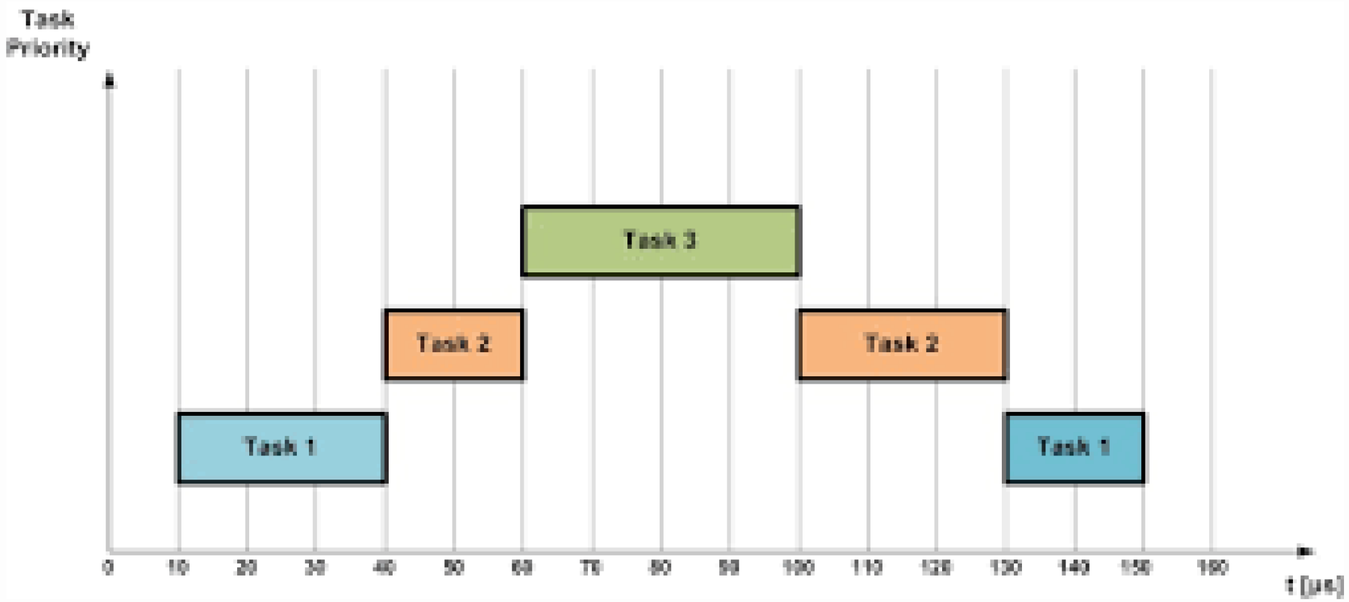
VIRTUAL TASK SCHEDULER



AKULA SATYA SAI SIVA RAMA KRISHNA
SAI KIRAN REDDY KASALA
AMARNADH PEAPALLA

TABLE OF CONTENTS

01 Project Objective

02 What Why How ?

03 System Specifications

04 Explain Code Base and files

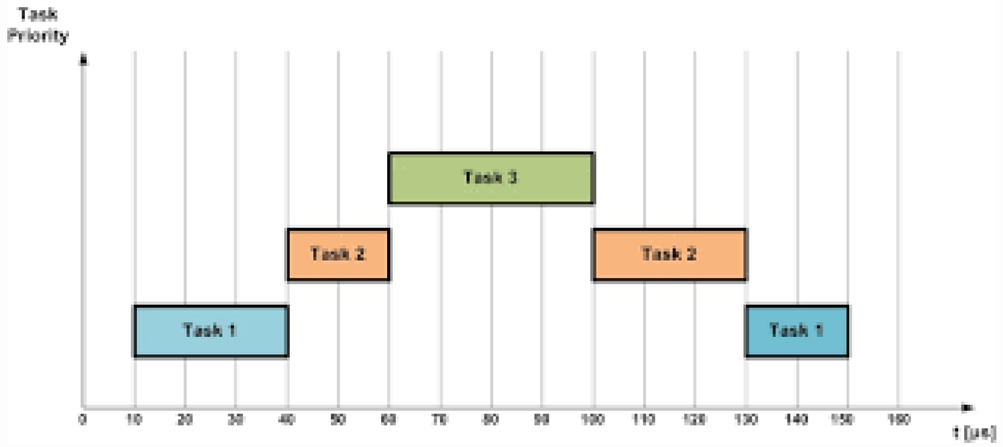
05 Explain Compiler and Linker scripts

06 How this scheduler works

07 Room for improvement

08 Queries

PROJECT OBJECTIVE



Creation of a Virtual Task Scheduler

Scheduler : The Task Scheduler enables you to automatically perform routine tasks on a chosen computer.

