**osKernel.h**

Through Main we will l only be able to access these functions

#if\_not\_defined

#define

void isKernelLaunch(uint32\_t quanta);

void osKernelInit(void);

uint8\_t osKernelAddThreads(void(\*task0)(void),

void(\*task1)(void),

void(\*task2)(void));

void addPeriodicTaskInfo(uint32\_t CPU,uint32\_t DEADLINE,uint32\_t PERIOD,uint32\_t RELEASE,uint32\_t REMAIN,uint32\_t task);

#end

The reason being to separate Stackframe initialization separate from main program so that useless stack manipulation should not be done

**Main.c**

Here we include osKernel.h

Define number and SVC call name method

int \_\_svc(0x00) svc\_service\_init();

int \_\_svc(0x01) svc\_service\_kernelLaunch();

Next we give prototype for SVC handler

void SVC\_Handler\_C(unsigned int \* svc\_args);

Checking which stack is being used

\_\_asm void SVC\_Handler(void)

{

TST lr, #4 //checks which stack is in use

ITE EQ

MRSEQ r0, MSP

MRSNE r0, PSP

IMPORT SVC\_Handler\_C

B SVC\_Handler\_C

}

Bit Description value

|  |  |  |
| --- | --- | --- |
| 2 | Return stack | 1 (Return with Process Stack) or 0 (Return with Main Stack) |

void SVC\_Handler\_C(unsigned int \* svc\_args){

unsigned int svc\_number;

svc\_number = ((char\*)svc\_args[6])[-2];// just for extracting SVC number

switch(svc\_number){

case 0:

osKernelInit();

addPeriodicTaskInfo(2,4,6,0,0,0);

addPeriodicTaskInfo(4,6,8,0,0,1);

addPeriodicTaskInfo(6,8,10,0,0,2);

osKernelAddThreads(&task0,&task1,&task2);

break;

case 1:

isKernelLaunch(quanta);

break;

default:

break;

}

}

The code (svc\_number = ((char \*)svc\_args[6])[-2])

svc\_args[6] is the stacked PC. It is the address of the instruction after the SVC.

The SVC instruction is located in the memory address [stacked\_PC - 2], because SVC is a 2 byte instruction.

The SVC number is the lower byte of the instruction.

So if without the "[-2]", (char \*)svc\_args[6] read the lower byte of the instruction after the SVC.

Adding the "[-2]", it handles that as an array of bytes and therefore reads the SVC number.

osKernel.c

#define BUS\_FREQUENCY 16000000//we are going to divide 16000 by 1000 and store result in millis prescalar

uint32\_t MILLIS;//to get unit of time multiply by( millis and quanta)

**Method:**

void osKernelInit(void){

\_\_disable\_irq();

MILLIS = BUS\_FREQUENCY/1000;

NVIC\_SetPriority(PendSV\_IRQn, 0xff); /\* Lowest possible priority \*/

NVIC\_SetPriority(SysTick\_IRQn, 0x00); /\* Highest possible priority \*/

}

**DATA STRUCTURE:**

Global Data structure for accepting parameters

This will be used to assign values to Task Frames

struct PERIOD{

int32\_t CPU;

int32\_t DEADLINE;

int32\_t PERIOD;

int32\_t RELEASE;

int32\_t REMAIN;

int32\_t task;

};

typedef struct PERIOD periodic;

periodic periodic\_tasks[NUM\_OF\_THREADS];

Method to initialize values(parameters) of task

void addPeriodicTaskInfo(uint32\_t CPU,uint32\_t DEADLINE,uint32\_t PERIOD,uint32\_t RELEASE,uint32\_t REMAIN,uint32\_t task){

periodic\_tasks[task].CPU=CPU;

periodic\_tasks[task].DEADLINE=DEADLINE;

periodic\_tasks[task].PERIOD=PERIOD;

periodic\_tasks[task].RELEASE=RELEASE;

periodic\_tasks[task].REMAIN=REMAIN;

periodic\_tasks[task].task=task;

}

The following function implemented helps to assign parameters to tasks that later will be used to assign to task stacks,the reason they are separated as to prevent access to task Frames

Structure of TCB

struct tcb{

int32\_t \*stack\_pointer; //32 address pointer to top of stack

struct tcb \*next\_pointer;

};

typedef struct tcb tcb\_type;

tcb\_type tcbs[NUM\_OF\_THREADS];//array for 3 task

tcb\_type \*current\_ptr\_task;// accessing current Task Stack frame

Data structure:

int32\_t TCB\_STACK\_GLOBAL[NUM\_OF\_THREADS][STACKSIZE];

2D array of 32 bit int whose first index is task id and second index give respective starting address of task frame

Method to initialise Task stack frames:

void osKernelStackInit(int i){

**From above initialised tcb-->tcbs[i].stack\_pointer = &TCB\_STACK\_GLOBAL[i][STACKSIZE -16]; //Stack pointer points to top of task**

