#LEARN IN DEPTH

#Be professional in embedded system

Embedded Linux (PART 4)

- ALARM
- READ/WRITE DIRECTORIES/FILES
- PTHREAD
- PTHREAD MUTEXES
- SOCKET PROGRAMMING
- LINUX COMMAND LINE INTERFACE CLI
- LINUX FILE-SYSTEM HIERARCHY

ENG.KEROLES SHENOUDA

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

#LEARN IN DEPTH

#Be professional in embedded system

Alarm

Alarm

- set an alarm clock for delivery of a signal

```
#include <unistd.h>
unsigned int alarm(unsigned int seconds);
```

- **alarm**() arranges for a **SIGALRM** signal to be delivered to the process in *seconds*.
- **SIGALRM** is the alarm signal that is generated when the timer set by the alarm function goes off.
- If *seconds* is zero, no new **alarm**() is scheduled.

alarm API - alarm

* Function

  The ***alarm*** API requests the kernel to send the SIGALRM signal after a certain number of real clock seconds.

* Include: *<signal.h>*
* Summary: unsigned *int* ***alarm****(unsigned int time_interval);*
* Return:

  Success: **the number of CPU seconds left in the process timer**;

  Failure: -1; Sets errno: Yes

* Argument

  time_interval: the number of CPU seconds elapsed time, after which the kernel will send the SIGALRM signal to the calling process.

#LEARN IN DEPTH

#Be professional in embedded system

Alarm-lab1 what is the output ?

main.c

```c
/*
 * main.c
 *
 *  Created on: Dec 10, 2019
 *      Author: Keroles Shenouda
 *      www.Learn-in-depth.com
 */
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
void signals_lab ();

int main ()
{
    signals_lab () ;
    return 0 ;
}

void sig_handler(int signum)
{
    printf("Alarm delivered signum is %d\n", signum);
    system("/bin/ls");
}
void signals_lab ()
{
        alarm(5);
        signal(SIGALRM, sig_handler);

        //start executing from this line only if not exited in signal handler.
        for(;;)
            sleep(1);
}
```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

#LEARN IN DEPTH

#Be professional in embedded system

Alarm-lab1 what is the output ?

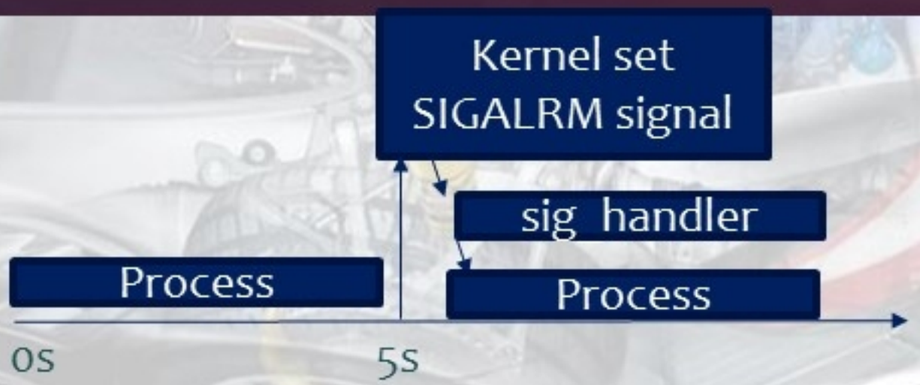
main.c

```c
/*
 * main.c
 *
 *  Created on: Dec 10, 2019
 *      Author: Keroles Shenouda
 *      www.Learn-in-depth.com
 */
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
void signals_lab ();

int main ()
{
    signals_lab () ;
    return 0 ;
}

void sig_handler(int signum)
{
    printf("Alarm delivered signum is %d\n", signum);
    system("/bin/ls");
}
void signals_lab ()
{
        alarm(5);
        signal(SIGALRM, sig_handler);

        //start executing from this line only if not exited in signal handler.
        for(;;)
            sleep(1);
}
```

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/POSIX/ws/Po
LABS/Debug$ ./Posix_LABS
Alarm delivered signum is 14
main.d  main.o  makefile  objects.mk  Posix_LABS  sources.mk  subdir.mk
```

Embedded Linux

ENG.KEROLES SHENOUDA



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Alarm-lab1 what is the output ?
To measure the time use time ./your_executable

#LEARN IN DEPTH

#Be professional in embedded system

main.c

```c
/*
 * main.c
 *
 *  Created on: Dec 10, 2019
 *      Author: Keroles Shenouda
 *      www.Learn-in-depth.com
 */
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
void signals_lab ();

int main ()
{
    signals_lab () ;
    return 0 ;
}

void sig_handler(int signum)
{
    printf("Alarm delivered signum is %d\n", signum);
    exit (0);
}
void signals_lab ()
{
    alarm(5);
    signal(SIGALRM, sig_handler);

    //start executing from this line only if not exited in signal handler.
    for(;;)
        sleep(1);
}
```

```
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_Linux/POSIX/ws/Posix_LABS/
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/POSIX/ws/Posix_
LABS/Debug$ time ./Posix_LABS
Alarm delivered signum is 14

real    0m5.001s
user    0m0.000s
sys     0m0.000s
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/POSIX/ws/Posix_
LABS/Debug$
```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

What will happen in this case if we doesn't set the alarm handler ?
You can debug the kernel by strace ./executlable_file

main.c

```c
/*
 * main.c
 *
 *  Created on: Dec 10, 2019
 *      Author: Keroles Shenouda
 *      www.Learn-in-depth.com
 */
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
void signals_lab ();

int main ()
{
    signals_lab () ;
    return 0 ;
}

void sig_handler(int signum)
{
    printf("Alarm delivered signum is %d\n", signum);
    exit (0);
}
void signals_lab ()
{
    alarm(5);
//  signal(SIGALRM, sig_handler);

    //start executing from this line only if not exited in signal handler.
    for(;;)
        sleep(1);
}
```

What will happen in this case if we doesn't set the alarm handler ?
You can debug the kernel by strace ./executlable_file

main.c

```c
/*
 * main.c
 *
 *  Created on: Dec 10, 2019
 *      Author: Keroles Shenouda
 *      www.Learn-in-depth.com
 */
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
void signals_lab ();

int main ()
{
    signals_lab () ;
    return 0 ;
}

void sig_handler(int signum)
{
    printf("Alarm delivered signum is %d\n", signum);
    exit (0);
}
void signals_lab ()
{
    alarm(5);
//  signal(SIGALRM, sig_handler);

    //start executing from this line only if not exited in signal handler.
    for(;;)
        sleep(1);
}
```

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/POSIX/ws/Posix_LABS/Debug$ ./Posix_LABS
Alarm clock
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/POSIX/ws/Posix_LABS/Debug$
```

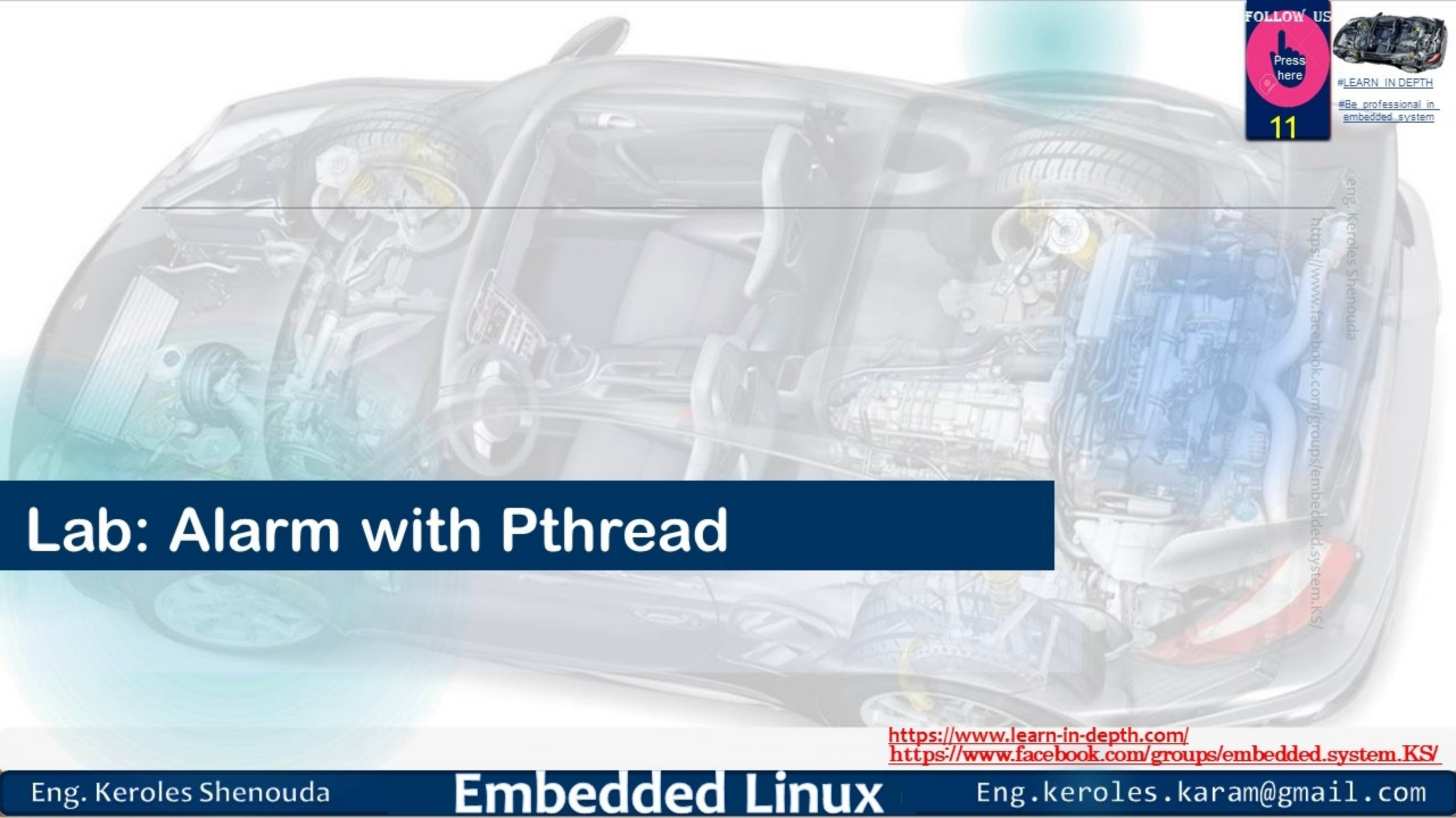
What that means?
Think_in_depth
This is Crash

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

What will happen in this case if we doesn't set the alarm handler ?
You can debug the kernel by strace ./executlable_file

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/POSIX/ws/Posix_
LABS/Debug$ strace ./Posix_LABS
execve("./Posix_LABS", ["./Posix_LABS"], [/* 66 vars */]) = 0
brk(NULL)                               = 0x8643000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap2(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7711000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=112722, ...}) = 0
mmap2(NULL, 112722, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb76f5000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/i386-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\1\1\1\3\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\204\1\0004\0\0\0"..., 512) =
 512
fstat64(3, {st_mode=S_IFREG|0755, st_size=1802928, ...}) = 0
mmap2(NULL, 1808924, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0xb753b000
mmap2(0xb76ef000, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1b3000) = 0xb76ef000
mmap2(0xb76f2000, 10780, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1,
 0) = 0xb76f2000
close(3)                                = 0
set_thread_area({entry_number:-1, base_addr:0xb77137c0, limit:1048575, seg_32bit:1, con
tents:0, read_exec_only:0, limit_in_pages:1, seg_not_present:0, useable:1}) = 0 (entry_
number:6)
mprotect(0xb76ef000, 8192, PROT_READ)   = 0
mprotect(0x8049000, 4096, PROT_READ)    = 0
mprotect(0xb773b000, 4096, PROT_READ)   = 0
munmap(0xb76f5000, 112722)              = 0
alarm(5)                                = 0
nanosleep({1, 0}, 0xbfb60608)           = 0
nanosleep({1, 0}, 0xbfb60608)           = 0
nanosleep({1, 0}, {0, 116500324})       = ? ERESTART_RESTARTBLOCK (Interrupted by signa
l)
```

--- SIGALRM {si_signo=SIGALRM, si_code=SI_KERNEL} ---
+++ killed by SIGALRM +++
Alarm clock



Lab: Alarm with Pthread

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Lab: Alarm with Pthread

- Create Two process (parent and child)
- The parent process get alarm after 5 sec
- The Child process get alarm after 10 sec
- As shown
- And that will run forever

```
embedded_system_ks@embedded-KS: ~
embedded_system_ks@embedded-KS:~$ gcc two_process_alarm.c
embedded_system_ks@embedded-KS:~$ ./a.out
 am the parent! getpid=5234
 am the child! getpid=5235

Alarm delivered >>>>>> each 5 sec (sig_handler_parent_process) signum is 14 getpid=5234

Alarm delivered >>>>>> each 5 sec (sig_handler_parent_process) signum is 14 getpid=5234

Alarm delivered >>>>>>  >>>>>> each 10 sec  (sig_handler_child_process) signum is 14 getpid=5235

Alarm delivered >>>>>> each 5 sec (sig_handler_parent_process) signum is 14 getpid=5234


Alarm delivered >>>>>>  >>>>>> each 10 sec  (sig_handler_child_process) signum is 14 getpid=5235
Alarm delivered >>>>>> each 5 sec (sig_handler_parent_process) signum is 14 getpid=5234

Alarm delivered >>>>>> each 5 sec (sig_handler_parent_process) signum is 14 getpid=5234

Alarm delivered >>>>>> each 5 sec (sig_handler_parent_process) signum is 14 getpid=5234

Alarm delivered >>>>>>  >>>>>> each 10 sec  (sig_handler_child_process) signum is 14 getpid=5235

Alarm delivered >>>>>> each 5 sec (sig_handler_parent_process) signum is 14 getpid=5234

Alarm delivered >>>>>>  >>>>>> each 10 sec  (sig_handler_child_process) signum is 14 getpid=5235

Alarm delivered >>>>>> each 5 sec (sig_handler_parent_process) signum is 14 getpid=5234
```

#LEARN IN DEPTH

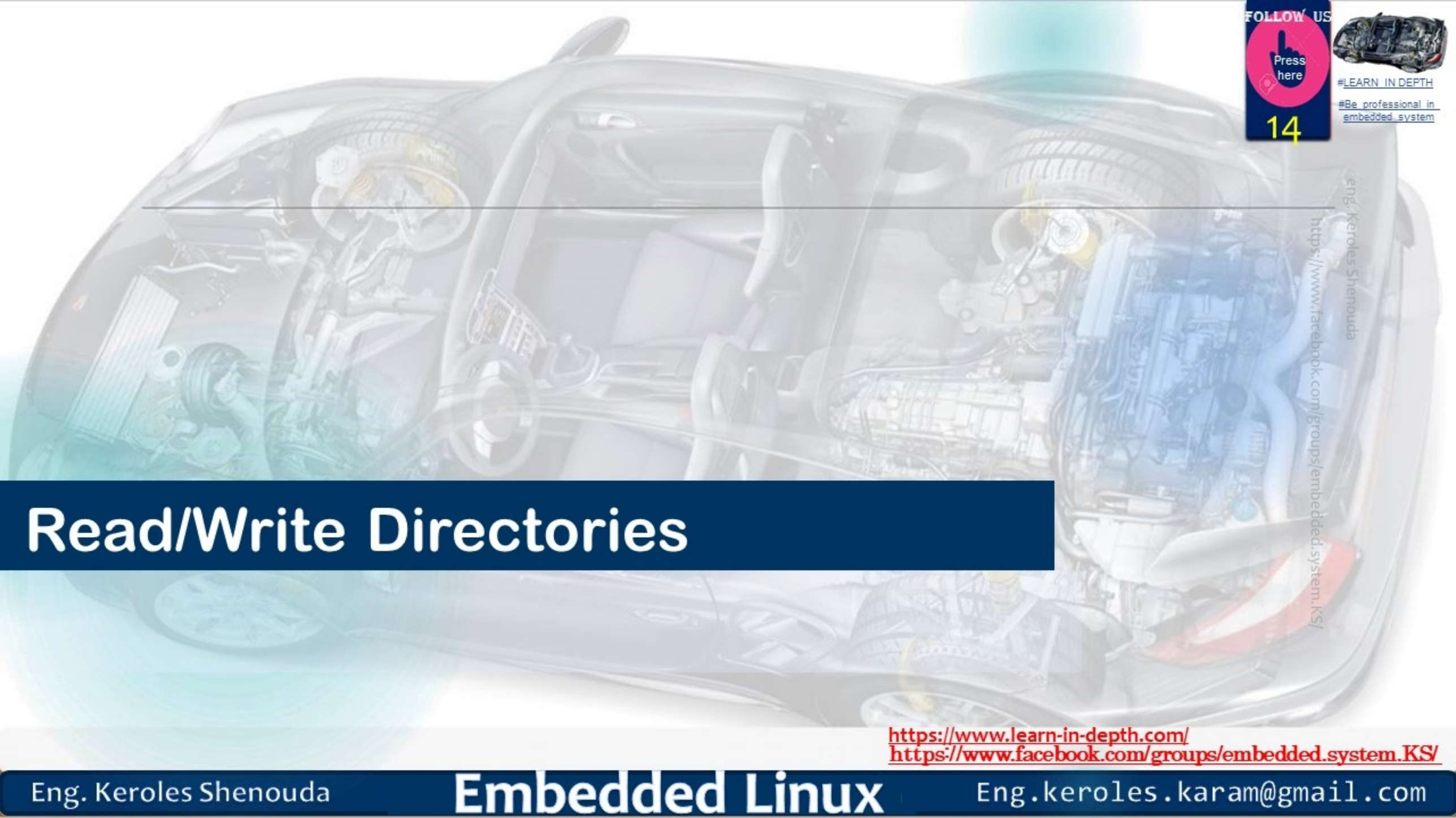
#Be professional in embedded system

Lab: Alarm Solution

```c
/*
 * main.c
 *
 *  Created on: Nov 10, 2019
 *      Author: Keroles Shenouda
 */
//////////////////////////////

#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>

void sig_handler_parent_process(int signum)
{
        printf("\n Alarm delivered >>>>>> each 5 sec (sig_handler_parent_process) signum is %d getpid=%d \n", signum,getpid());
        alarm(5);
        signal(SIGALRM, sig_handler_parent_process);
}
void sig_handler_child_process(int signum)
{
        printf("\n Alarm delivered >>>>>>  >>>>>> each 10 sec  (sig_handler_child_process) signum is %d getpid=%d \n", signum,getpid());
        alarm(10);
        signal(SIGALRM, sig_handler_child_process);
}
int  main(int argc, char **argv)
{
        pid_t pid,pid_parent,pid_child;
        pid = fork ();


        if (pid > 0) //Parent Process
        {
                printf ("I am the parent! getpid=%d \n", getpid());
                alarm(5);
                signal(SIGALRM, sig_handler_parent_process);

