HARVARD
Extension School

Principles of Operating Systems

CSCI E-92

Term-Project Presentation

Presented by:
Jyot Buch

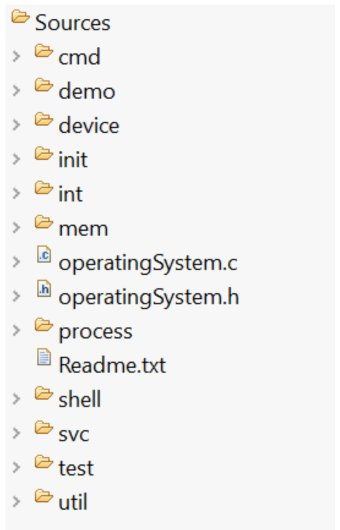
May 9th 2017, 9:00 PM

Agenda

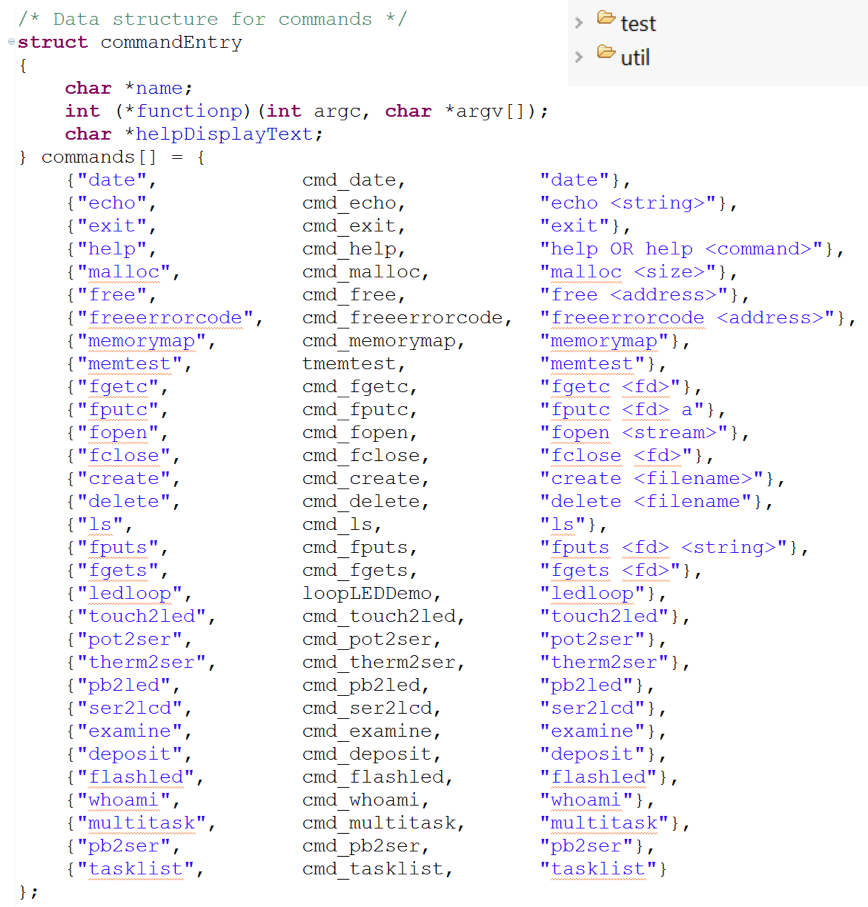
- Overview of my Operating System Implementation on Kinetics TWR-K70F120M
  - Shell & Shell Commands and Memory Management
  - Devices, File System and Supervisor Call Architecture
  - Multiprogramming
- Overview of Morse Code Translation
  - What is Morse Code ?
  - Morse LED Device implementation and use of PDB0 timer
  - How to use these devices ?
- What is different in my implementation ?
- Demo