Virtual Scheduler : A kernel or scheduler that is abstracted from underlying Hardware (*as much as possible)

WHAT
WHY
HOW
?

What have we created ?
A basic **priority based non-preemptive** kernel or task scheduler

Why did we do it ?

To understand what it takes to write a kernel from scratch

How did we do it ?

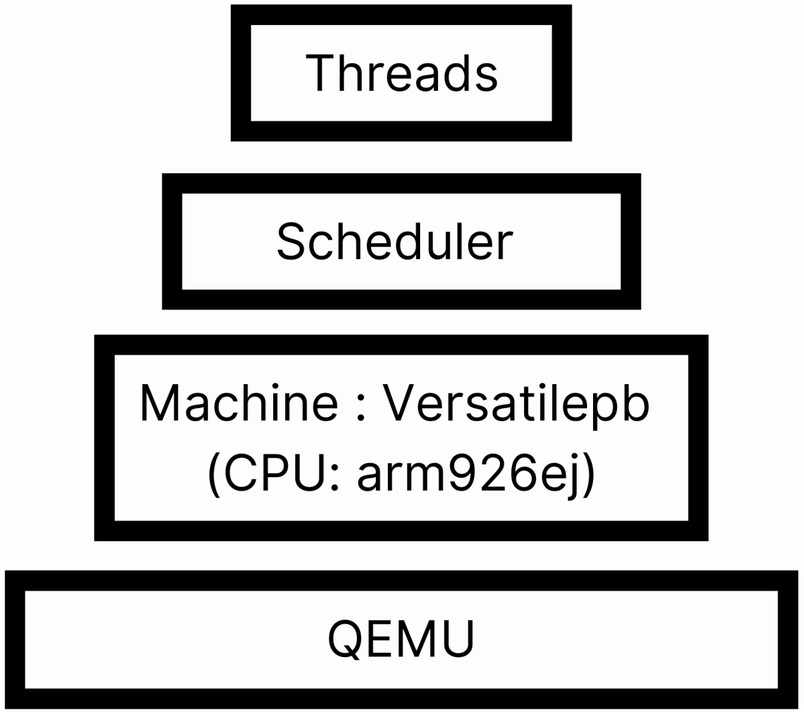
To Run this bare-metal Scheduler Code we choose QEMU as the Target Emulator.

Priority Based

Priority scheduling is a method of scheduling processes based on priority. In this method, the scheduler chooses the tasks to work as per the priority

Non-Preemptive

In non-preemptive scheduling, any new process has to wait until the running process finishes its CPU cycle



SYSTEM SPECIFICATIONS

CPU Arch :	Arm arm926ej
CPU ISA :	ARMv5
Language(s):	C and ARM Assembly
Compiler:	arm-none-eabi-gcc toolchain
Target:	QEMU, qemu-system-arm, versatilepb
Scheduler:	Non-Preemptive Priority Based No Timers Pool Based Delay No System Calls Seperate user and Kernel Spaces

Code base and Files:

Total 13 Files:

4 Header files [**types.h, utils.h, thread.h, memory.h**]

4 Source files. [**entry.c, utils.c, thread.c, memory.c**]

1 Assembly file [**startup.s**]

1 linker script [**boot.ld**]

1 build script [**build_commands.sh**]

