Advanced File operations

Advanced File Operations

ioctl() : read() & write() system calls doesn’t support to control device specific Parameters. To control and to get Device specific parameters use ioctl()

ioctl referred as input and output Control.
Ioctl is a system call for device-specific input/output Operations which can not be expresses by a regular System calls. (in kernel it is come under Device Management subsystem)

The simple files of a Linux-based system can easily be read or
Written using simple input and output operations.
Like read() and write() system calls.
However, there are some complex types of files too that cannot
Be accessed with the help of simple input and output functions.
So there are specific system calls which are used for those
special Files.
The special files are like device files they reside within Linux-based
“/dev” directory.

ioctl() system call

File descriptor associate with device file

```
int ioctl(int fd, unsigned long request, ...);
```

RETURN

Success: Zero

ERROR: Negative Number

Device commands

ioctl system call example:

```
#include<stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include<sys/ioctl.h>
#include <linux/input.h>
int main(int argc, char *argv[])
{
    int fd1,retfd;
    char buf[4096];
    int val;
    char name[256] ;
    //argv[1] has in format /dev/input/event0
    fd1 = open(argv[1],O_RDONLY);
    if(fd1<2)
        {
            printf(“Open Fails”);
            return -1;
        }
    //function call to get device name
    ioctl(fd1,EVIOCGNAME(sizeof(name)),name);
    printf(“Input device name: \”%s\”\n”,name);
    return 0;
}
```

This is the program to findout the device name, for which this device file is created.

If we give arguement as /*dev*/input/event0

Then we will get corresponding device name for that device file (/dev/input/evvent1).

Advanced File Operations

select(): read() & write() system calls are blocking for single file descriptor select() system call blocks for multiple descriptor rather single descriptor.

Select system call allow a program to moniter multiple file descriptors, waiting until one or more of the file descriptors become “ready” for some class of I/O

select() system call

No. of
File descriptor

readfds will be watched to see if characters become available for reading

writefds will be watched to see if space is available for write

```
int select(int nfds, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);
```

exceptfds will be watched for exceptional conditions

should block waiting for a file descriptor to become ready

RETURN

Success: No. of FD's contained in the three returned descriptor sets (that is,

The total number of bits that are set in readfds, writefds, exceptfds)

TIMEOUT: Return ZERO

ERROR: Negative Number

```
void FD_CLR(int fd, fd_set *set);
int  FD_ISSET(int fd, fd_set *set);
void FD_SET(int fd, fd_set *set);
void FD_ZERO(fd_set *set);
```