Shell, Shell Commands and Memory Management

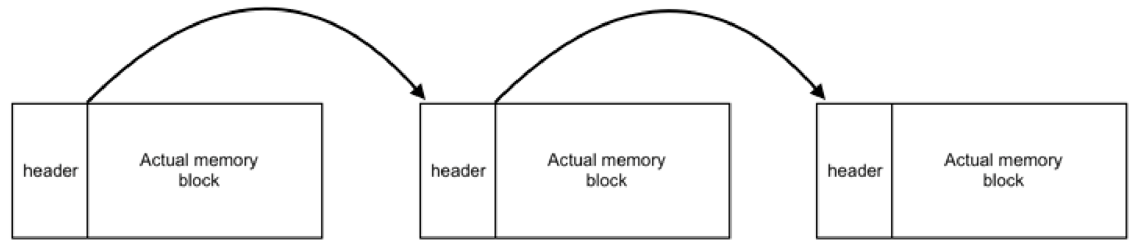
- Shell being the first process in multi-programming environment
- First Fit is used for memory management i.e. myMalloc and myFree infrastructure
- myFreeErrorCode
- freeMemoryForPID(pid_t pid)
- uint64_t computeMemoryForPID(pid_t pid)



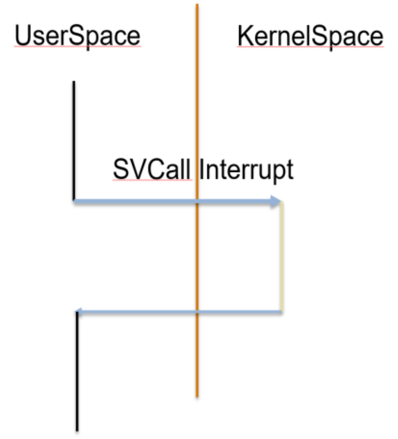
```
Sources
  cmd
  demo
  device
  init
  int
  mem
  operatingSystem.c
  operatingSystem.h
  process
  Readme.txt
  shell
  svc
  test
  util
```



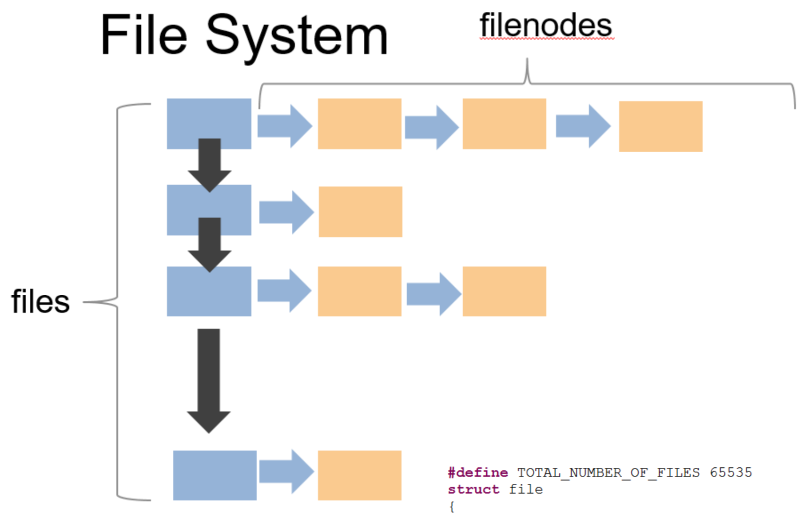
```
/* Data structure for commands */
struct commandEntry
{
    char *name;
    int (*functionp)(int argc, char *argv[]);
    char *helpDisplayText;
} commands[] = {
    {"date",           cmd_date,           "date"},
    {"echo",           cmd_echo,           "echo <string>"},
    {"exit",           cmd_exit,           "exit"},
    {"help",           cmd_help,           "help OR help <command>"},
    {"malloc",         cmd_malloc,         "malloc <size>"},
    {"free",           cmd_free,           "free <address>"},
    {"freeerrorcode",  cmd_freeerrorcode,  "freeerrorcode <address>"},
    {"memorymap",      cmd_memorymap,      "memorymap"},
    {"memtest",        tmemtest,           "memtest"},
    {"fgetc",          cmd_fgetc,          "fgetc <fd>"},
    {"fputc",          cmd_fputc,          "fputc <fd> a"},
    {"fopen",          cmd_fopen,          "fopen <stream>"},
    {"fclose",         cmd_fclose,         "fclose <fd>"},
    {"create",         cmd_create,         "create <filename>"},
    {"delete",         cmd_delete,         "delete <filename"},
    {"ls",             cmd_ls,             "ls"},
    {"fputs",          cmd_fputs,          "fputs <fd> <string>"},
    {"fgets",          cmd_fgets,          "fgets <fd>"},
    {"ledloop",        loopLEDDemo,        "ledloop"},
    {"touch2led",      cmd_touch2led,      "touch2led"},
    {"pot2ser",        cmd_pot2ser,        "pot2ser"},
    {"therm2ser",      cmd_therm2ser,      "therm2ser"},
    {"pb2led",         cmd_pb2led,         "pb2led"},
    {"ser2lcd",        cmd_ser2lcd,        "ser2lcd"},
    {"examine",        cmd_examine,        "examine"},
    {"deposit",        cmd_deposit,        "deposit"},
    {"flashled",       cmd_flashled,       "flashled"},
    {"whoami",         cmd_whoami,         "whoami"},
    {"multitask",      cmd_multitask,      "multitask"},
    {"pb2ser",         cmd_pb2ser,         "pb2ser"},
    {"tasklist",       cmd_tasklist,       "tasklist"}
};
```