TCB\_STACK\_GLOBAL[i][STACKSIZE -1] = 0x01000000;//run in thumb mode xPSR

TCB\_STACK\_GLOBAL[i][STACKSIZE-3] = 0x14141414; //R14 |

TCB\_STACK\_GLOBAL[i][STACKSIZE-4] = 0x12121212; //R12 |

TCB\_STACK\_GLOBAL[i][STACKSIZE-5] = 0x03030303; //R3 |dummy

TCB\_STACK\_GLOBAL[i][STACKSIZE-6] = 0x02020202; //R2 |values

TCB\_STACK\_GLOBAL[i][STACKSIZE-7] = 0x01010101; //R1 |given

TCB\_STACK\_GLOBAL[i][STACKSIZE-8] = 0x00000000; //R0 |

/\*We have to save manually\*/

TCB\_STACK\_GLOBAL[i][STACKSIZE-9] = 0x11111111; //R11

TCB\_STACK\_GLOBAL[i][STACKSIZE-10] = 0x10101010; //R10

TCB\_STACK\_GLOBAL[i][STACKSIZE-11] = periodic\_tasks[i].CPU; //R9 CPU

TCB\_STACK\_GLOBAL[i][STACKSIZE-12] = periodic\_tasks[i].REMAIN; //R8 remain

TCB\_STACK\_GLOBAL[i][STACKSIZE-13] = periodic\_tasks[i].RELEASE; //R7 release

TCB\_STACK\_GLOBAL[i][STACKSIZE-14] = periodic\_tasks[i].PERIOD; //R6 PERIOD

TCB\_STACK\_GLOBAL[i][STACKSIZE-15] = periodic\_tasks[i].DEADLINE; //R5 DEADLINE

TCB\_STACK\_GLOBAL[i][STACKSIZE-16] = 0x04040404; //R4

}

The global DS we initialised for accepting parameters for task now helps initialize their

Respective Task Frame

**Method:**

uint8\_t osKernelAddThreads(void(\*task0)(void),

void(\*task1)(void),

void(\*task2)(void))

Task task as input parameters

Call the Task initialise method

-->osKernelStackInit(0);

**Makes PC points to Task0 adrdress**

TCB\_STACK\_GLOBAL[0][STACKSIZE -2] =(int32\_t)(task0);//PC for task0

current\_ptr\_task = &tcbs[0];//makes current pointer points to tcb of task 0

**Method:**

void isKernelLaunch(uint32\_t quanta)

SysTick->LOAD =(quanta \* MILLIS)-1;//ALL COUNTING BEGINS FROM 0 SO -1

SysCount = SysCount+((quanta \* MILLIS)-1);//used later

SysTick->CTRL = 0x00000007;//trigger interrupt

Pend\_trigger();

void Pend\_trigger(void){

SCB->ICSR |= SCB\_ICSR\_PENDSVSET\_Msk;//Trigger PendSV

}

**ASSEMBLY CODE:**

AREA |.text|,CODE,READONLY,ALIGN=2

THUMB

PRESERVE8

EXTERN current\_ptr\_task //external current\_pointer\_task defined above

EXPORT SysTick\_Handler

EXPORT PendSV\_Handler

IMPORT SCHEDULAR ;SCHEDULAR is a C module which is defined below,it basically sets current pointer to task with least laxivity,the logic is given below for least laxivity selection

SysTick\_Handler ;save r0,r1,r2,r3,r12,lr,pc,psr

CPSID I;disable any interrupts

PUSH {R4-R11}

LDR R0,=current\_ptr\_task

LDR R1,[R0]

STR SP,[R1] ;stack pointer to top of task

PUSH {R0,LR}

BL SCHEDULAR;Defined below

POP {R0,LR}

LDR R1,[R0] ;R1=CURRENT POINTER

LDR SP,[R1] ;SP = CURRENTPT ->STACKPT

POP {R4-R11}

CPSIE I

BX LR ;RESTORED AUTOMATICALLY BY PROCESSOR

**Method Schedular:**

void SCHEDULAR(void){

uint32\_t laxitvity=0;

uint32\_t min\_lax=0;

uint32\_t recent;//Which Tsk to run now

for(int i=0;i<3;i++){

//[STACKSIZE-11:cpu][ STACKSIZE-12:remain][ STACKSIZE-13:release] [STACKSIZE-14: period][ STACKSIZE-15:Deadline]

if((SysCount%TCB\_STACK\_GLOBAL[i][STACKSIZE-14])-TCB\_STACK\_GLOBAL[i][STACKSIZE-13] ==0){

TCB\_STACK\_GLOBAL[i][STACKSIZE-15] = SysCount + TCB\_STACK\_GLOBAL[i][STACKSIZE-14];

TCB\_STACK\_GLOBAL[i][STACKSIZE-12] = TCB\_STACK\_GLOBAL[i][STACKSIZE-11];

}

if(TCB\_STACK\_GLOBAL[i][STACKSIZE-12] >0){

laxitvity = TCB\_STACK\_GLOBAL[i][STACKSIZE-15]-(((SysCount%TCB\_STACK\_GLOBAL[i][STACKSIZE-14])-TCB\_STACK\_GLOBAL[i][STACKSIZE-13])+TCB\_STACK\_GLOBAL[i][STACKSIZE-12]);

if(min\_lax > laxitvity){

recent = i;

min\_lax =laxitvity;

}

}

}

if(TCB\_STACK\_GLOBAL[recent][STACKSIZE-12] !=0){

TCB\_STACK\_GLOBAL[recent][STACKSIZE-12]--;

}

if(current\_ptr\_task != &tcbs[recent]){ //based on LLF we chose recent

current\_ptr\_task = &tcbs[recent];

//current\_ptr\_task = (tcbs +recent\*sizeof(tcbs[recent]));

}

else{

tcbs[recent].next\_pointer = current\_ptr\_task;

}

}

**ASSEMBLY CODE FOR PENDSV HANDLER(loading new Context):**

PendSV\_Handler

CPSID I;disable any interrupts

PUSH {R4-R11}

LDR R0,=current\_ptr\_task

LDR R2,[R0] ;R2 = R2=CURRENT PTR

LDR SP,[R2] ;R2 =CURRENT PTR->STACK TOP

POP {R4-R11}

POP {R0-R3}

POP {R12}

ADD SP,SP,#4

POP {LR}

ADD SP,SP,#4

CPSIE I

BX LR

ALIGN

END