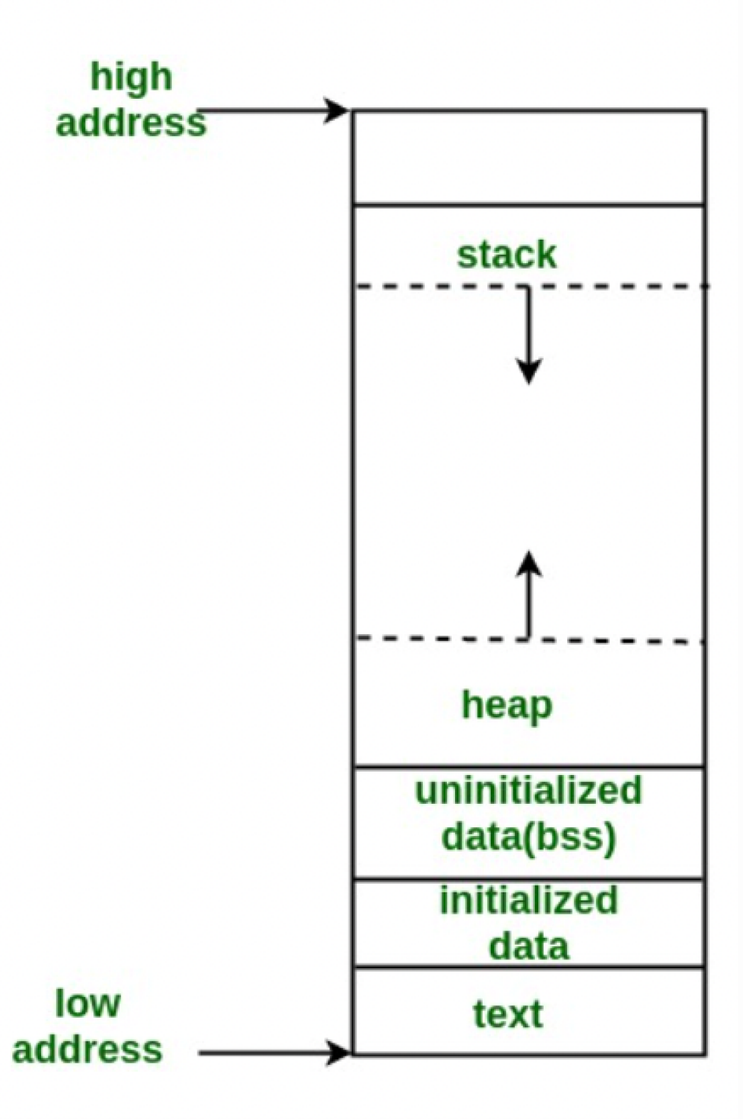
1 ELF file [**np_scheduler.elf**]

1 BIN file [**np_scheduler.bin**]

Build Command

```
$ build_commands.sh
1   arm-none-eabi-gcc -g -c -O0 -mcpu=arm926ej-s *.c
2   arm-none-eabi-as -g -mcpu=arm926ej-s startup.s -o startup.o
3   arm-none-eabi-ld -T boot.ld *.o -o np_scheduler.elf
4   arm-none-eabi-objcopy -O binary np_scheduler.elf np_scheduler.bin
5
6   # Run np_scheduler.bin using QEMU
7   # qemu-system-arm -M versatilepb -nographic -kernel np_scheduler.bin
```

Memory Layout:




```
sai@Akulas-MacBook-Air np_scheduler % arm-none-eabi-objdump -d startup.o

startup.o:     file format elf32-littlearm


Disassembly of section .text:

00000000 <_MyApp>:
   0:   e59fd038        ldr     sp, [pc, #56]   @ 40 <_delay_loop+0x20>
   4:   e59f0038        ldr     r0, [pc, #56]   @ 44 <_delay_loop+0x24>
   8:   ebfffffe        bl      0 <entry>
   c:   eafffffe        b       c <_MyApp+0xc>

00000010 <panic>:
  10:   ef000000        svc     0x00000000
  14:   eafffffe        b       14 <panic+0x4>

00000018 <delay>:
  18:   e1a01000        mov     r1, r0

0000001c <_ploop>:
  1c:   e3e004ff        mvn     r0, #-16777216  @ 0xff000000

00000020 <_delay_loop>:
  20:   e1a00000        nop                     @ (mov r0, r0)
  24:   e2500001        subs    r0, r0, #1
  28:   e3500000        cmp     r0, #0
  2c:   1afffffb        bne     20 <_delay_loop>
  30:   e2511001        subs    r1, r1, #1
  34:   e3510000        cmp     r1, #0
  38:   1afffff7        bne     1c <_ploop>
  3c:   e12fff1e        bx      lr
```

Linker Script

boot.ld

```
1   ENTRY(_MyApp)
2   SECTIONS
3   {
4    . = 0x10000;  /* Specified in QEMU versatilepb */
5    .startup . : { startup.o(.text) }
6    .text : { *(.text) }
7    .data : { *(.data) }
8    .bss : { *(.bss COMMON) }
9    . = ALIGN(8);
10   heap_start = .;
11   . = . + 0x10000; /* 64KB of heap memory */
12   . = ALIGN(8);
13   . = . + 0x1000; /* 4KB of stack memory */
14   stack_top = .;
15  }
```