Devices and File system Implementation & Supervisor Call Architecture



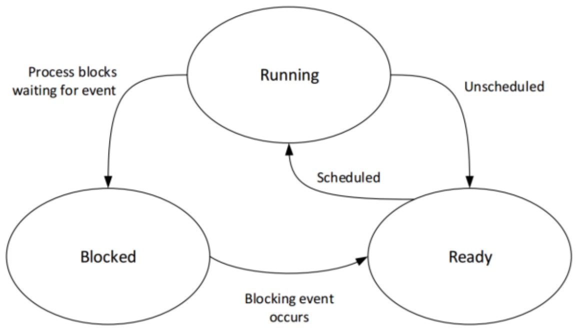
File System

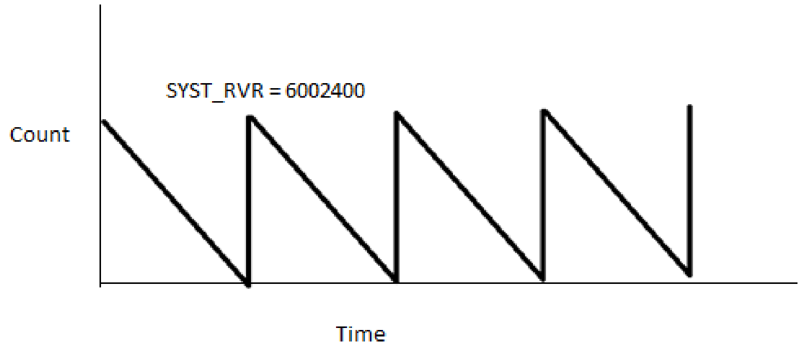


```c
#define TOTAL_NUMBER_OF_FILES 65535
struct file
{
    bool isFree;
    struct file_node *head; /* Head of the file node - some memory location for the file */
    uint32_t noOfFileNodes; /* Number of file nodes */
    char name[256]; /* Name of the file which may be up to 256 characters */
    /* uint8_t permissions; */ /* May be in future to provide the permissions for file access */
    enum file_type type; /* File or Directory */
    uint32_t file_size; /* Size of the file which is sum of all the file_node sizes */
};

struct file *allFiles;

/* Global variable which says how many files exists in a system */
int numOfFiles = 0;

/* SINGLe FILE_NODE_DATA_SIZE_LIMIT */
#define SINGLE_FILE_NODE_DATA_SIZE 1018

struct file_node
{
    uint8_t data[SINGLE_FILE_NODE_DATA_SIZE];
    uint16_t filled_size; /* Filled Size for particular node */
    struct file_node *next;
};
```

