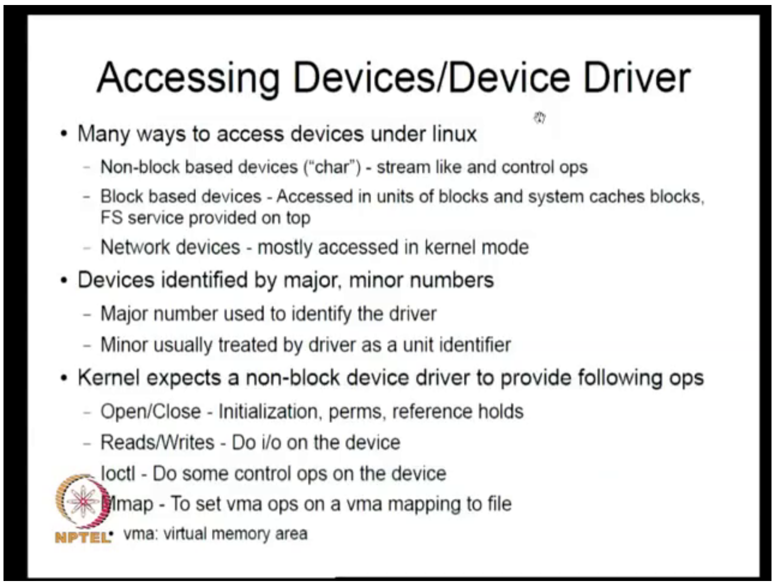
# Storage Systems

**Dr. K. Gopinath**
**Department of Computer Science and Engineering**
**Indian Institute of Science, Bangalore**

## Storage Interfaces and Device Driver
**Lecture – 17**
**Device Drivers_Part 1: Introduction, User-Memory-CPU interactions in Device Driver**

Welcome again to the NPTL course on storage systems. So, in the previous class we had some general discussion on device drivers.

(Refer Slide Time: 00:28)



### Accessing Devices/Device Driver

- Many ways to access devices under linux
  - Non-block based devices ("char") - stream like and control ops
  - Block based devices - Accessed in units of blocks and system caches blocks, FS service provided on top
  - Network devices - mostly accessed in kernel mode
- Devices identified by major, minor numbers
  - Major number used to identify the driver
  - Minor usually treated by driver as a unit identifier
- Kernel expects a non-block device driver to provide following ops
  - Open/Close - Initialization, perms, reference holds
  - Reads/Writes - Do i/o on the device
  - Ioctl - Do some control ops on the device
  - Mmap - To set vma ops on a vma mapping to file
    - vma: virtual memory area

We will go into some more details in this class. And also, you look at an example. I think again to summarize a bit, the many ways to access devices in many systems including Linux. A what are called char devices, char devices, we will pronounce it differently. Basically, these are what are called non-block based devices, basically is a stream.

So, they do not have any markers. When it is a basically stream, or you can also do control operations on it. Block based devices on their hand have very clear markers, where it some block starts when it ends for example, 512 bytes or 4 kilobytes whatever right. That is in different being beginning to the block and end to the block. And it is the information is structured in terms of blocks. And typically, the system caches the blocks. The basic reason is that, when you are dealing with blocks the reason why you are dealing with blocks is because of a disability considerations. Because you want you want to access a very large stuff. If you access that smaller levels you need the addressability is going to be not enough.

So, therefore, because you are dealing with such things, it turns out that you are dealing with typical slow devices in the caching also helps here. And typically, most file systems are expected to be running on top of such block devices. You also have things like network devices. Often times it is not exported out as a device that users can use directly. The usual accessed in kernel mode, what I mean by that is if you have a disk for example, it can be exported as slash dev slash disk one hda. And if you are a user program of course, it could be privileged user program.

For example, a process that is owned by route say route also is a user in the system. And user also can start a process on the console or wherever right. So, any process that he starts has got group privileges but still use a program it is not a kernel code. It is still using system calls interface.

Now, if you look at block based devices or character based devices, most unixs and Linux, they will export that particular device as some slash dev slash hda whatever tty etcetera; that means, that a privileged program which has enough privileges to open the device can directly read and write.

Let me the user program; that is, without any kernel code can directly access that particular device. Whereas, the network device what we is being mentioned here most likely kernel mode means that you have to have your code if you want to access these things inside a kernel itself. That is what if I mention.

