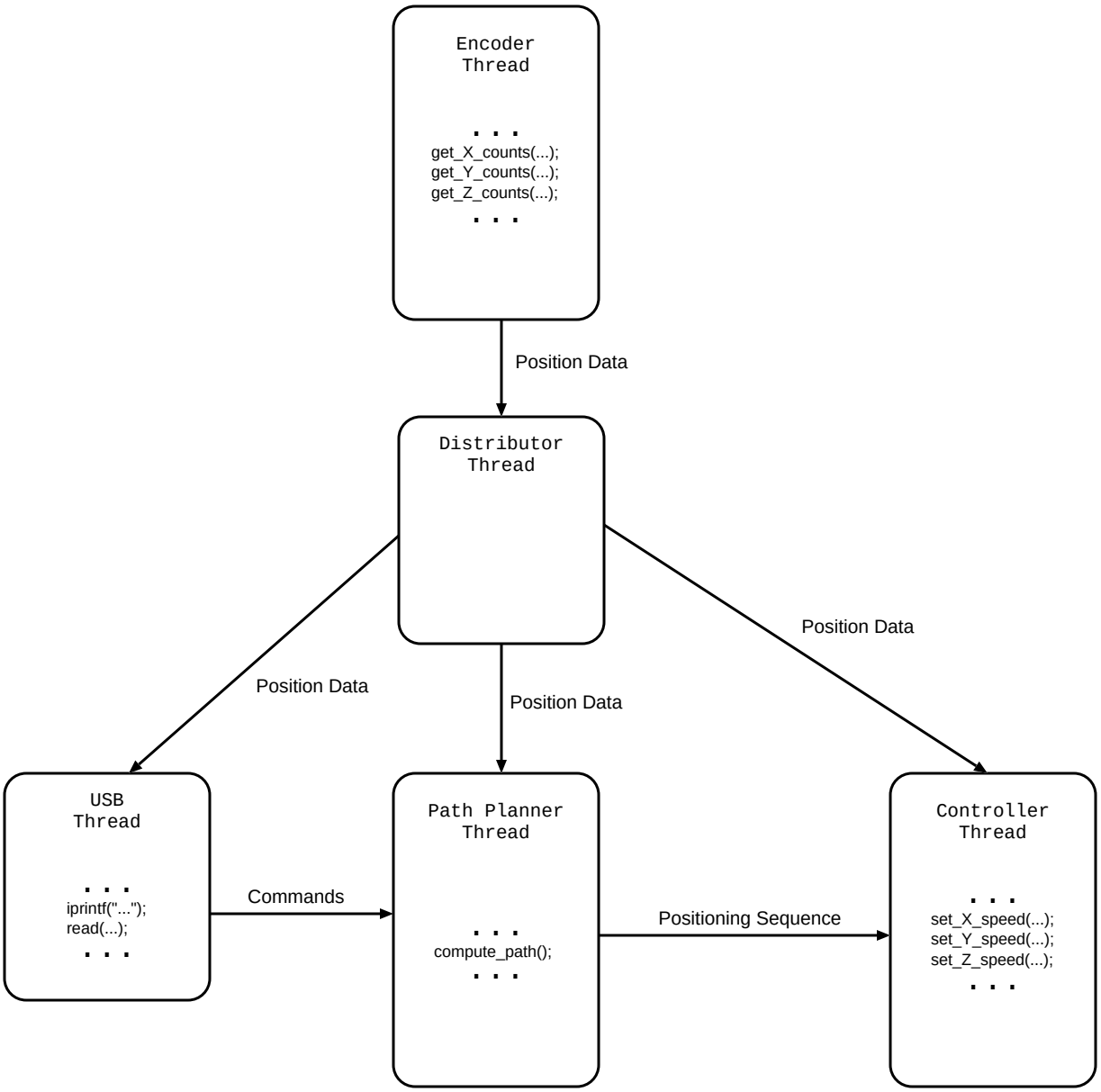
Advantages:
* Code is less intertwined; each coroutine has its own while loop that only needs to focus on one task
* No need for FLP code if we want the appearance of concurrency (but we do need to yield to other coroutines on a regular basis)
* Fairly simple to implement

Disadvantages:
* (Windows98) One coroutine, if it goes awry, can block other coroutines from running
* We need to yield to other coroutines on a regular basis, which is error−prone (depends on code author to remember to do this) and non−maintainable
* Each coroutine needs its own stack, which decreases available RAM


set_X_speed(...);
set_Y_speed(...);
set_Z_speed(...);
PWM ISR
Servo Amplifier
Encoder
Capture ISR
get_X_counts(...);
get_Y_counts(...);
get_Z_counts(...);