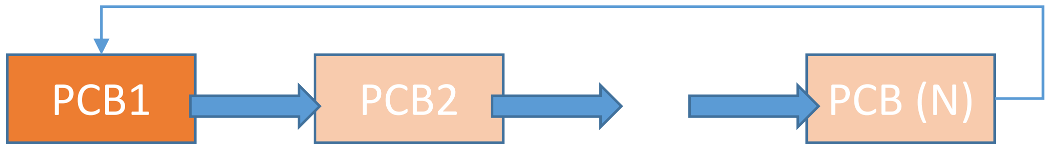
```c
struct deviceEntry
{
    char *name;
    int majorDeviceID;
    int minorDeviceID;
} allDevices[] = {
        {"/dev/led/orange",         LED,          ORANGE},
        {"/dev/led/yellow",         LED,          YELLOW},
        {"/dev/led/blue",           LED,          BLUE},
        {"/dev/led/green",          LED,          GREEN},
        {"/dev/pushbutton/pb1",     PUSHBUTTON,   PB1},
        {"/dev/pushbutton/pb2",     PUSHBUTTON,   PB2},
        {"",                        FILESYSTEM,   FILE_REGULAR},
        {"/dev/uart/2",             UART_COMM,    UART_2},
        {"/dev/lcdc/rgb",           LCDC_DEV,     RGB},
        {"/dev/adc/pot",            ADC,          POTENTIOMETER},
        {"/dev/adc/temp",           ADC,          TEMPRATURE_SENSOR},
        {"/dev/touch/ts1",          TOUCH,        TS1},
        {"/dev/touch/ts2",          TOUCH,        TS2},
        {"/dev/touch/ts3",          TOUCH,        TS3},
        {"/dev/touch/ts4",          TOUCH,        TS4},
        {"/dev/morse/led/orange",   MORSE_LED,    ORANGE},
        {"/dev/morse/led/yellow",   MORSE_LED,    YELLOW},
        {"/dev/morse/led/blue",     MORSE_LED,    BLUE},
        {"/dev/morse/led/green",    MORSE_LED,    GREEN},
};
```

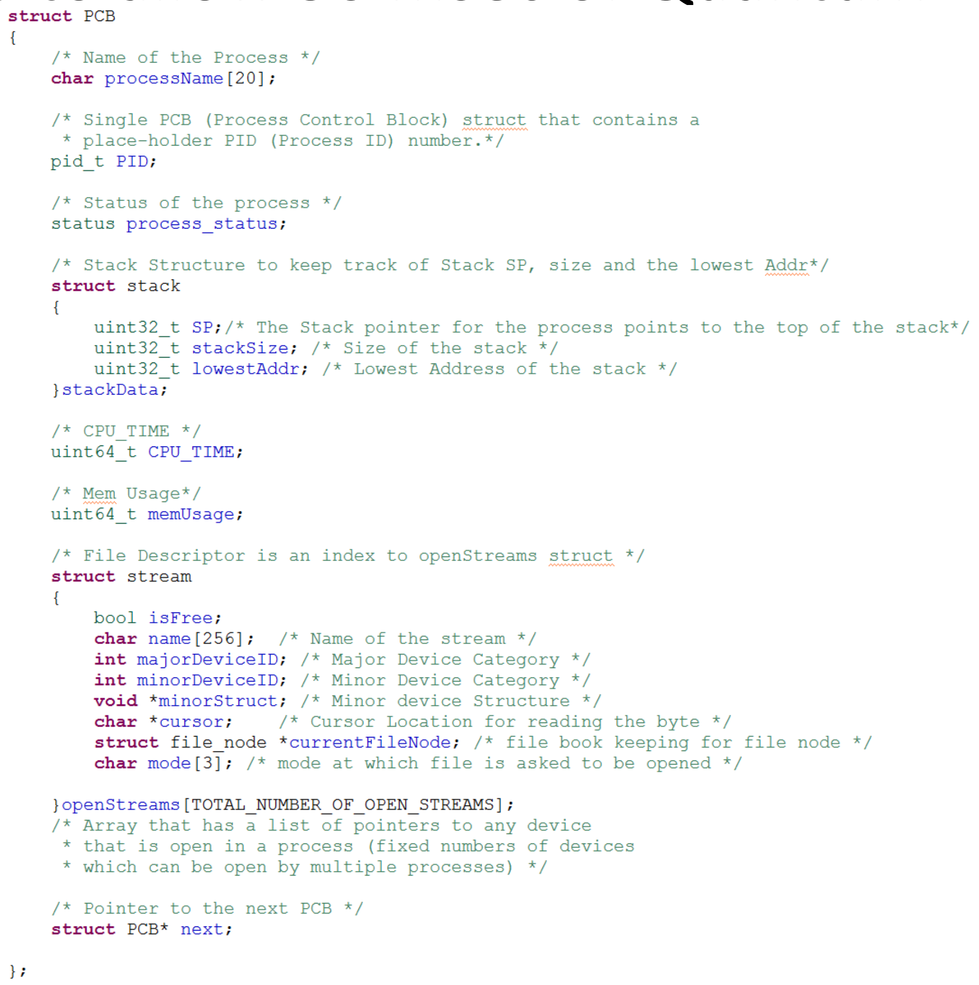
Multi Tasking





- SysTick Timer Reload Value : 6002400 to allow 50 msec of Quantum to each process
- Round Robbin Scheduling (Still Working)
- Linked list of PCBs