Of course, other up some operating systems, the expose network devices even protocols outside. There is a operating system called plan 9, some of you might have heard of it. It exports even TCP. You can access it as slash devs, something slash TCP slash ip etcetera. One read from a TCP usually can say read from slash dev something TCP etcetera. This is way to do it.

Typically, devices are identified by some major and minor numbers. Why do you need all this major and minor numbers? Basically, because you want to keep system dynamically configurable; you are not really constraining system to have exactly this number of devices. So, if you do not know what device have only put on taken off right? If you want to refer to them all right. So, suppose I am able to say that when I attach a device, I give it some number as a part of attaching a device electrically attach it, but then I say now then I attached it. Please give me a corresponding let us say way to access a device.

Simplest thing that people have figured out is have give giving it something a major number. So, example I can see the my tape device, I have got major number 7. Let us say, it can use the it can use that number and see where to say that I am interested in tape number 7. Suppose I take the device off, let us I have maximum, let us say my major number is let us say 8 bits; that means, I can have maximum off 256 types of devices. And then I can later once I remove 7, the tape device I can attach some other thing, like a CD ROM for example, I will say reuse that number slot 7. I can do those kind of things.

So, you can always say why not use some user-friendly names. You can always have user friendly name, but you have to pass it again you have to pass it on again it takes time. So, idea is to avoid those things. So, that you have some numbers. So, it makes it easier just like file descriptors if you have familiars file descriptors, you will notice that once you open a file you get a file descriptor. In the beginning at open time you pass it the full file name slash some things slash some things slash something. That takes some time. And then once you get a fd then you do not have go through the process. That is basically what you think that (Refer Time: 06:36)

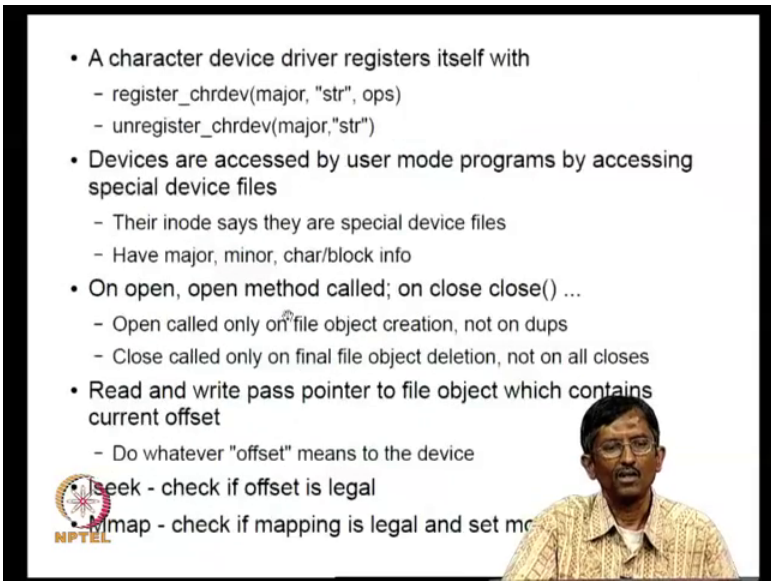
The minor numbers are you can have many many types of units of the same type. For example, I have a tape controller. I might have lots of tape devices. So, tape control might have a major number and a the tape unit reserve might have the units have that number. For example, you will be able to take a regular audio system right. It has got 2 tape drives right we can call it tapes you might have one controller and I have 2 tape drives one which does a recording. One which does plays or one is does both recording and play whenever we play plays, whatever right.

So, the minor number tells you how many things. So, you can have a big system with thousand disks. Impossible where you can identify each disk by it is minor number that also is possible. So, it is unit identifier. So, normally when you have a so, the example I am going to discuss today is a the character device, or and it is typically if you are talking about a character device and non-block device you need to provide the following things. How to open it up? Open the device close the device, how to do I/O on it? And how to do some control operations in it? I was sit elements I/O control; means, you are trying to control how the device operates

So, for example, somebody might want to try to do something slightly non-standard with the device. Let us say for example, the block size is something on a particular tape, and you want to change it to some other thing. Let us say, I am was giving and an example and I was there maybe used to configure that part. Essentially you can use it for configuring the device. Or it could be that there are some interrupt lines you want to tell it to use the particular interrupt line. Because how the system the typical interrupt line it is using is being use somebody else. And I want to say you are what you typically prefer is not available use the something else. Who knows this kind of things may be possible.

