

ATTEMPT TO DESIGN A REAL TIME OPERATING SYSTEM

Github Link for the Project:

https://github.com/lazyswan/Kernel_Components

Swanand Sapre

scsapre@gmail.com

669-226-7772

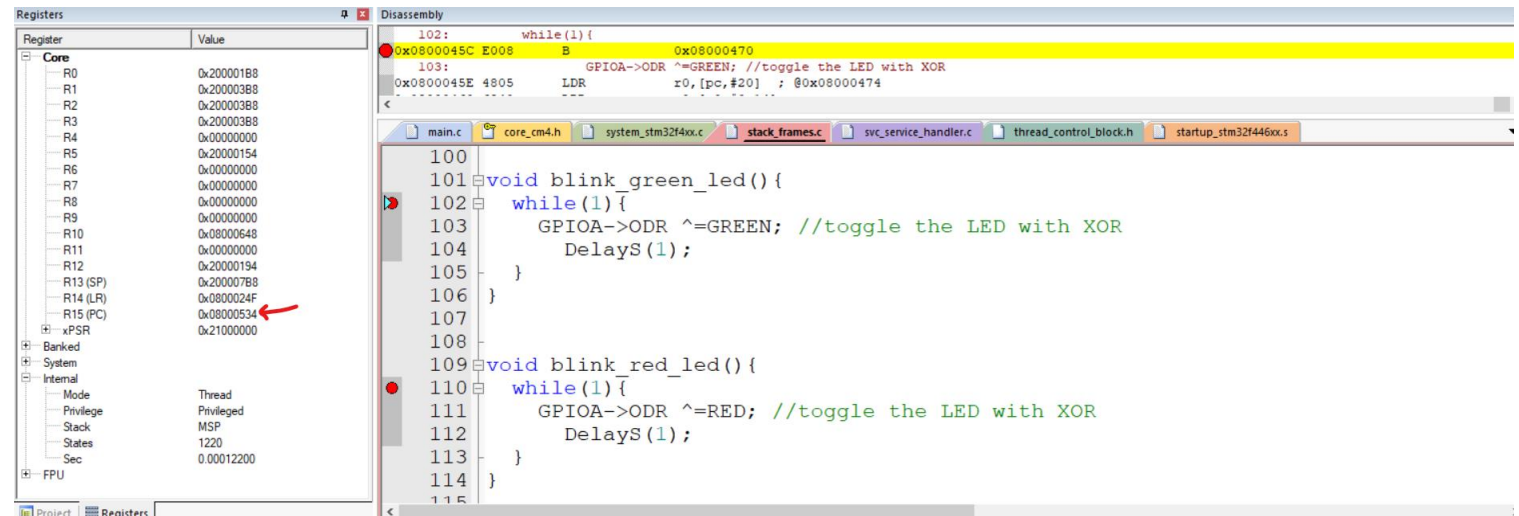
CRAPPY WAY OF TASK-SWITCH

Address of Thread
blink_green_led() : 0x0800045C

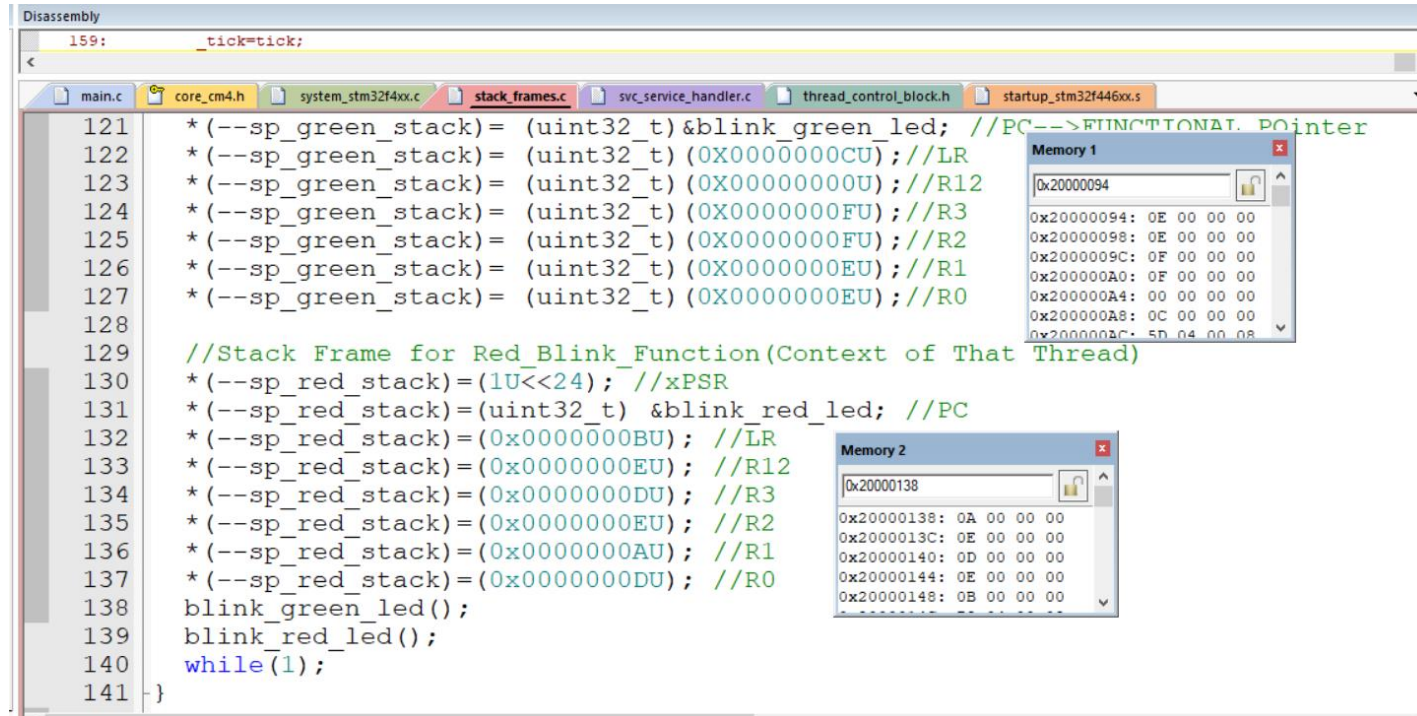
Address of Thread blink_red_led() :
0x08000478

Load PC with one of this address to
perform switch,

But its not context switch, since we
are not switching context(registers)
just the PC and program might
crash.



INITIALISING STACK FRAMES AND CHANGING SP IN ISR_HANDLER



The screenshot shows a disassembler window with a C code editor. The code is for a tick handler function. It initializes a green stack frame and then a red stack frame. Two memory windows are open: 'Memory 1' at address 0x20000094 and 'Memory 2' at address 0x20000138. The code is as follows:

```
159:  _tick=tick;