Encoder Thread
. . .
get_X_counts(...);
get_Y_counts(...);
get_Z_counts(...);
. . .
Position Data
Position Data
Position Data
USB Thread
. . .
iprintf("...");
read(...);
. . .
Controller Thread
. . .
set_X_speed(...);
set_Y_speed(...);
set_Z_speed(...);
. . .
Commands
Path Planner Thread
. . .
compute_path();
. . .
Positioning Sequence


set_X_speed(...);
set_Y_speed(...);
set_Z_speed(...);
PWM ISR
Servo Amplifier
Encoder
Capture ISR
get_X_counts(...);
get_Y_counts(...);
get_Z_counts(...);


Encoder Thread
. . .
get_X_counts(...);
get_Y_counts(...);
get_Z_counts(...);
. . .
Position Data
Distributor Thread
Position Data
Position Data
Position Data
USB Thread
. . .
iprintf("...");
read(...);
. . .
Commands
Path Planner Thread
. . .
compute_path();
. . .
Positioning Sequence
Controller Thread
. . .
set_X_speed(...);
set_Y_speed(...);
set_Z_speed(...);
. . .

```
1  /* Demonstrate the use of setjmp/longjmp to implement coroutines */
2
3  #include <stdio.h>
4  #include <setjmp.h>
5  #include <stdlib.h>
6
7  #define STACK_SIZE 4096
8
9  jmp_buf process1_buf;
10 jmp_buf process2_buf;
11
12 void process1(void)
13 {
14   volatile unsigned count = 0;
15
16   // Start of Process #1
17   (void) setjmp(process1_buf);
18
19   printf("In process 1 -- pass %d\n", ++count);
20
21   longjmp(process2_buf, 1);
22 }
23
24 void create_process1(void *stackptr)
25 {
26   // 64-bit stack pointer --> save in 64-bit integer
27   register unsigned long savedstack;
28
29   // Change SP prior to calling setjmp so that longjmp will
30   // start the process with 'stackptr'. More robust is to
31   // write a new version of setjmp that takes a SP as a
32   // parameter so no manual assembly code will be needed.
33   asm volatile (
34       // ia64 assembly language will only work on ia64 systems
35       "movq %%rsp,%[savedstack]\n" // savedstack <-- SP
36    "   movq %[stackptr], %%rsp"    // SP <-- stackptr
37             : [savedstack] "=r" (savedstack): [stackptr] "r" (stackptr)
38       );
39
40   if (setjmp(process1_buf) == 0) {
41     // Restore "normal" stack prior to return
42     // ia64 assembly language will only work on ia64 systems
43     asm volatile ("movq %[savedstack],%%rsp" : : [savedstack] "r" (savedstack));
44   } else {
45     // We got here through longjmp
46     process1();
47   }
48 }
49
50 void process2(void)
51 {
52   volatile unsigned count = 0;
53
54   // Start of Process #2
55   (void) setjmp(process2_buf);
56
57   printf("In process 2 -- pass %d\n", ++count);
58
59   if (count < 5) longjmp(process1_buf, 1);
60
61   exit(0);
62 }
63
```

```
64  void create_process2(void *stackptr)
65  {
66    register unsigned long savedstack;
67
68    asm volatile (
69        "movq %%rsp,%[savedstack];" // savedstack <-- SP
70     "    movq %[stackptr], %%rsp"    // SP <-- stackptr
71              : [savedstack] "=r" (savedstack): [stackptr] "r" (stackptr)
72        );
73
74    if (setjmp(process2_buf) == 0) {
75      asm volatile ("movq %[savedstack],%%rsp" : : [savedstack] "r" (savedstack));
76    } else {
77      process2();
78    }
79  }
80
81  int main(void)
82  {
83    // Yes, I really should be checking the return value of
84    // malloc to make sure it doesn't return 0.
85    create_process1((char *)malloc(STACK_SIZE) + STACK_SIZE);
86    create_process2((char *)malloc(STACK_SIZE) + STACK_SIZE);
87    longjmp(process1_buf, 1);
88
89    return 0;
90  }
91
92  /* Compile with:
93
94       gcc -g -O0 -o crdemo crdemo.c
95  */
```

```
In process 1 -- pass 1
In process 2 -- pass 1
In process 1 -- pass 2
In process 2 -- pass 2
In process 1 -- pass 3
In process 2 -- pass 3
In process 1 -- pass 4
In process 2 -- pass 4
In process 1 -- pass 5
In process 2 -- pass 5
```

```
1  #include <stdio.h>
2  #include <setjmp.h>
3  #include <stdlib.h>