There is also memory mapped operations. I think I will skip this right now.

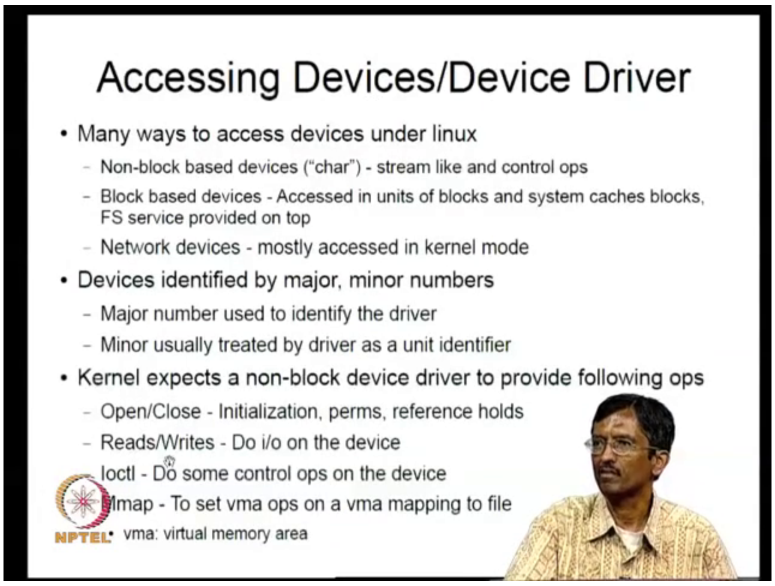
(Refer Slide Time: 09:08)



- A character device driver registers itself with
  - register_chrdev(major, "str", ops)
  - unregister_chrdev(major,"str")
- Devices are accessed by user mode programs by accessing special device files
  - Their inode says they are special device files
  - Have major, minor, char/block info
- On open, open method called; on close close() ...
  - Open called only on file object creation, not on dups
  - Close called only on final file object deletion, not on all closes
- Read and write pass pointer to file object which contains current offset
  - Do whatever "offset" means to the device
- lseek - check if offset is legal
- mmap - check if mapping is legal and set mc

So, it typically a character device driver registers itself with a register character device major operations. Basically, what is it? We are basically saying at this particular major number, there is going to be a device whose operations are given by a vector of operations.

(Refer Slide Time: 09:36)

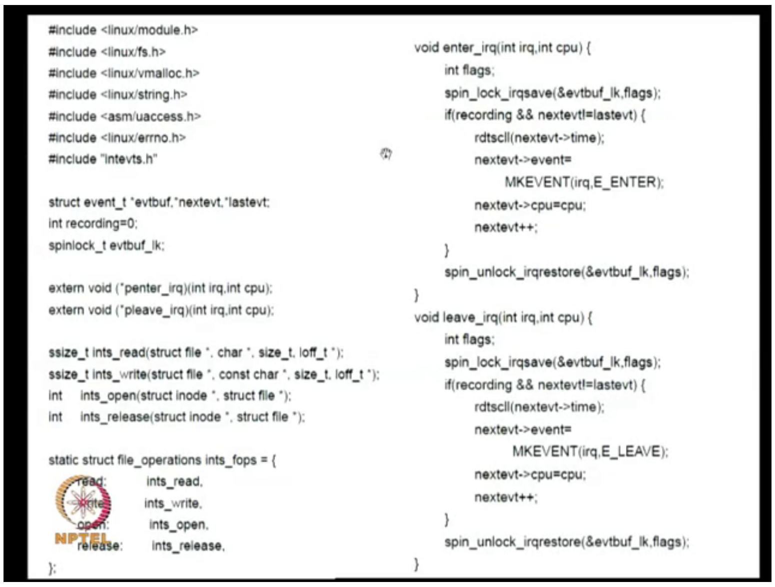


## Accessing Devices/Device Driver

- Many ways to access devices under linux
  - Non-block based devices ("char") - stream like and control ops
  - Block based devices - Accessed in units of blocks and system caches blocks, FS service provided on top
  - Network devices - mostly accessed in kernel mode
- Devices identified by major, minor numbers
  - Major number used to identify the driver
  - Minor usually treated by driver as a unit identifier
- Kernel expects a non-block device driver to provide following ops
  - Open/Close - Initialization, perms, reference holds
  - Reads/Writes - Do i/o on the device
  - Ioctl - Do some control ops on the device
  - Mmap - To set vma ops on a vma mapping to file
    - vma: virtual memory area

And this this operation is typically what we discussed before, open closed, read write sector. To give an example, we can look at this one. So, this is an example.