        }
        else if (!pid) //pid =0 Child  Process
        {
                printf ("I am the child! getpid=%d \n", getpid());
                alarm(10);
                signal(SIGALRM, sig_handler_child_process);
        }
        else if (pid < 0)
                perror ("fork");


        while (1);////infinite loop for parent

}
//////////////////////////////
```

```
embedded_system_ks@embedded-KS: ~
embedded_system_ks@embedded-KS:~$ gcc two_process_alarm.c
embedded_system_ks@embedded-KS:~$ ./a.out
 am the parent! getpid=5234
 am the child! getpid=5235

Alarm delivered >>>>>> each 5 sec (sig_handler_parent_process) signum is 14 getpid=5234

Alarm delivered >>>>>> each 5 sec (sig_handler_parent_process) signum is 14 getpid=5234

Alarm delivered >>>>>>  >>>>>> each 10 sec  (sig_handler_child_process) signum is 14 getpid=5235

Alarm delivered >>>>>> each 5 sec (sig_handler_parent_process) signum is 14 getpid=5234


Alarm delivered >>>>>>  >>>>>> each 10 sec  (sig_handler_child_process) signum is 14 getpid=5235
Alarm delivered >>>>>> each 5 sec (sig_handler_parent_process) signum is 14 getpid=5234

Alarm delivered >>>>>> each 5 sec (sig_handler_parent_process) signum is 14 getpid=5234

Alarm delivered >>>>>> each 5 sec (sig_handler_parent_process) signum is 14 getpid=5234

Alarm delivered >>>>>>  >>>>>> each 10 sec  (sig_handler_child_process) signum is 14 getpid=5235

Alarm delivered >>>>>> each 5 sec (sig_handler_parent_process) signum is 14 getpid=5234

Alarm delivered >>>>>>  >>>>>> each 10 sec  (sig_handler_child_process) signum is 14 getpid=5235

Alarm delivered >>>>>> each 5 sec (sig_handler_parent_process) signum is 14 getpid=5234
```

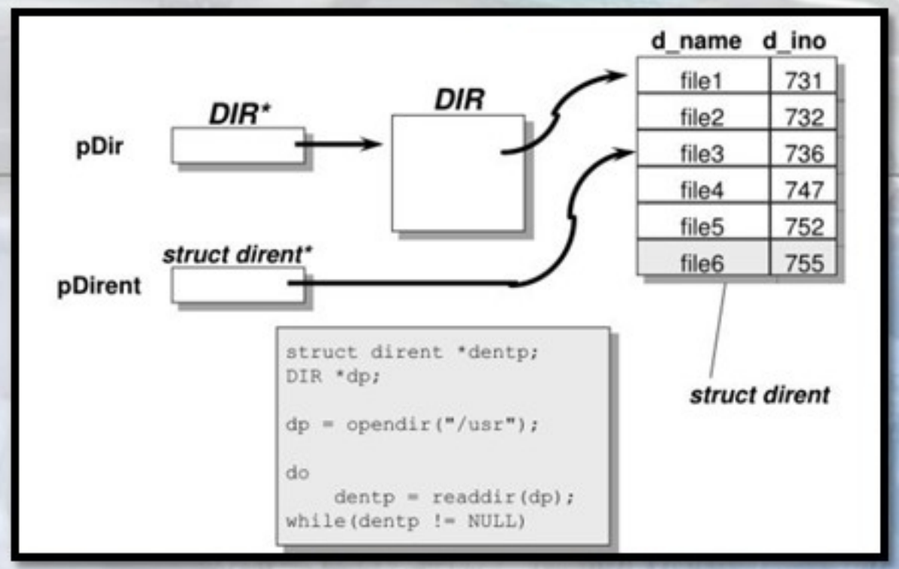
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



Read/Write Directories

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Directories



- Accessing directories:
  - Open a directory
  - Iterate through its contents
  - Close the directory
- Opening a directory:

```
DIR *opendir(const char* name);
```

  - Opens a directory given by `name` and provides a pointer `DIR*` to access files within the directory.
- Don't forget to close the directory when done:

```
int closedir(DIR *dirp);
```

To get more informationes:

[man 3 opendir]

[man 3 closedir]

Example

main.c

```c
/*
 * main.c
 *
 *  Created on: Dec 10, 2019
 *      Author: Keroles Shenouda
 *      www.Learn-in-depth.com
 */
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <dirent.h>
#include <string.h>

void signals_lab ();
Dir_example (int argc, char *argv[]);

int main (int argc, char *argv[])
{
//  signals_lab () ;
    Dir_example ( argc, argv);
    return 0 ;
}

/* this can serve as the ls command implementation in c language */

int Dir_example (int argc, char *argv[])
{
    /* DIR:- Opaque data type representing directory stream */
    DIR  *dp;
    printf("argv[0]=%s\n", argv[0]);
    printf("argv[1]=%s\n", argv[1]);
    /* dirent is a structure which cantains information about a file like name, inode, offset etc */
    struct dirent   *dirp;

    if(argc !=2 )
    {
        printf("Please provide the directory to list out\n");
        exit(1);
    }

    if(( dp = opendir(argv[1])) == NULL)
    {
        printf("Cannot open directory error=%s\n", strerror(errno));
        exit(1);
    }

    while(( dirp = readdir(dp)) != NULL)
        printf("%s\n", dirp->d_name);

    closedir(dp);
    exit(0);
```

Example

main.c

```c
/*
 * main.c
 *
 *  Created on: Dec 10, 2019
 *      Author: Keroles Shenouda
 *      www.Learn-in-depth.com
 */
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <dirent.h>
#include <string.h>

void signals_lab ();
Dir_example (int argc, char *argv[]);

int main (int argc, char *argv[])
{
//  signals_lab () ;
    Dir_example ( argc, argv);
    return 0 ;
}

/* this can serve as the ls command implementation in c language */

int Dir_example (int argc, char *argv[])
{
    /* DIR:- Opaque data type representing directory stream */
    DIR  *dp;
    printf("argv[0]=%s\n", argv[0]);
    printf("argv[1]=%s\n", argv[1]);
    /* dirent is a structure which cantains information about a file like name, inode, offset etc */
    struct dirent   *dirp;

    if(argc !=2 )
    {
        printf("Please provide the directory to list out\n");
        exit(1);
    }

    if(( dp = opendir(argv[1])) == NULL)
    {
        printf("Cannot open directory error=%s\n", strerror(errno));
        exit(1);
    }

    while(( dirp = readdir(dp)) != NULL)
        printf("%s\n", dirp->d_name);

    closedir(dp);
    exit(0);
```

```c
    while(( dirp = readdir(dp)) != NULL)
        printf("%s\n", dirp->d_name);

    closedir(dp);
    exit(0);

oid sig_handler(int signum)

    printf("Alarm delivered sig
    exit (0);
```

- d_ino : ?
- d_name : char [256]
- d_off : ?
- d_reclen : unsigned short int
- d_type : unsigned char

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

#LEARN IN DEPTH

#Be professional in embedded system

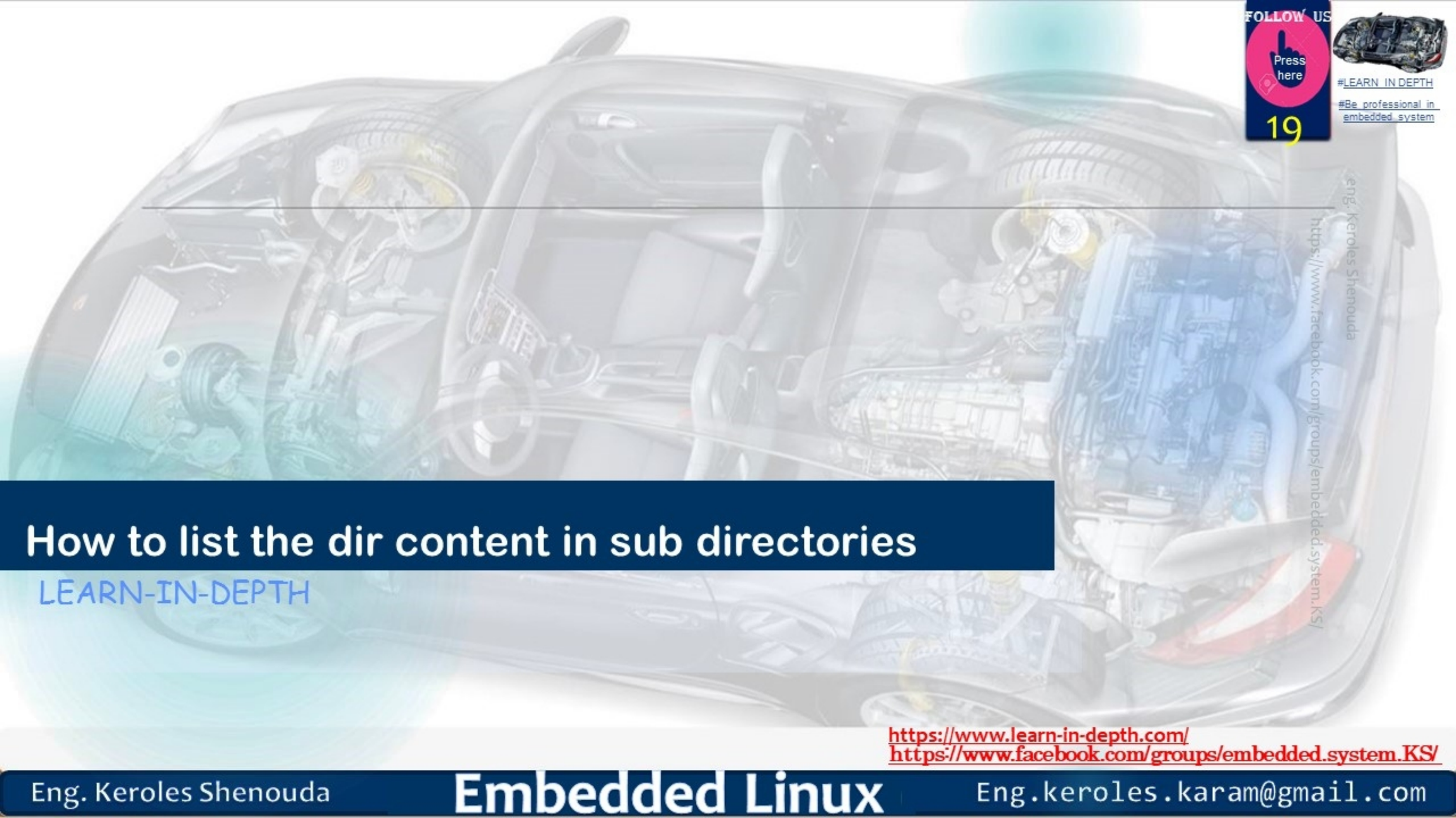
Example

main.c

```c
/*
 * main.c
 *
 *  Created on: Dec 10, 2019
 *      Author: Keroles Shenouda
 *      www.Learn-in-depth.com
 */
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <dirent.h>
#include <string.h>

void signals_lab ();
Dir_example (int argc, char *argv[]);

int main (int argc, char *argv[])
{
//  signals_lab () ;
    Dir_example ( argc, argv);
    return 0 ;
}

/* this can serve as the ls command implementation in c language */

int Dir_example (int argc, char *argv[])
{
    /* DIR:- Opaque data type representing directory stream */
    DIR  *dp;
    printf("argv[0]=%s\n", argv[0]);
    printf("argv[1]=%s\n", argv[1]);
    /* dirent is a structure which cantains information about a file like name, inode, offset etc */
    struct dirent   *dirp;

    if(argc !=2 )
    {
        printf("Please provide the directory to list out\n");
        exit(1);
    }

    if(( dp = opendir(argv[1])) == NULL)
    {
        printf("Cannot open directory error=%s\n", strerror(errno));
        exit(1);
    }

    while(( dirp = readdir(dp)) != NULL)
        printf("%s\n", dirp->d_name);

    closedir(dp);
    exit(0);
```

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/POSIX/ws/Posix_
LABS/Debug$ ./Posix_LABS ../Debug/
argv[0]=./Posix_LABS
argv[1]=../Debug/
main.o
..
main.d
objects.mk
sources.mk
subdir.mk
Posix_LABS
makefile
.
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/POSIX/ws/Posix_
LABS/Debug$ ls -la
total 184
drwxr-xr-x 2 embedded_system_ks embedded_system_ks  4096 ري د 10 05:18 .
drwxr-xr-x 3 embedded_system_ks embedded_system_ks  4096 ري د 10 02:53 ..
-rw-r--r-- 1 embedded_system_ks embedded_system_ks    18 ري د 10 05:18 main.d
-rw-r--r-- 1 embedded_system_ks embedded_system_ks 90980 ري د 10 05:18 main.o
-rw-r--r-- 1 embedded_system_ks embedded_system_ks   970 ري د 10 05:18 makefile
-rw-r--r-- 1 embedded_system_ks embedded_system_ks   231 ري د 10 02:53 objects.mk
-rwxr-xr-x 1 embedded_system_ks embedded_system_ks 65272 ري د 10 05:18 Posix_LABS
-rw-r--r-- 1 embedded_system_ks embedded_system_ks   390 ري د 10 05:18 sources.mk
-rw-r--r-- 1 embedded_system_ks embedded_system_ks   644 ري د 10 05:18 subdir.mk
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/POSIX/ws/Posix_
LABS/Debug$
```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/



How to list the dir content in sub directories

LEARN-IN-DEPTH

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

#LEARN IN DEPTH

#Be professional in embedded system

How to list the dir content in sub directories

```c
/*
 * main.c
 *
 *  Created on: Dec 10, 2019
 *      Author: Keroles Shenouda
 *      www.Learn-in-depth.com
 */
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <dirent.h>
#include <string.h>

void signals_lab ();
Dir_example (int argc, char *argv[]);
void listdir(const char *pathname);

int main (int argc, char *argv[])
{
//  signals_lab () ;
//  Dir_example ( argc, argv);
    DIR  *dp;
    struct dirent   *dirp;

    if( argc !=2 )
    {
        printf("Please provide the directory to list out\n");
        exit(1);
    }
    listdir(argv[1]);
    return 0 ;
}
```

```c
/* this can serve as the ls command implementation in c language */

void listdir(const char *pathname)
{
    DIR  *dp;
    struct dirent   *dirp;
    char PATH[259] = {0};

    if ((dp = opendir(pathname)) == NULL)
        return;
    while(( dirp = readdir(dp)) != NULL)
    {

        if(dirp->d_type == DT_DIR)
        {
            if( (strcmp(dirp->d_name,".") == 0) || (strcmp(dirp->d_name, "..") == 0) )
                continue;

            printf("%s\n", dirp->d_name);
            snprintf(PATH, sizeof(PATH)-1, "%s/%s",pathname,dirp->d_name);
            listdir(PATH);
        }
        else
        {
            printf("%s\n", dirp->d_name);
        }
    }
    closedir(dp);
}
```

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

#LEARN IN DEPTH

#Be professional in embedded system

How to list the dir content in sub directories

```
DESCRIPTION
       The  readdir()  function returns a pointer to a dirent structure representing
       the next directory entry in the directory stream  pointed  to  by  dirp.   It
       returns  NULL  on  reaching  the  end  of the directory stream or if an error
       occurred.

       In the glibc implementation, the dirent structure is defined as follows:

           struct dirent {
               ino_t          d_ino;       /* Inode number */
               off_t          d_off;       /* Not an offset; see below */
               unsigned short d_reclen;    /* Length of this record */
               unsigned char  d_type;      /* Type of file; not supported
                                              by all filesystem types */
               char           d_name[256]; /* Null-terminated filename */
           };
```

```
d_type This field contains a value indicating the file type, making it possi-
       ble to avoid the expense of calling lstat(2) if further actions depend
       on the type of the file.

       When a suitable feature test  macro  is  defined  (_DEFAULT_SOURCE  on
       glibc  versions  since 2.19, or _BSD_SOURCE on glibc versions 2.19 and
       earlier), glibc defines the following macro constants  for  the  value
       returned in d_type:

       DT_BLK      This is a block device.

       DT_CHR      This is a character device.

       DT_DIR      This is a directory.

       DT_FIFO     This is a named pipe (FIFO).

       DT_LNK      This is a symbolic link.

       DT_REG      This is a regular file.

       DT_SOCK     This is a UNIX domain socket.
```

```c
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <dirent.h>

/* this can serve as the ls command implementation in c language */

void listdir(const char *pathname)
{
    DIR  *dp;
    struct dirent   *dirp;
    char PATH[259] = {0};

    if ((dp = opendir(pathname)) == NULL)
        return;
    while(( dirp = readdir(dp)) != NULL)
    {

        if(dirp->d_type == DT_DIR)
        {
            if( (strcmp(dirp->d_name,".") == 0) || (strcmp(dirp->d_name, "..") == 0) )
                continue;

            printf("%s\n", dirp->d_name);
            snprintf(PATH, sizeof(PATH)-1, "%s/%s",pathname,dirp->d_name);
            listdir(PATH);
        }
        else
        {
            printf("%s\n", dirp->d_name);
        }
    }
    closedir(dp);
```

For ignore the hidden file

Recursive Calling

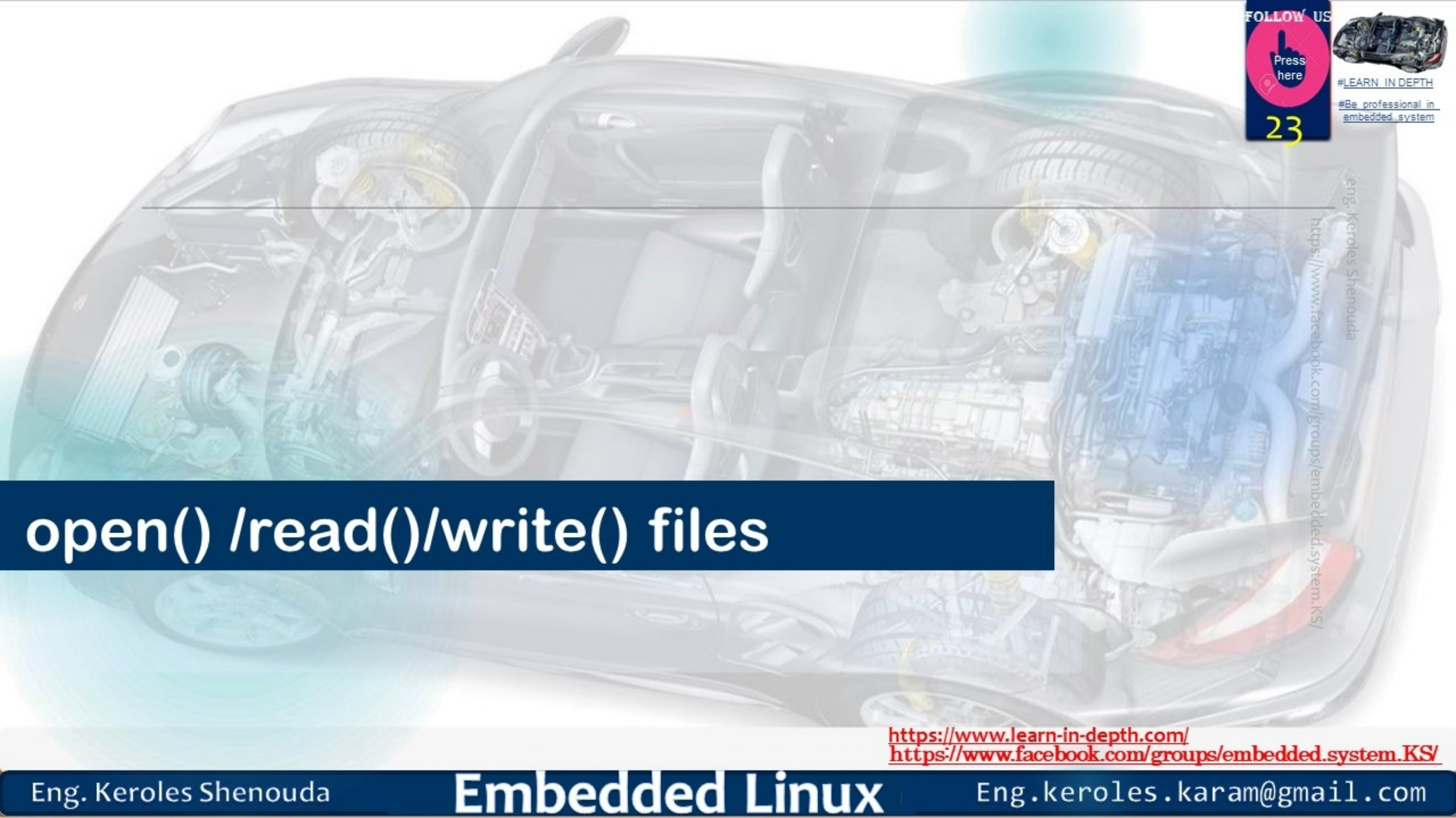
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