121  *(&sp_green_stack)= (uint32_t)&blink_green_led; //PC-->FUNCTIONAL Pointer
122  *(&sp_green_stack)= (uint32_t) (0X0000000CU); //LR
123  *(&sp_green_stack)= (uint32_t) (0X00000000U); //R12
124  *(&sp_green_stack)= (uint32_t) (0X0000000FU); //R3
125  *(&sp_green_stack)= (uint32_t) (0X0000000FU); //R2
126  *(&sp_green_stack)= (uint32_t) (0X0000000EU); //R1
127  *(&sp_green_stack)= (uint32_t) (0X0000000EU); //R0
128
129  //Stack Frame for Red Blink Function(Context of That Thread)
130  *(&sp_red_stack)= (1U<<24); //xPSR
131  *(&sp_red_stack)= (uint32_t) &blink_red_led; //PC
132  *(&sp_red_stack)= (0x0000000BU); //LR
133  *(&sp_red_stack)= (0x0000000EU); //R12
134  *(&sp_red_stack)= (0x0000000DU); //R3
135  *(&sp_red_stack)= (0x0000000EU); //R2
136  *(&sp_red_stack)= (0x0000000AU); //R1
137  *(&sp_red_stack)= (0x0000000DU); //R0
138  blink_green_led();
139  blink_red_led();
140  while(1);
141 }
```