Scheduling Tasks

Analyzing Scheduler Logs

```
Hello, world!
New: -> ID: 1  NAME: THREAD_1  PRIORITY : 8
New: -> ID: 2  NAME: THREAD_2  PRIORITY : 5
New: -> ID: 3  NAME: THREAD_3  PRIORITY : 6
New: -> ID: 4  NAME: THREAD_4  PRIORITY : 8
----------------- START OF SCHEDULER -------------
Running: THREAD_1 Priority: 8
Running: THREAD_4 Priority: 8
 BLOCK [1]  Block ID [1]
 Heap Start [71560] Heap ptr Allocated [71560]
New: -> ID: 5  NAME: THREAD_5  PRIORITY : 12
Running: THREAD_5 Priority: 12
Running: THREAD_3 Priority: 6
Running: THREAD_2 Priority: 5
----------------- END OF SCHEDULER -------------
```

Startup & Entry Code :

startup.s

```asm
.global _MyApp
.global delay
.global panic
.global heap_start

_MyApp:
    LDR sp, =stack_top
    LDR R0, =heap_start
    BL entry
    B .

panic:
    SVC 0
    BAL .

delay:
    MOV R1, R0
_ploop:
    MOV R0, #0xFFFFFF
_delay_loop:
```

```c
int entry(uint8_t * heap){
        inti_memory(heap);
        print("Hello, world! \n");
        delay( TICKS_S(1) );

        thread_handler_t thread1;
        memset(&thread1, 0, sizeof(thread_handler_t));
        thread_init(&thread1, 1, "THREAD_1", 8, thread_fun);
        thread_create(&thread1);

        thread_handler_t thread2;
        memset(&thread2, 0, sizeof(thread_handler_t));
        thread_init(&thread2, 2, "THREAD_2", 5, thread_fun);
        thread_create(&thread2);


        thread_handler_t thread3;
        memset(&thread3, 0, sizeof(thread_handler_t));
        thread_init(&thread3, 3, "THREAD_3", 6, thread_fun);
        thread_create(&thread3);


        thread_handler_t thread4;
        memset(&thread4, 0, sizeof(thread_handler_t));
        thread_init(&thread4, 4, "THREAD_4", 8, thread_fun2);
        thread_create(&thread4);

        // print_tcb_list();
        // print_heap_block_list();
        thread_np_scheduler();

    return 0;
}
```

Dynamic Memory Management:

memory.c > malloc(size_t)

```c
#include "memory.h"

/* Heap Size is Coupled with Linker Script */
#define HEAP_SIZE   0x10000
#define BLOCK_SIZE  64
#define BLOCK_COUNT (HEAP_SIZE/BLOCK_SIZE)


typedef struct heap_memory_handler{
    uint8_t *heap_start;
    uint8_t block[BLOCK_COUNT];
    uint32_t total_blocks;
    uint32_t available_blocks;
    uint32_t total_size;
    uint32_t available_size;
} heap_memory_handler_t;

static heap_memory_handler_t _heap_memory_handler = { NULL, };


void inti_memory(uint8_t *heap){
    memset((void*)&_heap_memory_handler, 0, sizeof(heap_memory_handler_t));
    _heap_memory_handler.heap_start = heap;
    _heap_memory_handler.available_blocks=BLOCK_COUNT;
    _heap_memory_handler.total_blocks=BLOCK_COUNT;
    _heap_memory_handler.available_size=HEAP_SIZE;
    _heap_memory_handler.total_size=HEAP_SIZE;
    memset((void*)_heap_memory_handler.heap_start, 0, HEAP_SIZE);
}
```

```c
void* malloc(size_t size){
    static uint8_t  block_id = 1;
    uint8_t *ptr = NULL, served=0;
    uint32_t blocks_req = (size/BLOCK_SIZE) + ((size%BLOCK_SIZE)!=0);
    if(blocks_req<= _heap_memory_handler.available_blocks){
        for(uint32_t i=0; (i<BLOCK_COUNT)&&(!served); i++){
            if(_heap_memory_handler.block[i]==0){
                for(uint32_t j=0; j<blocks_req; j++){
                    _heap_memory_handler.block[i+j] = block_id;
                    served=1;
                    print(" BLOCK [%u]  Block ID [%u] \n", (i+j+1), block_id);
                }
                if(served){
                    ptr = (uint8_t*)(_heap_memory_handler.heap_start + (i*BLOCK_SIZE));
                    block_id++;
                    _heap_memory_handler.available_blocks -= blocks_req;
                    _heap_memory_handler.available_size -= (blocks_req*BLOCK_SIZE);
                }
            }
        }
    }
    print(" Heap Start [%u] Heap ptr Allocated [%u] \n", _heap_memory_handler.heap_start, ptr);
    return (void *)ptr;
}
```

Room for Improvement:

- Timers
- Interrupt Handlers
- Preemptive Scheduling
- Mutex, Semaphores
- More sophisticated Memory management
- Facilitate Process Management
- System Calls
- etc ...


Any
Question


Thank
you!