4  #include "inc/hw_memmap.h"
5  #include "inc/hw_types.h"
6  #include "inc/hw_nvic.h"
7  #include "inc/lm3s6965.h"
8  #include "driverlib/debug.h"
9  #include "driverlib/gpio.h"
10 #include "driverlib/sysctl.h"
11 #include "driverlib/uart.h"
12 #include "rit128x96x4.h"
13
14 #define STACK_SIZE 2048
15
16 jmp_buf process1_buf;
17 jmp_buf process2_buf;
18
19 void process1(void)
20 {
21   volatile unsigned count = 0;
22
23   // Start of Process #1
24   (void) setjmp(process1_buf);
25
26   iprintf("In process 1 -- pass %d\r\n", ++count);
27
28   longjmp(process2_buf, 1);
29 }
30
31 void create_process1(void *stackptr)
32 {
33   register unsigned savedstack;
34   asm volatile (
35       "mov %[savedstack], sp;"
36       "mov sp,%[stackptr]"
37             : [savedstack] "=r" (savedstack): [stackptr] "r" (stackptr)
38       );
39
40   if (setjmp(process1_buf) == 0) {
41     asm volatile ("mov sp,%[savedstack]" : : [savedstack] "r" (savedstack));
42   } else {
43     process1();
44   }
45 }
46
47 void process2(void)
48 {
49   volatile unsigned count = 0;
50
51   // Start of Process #2
52   (void) setjmp(process2_buf);
53
54   iprintf("In process 2 -- pass %d\r\n", ++count);
55
56   if (count < 5) longjmp(process1_buf, 1);
57
58   exit(0);
59 }
60
61 void create_process2(void *stackptr)
62 {
63   register unsigned savedstack;
```

```
64    asm volatile (
65          "mov %[savedstack], sp;"
66          "mov sp,%[stackptr]"
67                : [savedstack] "=r" (savedstack): [stackptr] "r" (stackptr));
68
69    if (setjmp(process2_buf) == 0) {
70      asm volatile ("mov sp,%[savedstack]" : : [savedstack] "r" (savedstack));
71    } else {
72      process2();
73    }
74  }
75
76  void exit(int status)
77  {
78    (void)status;
79    while (1) {}
80  }
81
82  void main(void)
83  {
84    void *stack;
85
86    // Set the clocking to run directly from the crystal.
87    SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN |
88                   SYSCTL_XTAL_8MHZ);
89
90    // Initialize the OLED display and write status.
91    RIT128x96x4Init(1000000);
92    RIT128x96x4StringDraw("UART Echo",             36,  0, 15);
93    RIT128x96x4StringDraw("Port:  Uart 0",        12, 16, 15);
94    RIT128x96x4StringDraw("Baud:  115,200 bps",   12, 24, 15);
95    RIT128x96x4StringDraw("Data:  8 Bit",          12, 32, 15);
96    RIT128x96x4StringDraw("Parity: None",           12, 40, 15);
97    RIT128x96x4StringDraw("Stop:  1 Bit",          12, 48, 15);
98
99    // Enable the peripherals used by this example.
100   SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
101   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
102
103   // Set GPIO A0 and A1 as UART pins.
104   GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
105
106   // Configure the UART for 115,200, 8-N-1 operation.
107   UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
108                       (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
109                        UART_CONFIG_PAR_NONE));
110
111   // Create coroutines
112   stack = malloc(STACK_SIZE);
113   if (stack != 0) {
114     create_process1(stack + STACK_SIZE);
115   } else {
116     iprintf("Out of memory!\r\n");
117     exit(0);
118   }
119
120   stack = malloc(STACK_SIZE);
121   if (stack != 0) {
122     create_process2(stack + STACK_SIZE);
123   } else {
124     iprintf("Out of memory!\r\n");
125     exit(0);
126   }
```

```
127
128   // Start coroutines
129   longjmp(process1_buf, 1);
130 }
131
132 /* Compile with:
133  *
134  *      CC -o crdemotarget.elf -Os -Tlinkscript.x -Wl,--entry,ResetISR
135  *         -Wl,-Map,crdemotarget.map -I${STELLARISWARE}
136  *         -L${STELLARISWARE}/driverlib/gcc
137  *         crdemotarget.c startup_gcc.c syscalls.c rit128x96x4.c -ldriver
138  */
```