Memory 1 window (0x20000094):

0x20000094:	0E 00 00 00
0x20000098:	0E 00 00 00
0x2000009C:	0F 00 00 00
0x200000A0:	0F 00 00 00
0x200000A4:	00 00 00 00
0x200000A8:	0C 00 00 00
0x200000AC:	5D 04 00 08

Memory 2 window (0x20000138):

0x20000138:	0A 00 00 00
0x2000013C:	0E 00 00 00
0x20000140:	0D 00 00 00
0x20000144:	0E 00 00 00
0x20000148:	0B 00 00 00

```
150 //Sys tick handler, ISR executed in everytick intr
151 void SysTick_Handler(){
152     ++tick;
153     //In this handler we change the value of stack pointer to load the
154     //different context, custom stack frames.
155
156 }
157
```

BUILDING KERNEL INTERNALS____ SIMPLE KERNEL WITH JUST 3 THREADS .

Supported Features:

- 3 Threads.
- Round Robin Scheduler.
- Task Yield Functionality to support Co-operative Scheduling
- Semaphore.
- Cooperative Semaphore.

THREAD CONTROL BLOCK

```
11
12 #define NUM_OF_THREADS 3
13 #define STACK_SIZE 100
14 #define BUS_FREQ 16000000
15 #define SYSPRI3 (*(volatile uint32_t *)0xE000ED20)
16 #define INTRCTRL (*(uint32_t volatile *)0xE000ED04)
17
18 void osSchedulerLaunch(void);
19 uint32_t MILLIS_PRESCALAR;
20
21 //Thread Control Block
22 typedef struct tcb{
23     int32_t *sp; //stack pointer
24     struct tcb *next;
25 }tcb_t;
26
27 //Similar to process descriptor table;
28 //This one is thread descriptor table/Array of TCB.
29 tcb_t tcbs[NUM_OF_THREADS];
30
31 //Holds Pointer to current running Process
32 tcb_t *current_tcb;
33
34 //stack area for all the threads in form of 2-d matrix
35 int32_t TCB_STACK[NUM_OF_THREADS][STACK_SIZE];
36
37 /*
```

KERNEL STACK INIT

```
36
37  /*
38   Initialises the stack,
39   PC and xPSR for each Thread.
40   thread_num: 0,1,2
41  */
42  void osKernelStackInit(int thread_num) {
43
44      //Stack pointer initialised to bottom of its stack.
45      tcbs[thread_num].sp = &TCB_STACK[thread_num][STACK_SIZE-16];
46
47      //Runing in the thumb mode | xpsr rx Bit 24.
48      TCB_STACK[thread_num][STACK_SIZE-1]=1<<24;
49
50  }
51  /*
```

CREATING THREADS

```
50 -;
51 /*
52 This function decides which thread to run,
53 this function must be atomic, its critical section
54 */
55 uint8_t osKernelAddThreads(void (*task0)(), void (*task1)(), void (*task2)()) {
56
57     //connected all the tcbs in circular LL fashion
58     __disable_irq();
59
60     tcbs[0].next=&tcbs[1];
61     tcbs[1].next=&tcbs[2];
62     tcbs[2].next=&tcbs[0];
63
64     //initialise PC with task0, callback function
65     osKernelStackInit(0);
66     TCB_STACK[0][STACK_SIZE-2]= (int32_t)(task0);
67
68     //initialise PC with task1, callback function
69     osKernelStackInit(1);
70     TCB_STACK[1][STACK_SIZE-2]= (int32_t)(task1);
71
72     //initialise PC with task2, callback function
73     osKernelStackInit(2);
74     TCB_STACK[2][STACK_SIZE-2]= (int32_t)(task2);
75
76     //First Task to be scheduled is task0
77     current_tcb= &tcbs[0];
78     __enable_irq();
79
80
81     return 1;
82 }
```