(Refer Slide Time: 09:44)

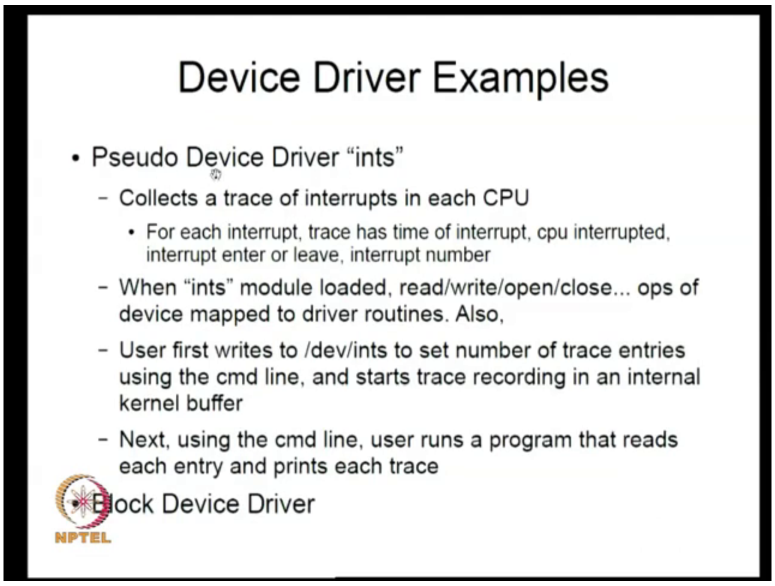


```
#include <linux/module.h>
#include <linux/fs.h>
#include <linux/vmalloc.h>
#include <linux/string.h>
#include <asm/uaccess.h>
#include <linux/errno.h>
#include "intevts.h"

struct event_t *evtbuf,*nextevt,*lastevt;
int recording=0;
spinlock_t evtbuf_lk;

extern void (*penter_irq)(int irq,int cpu);
extern void (*pleave_irq)(int irq,int cpu);

ssize_t ints_read(struct file *, char *, size_t, loff_t *);
ssize_t ints_write(struct file *, const char *, size_t, loff_t *);
int     ints_open(struct inode *, struct file *);
int     ints_release(struct inode *, struct file *);

static struct file_operations ints_fops = {
    read:       ints_read,
    write:      ints_write,
    open:       ints_open,
    release:    ints_release,
};
```

```
void enter_irq(int irq,int cpu) {
    int flags;
    spin_lock_irqsave(&evtbuf_lk,flags);
    if(recording && nextevt!=lastevt) {
        rdtscll(nextevt->time);
        nextevt->event=
            MKEVENT(irq,E_ENTER);
        nextevt->cpu=cpu;
        nextevt++;
    }
    spin_unlock_irqrestore(&evtbuf_lk,flags);
}
void leave_irq(int irq,int cpu) {
    int flags;
    spin_lock_irqsave(&evtbuf_lk,flags);
    if(recording && nextevt!=lastevt) {
        rdtscll(nextevt->time);
        nextevt->event=
            MKEVENT(irq,E_LEAVE);
        nextevt->cpu=cpu;
        nextevt++;
    }
    spin_unlock_irqrestore(&evtbuf_lk,flags);
}
```

We are talking about something called what I am discussing here is a what is called as a pseudo device driver interrupts.

(Refer Slide Time: 09:51)



## Device Driver Examples

- Pseudo Device Driver "ints"
  - Collects a trace of interrupts in each CPU
    - For each interrupt, trace has time of interrupt, cpu interrupted, interrupt enter or leave, interrupt number
  - When "ints" module loaded, read/write/open/close... ops of device mapped to driver routines. Also,
  - User first writes to /dev/ints to set number of trace entries using the cmd line, and starts trace recording in an internal kernel buffer
  - Next, using the cmd line, user runs a program that reads each entry and prints each trace
- Block Device Driver

What is pseudo device driver? It is actually no real device there, but I am giving you some way to access some information is not otherwise excessively what is in the corner.

So, I am basically I am going through interface. The interface gives me some information in a structure and in a very safe manner some information. That I can gets make on what is information I am looking for here I want to collect a trace of interrupts in each CPU. In a real system right, there are lot of interrupts going on. And let us say I am talking about a multi CPU system, interrupts are going on.