How to list the dir content in sub directories

#LEARN IN DEPTH

#Be professional in embedded system

```
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_Linux/POSIX/ws/Posix_LABS/Debug
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/POSIX/ws/Posix_LABS/Debug$ ./Posix_LABS ../../Posix_LABS/
.project
.cproject

>>>Debug
main.o
main.d
objects.mk
sources.mk
subdir.mk
Posix_LABS
makefile
main.c
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/POSIX/ws/Posix_LABS/Debug$ ls
main.d  main.o  makefile  objects.mk  Posix_LABS  sources.mk  subdir.mk
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/POSIX/ws/Posix_LABS/Debug$ ls ../
Debug  main.c
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/POSIX/ws/Posix_LABS/Debug$ ls ../ -a
.  ..  .cproject  Debug  main.c  .project
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/POSIX/ws/Posix_LABS/Debug$
```



open() /read()/write() files

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

C Standard Library File I/O

- So far you've used the C standard library to access files
  - Use a provided `FILE*` *stream* abstraction
  - `fopen()`, `fread()`, `fwrite()`, `fclose()`, `fseek()`
- These are convenient and portable
  - They are buffered
  - They are implemented using lower-level OS calls

Lower-Level File Access

- Most Linux support a common set of lower-level file access APIs:
  POSIX - Portable Operating System Interface
  - `open()`, `read()`, `write()`, `close()`, `lseek()`

open()/close()

- To open a file:
  - Pass in the filename and access mode
    - Similar to `fopen()`
  - Get back a "file descriptor"
    - Similar to `FILE*` from `fopen()`, but is just an `int`
    - Defaults: `0` is `stdin`, `1` is `stdout`, `2` is `stderr`

```c
#include <fcntl.h>    // for open()
#include <unistd.h>   // for close()
  ...
  int fd = open("foo.txt", O_RDONLY);
  if (fd == -1) {
    perror("open failed");
    exit(EXIT_FAILURE);
  }
  ...
  close(fd);
```

Reading from a File

```
ssize_t read(int fd, void* buf, size_t count);
```

- Returns the number of bytes read
  - Might be fewer bytes than you requested (!!!)
  - Returns 0 if you're already at the end-of-file
  - Returns -1 on error (and sets errno)
- There are some surprising error modes (check errno)
  - EBADF: bad file descriptor
  - EFAULT: output buffer is not a valid address
  - EINTR: read was interrupted, please try again (ARGH!!!! 😭😭)
  - And many others…

example

```c
/*
 * main.c
 *
 *  Created on: Dec 10, 2019
 *      Author: Keroles Shenouda
 *      www.Learn-in-depth.com
 */
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <dirent.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

void signals_lab ();
Dir_example (int argc, char *argv[]);
void listdir(const char *pathname);
void myRead (int argc, char *argv[]);

int main (int argc, char *argv[])
{
    myRead ( argc, argv);
    return 0 ;
}
```

```c
void myRead (int argc, char *argv[])
{
    int fd = -1;
    char buffer[10] = {0};
    int ret = 0;

    if(argc != 2)
    {
        printf("Please provide the filename as argument to read\n");
        exit(EXIT_FAILURE);
    }
    fd = open("./test.txt", O_RDONLY);

    if(fd == -1)
    {
        printf("Error in opening the file error=%s\n", strerror(errno));
        exit(EXIT_FAILURE);
    }
    while (ret = read(fd, (void *)buffer, sizeof(buffer)) >0 )
    {
        printf("%s", buffer);
        memset(buffer, 0 , sizeof(buffer));
    }

    close(fd);

    return EXIT_SUCCESS;

}
```

#LEARN IN DEPTH

#Be professional in embedded system

example

test.txt (Embedded_Linux /media/embedded_system_ks/Embedded_Linux/POSIX/

Open Save

```
1 /*
2  * main.c
3  *
4  *  Created on: Dec 10, 2019
5  *      Author: Keroles Shenouda
6  *      www.Learn-in-depth.com
7  */
```

```
LABS/Debug$ gedit test.txt &
[3] 10114
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embe
LABS/Debug$ ./Posix_LABS test.txt
/*
 * main.c
 *
 *  Created on: Dec 10, 2019
 *      Author: Keroles Shenouda
 *      www.Learn-in-depth.com
 */
[3]+  Done                    gedit test.txt
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embe
LABS/Debug$
```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Write/Read Example

main.c | dirent.h | dirent.h

```c
/*
 * main.c
 *
 *  Created on: Dec 10, 2019
 *      Author: Keroles Shenouda
 *      www.Learn-in-depth.com
 */
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <dirent.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>


void myRead_write (int argc, char *argv[]);

int main (int argc, char *argv[])
{
    myRead_write ( argc, argv);
    return 0 ;
}
```

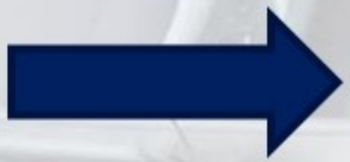
```c
void myRead_write (int argc, char *argv[])
{
    int fd_in = -1;
    int fd_out = -1;

    char buffer[10] = {0};
    int ret = 0;
    int retw = 0;
    char *INPUT_FILENAME;
    char *OUTPUT_FILENAME;

    if(argc != 3)
    {
        printf("Please provide the filename as argument to read and for writing\n");
        printf("./a.out test_read test_write\n");
        exit(EXIT_FAILURE);
    }

    INPUT_FILENAME = argv[1];
    OUTPUT_FILENAME = argv[2];

    fd_in = open(INPUT_FILENAME, O_RDONLY);

    if(fd_in == -1)
    {
        printf("Error in opening the input file  error_num=%d  %s\n", errno, strerror(errno) );
        exit(EXIT_FAILURE);
    }
     /*
     r = 4
     w = 2
     x = 1
     user: 4 + 1 = 5
     group  4
     other  4
     */
//O_RDWR read Write
//O_TRUNC emove the previouse content
//O_CREAT = create a new one if it is not exist
// 0544 rw r r
     fd_out = open(OUTPUT_FILENAME, O_RDWR | O_TRUNC | O_CREAT,0544);

     if(fd_out == -1)
     {
         printf("Error in opening the output file  error_num=%d  %s\n", errno, strerror(errno) );
         close(fd_in); //Note we are closing the handle of fd_in as it was already openened
         exit(EXIT_FAILURE);
     }

     while ((ret = read(fd_in, (void *)buffer, sizeof(buffer)-1 )) >0 )
     {
         retw = write(fd_out, buffer, ret);

         if(retw != ret)
         {
             printf("Error in writing complete data. error=%s\n", strerror(errno));
                 close(fd_out);
                 close(fd_in);
             exit(0);
         }
         memset(buffer, 0 , sizeof(buffer));
     }

     close(fd_in);
     close(fd_out);

     return EXIT_SUCCESS;

}
```

Write/Read Example

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/POSIX/ws/Posix_
LABS/Debug$ ./Posix_LABS test.txt  new_one.txt
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/POSIX/ws/Posix_
```

test.txt

```
1 /*
2  * main.c
3  *
4  *  Created on: Dec 10, 2019
5  *      Author: Keroles Shenouda
6  *      www.Learn-in-depth.com
7  */
```



new_one.txt [Read-Only] (Embedded_Linux /

Open

```
1 /*
2  * main.c
3  *
4  *  Created on: Dec 10, 2019
5  *      Author: Keroles Shenouda
6  *      www.Learn-in-depth.com
7  */
```

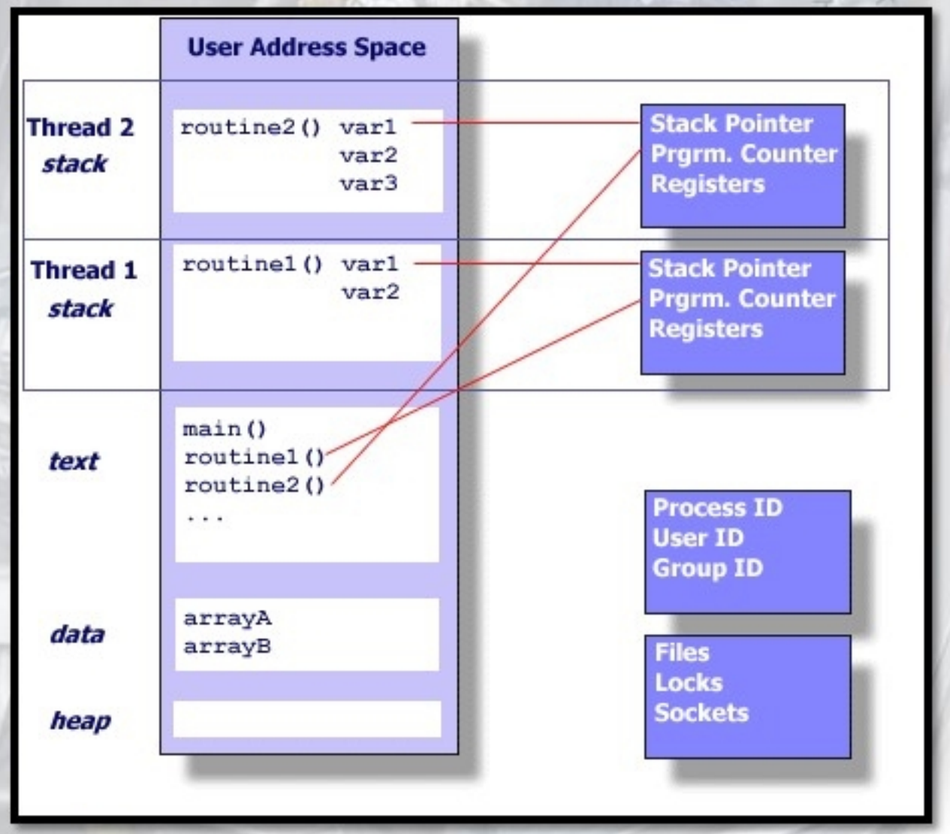
Headers

- Linux system programming revolves around a handful of headers.
- Both the kernel itself and *glibc* provide the headers used in system-level programming.
- These headers include the standard C (for example, <string.h>).

The pthread library

- In Linux, when a **new process is created**, it already contains **a thread, used to execute the main() function**
- A process is created by **the operating system**
- Additional threads can be created using the **pthread** library, which is part of the C library
- all threads inside a given process will share the same address space, the same set of open files, etc
- The pthread library also provide thread synchronization primitives: mutexes and conditions
- This pthread library has its own header : pthread.h
- Applications using pthread function calls should be explicitly

linked with the pthread library **gcc -o app app.c -l pthread**



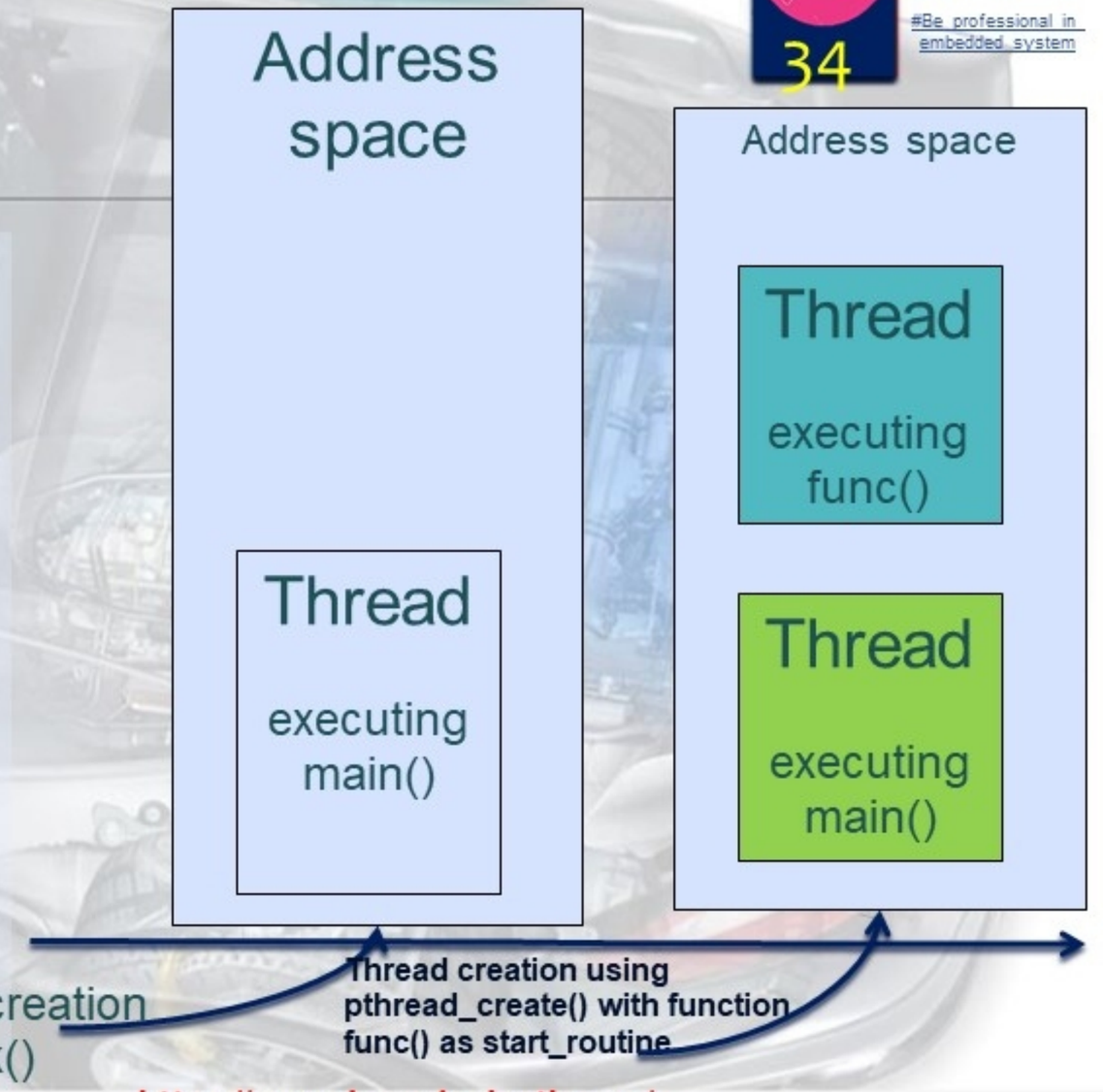
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Creating a new thread

- The function to create a new thread is pthread_create()

```
int pthread_create(pthread_t * thread,
pthread_attr_t * attr,
void *(*start_routine)(void *),
void * arg);
```

- thread is a pointer to a pthread_t structure that will be initialized by the function. Later, this structure can be used to reference the thread.
- Attr is a pointer to an optional structure pthread_attr_t. This structure can be manipulated using pthread_attr_*() functions. It can be used to set various attributes of the threads (detach policy, scheduling policy, etc.)
- start_routine is the function that will be executed by the thread
- arg is the private data passed as argument to the start_routine function



Lab: what is the output for the two cases

main.c | subdir.mk

```c
/*
 * main.c
 *
 *  Created on: Nov 10, 2019
 *      Author: Keroles Shenouda
 */

////////////////////////////
#include <pthread.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <sched.h>

void *thread(void *data)
{
    while(1) {
        printf(" Hello world from thread \n");
    }
}

int main(void) {
    pthread_t th;
    pthread_create(& th, NULL, thread, NULL);
//  while(1) ;
    return 0;
}
```

```
embedded_system_ks@embedded-KS: /media/embedded
embedded_system_ks@embedded-KS:/media/embedded_s
first_term/ws/example_1$ gcc main.c -lpthread
embedded_system_ks@embedded-KS:/media/embedded_s
```

main.c | subdir.mk

```c
/*
 * main.c
 *
 *  Created on: Nov 10, 2019
 *      Author: Keroles Shenouda
 */

////////////////////////////
#include <pthread.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <sched.h>

void *thread(void *data)
{
    while(1) {
        printf(" Hello world from thread \n");
    }
}

int main(void) {
    pthread_t th;
    pthread_create(& th, NULL, thread, NULL);
    while(1) ;
    return 0;
}
```

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Lab: what is the output for the two cases

main.c | subdir.mk

```c
/*
 * main.c
 *
 *  Created on: Nov 10, 2019
 *      Author: Keroles Shenouda
 */

////////////////////////////
#include <pthread.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <sched.h>

void *thread(void *data)
{
    while(1) {
        printf(" Hello world from thread \n");
    }
}

int main(void) {
    pthread_t th;
    pthread_create(& th, NULL, thread, NULL);
//  while(1) ;
    return 0;
}
```

```
first_term/ws/example_1$ ./a.out
embedded_system_ks@embedded-KS:/media/embedded_s
first_term/ws/example_1$
```

main.c | subdir.mk

```c
/*
 * main.c
 *
 *  Created on: Nov 10, 2019
 *      Author: Keroles Shenouda
 */