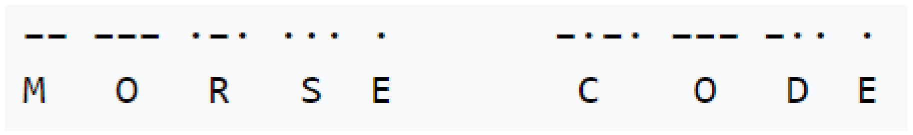
```
struct PCB
{
    /* Name of the Process */
    char processName[20];

    /* Single PCB (Process Control Block) struct that contains a
     * place-holder PID (Process ID) number.*/
    pid_t PID;

    /* Status of the process */
    status process_status;

    /* Stack Structure to keep track of Stack SP, size and the lowest Addr*/
    struct stack
    {
        uint32_t SP;/* The Stack pointer for the process points to the top of the stack*/
        uint32_t stackSize; /* Size of the stack */
        uint32_t lowestAddr; /* Lowest Address of the stack */
    }stackData;

    /* CPU_TIME */
    uint64_t CPU_TIME;

    /* Mem Usage*/
    uint64_t memUsage;

    /* File Descriptor is an index to openStreams struct */
    struct stream
    {
        bool isFree;
        char name[256];  /* Name of the stream */
        int majorDeviceID; /* Major Device Category */
        int minorDeviceID; /* Minor Device Category */
        void *minorStruct; /* Minor device Structure */
        char *cursor;    /* Cursor Location for reading the byte */
        struct file_node *currentFileNode; /* file book keeping for file node */
        char mode[3]; /* mode at which file is asked to be opened */

    }openStreams[TOTAL_NUMBER_OF_OPEN_STREAMS];
    /* Array that has a list of pointers to any device
     * that is open in a process (fixed numbers of devices
     * which can be open by multiple processes) */

    /* Pointer to the next PCB */
    struct PCB* next;

};
```

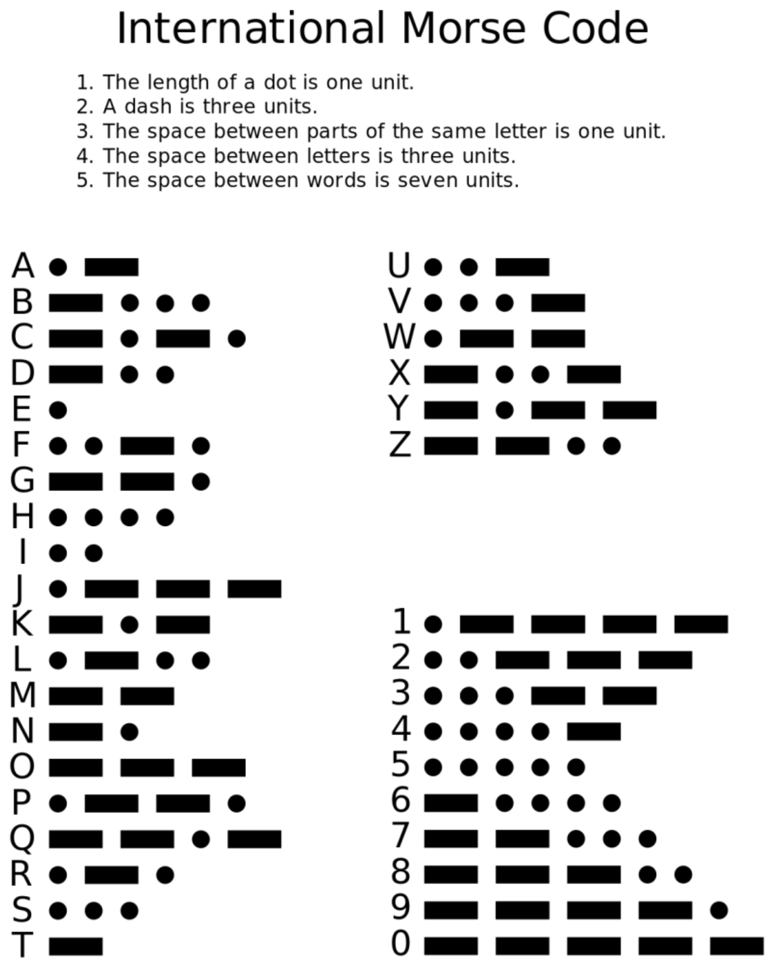
Term-Project: Morse Code Translation

- What is Morse Code ?
  - Morse code is a method of transmitting text information as a series of on-off tones, lights, or clicks that can be directly understood by a skilled listener
  - The International Morse Code encodes the ISO basic Latin alphabet, some extra Latin letters, the Arabic numerals and a small set of punctuation and procedural signals (prosigns) as standardized sequences of short and long signals called "dots" and "dashes", or "dits" and "dahs", as in amateur radio practice.