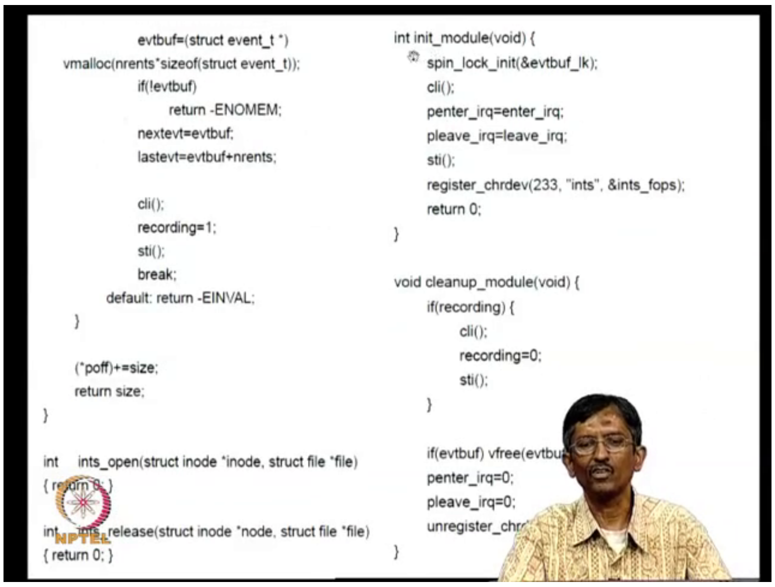
All the time I want to write something which enables me to trace have a trace of interrupts when it this trace when it this interrupt come here which CPU handle it. How long did it take for you to handle it? Example I want to know information like then there is the time with interrupt this CPU got interrupted. When was the tiny chunk this time with interrupt is it the entry time or the exit time and the interrupt number. Basically, I want to count interrupt number one is I want to start is 1 2 3 4 I just want to keep record it.

So, this typic this device typically not available to you. Typically, provided some crazy operating system will give you this, but usually nobody does it but this information available in the kernel. Somewhere it is their question is how do I get is out that is the question. So, what we are going to do is you are going to device a pseudo device driver which can be loaded on to the system. Now who can load the system? Who can load this device driver? Only a authorized person. A person has appropriate privileges. There is another person cannot do it. Because this code that they are exhibiting a sensitive. You can essentially when you insert this particular code in the kernel you have access to the kernel there are structures, you can clobber it. You can change it. You can read from it you can do various things.

Now, that is a bit let us say you are giving lot of privileges to somebody. And it can be complete it can really mess up things. So, what is this and why it has to be done with privilege parties. And that is why it is if you are privilege party can load this particular pseudo device driver, then what you have to do is; you should be able to register the device, and then that operations correspond the device has to be set up.

So, what we are going to do is; we are going to register the device, and then I am going to say that please make sure that whereas, a read actually you make this. So, let us look at that again.

(Refer Slide Time: 13:04)



```
            evtbuf=(struct event_t *)
    vmalloc(nrents*sizeof(struct event_t));
            if(!evtbuf)
                return -ENOMEM;
            nextevt=evtbuf;
            lastevt=evtbuf+nrents;

            cli();
            recording=1;
            sti();
            break;
        default: return -EINVAL;
    }

    (*poff)+=size;
    return size;
}

int    ints_open(struct inode *inode, struct file *file)
{ return 0; }

int    ints_release(struct inode *node, struct file *file)
{ return 0; }
```

```
int init_module(void) {
    spin_lock_init(&evtbuf_lk);
    cli();
    penter_irq=enter_irq;
    pleave_irq=leave_irq;
    sti();
    register_chrdev(233, "ints", &ints_fops);
    return 0;
}

void cleanup_module(void) {
    if(recording) {
        cli();
        recording=0;
        sti();
    }

    if(evtbuf) vfree(evtbu
    penter_irq=0;
    pleave_irq=0;
    unregister_chr
}
```

So, this is example what we are doing is we are going to do when you load the things, we are going to do in it module. That piece of code that is going to be put in the kernel. It is going to initialize something.