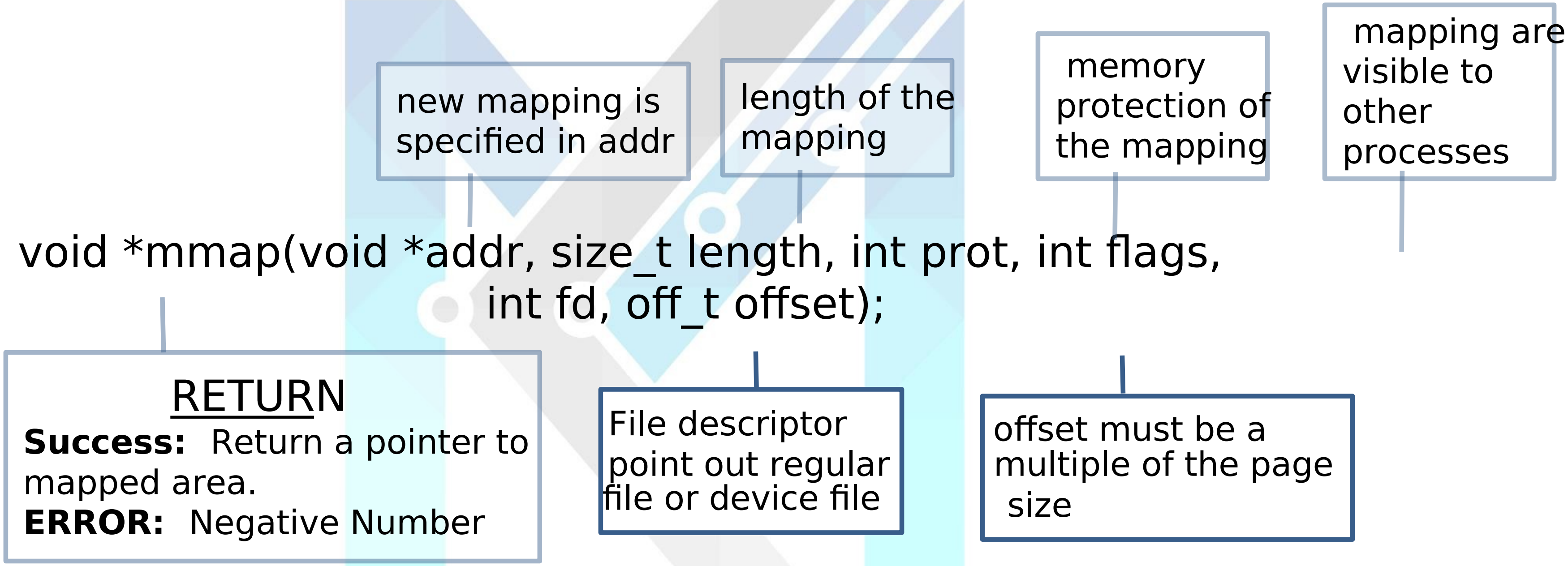
select() system call example:

```
#include<stdio.h>
#include<sys/time.h>
#include<sys/types.h>
#include<unistd.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<linux/input.h>
int main(int argc, char *argv[])
{
    fd_set rfds;
    struct timeval tv;
    int retval;
      /*Watch stdin (fd 0) to see when it has input*/
      FD_ZERO(&rfds);
      FD_SET(0,&rfds);
/* Wait up to five seconds. */
    tv.tv_sec = 5;
    tv.tv_usec = 0;
    retval = select(1, &rfds, NULL, NULL, &tv);
    /* Don’t rely on the value of tv now!*/
   if(retval == 0) {
        printf("select timeout:\n");
    }
    else if(retval == -1){
        printf("fail to select\n");
    }
    else{
      printf("data is avaliable\n");
    }
   return 0;
}
```

Advanced File Operations

mmap(): using mmap() system calls to map device buffers into running process Memory .

mmap() system call



new mapping is specified in addr

length of the mapping

memory protection of the mapping

mapping are visible to other processes

```
void *mmap(void *addr, size_t length, int prot, int flags,
                int fd, off_t offset);
```