////////////////////////////
#include <pthread.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <sched.h>

void *thread(void *data)
{
    while(1) {
        printf(" Hello world from thread \n");
    }
}

int main(void) {
    pthread_t th;
    pthread_create(& th, NULL, thread, NULL);
    while(1) ;
    return 0;
}
```

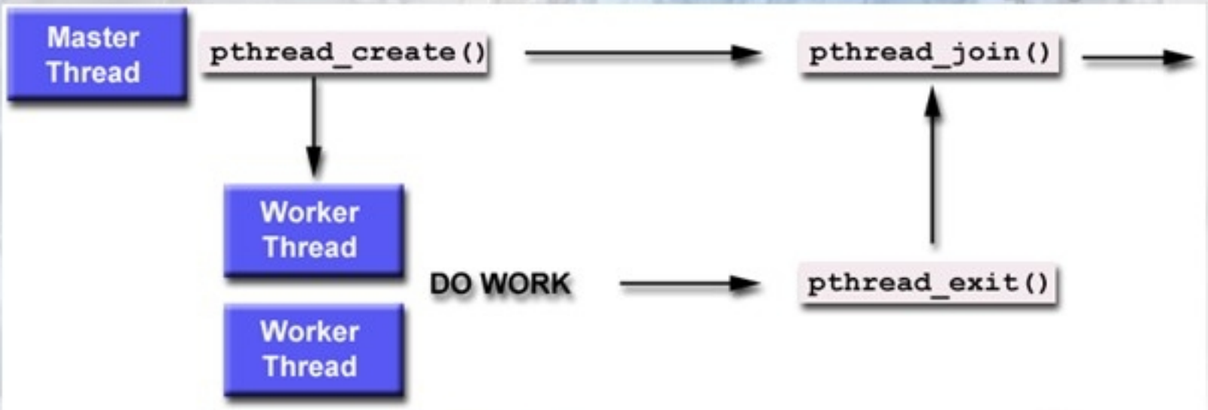
embedded_system_ks@embedded-

```
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
```

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Joinable and detached threads

- When the main() function exits, all threads of the application are **destroyed**
- "Joining" is one way to accomplish synchronization between threads.
- The **pthread_join()** function call can be used to **suspend** the **execution** of a **thread until another thread terminates**
  - This function **must** be called **in order to** ***release*** the ressources used by the thread, **otherwise it remains as** ***zombie***.



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

#LEARN IN DEPTH

#Be professional in embedded system

Thread join, code sample

main.c subdir.mk

```c
/*
 * main.c
 *
 *  Created on: Nov 10, 2019
 *      Author: Keroles Shenouda
 */

////////////////////////////
#include <pthread.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <sched.h>

void *thread(void *data)
{
    int i;
    for (i = 0; i < 3; i++) {
        printf(" Hello world from thread \n");
    }
}

int main(void) {
    pthread_t th;
    pthread_create(&th, NULL, thread, NULL);
    pthread_join(th, NULL);

    return 0;
}
```

```
embedded_system_ks@embedded-KS: /media/embedded_sys
embedded_system_ks@embedded-KS:/media/embedded_syste
first_term/ws/example_1$ gcc main.c -lpthread
embedded_system_ks@embedded-KS:/media/embedded_syste
first_term/ws/example_1$ ./a.out
 Hello world from thread
 Hello world from thread
 Hello world from thread
embedded_system_ks@embedded-KS:/media/embedded_syste
first_term/ws/example_1$
```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

detached threads

- Threads can also be detached, in which case they become independent.
- <u>The parent thread doesn't need to wait</u>: the tid storage is reclaimed when the thread is done.
  - Mainly to save space.

```
int pthread_detach(pthread_t tid)
```

Detaching self :

```
pthread_detach(pthread_self())
```

main.c ⊠ subdir.mk

```c
/*
 * main.c
 *
 *  Created on: Nov 10, 2019
 *      Author: Keroles Shenouda
 */

/////////////////////////////
#include <pthread.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <sched.h>

void *thread(void *data)
{
    int i;
    for (; ; ) {
        printf(" Hello world from thread \n");
        //The parent thread doesn't need to wait
        pthread_detach(pthread_self());

    }
}

int main(void) {
    pthread_t th;
    pthread_create(&th, NULL, thread, NULL);
    sleep(1);
    pthread_join(th, NULL);

    return 0;
}
```

```
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
Hello world from thread
embedded_system_ks@embedded-KS:/me
first_term/ws/example_1$
```

Thread cancelation

- It is also possible to cancel a thread from another thread using the pthread_cancel() function, passing the pthread_t structure of the thread to cancel.

What will be the output ?
Think in depth ☺

```c
////////////////////////////////
/*
 * main.c
 *
 *  Created on: Nov 10, 2019
 *      Author: Keroles Shenouda
 */
////////////////////////////////
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>
#include <pthread.h>

void *thread()
{
        while (1)
        { printf ("thread Hello \n");
        }


}
int  main(int argc, char **argv)
{
        int s , res ;
        pthread_t th ;
        pthread_create(&th , NULL , thread , NULL);
        sleep (1);   /* Give thread a chance to get started */
        printf("main(): sending cancellation request\n");
        s = pthread_cancel(th);
        if (s != 0)
                perror ("pthread_cancel");
        /* Join with thread to see what its exit status was */
        //s = pthread_join(th, &res);
        //if (res == PTHREAD_CANCELED)
                //printf("main(): thread was canceled\n");

        while (1) ;
        return 0 ;

}

```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Thread cancelation

```c
////////////////////////////////

/*
 * main.c
 *
 *  Created on: Nov 10, 2019
 *      Author: Keroles Shenouda
 */
////////////////////////////////

#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>
#include <pthread.h>

void *thread()
{
        while (1)
        { printf ("thread Hello \n");
        }


}
int  main(int argc, char **argv)
{
        int s , res ;
        pthread_t th ;
        pthread_create(&th , NULL , thread , NULL);
        sleep (1);   /* Give thread a chance to get started */
        printf("main(): sending cancellation request\n");
        s = pthread_cancel(th);
        if (s != 0)
                perror ("pthread_cancel");
        /* Join with thread to see what its exit status was */
        //s = pthread_join(th, &res);
        //if (res == PTHREAD_CANCELED)
                //printf("main(): thread was canceled\n");

        while (1) ;
        return 0 ;

}

```

```
embedded_system_ks@embedded-KS: ~
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

What will be the output ?
Think in depth ☺

```c
////////////////////////////////

/*
 * main.c
 *
 *  Created on: Nov 10, 2019
 *      Author: Keroles Shenouda
 */
////////////////////////////////

#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>
#include <pthread.h>

void *thread()
{
        while (1)
        { printf ("thread Hello \n");
        }


}
int  main(int argc, char **argv)
{
        int s , res ;
        pthread_t th ;
        pthread_create(&th , NULL , thread , NULL);
        sleep (1);   /* Give thread a chance to get started */
        printf("main(): sending cancellation request\n");
        s = pthread_cancel(th);
        if (s != 0)
                perror ("pthread_cancel");
        /* Join with thread to see what its exit status was */
        s = pthread_join(th, &res);
        if (res == PTHREAD_CANCELED)
                printf("main(): thread was canceled\n");

        while (1) ;
        return 0 ;

}
```


```c
////////////////////////////////

/*
 * main.c
 *
 *  Created on: Nov 10, 2019
 *      Author: Keroles Shenouda
 */
////////////////////////////////

#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>
#include <pthread.h>

void *thread()
{
        while (1)
        { printf ("thread Hello \n");
        }


}
int  main(int argc, char **argv)
{
        int s , res ;
        pthread_t th ;
        pthread_create(&th , NULL , thread , NULL);
        sleep (1);   /* Give thread a chance to get started */
        printf("main(): sending cancellation request\n");
        s = pthread_cancel(th);
        if (s != 0)
                perror ("pthread_cancel");
        /* Join with thread to see what its exit status was */
        s = pthread_join(th, &res);
        if (res == PTHREAD_CANCELED)
                printf("main(): thread was canceled\n");

        while (1) ;
        return 0 ;

}
```

```
embedded_system_ks@embedded-KS: ~
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
thread Hello
main(): thread was canceled
```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

pthread mutexes

- The pthread library provides a mutual exclusion primitive, the **pthread_mutex**.
- Declaration and initialization of a pthread mutex
  - pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
  - **Or: at run time**
    - pthread_mutex_t lock;
    - pthread_mutex_init(& lock, NULL);
    - pthread_mutex_destroy(& lock);
- The second argument to pthread_mutex_init() is a set of mutexspecific attributes, in the form of a pthread_mutexattr_t structure that can be initialized and manipulated using pthread_mutexattr_*() functions.

Mutex locks: lock

- pthread_mutex_lock(pthread_mutex_t *mutex);
  - Tries to acquire the lock specified by mutex
  - If mutex is already locked, then the calling thread blocks until mutex is **unlocked**.

Mutex locks: unlock

- pthread_mutex_unlock(pthread_mutex_t *mutex);
  - If the calling thread has mutex currently locked, this will unlock the mutex.
  - If other threads are blocked waiting on this mutex, **one will unblock and acquire mutex.**
- **Which one is determined by the scheduler**

Mutex example

What will be the output ?
Think in depth ☺

main.c | clone() at clone.S:94 0xb7eca894

```
////////////////////////////////
/*
 * main.c
 *
 *  Created on: Nov 10, 2019
 *      Author: Keroles Shenouda
 */
////////////////////////////////
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
pthread_mutex_t fill_mutex;
int arr[10];
int flag=0;
pthread_cond_t cond_var=PTHREAD_COND_INITIALIZER;
void *fill() {
    int i=0;
    for(i=0;i<4;i++) {
        pthread_mutex_lock(&fill_mutex);
        arr[i] = i ;
        pthread_mutex_unlock(&fill_mutex);
        sleep (1);
    }
}

void *read() {
    int i=0;



    printf("Values filled in array are");
    for(i=0;i<4;i++) {
        sleep (1);
        pthread_mutex_lock(&fill_mutex);
        printf("\n %d \n",arr[i]);
        pthread_mutex_unlock(&fill_mutex);

    }
}

int main() {

    pid_t x ;
    pthread_t thread_id,thread_id1;
    int ret;
    void *res;
    ret=pthread_create(&thread_id,NULL,&fill,NULL);
    ret=pthread_create(&thread_id1,NULL,&read,NULL);
    printf("\n Created threads");
    pthread_join(thread_id,&res);
    pthread_join(thread_id1,&res);
    return 0 ;
}
```

Mutex example

What will be the output ?
Think in depth ☺

```
Created threadsValues filled in array are
0

1

2

3
```

main.c | clone() at clone.S:94 0xb7eca894

```
////////////////////////////////
/*
 * main.c
 *
 *  Created on: Nov 10, 2019
 *      Author: Keroles Shenouda
 */
////////////////////////////////
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
pthread_mutex_t fill_mutex;
int arr[10];
int flag=0;
pthread_cond_t cond_var=PTHREAD_COND_INITIALIZER;
void *fill() {
    int i=0;
    for(i=0;i<4;i++) {
        pthread_mutex_lock(&fill_mutex);
        arr[i] = i ;
        pthread_mutex_unlock(&fill_mutex);
        sleep (1);
    }
}

void *read() {
    int i=0;



    printf("Values filled in array are");
    for(i=0;i<4;i++) {
        sleep (1);
        pthread_mutex_lock(&fill_mutex);
        printf("\n %d \n",arr[i]);
        pthread_mutex_unlock(&fill_mutex);

    }
}

int main() {

    pid_t x ;
    pthread_t thread_id,thread_id1;
    int ret;
    void *res;
    ret=pthread_create(&thread_id,NULL,&fill,NULL);
    ret=pthread_create(&thread_id1,NULL,&read,NULL);
    printf("\n Created threads");
    pthread_join(thread_id,&res);
    pthread_join(thread_id1,&res);
    return 0 ;
}
```

#LEARN IN DEPTH

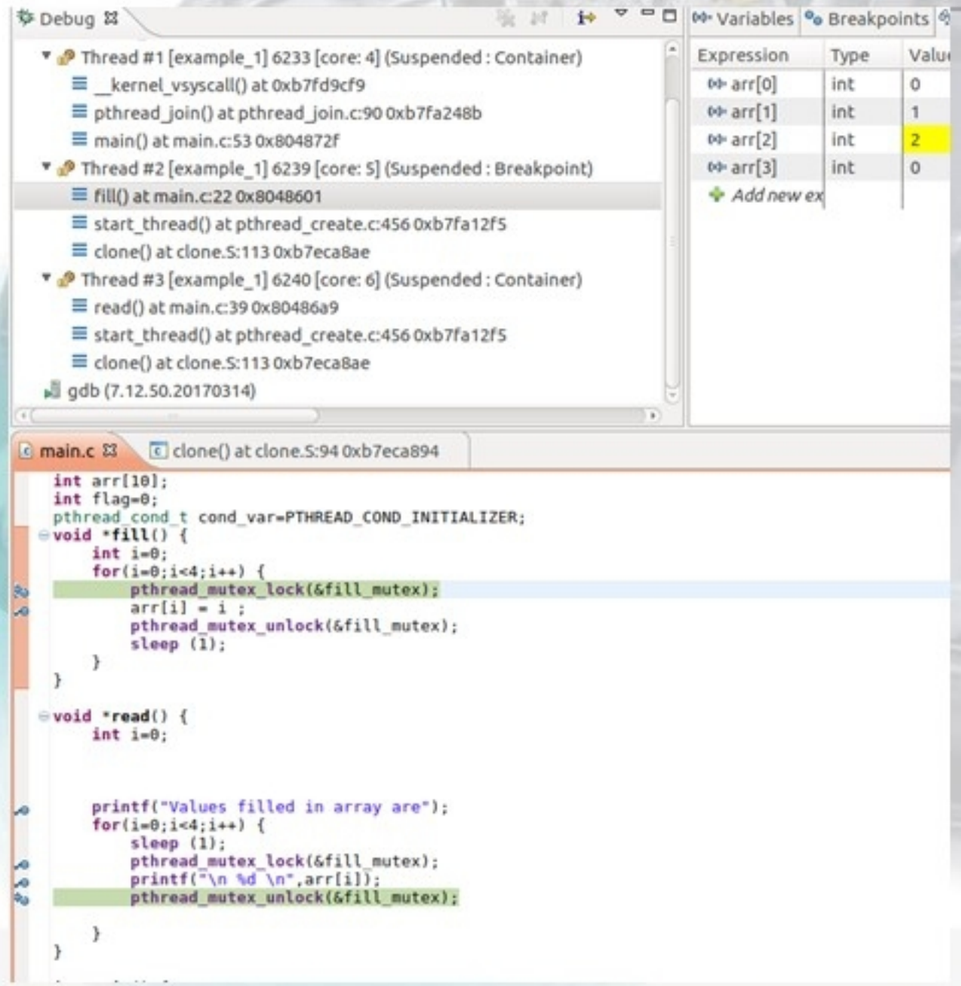
#Be professional in embedded system

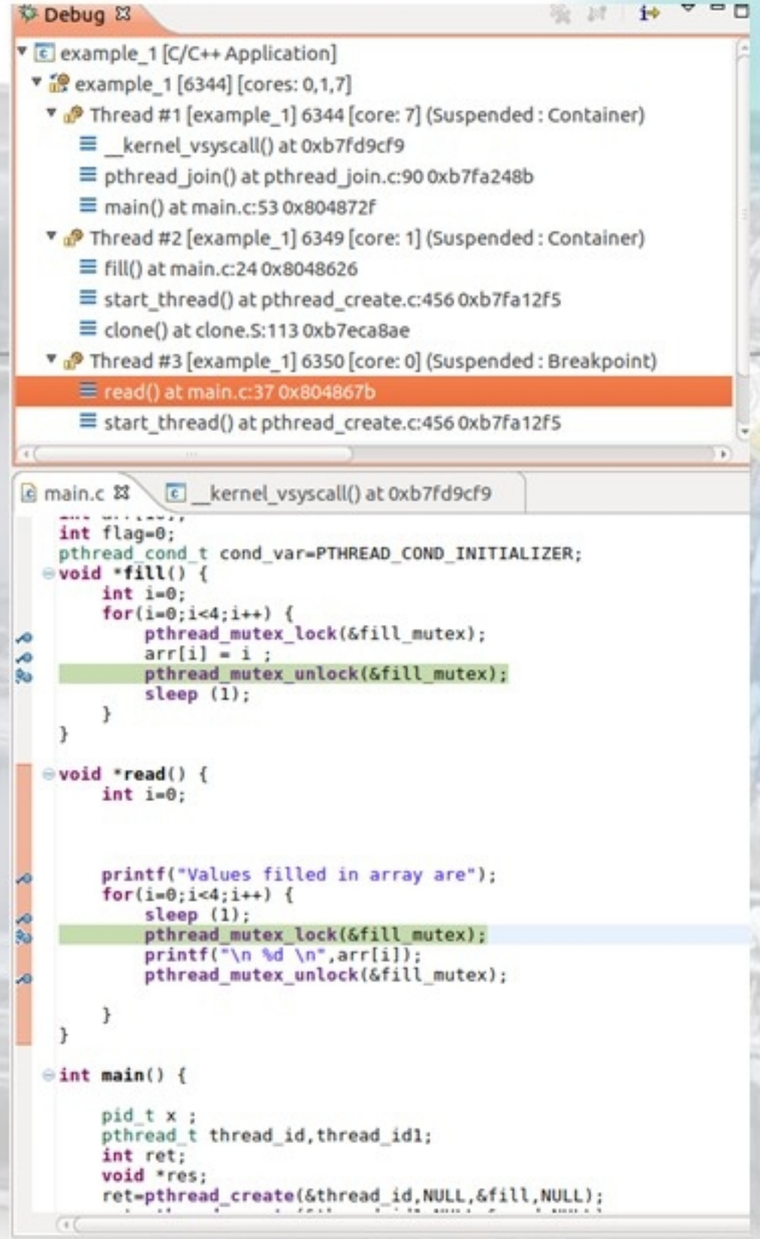
Learn-in-depth

Why three threads ☺ ?

Mutex example

Mutex example





https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

```c
////////////////////////////////

/*
 * main.c
 *
 *  Created on: Nov 10, 2019
 *      Author: Keroles Shenouda
 */
////////////////////////////////
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
pthread_mutex_t fill_mutex;
int arr[10], iii;
int flag=0;
pthread_cond_t cond_var=PTHREAD_COND_INITIALIZER;
void *fill() {
        int i=0;
        for(i=0;i<4;i++) {
                pthread_mutex_lock(&fill_mutex);
                iii = i ;
                pthread_mutex_unlock(&fill_mutex);
                sleep (1);
        }
}

void *read() {
        int i=0;



        printf("Values filled in array are");
        for(i=0;i<4;i++) {
                sleep (1);
                pthread_mutex_lock(&fill_mutex);
                printf("\n %d \n",iii);
                pthread_mutex_unlock(&fill_mutex);

        }
}

int main() {

        pid_t x ;
        pthread_t thread_id,thread_id1;
        int ret;
        void *res;
        ret=pthread_create(&thread_id,NULL,&fill,NULL);
        ret=pthread_create(&thread_id1,NULL,&read,NULL);
        printf("\n Created threads");
        pthread_join(thread_id,&res);
        pthread_join(thread_id1,&res);
        return 0 ;
}

```


#LEARN IN DEPTH

#Be professional in embedded system

What will be the output ?