KERNEL LAUNCH

```
82 - },
83 void osKernelInit() {
84     __disable_irq();
85     MILLIS_PRESCALAR=BUS_FREQ/1000;
86 }
87
88
89 void osKernelLaunch(uint32_t quanta) {
90     //initialising SysTick
91     SysTick->CTRL=0;
92     SysTick->VAL=0;
93     SYSPRI3 = (SYSPRI3&0x00FFFFFF) | 0xE0000000; //priority to 7
94
95     //SysTick Interrupts after @ LOAD value | quanta
96     SysTick->LOAD=(quanta*MILLIS_PRESCALAR)-1;
97
98     SysTick->CTRL = 0x00000007;
99
100     //This is assembly Function
101     osSchedulerLaunch();
102 }
103
```


TASK YEILD

```
109 void osThreadYeild(){
110
111     //reset the counter value to 0;
112     SysTick->VAL=0;
113     //this will generate a
114     //sysTick Intr and then sysTick Handler will perform a context SW.
115     INTRCTRL= 1<<26;
116
117 }
```

ASSEMBLY CODE

```
1      AREA |.text|, CODE, READONLY, ALIGN=2
2      THUMB
3      EXTERN  current_tcb
4      EXPORT  SysTick_Handler
5      EXPORT  osSchedulerLaunch
6
7
8      SysTick_Handler                ;save r0,r1,r2,r3,r12,lr,pc,psr
9          CPSID    I                ;Disables the INTR
10         PUSH     {R4-R11}          ;save r4,r5,r6,r7,r8,r9,r10,r11
11         LDR      R0, =current_tcb  ; R0=current_tcb(which is a pointer)
12         LDR      R1, [R0]          ; r1= currentPt
13         STR      SP, [R1]
14         LDR      R1, [R1,#4]        ; r1 =currentPt->next
15         STR      R1, [R0]          ;currentPt =r1
16         LDR      SP, [R1]          ;SP= currentPt->stackPt
17         POP      {R4-R11}
18         CPSIE    I
19         BX       LR
20
21
22
23      osSchedulerLaunch
24         LDR      R0, =current_tcb
25         LDR      R2, [R0]          ; R2 =currentPt
26         LDR      SP, [R2]          ;SP = currentPt->stackPt
27         POP      {R4-R11}
28         POP      {R0-R3}
29         POP      {R12}
30         ADD      SP, SP, #4
31         POP      {LR}
32         ADD      SP, SP, #4
```

SEMAPHORE

```
4  scsapre@gmail.com
5  */
6
7  #include<stdint.h>
8
9  void osSemaphoreInit(int32_t *semaphore, int32_t value){
10     *semaphore=value;
11 }
12
13 void osSignalSet(int32_t *semaphore){
14     __disable_irq();
15     *semaphore+=1;
16     __enable_irq();
17 }
18
19 void osSignalWait(int32_t *semaphore){
20     __disable_irq();
21     while(*semaphore<=0){
22         __disable_irq();
23         __enable_irq();
24     }
25     *semaphore-=1;
26     __enable_irq();
27 }
```

COOPERATIVE SEM

```
23 |
24 | //instead of busywaiting for entire Quanta, task yeilds to another task
25 | void osSignalWait(int32_t *semaphore){
26 |     __disable_irq();
27 |     while(*semaphore<=0){
28 |         __disable_irq();
29 |         osThreadYeild();
30 |         __enable_irq();
31 |
32 |     }
33 |     *semaphore-=1;
34 |     __enable_irq();
35 | }
```