A U.S. Navy signalman sends Morse code signals in 2005.

1.short mark, dot or "dit" (■): 1
2.longer mark, dash or "dah" (▬): 111
3.intra-character gap (between the dots and dashes within a character): 0
4.short gap (between letters): 000
5.medium gap (between words): 0000000




International Morse Code

1. The length of a dot is one unit.
2. A dash is three units.
3. The space between parts of the same letter is one unit.
4. The space between letters is three units.
5. The space between words is seven units.

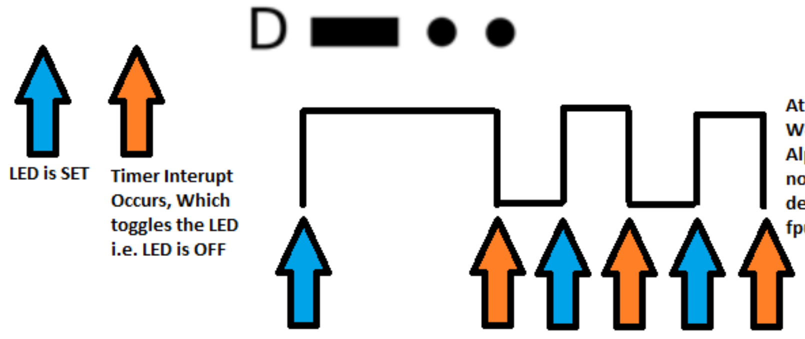
A	● ▬	U	● ● ▬
B	▬ ● ● ●	V	● ● ● ▬
C	▬ ● ▬ ●	W	● ▬ ▬
D	▬ ● ●	X	▬ ● ● ▬
E	●	Y	▬ ● ▬ ▬
F	● ● ▬ ●	Z	▬ ▬ ● ●
G	▬ ▬ ●		
H	● ● ● ●		
I	● ●		
J	● ▬ ▬ ▬		
K	▬ ● ▬	1	● ▬ ▬ ▬ ▬
L	● ▬ ● ●	2	● ● ▬ ▬ ▬
M	▬ ▬	3	● ● ● ▬ ▬
N	▬ ●	4	● ● ● ● ▬
O	▬ ▬ ▬	5	● ● ● ● ●
P	● ▬ ▬ ●	6	▬ ● ● ● ●
Q	▬ ▬ ● ▬	7	▬ ▬ ● ● ●
R	● ▬ ●	8	▬ ▬ ▬ ● ●
S	● ● ●	9	▬ ▬ ▬ ▬ ●
T	▬	0	▬ ▬ ▬ ▬ ▬

Ref: https://en.wikipedia.org/wiki/Morse_code

Morse LED Device implementation and use of PDB0 timer

- 4 Additional OUTPUT_ONLY Devices for each of the LEDs in Device Struct
- Named as “/dev/morse/led/orange” and such
- PDB0 Timer is used in the OneShotMode of Operation
- Device Method: morse_led_fputc
  - Identify the Character
  - Get the Index for the Morse code in table, based on isAlphabet(), isDigit(), isSpecialChars()
  - Loop through the string of Morse code i.e. “-…”
    - Switch Case statements for either . Or –
    - Turn ON the LED &
      Start a Timer with appropriate time duration to turn the LED OFF(i.e. Toggle)
- For Example: To Transmit Letter D
- Case Friendly implementation
  (same code gets transmitted for D and d)