Think in depth ☺

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

```c
////////////////////////////////

/*
 * main.c
 *
 *  Created on: Nov 10, 2019
 *      Author: Keroles Shenouda
 */
////////////////////////////////
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
pthread_mutex_t fill_mutex;
int arr[10], iii;
int flag=0;
pthread_cond_t cond_var=PTHREAD_COND_INITIALIZER;
void *fill() {
        int i=0;
        for(i=0;i<4;i++) {
                pthread_mutex_lock(&fill_mutex);
                iii = i ;
                pthread_mutex_unlock(&fill_mutex);
                sleep (1);
        }
}

void *read() {
        int i=0;



        printf("Values filled in array are");
        for(i=0;i<4;i++) {
                sleep (1);
                pthread_mutex_lock(&fill_mutex);
                printf("\n %d \n",iii);
                pthread_mutex_unlock(&fill_mutex);

        }
}

int main() {

        pid_t x ;
        pthread_t thread_id,thread_id1;
        int ret;
        void *res;
        ret=pthread_create(&thread_id,NULL,&fill,NULL);
        ret=pthread_create(&thread_id1,NULL,&read,NULL);
        printf("\n Created threads");
        pthread_join(thread_id,&res);
        pthread_join(thread_id1,&res);
        return 0 ;
}

```

```
embedded_system_ks@embedded-KS:~$ ./a.out

 Created threadsValues filled in array are
 0

 1

 2

 3
embedded_system_ks@embedded-KS:~$ clear
```

pthread conditions

- Conditions can be used to **suspend** a thread until a condition becomes true, as signaled by another thread.
- Initialization:
  - pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
  - Or at runtime:
    - pthread_cond_t cond;
      pthread_cond_init(& cond, NULL);
- Wait for the condition pthread_cond_wait(& cond, & mutex)
  The mutex will be released before waiting and taken again after the wait
- Signaling the condition
  - To one thread waiting, pthread_cond_signal(& cond);
  - To all threads waiting, pthread_cond_broadcast(& cond);

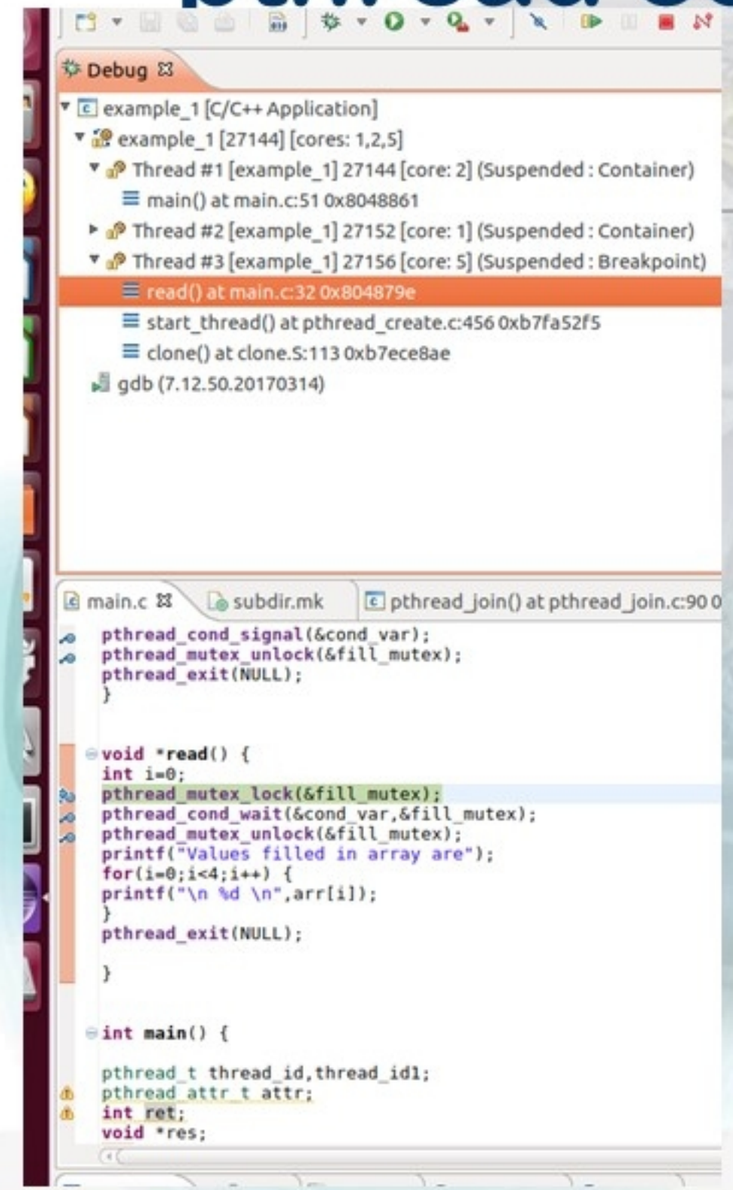
pthread conditions example

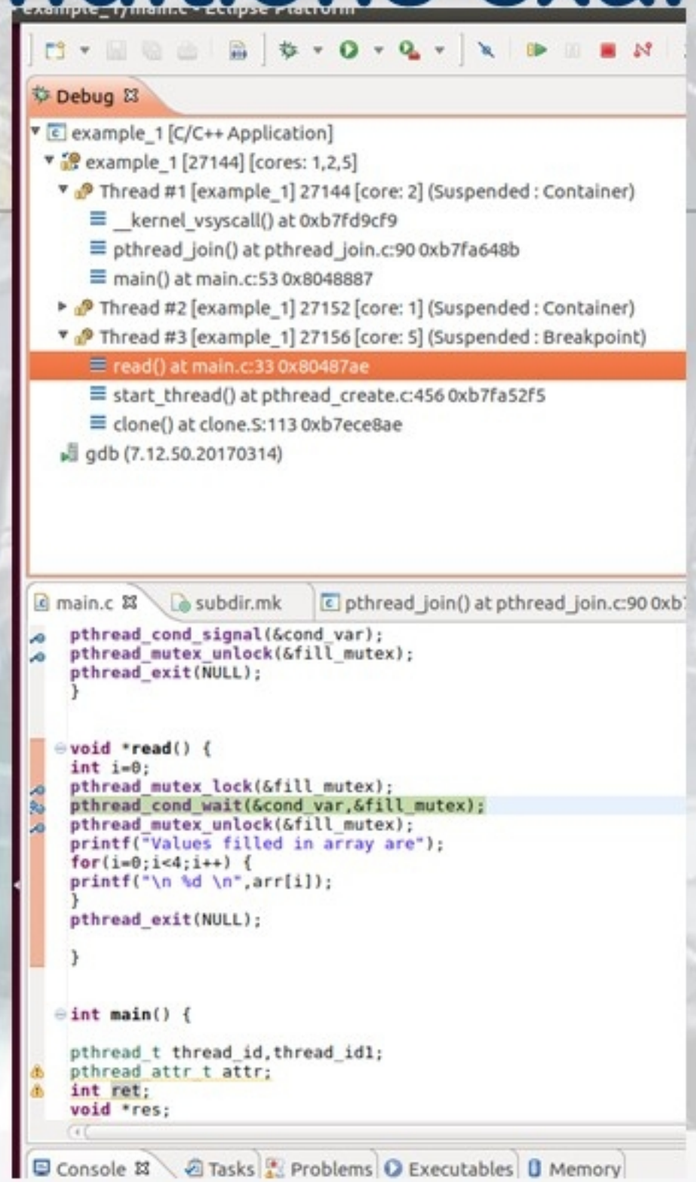
main.c | subdir.mk | __kernel_vsyscall() at 0xb7fd9cf9

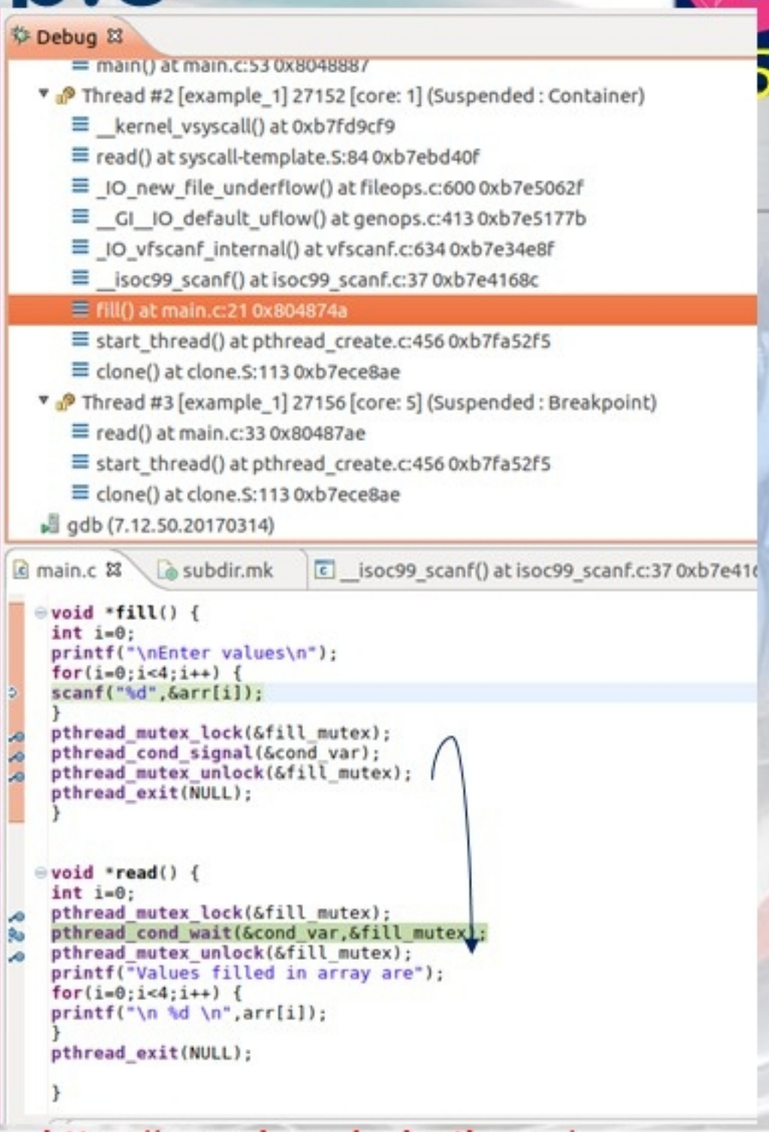
```c
/*
 * main.c
 *
 *  Created on: Nov 10, 2019
 *      Author: Keroles Shenouda
 */
////////////////////////////
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
pthread_mutex_t fill_mutex;
int arr[10];
int flag=0;
pthread_cond_t cond_var=PTHREAD_COND_INITIALIZER;
void *fill() {
int i=0;
printf("\nEnter values\n");
for(i=0;i<4;i++) {
scanf("%d",&arr[i]);
}
pthread_mutex_lock(&fill_mutex);
pthread_cond_signal(&cond_var);
pthread_mutex_unlock(&fill_mutex);
pthread_exit(NULL);
}

void *read() {
int i=0;
pthread_mutex_lock(&fill_mutex);
pthread_cond_wait(&cond_var,&fill_mutex);
pthread_mutex_unlock(&fill_mutex);
printf("Values filled in array are");
for(i=0;i<4;i++) {
printf("\n %d \n",arr[i]);
}
pthread_exit(NULL);}

int main() {

pthread_t thread_id,thread_id1;
pthread_attr_t attr;
int ret;
void *res;
ret=pthread_create(&thread_id,NULL,&fill,NULL);
ret=pthread_create(&thread_id1,NULL,&read,NULL);
printf("\n Created threads");
pthread_join(thread_id,&res);
pthread_join(thread_id1,&res);
return 0 ;
}
```

```
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_ided_
Linux/Embedded_Linux/first_term/ws/example_1$ gcc main.c -lpthread ıread
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_ided_
Linux/Embedded_Linux/first_term/ws/example_1$ ./a.out

 Created threads
Enter values
1
2
3
4
Values filled in array are
 1

 2

 3

 4
```