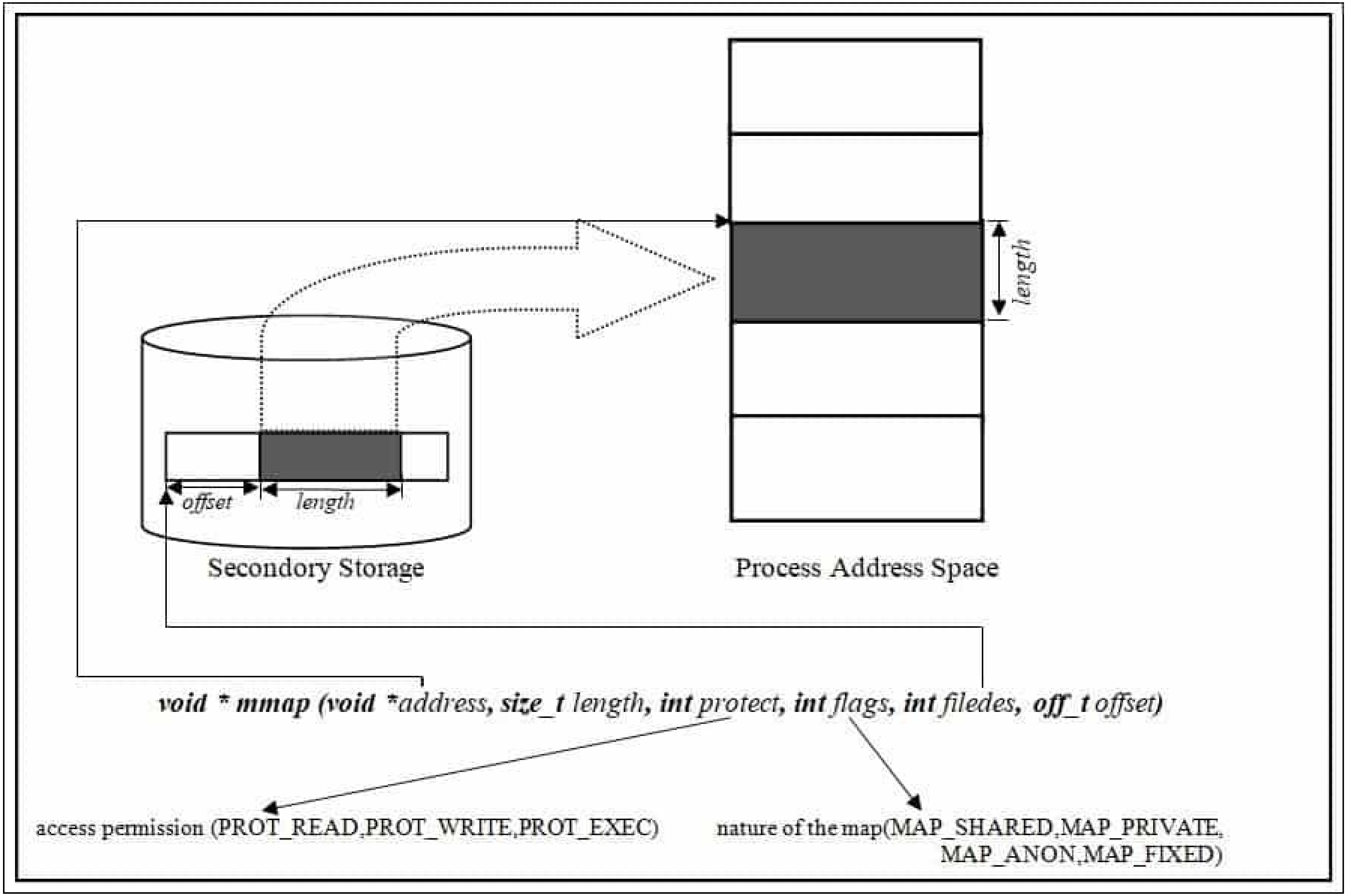
RETURN

Success: Return a pointer to mapped area.

ERROR: Negative Number

File descriptor point out regular file or device file

offset must be a multiple of the page size



***void * mmap (void** *address, **size_t** length, **int** protect, **int** flags, **int** filedes, **off_t** offset)*

access permission (PROT_READ,PROT_WRITE,PROT_EXEC)

nature of the map(MAP_SHARED,MAP_PRIVATE,
MAP_ANON,MAP_FIXED)

mmap() system call example:

```
# include <unistd.h>
# include <sys/types.h>
# include <sys/mman.h>
# include <sys/stat.h>
# include <fcntl.h>
# include <stdio.h>

int main(){
        int fd,i,ret;
        unsigned char *filedata= NULL,*temp;

        fd = open("pres.txt",O_RDWR);
        getchar();

        filedata = (char *) mmap((void*)0,1,PROT_READ|
PROT_WRITE, MAP_SHARED,fd,0);
getchar();
        // now we can access the content of the file as if it is part of
        // our process starting from the memory pointed by filedata.
        temp = filedata;
        for(i=0;i<4;i++,filedata++)
        {
                *filedata = (char)(i+65);
                printf("\n %c\n",(char)(i+65));
        }
        getchar();

/*       i = munmap(temp,6);
        getchar();
        if( i != 0)
                printf(" failed to unmap\n");
                */
}
```

Day 2 Assignments:

1. Write a program show the /dev/input/event0 device name?
2. Write a program to read framebuffer fixed size information?

   Hint: Device name is /dev/fb0.
3. WAP your own version of cat command using mmap system call?
4. Write an Linux System Programming copy one file content to another file using mmap() system call.