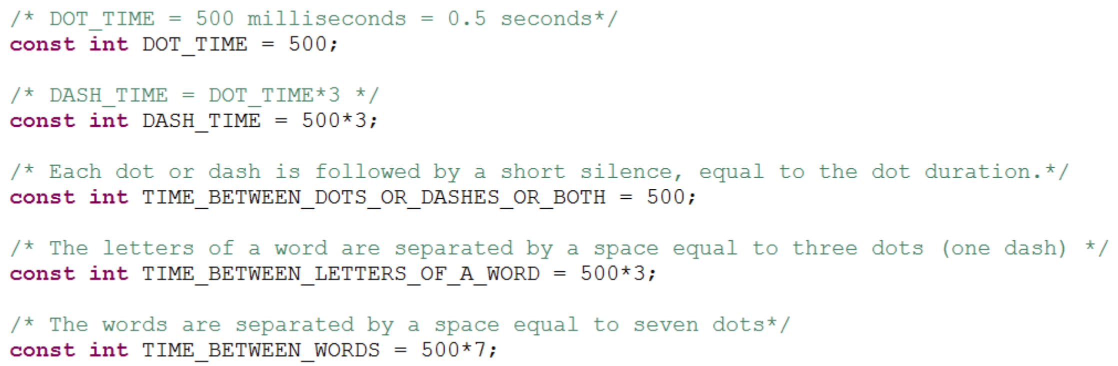
```c
struct
{
    char* morseCode;
    char asciiChar;
} morseCharTable[] = {
        {".-",    'A'},
        {"-...",  'B'},
        {"-.-.",  'C'},
        {"-..",   'D'},
        {".",     'E'},
        {"..-.",  'F'},
        {"--.",   'G'},
        {"....",  'H'},
        {"..",    'I'},
        {".---",  'J'},
        {"-.-",   'K'},
        {".-..",  'L'},
        {"--",    'M'},
        {"-.",    'N'},
        {"---",   'O'},
        {".--.",  'P'},
        {"--.-",  'Q'},
        {".-.",   'R'},
        {"...",   'S'},
        {"-",     'T'},
        {"..-",   'U'},
        {"...-",  'V'},
        {".--",   'W'},
        {"-..-",  'X'},
        {"-.--",  'Y'},
        {"--..",  'Z'}
};
```


D ▬ ● ●
LED is SET
Timer Interupt Occurs, Which toggles the LED i.e. LED is OFF

At This point,
We do not know what is the next Alphbet/Digit/SpecialChar/Space going to be ? so we do not terminate with by a space of dot. That will be determined by the parent caller of fputc() in the case of fputs()

How to use these devices ?

- Simultaneously, my implementation display what was transmitted to the UART



```
/* DOT_TIME = 500 milliseconds = 0.5 seconds*/
const int DOT_TIME = 500;

/* DASH_TIME = DOT_TIME*3 */
const int DASH_TIME = 500*3;

/* Each dot or dash is followed by a short silence, equal to the dot duration.*/
const int TIME_BETWEEN_DOTS_OR_DASHES_OR_BOTH = 500;

/* The letters of a word are separated by a space equal to three dots (one dash) */
const int TIME_BETWEEN_LETTERS_OF_A_WORD = 500*3;

/* The words are separated by a space equal to seven dots*/
const int TIME_BETWEEN_WORDS = 500*7;
```

```
$ fopen /dev/morse/led/orange
Returned File Descriptor is: 3
The command executed with return status 0.

$ fputc 3 a
  . –
```

Following visualization and LED blinking is available for the device:

```
$ fputs 3 Hello World
....
.
.-..
.-..
---


.--
---
.-.
.-..
-..
```

So, What’s different in my implementation ?

- User Identities : admin, jyotbuch, guest
    - Only privileged user(admin) is allowed to change the timeofday clock
    - For unprivileged user, the clock comes as pre-initialized to 1st Jan 1970, 00:00:00 Midnight
- whoami() shell command to return the logged in username
- Morse Code Translation and Display using PDB0 timer
- Global ENV based PCB and PID reference:
    - That means, if the currentPCB(e.g. with PID 3) process is currently running and if user tries to kill, let say process with PID 8, in SVCKillImpl() when we try to close all the opened streams for PID 8, we can simply set the global ENV variable and use SVCfclose_mainImpl() function. NOTE that if you do not set the ENV, this function will close the stream of PID 3 processes, because it then internally uses currentPCB, instead of process 8th PCB.
- FileSystemInit() initializes allFiles to be owned by the OS Kernel, so even if the process gets killed, the file still exists
- Tasklist Shell command that reports following to the USER:
    - Currently Available processes and their names
    - PID Numbers
    - CPU time
    - Memory Occupied by that process

PID	Associated with
0	All Free Memory Blocks
1	Shell (filesystem and everything related to kernel)
2	Process 2 and So on….