pthread conditions example

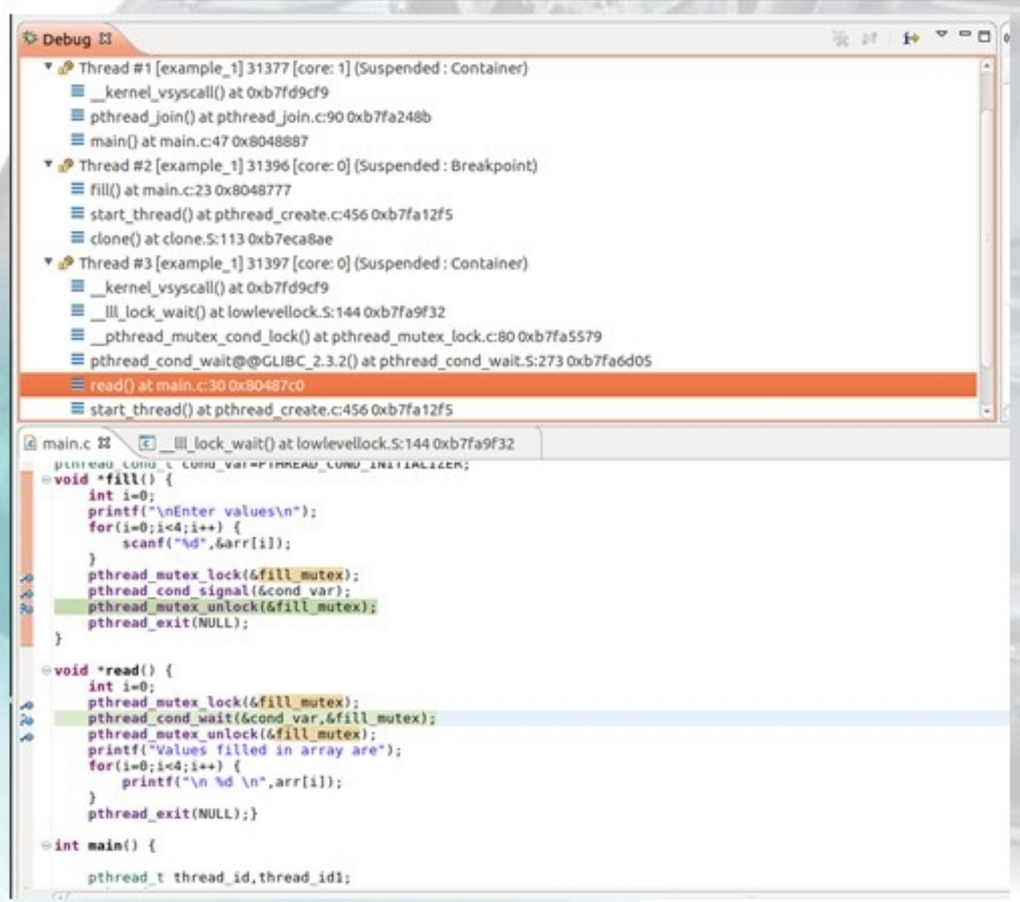


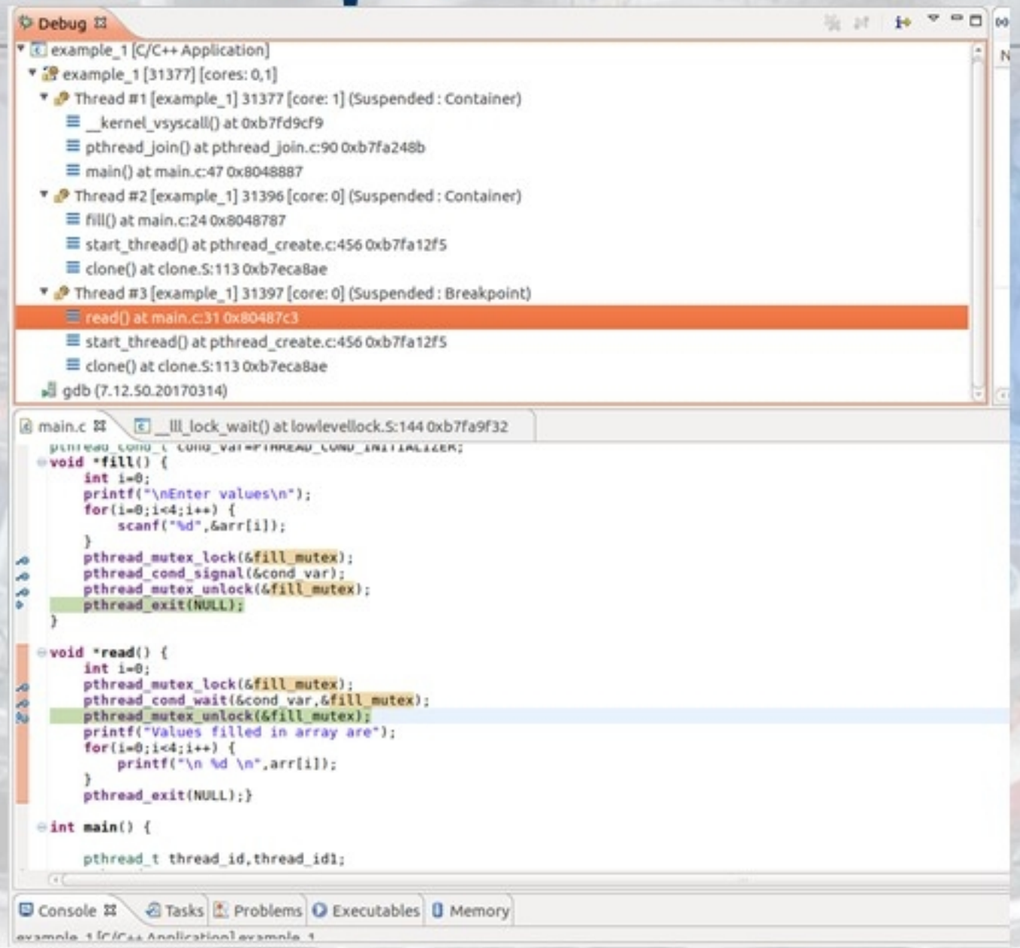




https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

pthread conditions example





https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Socket Programming

CLIENT-SERVER COMMUNICATION

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

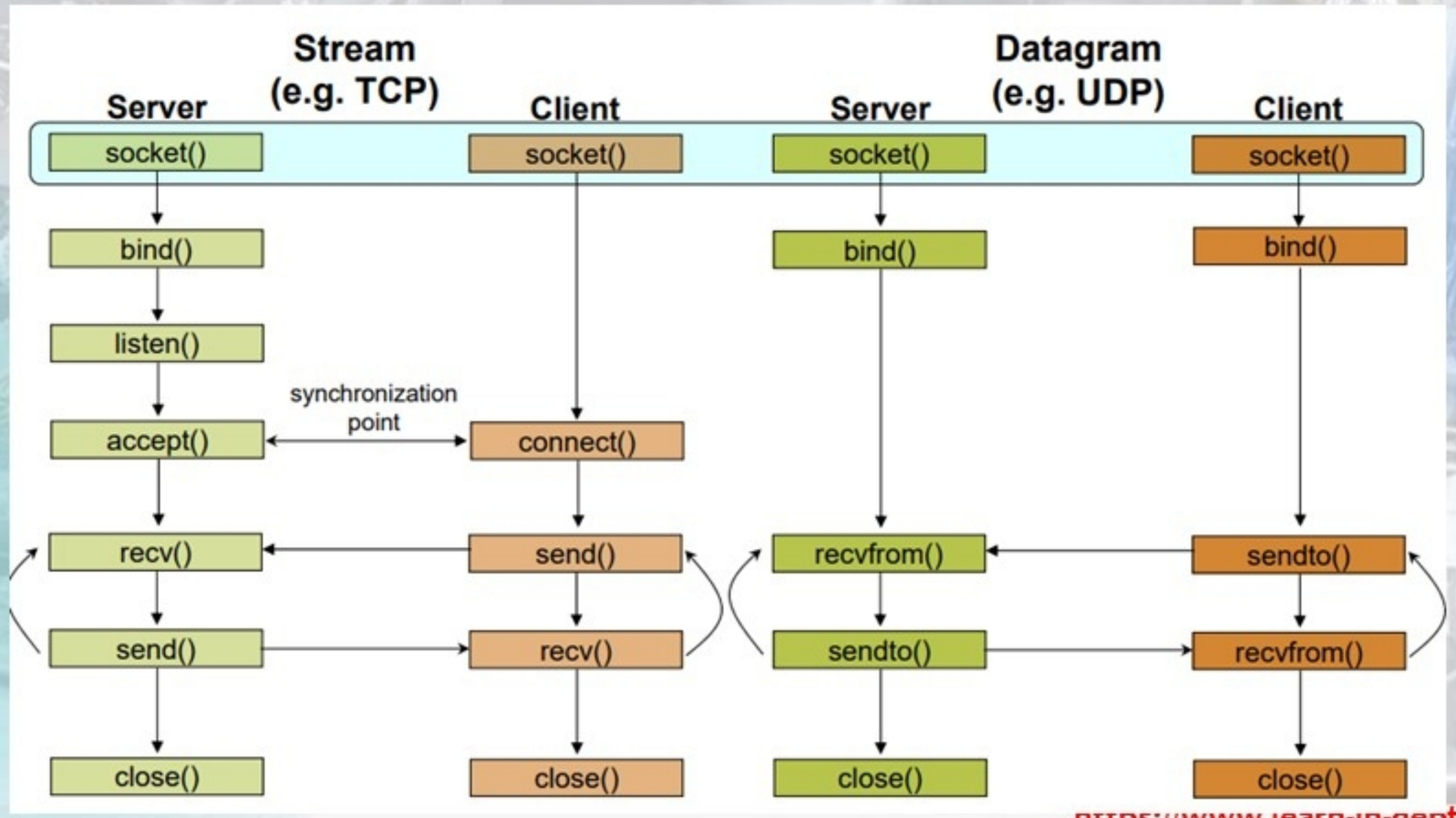
Client-Server communication

- Server
  - passively waits for and responds to clients
  - passive socket
- Client
  - initiates the communication
  - must know the address and the port of the server
  - active socket

Sockets - Procedures

Primitive	Meaning
Socket	Create a new communication endpoint
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection

Client - Server Communication - Unix



Socket creation in C: socket()

- int sockid = socket(family, type, protocol);
  - sockid: socket descriptor, an integer (like a file-handle)
    - family: integer, communication domain, e.g.,
      - PF_INET, IPv4 protocols, Internet addresses (typically used)
      - PF_UNIX, Local communication, File addresses
  - type: communication type
    - SOCK_STREAM - reliable, 2-way, connection-based service
    - SOCK_DGRAM - unreliable, connectionless, messages of maximum length
  - protocol: specifies protocol
    - IPPROTO_TCP IPPROTO_UDP
  - upon failure returns -1

Socket close in C: close()

- status = close(sockid);
  - sockid: the file descriptor (socket being closed)
  - status: 0 if successful, -1 if error
- Closing a socket
  - closes a connection (for stream socket)
  - frees up the port used by the socket

Assign address to socket: bind()

- int status = bind(sockid, &addrport, size);
  - sockid: integer, socket descriptor
  - addrport: struct sockaddr, the (IP) address and port of the machine
    - for TCP/IP server, internet address is usually set to INADDR_ANY, i.e., chooses any incoming interface
  - size: the size (in bytes) of the addrport structure
  - status: upon failure -1 is returned

struct sockaddr_in

```
struct sockaddr_in {
    unsigned short sin_family;     /* Internet protocol (AF_INET) */
    unsigned short sin_port;       /* Address port (16 bits) */
    struct in_addr sin_addr;       /* Internet address (32 bits) */
    char sin_zero[8];              /* Not used */
}
```

bind()- Example with TCP

```
int sockid;
struct sockaddr_in addrport;
sockid = socket(PF_INET, SOCK_STREAM, 0);

addrport.sin_family = AF_INET;
addrport.sin_port = htons(5100);
addrport.sin_addr.s_addr = htonl(INADDR_ANY);
if(bind(sockid, (struct sockaddr *) &addrport, sizeof(addrport))!= -1) {
   …}
```

listen for connections

- int status = listen(sockid, queueLimit);
  - sockid: integer, socket descriptor
  - queuelen: integer, # of active participants that can "wait" for a connection
  - status: 0 if listening, -1 if error
  - listen() is non-blocking: returns immediately

Establish Connection: connect() calling from the client

- The client establishes a connection with the server by calling connect()
- int status = connect(sockid, &foreignAddr, addrlen);
  - sockid: integer, socket to be used in connection
  - foreignAddr: struct sockaddr: address of the passive participant
  - addrlen: integer, sizeof(name)
  - status: 0 if successful connect, -1 otherwise
- connect() is blocking

Incoming Connection: accept()

- The server gets a socket for an incoming client connection by calling accept()
- int s = accept(sockid, &clientAddr, &addrLen);
- s: integer, the new socket (used for data-transfer)
- sockid: integer, the orig. socket (being listened on)
- clientAddr: struct sockaddr, address of the active participant
  - filled in upon return
  - addrLen: sizeof(clientAddr): value/result parameter
- accept() is blocking: waits for connection before returning

Exchanging data with stream socket

- int count = send(sockid, msg, msgLen, flags);
  - msg: const void[], message to be transmitted
  - msgLen: integer, length of message (in bytes) to transmit
  - flags: integer, special options, usually just 0
  - count: # bytes transmitted (-1 if error)
- int count = recv(sockid, recvBuf, bufLen, flags);
  - recvBuf: void[], stores received bytes
  - bufLen: # bytes received
  - flags: integer, special options, usually just 0
  - count: # bytes received (-1 if error)
- Calls are blocking returns only after data is sent / received

Server-client example lab

Server-client example

```
embedded_system_ks@embedded-KS: /media/embedded_system_ks/Embedded_Linux/POSIX/ws/server/Debu
This is from thread i=18
this is from main or parent thread j=17
This is from thread i=19
this is from main or parent thread j=18
this is from main or parent thread j=19
This is from thread i=20
This is from thread i=21
this is from main or parent thread j=20
this is from main or parent thread j=21
This is from thread i=22
this is from main or parent thread j=22
This is from thread i=23
this is from main or parent thread j=23
This is from thread i=24
this is from main or parent thread j=24
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/POSIX/ws/Posix_
LABS/Debug$ cd ../../
client/     .metadata/  Posix_LABS/ server/
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/POSIX/ws/Posix_
LABS/Debug$ cd ../../
client/     .metadata/  Posix_LABS/ server/
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/POSIX/ws/Posix_
LABS/Debug$ cd ../../server/
Debug/      .settings/
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/POSIX/ws/Posix_
LABS/Debug$ cd ../../server/Debug/
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/POSIX/ws/server
/Debug$ ./server
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/POSIX/ws/server
/Debug$ ^C
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/POSIX/ws/server
/Debug$ ./server
^C
embedded_system_ks@embedded-KS:/media/embedded_system_ks/Embedded_Linux/POSIX/ws/server
/Debug$ ./server
```

```
embedded_system_ks@embedded-KS: /media/embedd
00000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000
^C
embedded_system_ks@embedded-KS:/media/embedded_
/Debug$ ./client 192.168.1.10
Tue Dec 10 08:10:49 20190
Tue Dec 10 08:10:50 20190
Tue Dec 10 08:10:51 20190
Tue Dec 10 08:10:52 20190
Tue Dec 10 08:10:53 20190
Tue Dec 10 08:10:54 20190
Tue Dec 10 08:10:55 20190
Tue Dec 10 08:10:56 20190
Tue Dec 10 08:10:57 20190
Tue Dec 10 08:10:58 20190
Tue Dec 10 08:10:59 20190
Tue Dec 10 08:11:00 20190
Tue Dec 10 08:11:01 20190
Tue Dec 10 08:11:02 20190
Tue Dec 10 08:11:03 20190
Tue Dec 10 08:11:04 20190
Tue Dec 10 08:11:05 20190
Tue Dec 10 08:11:06 20190
Tue Dec 10 08:11:07 20190
Tue Dec 10 08:11:08 20190
Tue Dec 10 08:11:09 20190
Tue Dec 10 08:11:10 20190
Tue Dec 10 08:11:11 20190
Tue Dec 10 08:11:12 20190
Tue Dec 10 08:11:13 20190
Tue Dec 10 08:11:14 20190
Tue Dec 10 08:11:15 20190
Tue Dec 10 08:11:16 20190
Tue Dec 10 08:11:17 20190
```

Server.c


```c
/*
 * server.c
 *
 *  Created on: Dec 10, 2019
 *      Author: Keroles Shenouda
 */


#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <time.h>

int main(int argc, char *argv[])
{
    int listenfd = 0, connfd = 0;
    struct sockaddr_in serv_addr;

    char sendBuff[25];
    time_t ticks;

    listenfd = socket(AF_INET, SOCK_STREAM, 0);
    memset(&serv_addr, '0', sizeof(serv_addr));
    memset(sendBuff, '0', sizeof(sendBuff));

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_addr.sin_port = htons(5000);

    bind(listenfd, (struct sockaddr*)&serv_addr, sizeof(serv_addr));

    listen(listenfd, 10);

    connfd = accept(listenfd, (struct sockaddr*)NULL, NULL);

    while(1)
    {
        ticks = time(NULL);
        snprintf(sendBuff, sizeof(sendBuff), "%.24s\r\n", ctime(&ticks));
        write(connfd, sendBuff, strlen(sendBuff));

        //close(connfd);
        sleep(1);
     }
}
```

#LEARN IN DEPTH

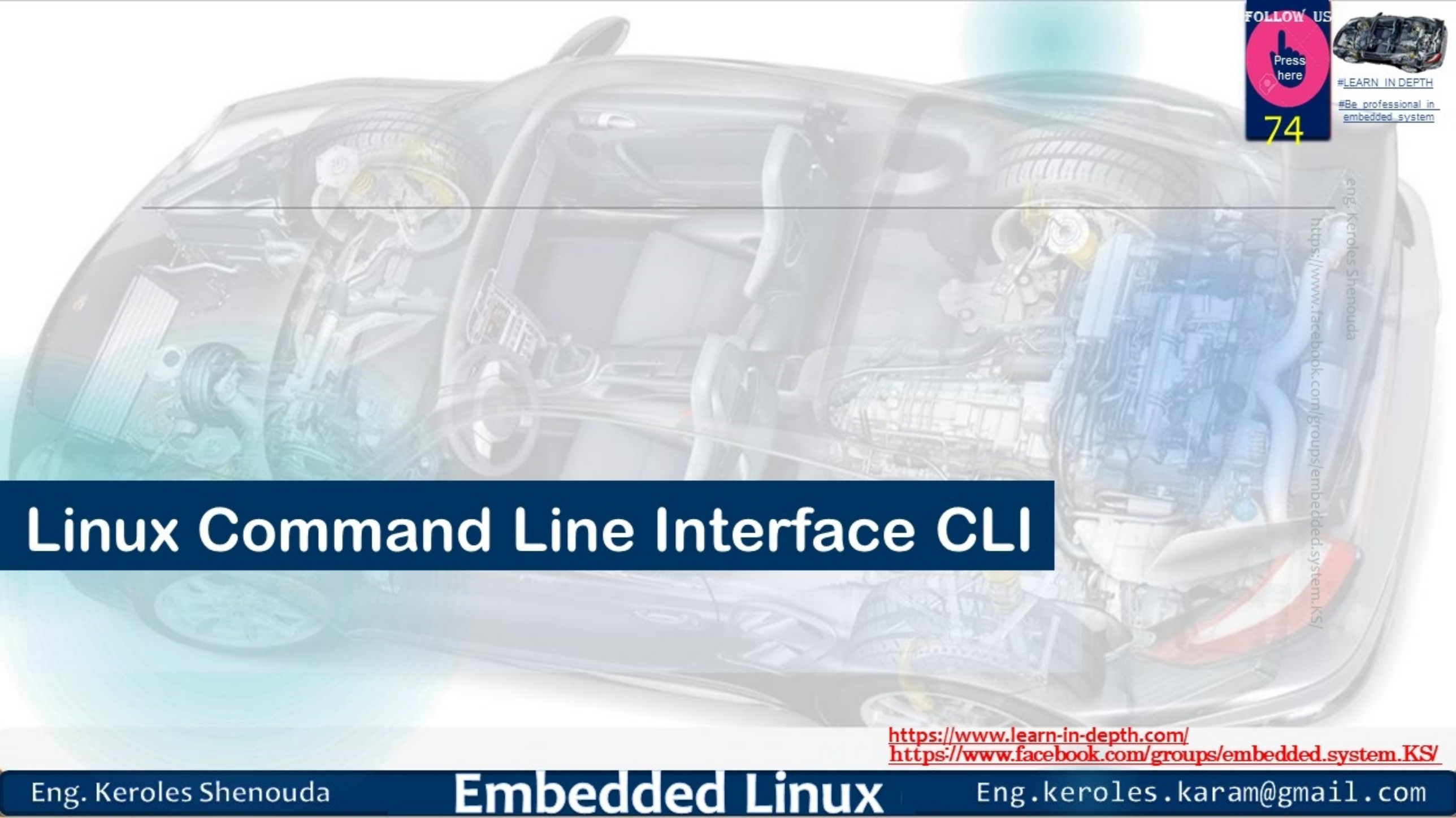
#Be professional in embedded system

Client.c

```c
/*
 * client.c
 *
 *  Created on: Dec 10, 2019
 *   Author: Keroles Shenouda
 */

#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <arpa/inet.h>

int main(int argc, char *argv[])
{
    int sockfd = 0, n = 0;
    char recvBuff[25];
    struct sockaddr_in serv_addr;

    if(argc != 2)
    {
        printf("\n Usage: %s <ip of server> \n",argv[0]);
        return 1;
    }

    memset(recvBuff, '0',sizeof(recvBuff));
    if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("\n Error : Could not create socket \n");
        return 1;
    }
```

```c
    memset(&serv_addr, '0', sizeof(serv_addr));

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(5000);
    serv_addr.sin_addr.s_addr = inet_addr(argv[1]);

    if( connect(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
    {
        printf("\n Error : Connect Failed \n");
        return 1;
    }

    while ( (n = read(sockfd, recvBuff, sizeof(recvBuff)-1)) > 0)
    {
        printf ("%s \n",recvBuff);

    }


    return 0;
}
```

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

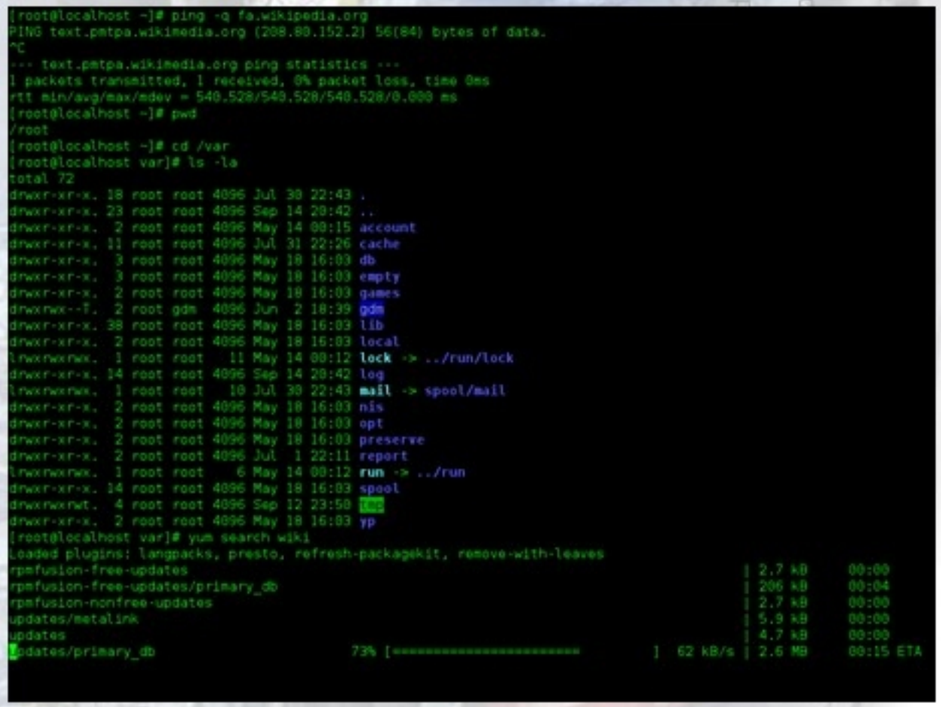


Linux Command Line Interface CLI

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Access of Linux by CLI

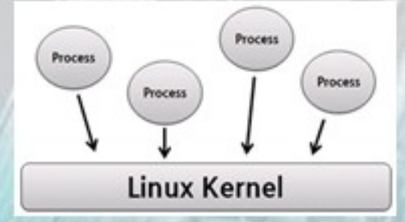
- Using the Command Line Interface • Looks like DOS, but with much more features !!!
- With CLI,
  - More Control, do things you can never reach with GUI
  - Faster, one command can do the effect of several menus, and mouse clicks
  - Enable automation and scripting
- GUI makes easy tasks easier…. But CLI makes difficult tasks possible
- Sometimes in the embedded systems, ***CLI is all what we have***



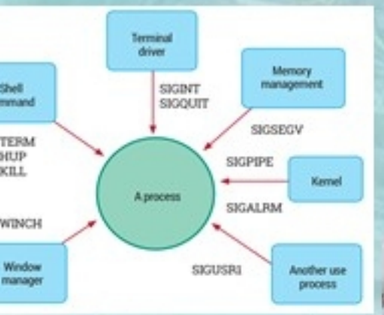
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Linux Command Line Interface
CLI