MAIN FUNCTION

```
1  #include "osKernel.h"
2  #define QUANTA 10
3  uint32_t count0, count1, count2;
4  int32_t semaphore1, semaphore2;
5  uint32_t volatile shared_resource=0;
6
7  void Task0(void) {
8      while(1) {
9          count0++;
10         osThreadYield();
11     }
12 }
13 void Task1(void) {
14     while(1) {
15         osSignalWait(&semaphore2);
16         shared_resource++;
17         osSignalSet(&semaphore1);
18     }
19 }
20 void Task2(void) {
21     while(1) {
22         osSignalWait(&semaphore1);
23         shared_resource--;
24         osSignalSet(&semaphore2);
25     }
26 }
27 int main(void) {
28     osKernelInit();
29     osKernelAddThreads(&Task0, &Task1, &Task2);
30     osKernelLaunch(QUANTA);
31     //both semaphores are set.
32     osSemaphoreInit(&semaphore1, 0);
33     osSemaphoreInit(&semaphore2, 1);
34 }
```

OUTPUT

The screenshot displays a debugger interface with three main panels:

- Registers Panel:** Shows the state of various registers. The 'Core' section lists R0 through R15 and xPSR. R0-R9 are zeroed out. R10 is 0x00000000. R11 is 0x00000000. R12 is 0x00000000. R13 (SP) is 0x20000358. R14 (LR) is 0x080003E7. R15 (PC) is 0x0800054A. xPSR is 0x01000000. The 'Internal' section shows Thread (15.05830130), Privileged (MSP), Stack (150583013), States (150583013), and Sec (15.05830130). The 'FPU' section is empty.
- Source Code Panel:** Displays the source code of the program. The code includes headers, defines constants, and implements three tasks (Task0, Task1, Task2) and a main function. The main function initializes two semaphores and calls the tasks.
- Watch Window:** Shows the values of variables being watched. The variables are count0, count1, count2, and shared_resource, all of type unsigned int. Their values are 5119835, 588081, 584598, and 3483 respectively.

Registers:

Register	Value
R0	0x20000018
R1	0x2000001C
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20000358
R14 (LR)	0x080003E7
R15 (PC)	0x0800054A
xPSR	0x01000000

Source Code:

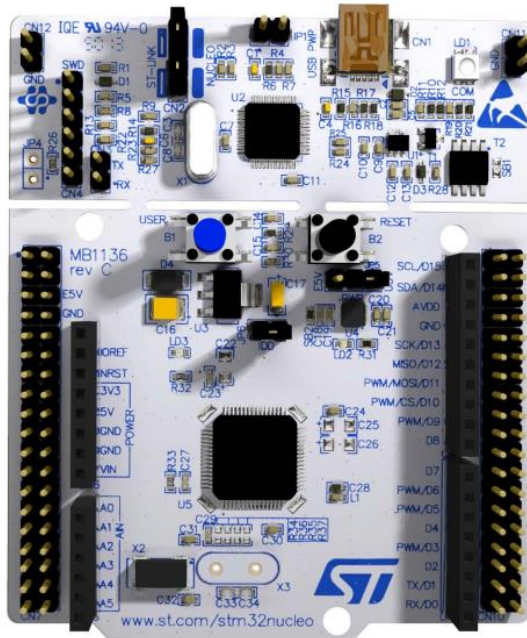
```
1 #include "osKernel.h"
2 #define QUANTA 100
3 uint32_t count0, count1, count2;
4 int32_t semaphore1, semaphore2;
5 uint32_t volatile shared_resource=0;
6
7 void Task0(void){
8     while(1) {
9         count0++;
10        //osThreadYield();
11    }
12}
13 void Task1(void){
14     while(1){
15         count1++;
16         osSignalWait(&semaphore2);
17         shared_resource++;
18         osSignalSet(&semaphore2);
19     }
20}
21
22 void Task2(void){
23     while(1){
24         count2++;
25         osSignalWait(&semaphore2);
26         shared_resource--;
27         osSignalSet(&semaphore2);
28     }
29}
30 int main(void){
31     //both semaphores are set.
32     osSemaphoreInit(&semaphore1,1);
33     osSemaphoreInit(&semaphore2,1);
34     osKernelInit();
```

Watch 1:

Name	Value	Type
count0	5119835	unsigned int
count1	588081	unsigned int
count2	584598	unsigned int
shared_resource	3483	unsigned int
< Enter expression >		

REFERENCES

Figure 1. STM32 Nucleo-64 board (1)



1. Picture not contractual.