Text Handling

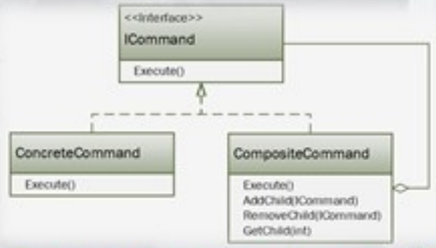


Process Management

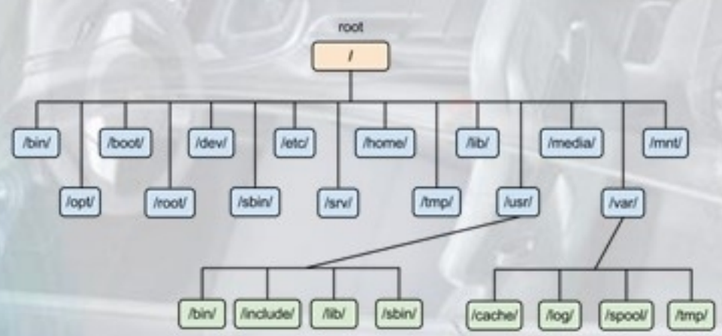


Sending Signals

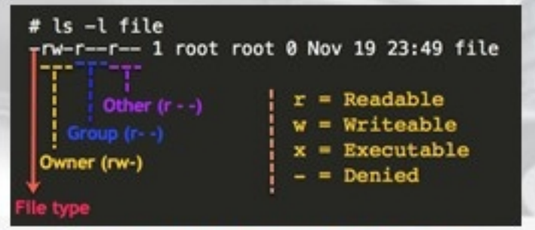
Searching for Files



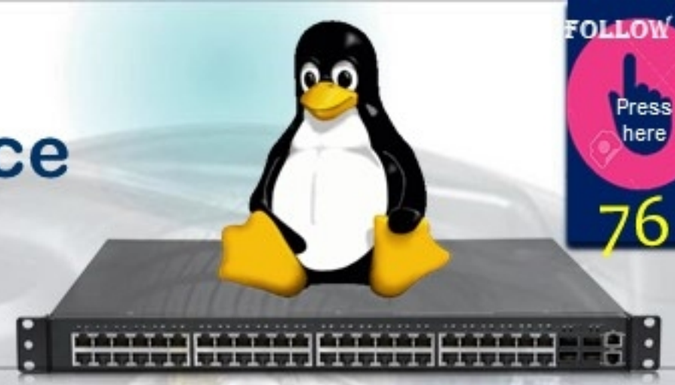
Composite Commands



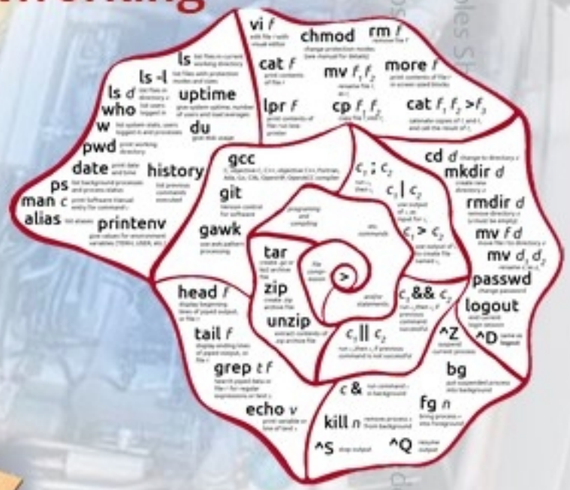
FileSystems



Users & Permissions



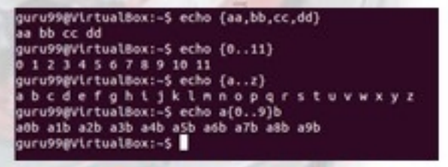
Networking



Learning about the Shell

Archiving & File Compression

Environment Variables



Regular Expressions

#LEARN IN DEPTH

#Be professional in embedded system

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

#LEARN IN DEPTH

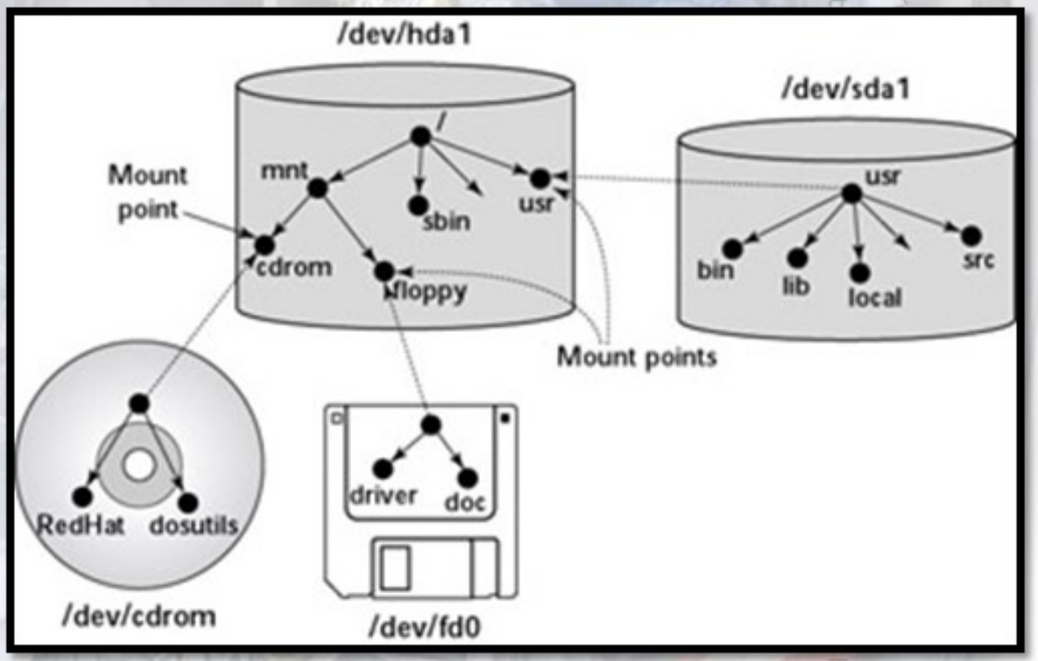
#Be professional in embedded system

Linux File-System Hierarchy

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Linux File-System Layout

- In Linux there is a single tree (unified file-system)
- The head of the tree is called the root '/'
- Plugging a device will add a branch (or sub-tree) to the filesystem … at the selected mounting point
- **Mounting** is the operation that you perform to cause the file system on a physical storage device (a hard-disk partition or a CD-ROM) **to appear as part of the Linux file system**



A *mount point* is a directory (typically an empty one) in the currently accessible *filesystem* on which an additional filesystem is *mounted* (i.e., logically attached).

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Basic Information about root filesystem (we will take it in depth later)

Directory	Description
/	The root filesystem **is the top-level directory** of the filesystem. **It must contain all of the files required to boot the Linux system before other filesystems are mounted**. It must include all of the required executables and libraries required to boot the remaining filesystems**. After the system is booted, all other filesystems are mounted on standard**, well-defined mount points as subdirectories of the root filesystem.
/root	This is not the root (/) filesystem. **It is the home directory for the root user (admin)**
/bin	The /bin directory contains **user executable files**. (system/common) binaries
/boot	Contains the static bootloader and kernel executable and configuration files required to boot a **Linux computer.** **/boot/vmlinuz/ Linux kernel image** **/boot/frup/grup.conf Bootlaoder configuration file** **/boot/initrd/ startup files used in booing**
/dev	This directory contains **the device files for every hardware device attached to the system**. These are not device drivers, rather they are files that represent each device on the computer and facilitate access to those devices. **This is a place holders for the device nodes.**

#LEARN IN DEPTH

#Be professional in embedded system

Basic Information about root filesystem (we will take it in depth later)

/etc	Contains the local system configuration files for the host computer. Ex. **/etc/fstab** contains a list of storage devices and their associated mount point. **/etc/passwd** contains a list of user accounts.
/home	**Home directory storage for user files**. **Each user has a subdirectory in /home**.
/lib	Contains **shared library** files that are required to boot the system.
/media	**A place to mount external removable media devices such as USB** thumb drives that may be connected to the host.
/mnt	**A temporary mountpoint for regular filesystems** (as in not removable media) that can be used while the administrator is repairing or working on a filesystem.
/opt	Optional files such as vendor supplied application programs should be located here. The user optionally installed it on this dir to be shared with another users

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

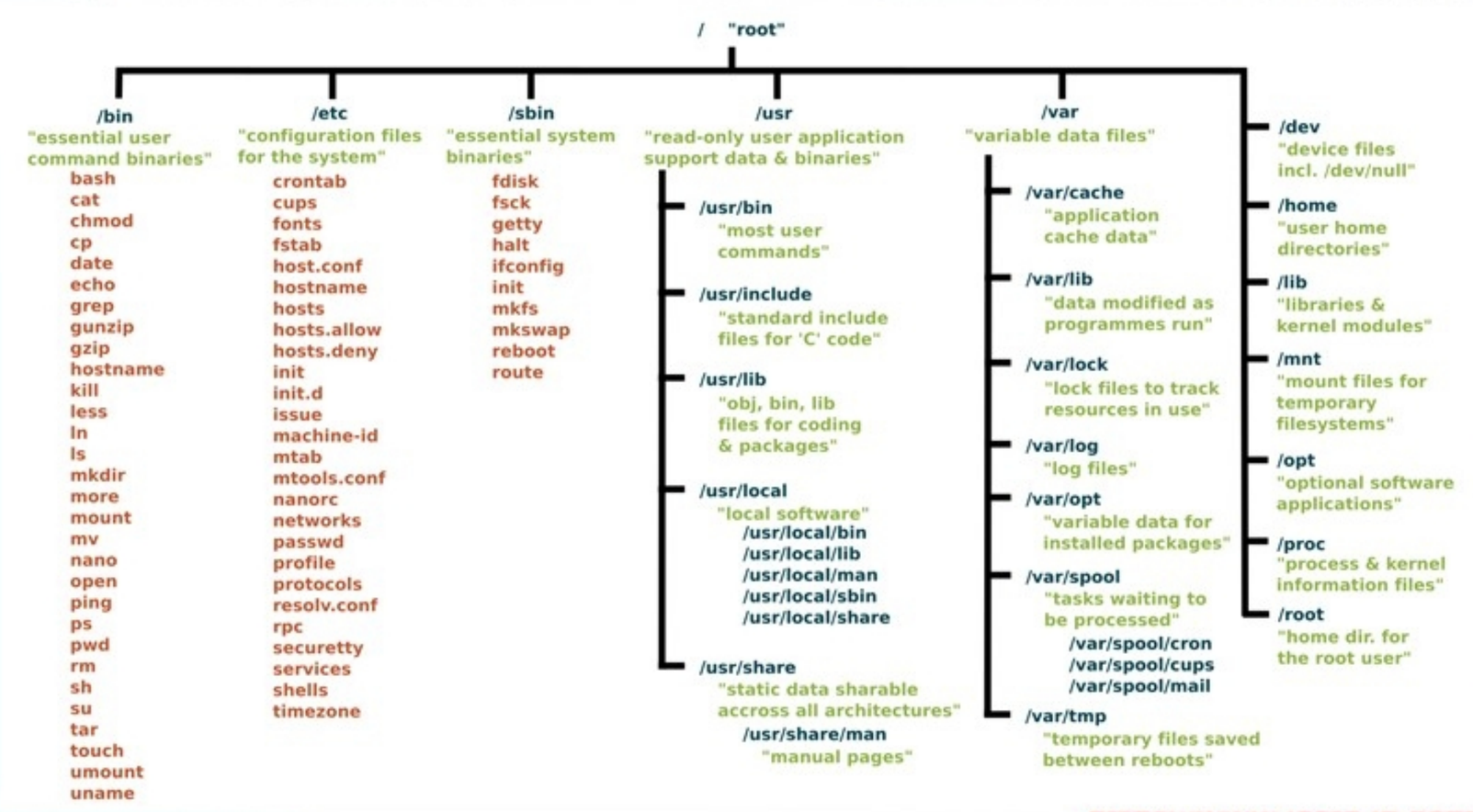
Basic Information about root filesystem (we will take it in depth later)

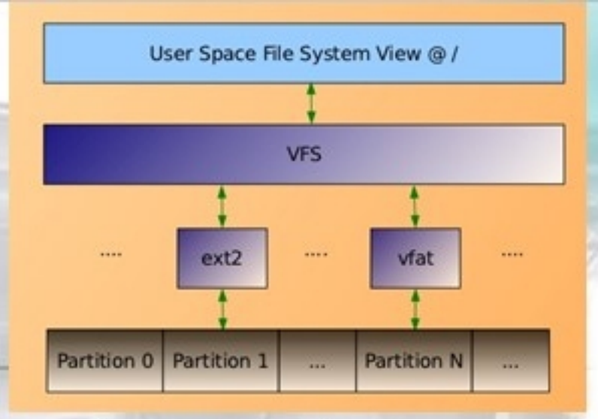
/sbin	System binary files. These are executables used for system administration.
/tmp	Temporary directory. Used by the operating system and many programs to store temporary files. Users may also store files here temporarily. Note that files stored here may be deleted at any time without prior notice.
/usr	**These are shareable between multi users** , read-only files, including executable binaries and libraries, man files, and other types of documentation.
/var	Variable data files are stored here. This can include things like **log files**, MySQL, and other database files, web server data files, email inboxes, and much more. /var/log log files /var/log/messages system log file for debugging

Basic Information about root filesystem (we will take it in depth later)

Virtual filesystem	/proc	Does not contain stored files, **it is just the front of some devices feedback (reading from it, is like asking a driver to provide information) Used to read information about system resources.**
	/sys	Same as /proc/
	/tmp	**Temporary space for use by the system**, cleaned upon reboot, so don't use this for saving any work!
	/lost+found	For Every disk partition, **contains files that were saved during failures are here.**

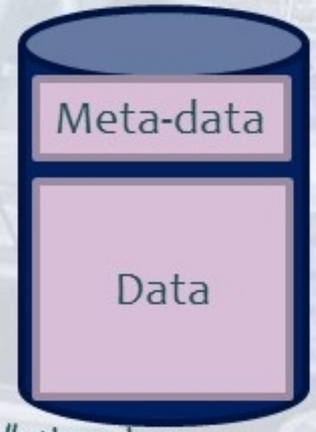
The Linux Filesystem

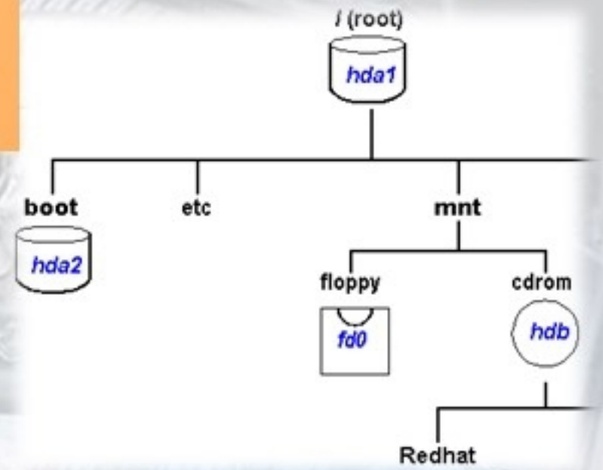


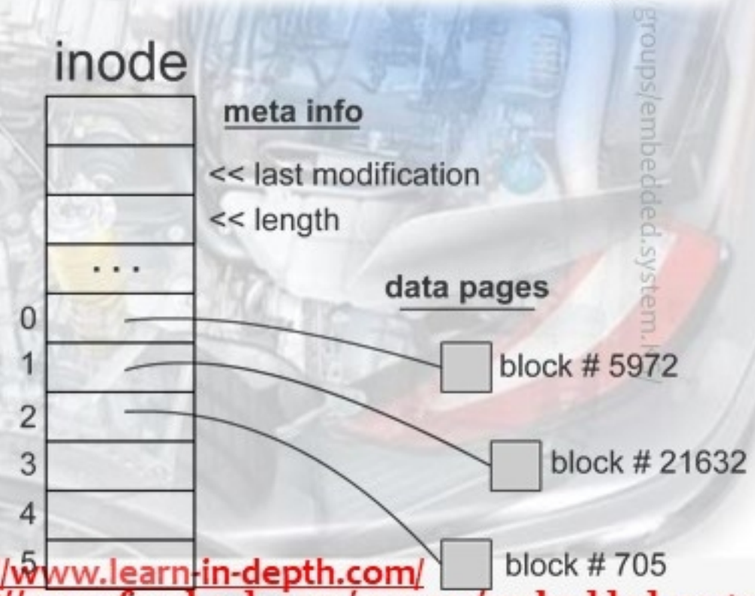


What is a File ??

- A file is a set of bytes that represent some content (pdf document, excel sheet, binary executable, …)
- The file is stored in a (partition in a) storage device as a single data block or **fragmented (devidded into different blocks)** into a group of data blocks (within the same partition)
- The fileSystem is responsible for managing **the data block(s),** and their representation to the user
- For this management, the fileSystem needs to maintain some extra info about the file which is called file **Meta-data**
  - File Size
  - File Owner (user & group)
  - File Permissions
  - Data of creation/last modification
  - Pointers to the file content data blocks
  - etc …
- These meta-data are stored in an "inode" stands "index node" structure

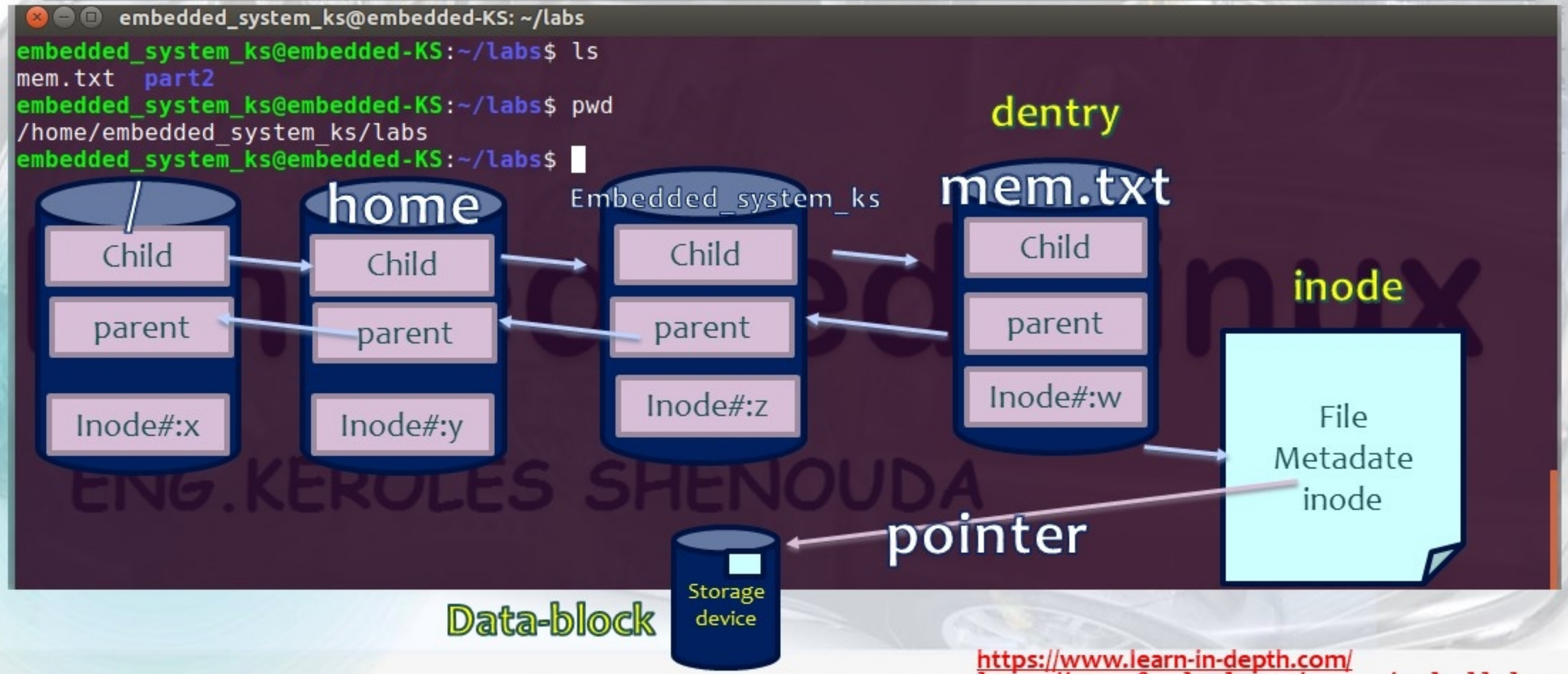






https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Example



Example Cont.

- Ls -i show the inode number for this file

```
embedded_system_ks@embedded-KS: ~/labs
embedded_system_ks@embedded-KS:~/labs$ ls mem.txt -i
414304 mem.txt
embedded_system_ks@embedded-KS:~/labs$
```

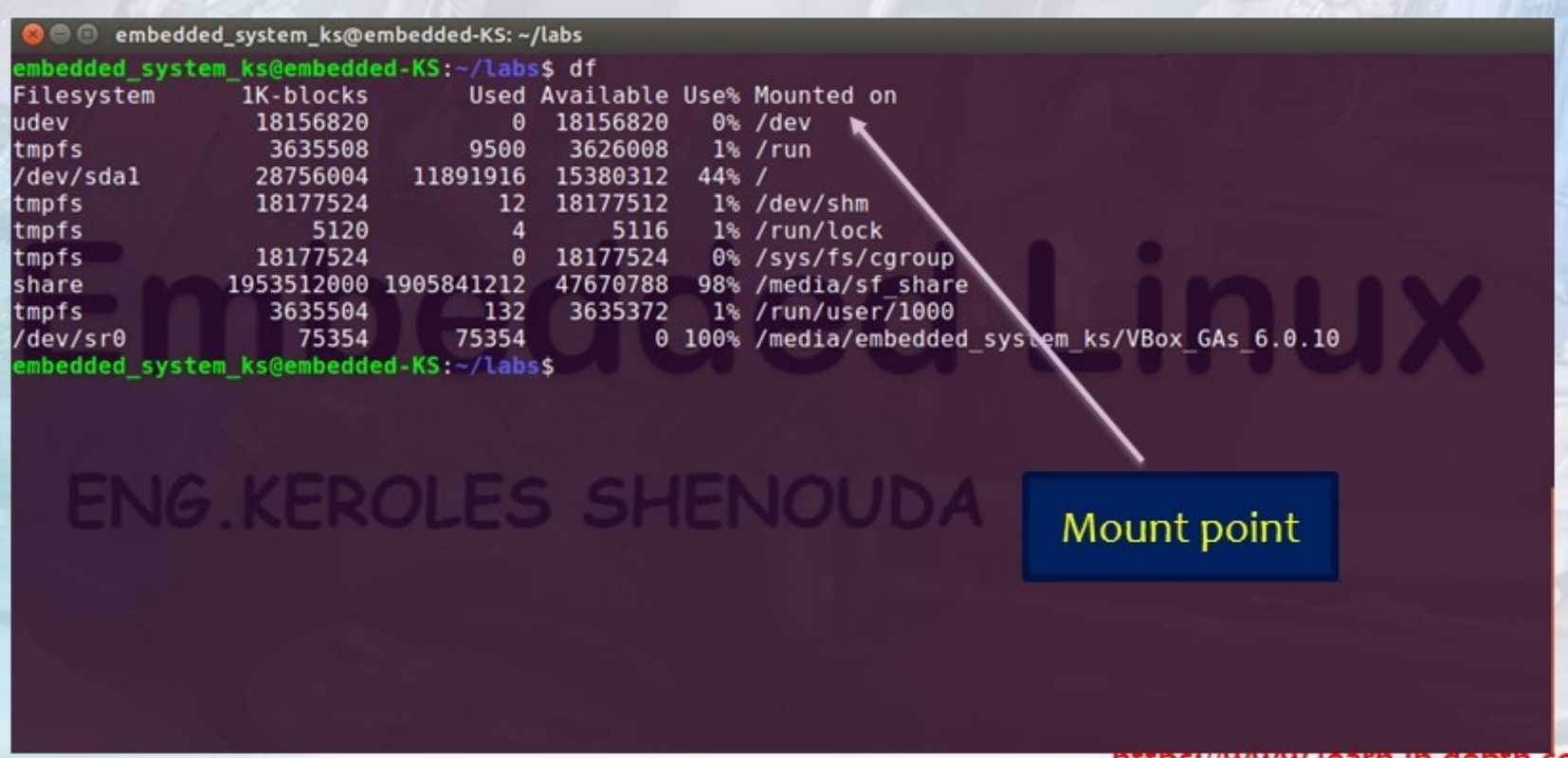
Stat command

```
embedded_system_ks@embedded-KS:~/labs$ stat mem.txt
  File: mem.txt
  Size: 10133534        Blocks: 19808      IO Block: 4096   regular file
Device: 801h/2049d      Inode: 414304      Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/embedded_system_ks)   Gid: ( 1000/embedded_system_ks)
Access: 2019-11-22 19:09:10.918715147 +0200
Modify: 2019-11-22 19:08:48.102715147 +0200
Change: 2019-12-20 21:42:21.022406753 +0200
 Birth: -
embedded_system_ks@embedded-KS:~/labs$
```


https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

$df command

- $ df (Show FileSystem Disk Space Usage)



$ df -i (Show FileSystem inode Usage)

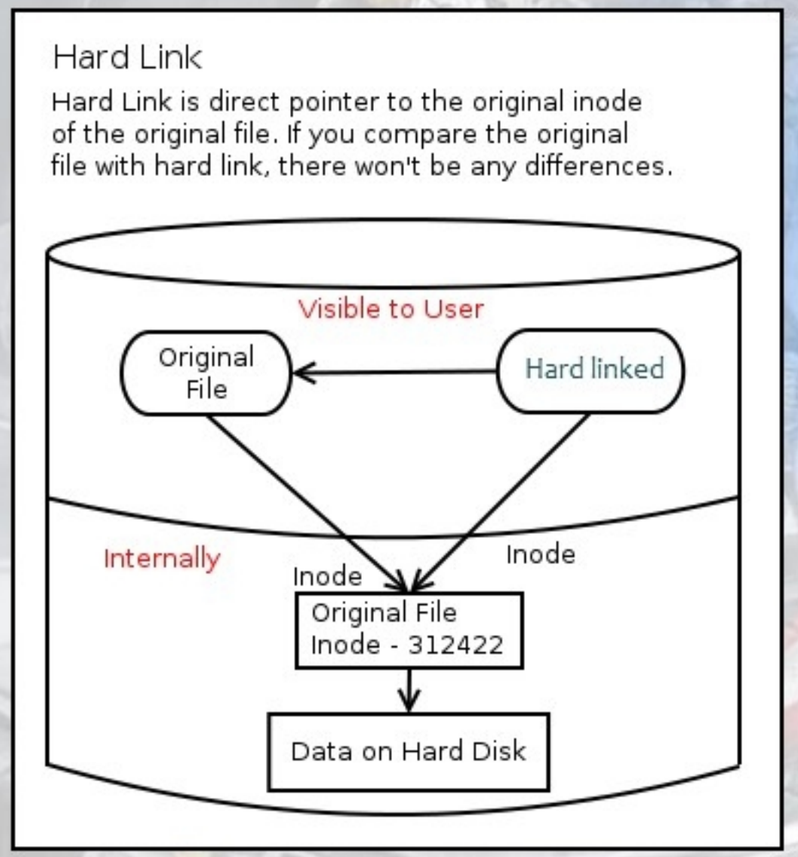
```
embedded_system_ks@embedded-KS: ~/labs
embedded_system_ks@embedded-KS:~/labs$ df  -i
Filesystem      Inodes   IUsed   IFree IUse% Mounted on
udev            123032     453  122579    1% /dev
tmpfs           133385     675  132710    1% /run
/dev/sda1      1835008  315818 1519190   18% /
tmpfs           133385       4  133381    1% /dev/shm
tmpfs           133385       5  133380    1% /run/lock
tmpfs           133385      16  133369    1% /sys/fs/cgroup
share             1000 -999000 1000000     - /media/sf_share
tmpfs           133385      72  133313    1% /run/user/1000
/dev/sr0             0       0       0     - /media/embedded_system_ks/VBox_GAs_6.0.10
embedded_system_ks@embedded-KS:~/labs$
```

$df –kh .

```
embedded_system_ks@embedded-KS: ~/labs
embedded_system_ks@embedded-KS:~/labs$ df -kh .
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1        28G   12G   15G  44% /
embedded_system_ks@embedded-KS:~/labs$
```

Hard LINKING FILES

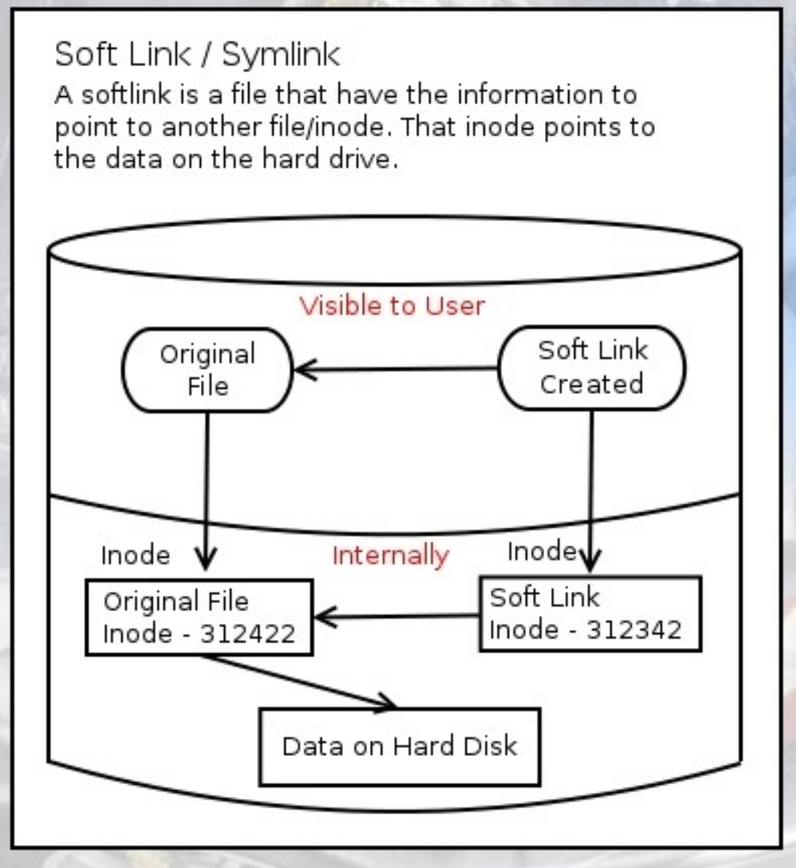
- in hard link, only **an entry into** directory structure is created for the file, **but it points to the inode location of the original file**.
  - Which means there **is no new inode creation in the hard link**. This can be explained like this:



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Soft LINKING FILES

- Soft links are very similar to what we say "Shortcut" in windows,
- is a way to link to a file or directory.
- it simply contains the pointer to the location of the destination file.
- In more technical words, in soft link, a new file is created **with a new inode**, which have the pointer to the inode location of the original file. This can be better explained with a diagram
- $ls -s <file_original> <full path soft link file>
  - Important Note, it should be full path



Example

l means symbolic link

d means directory

```
embedded_system_ks@embedded-KS: ~/labs
embedded_system_ks@embedded-KS:~/labs$ ln mem.txt hard_mem.txt
embedded_system_ks@embedded-KS:~/labs$ ln -s mem.txt soft_mem.txt
embedded_system_ks@embedded-KS:~/labs$ ls -l
total 19812
-rw-r--r-- 2 embedded_system_ks embedded_system_ks 10133534 فون 22 19:08 hard_mem.txt
-rw-r--r-- 2 embedded_system_ks embedded_system_ks 10133534 فون 22 19:08 mem.txt
drwxr-xr-x 2 embedded_system_ks embedded_system_ks     4096 سپ 20 21:42 part2
lrwxrwxrwx 1 embedded_system_ks embedded_system_ks        7 سپ 20 22:12 soft_mem.txt -> mem.txt
embedded_system_ks@embedded-KS:~/labs$ ls -li
total 19812
414304 -rw-r--r-- 2 embedded_system_ks embedded_system_ks 10133534 فون 22 19:08 hard_mem.txt
414304 -rw-r--r-- 2 embedded_system_ks embedded_system_ks 10133534 فون 22 19:08 mem.txt
414283 drwxr-xr-x 2 embedded_system_ks embedded_system_ks     4096 سپ 20 21:42 part2
408210 lrwxrwxrwx 1 embedded_system_ks embedded_system_ks        7 سپ 20 22:12 soft_mem.txt -> mem.txt
embedded_system_ks@embedded-KS:~/labs$
```

Create hardlink

Create softlink

Number of hardlinks

That means inode for mem.txt have two entries point on it

In the hard link have the same inode number

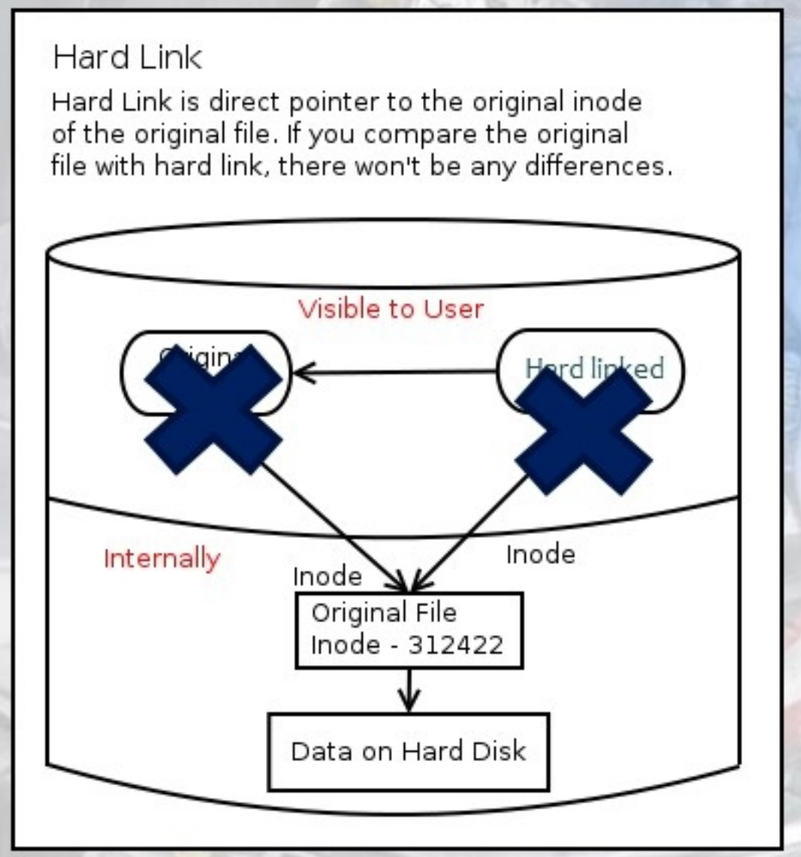
In the soft link have the different inode number

Hardlink constrains

- Only applicable for files, not used for directories
  - After implementing it for directories, a security hole was found Can cause loops of links which result in system faults So it was disabled in latest releases
- Does not work across filesystems
  - We just link using the inode# **But inode# is only unique within the same filesystem**
    - Hence, we can not link to a file in a different filesystem This is very limiting, specially Linux merges all the FS in a unified tree

To rm the hardlink and the original files

- You have to remove the original and hard link file to remove the inode and the d-entries



https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

Learn-in-depth.com

References

- Embedded Linux training
  - https://bootlin.com/training/
- Linux kernel and driver development training
- Yocto Project and OpenEmbedded development training
- Buildroot development training
- Building Embedded Linux Systems, 2nd Edition
- Mastering Embedded Linux Programming - Second Edition
- Christopher Hallinan, **Embedded Linux Primer,** Prentice Hall, 2006.
- **Silberschatz**, Galvin, and Gagne's **Operating System Concepts**, **Eighth Edition**.
- Robert love, linux kernel develoepemnt 3 rd edition 2010
- **Advanced Linux Programming By Mark Mitchell, Jeffrey Oldham, Alex Samuel**
- **Linux OS in Embedded Systems & Linux Kernel Internals(2/2)**
  - Memory Management, Paging, Virtual Memory, File system and its implementation, Secondary Storage(HDD), I/O systems
- Ahmed ElArabawy Courses at http://linux4embeddedsystems.com/

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

- Memory Management article
- Introduction to Operating Systems Class 9 - Memory Management
- Virtual Memory lecture for Introduction to Computer Architecture at Uppsala University.
- OS Lecture Note 13 - Address translation, Caches and TLBs
- CSE 331 Operating Systems Design lectures
- CSE 331 Virtual Memory
- CSE 332 Computer Organization and Architecture
- File Systems: Fundamentals.
- Linux System Programming
- System Calls, POSIX I/O
  CSE 333 Spring 2019, Justin Hsia
- Linux basic commands

https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

- **File Permissions in Linux/Unix with Example**
- Introduction to UNIX / Linux – 4
- **12.2 Basic I/O Concepts**
- Communicating with Hardware
- I/O Systems I/O Hardware Application I/O Interface
- COMPUTER ARCHITECTURE
- **OS**
- Linux Operating System
- W4118 Operating Systems, Instructor: Junfeng Yang
- CSNB334 Advanced Operating Systems 4. Process & Resource Management.
- Using the POSIX API
- **Linux Memory Management**
- Chapter 2: Operating-System Structures
- **POSIX Threads Programming**
- Pthreads: A shared memory programming model https://slideplayer.com/slide/8734550/
- Signal Handling in Linux Tushar B. Kute
- Introduction to Sockets Programming in C using TCP/IP