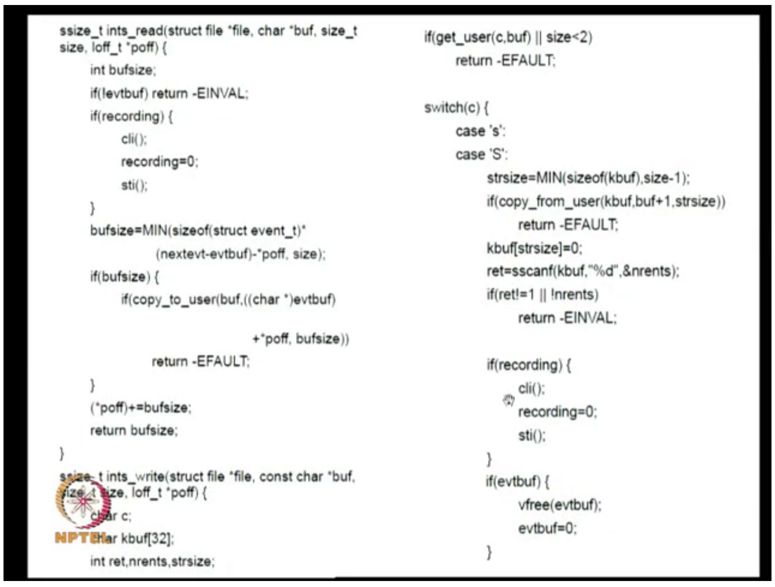
Now, this is basically a locked is required, and that is going to be initialized. This is nots nos mutual exclusion is being done here, it is just you are just taking some other called spin lock, and that is being initialized. Plus, installation, and what we are going to do is this particular init module is going to do this registration here. And it is saying that take 233 as the major number. Ints is the name of the it can you can you can give it anything name you like it is just for our convenience. And it says and ints fops. What is it saying it is saying that for this particular device, give me match up with this particular operations. So, what are the operations here these are operations here.

So, there is ints fops right here, you basically saying that on this device, whenever somebody says read, it actually I mean ints read basically is a function pointers. You can here installing this function pointers normally it will be some function pointers which is with no valid information beginning once you load it these function pointers have initialized to these values.

So, whenever I say read on this device I am I am asked to go and re ints underscore read. Whenever I say write on the device, because I will have a device by the name slash play let us say. Slash dev ints let us say I can open it I can just like you can read and write right, I can also say read from that slash dev ints then I am going to read from corresponding thing is going to be done in ints.

So, and this particular driver is going to supply this routines in ints readers.

(Refer Slide Time: 15:05)



```
ssize_t ints_read(struct file *file, char *buf, size_t
size, loff_t *poff) {
        int bufsize;
        if(!evtbuf) return -EINVAL;
        if(recording) {
                cli();
                recording=0;
                sti();
        }
        bufsize=MIN(sizeof(struct event_t)*
                        (nextevt-evtbuf)-*poff, size);
        if(bufsize) {
                if(copy_to_user(buf,((char *)evtbuf)

                                        +*poff, bufsize))
                        return -EFAULT;
        }
        (*poff)+=bufsize;
        return bufsize;
}
ssize_t ints_write(struct file *file, const char *buf,
size_t size, loff_t *poff) {
        char c;
        char kbuf[32];
        int ret,nrents,strsize;
```

```
if(get_user(c,buf) || size<2)
        return -EFAULT;

switch(c) {
        case 's':
        case 'S':
                strsize=MIN(sizeof(kbuf),size-1);
                if(copy_from_user(kbuf,buf+1,strsize))
                        return -EFAULT;
                kbuf[strsize]=0;
                ret=sscanf(kbuf,"%d",&nrents);
                if(ret!=1 || !nrents)
                        return -EINVAL;

                if(recording) {
                        cli();
                        recording=0;
                        sti();
                }
                if(evtbuf) {
                        vfree(evtbuf);
                        evtbuf=0;
                }
```

For example, here is a place where intensity is there, here is a place the write is known. So, basically, I am going to register the device and then I am going to give it a name the sorry major number, and I am going to set a corresponding routines for what I mean by read write etcetera open close all those things and that is being decided by this set of function pointers. And the function point is being defined here. As a files operations file (Refer Time: 15:36) operations this participant.

You can see that assume some more additional things also, in addition because these are some special things. Because we are doing a they are doing interrupt counting. This is something unusual for this particular driver. Not every driver will be doing these things. So, it is got other additional functions. And what are the and that I am going to initialize. This it turns out that is p enter irq is the name of the function that is called by the kernel on entry of the interrupt and it is user specified rule usually if it is null nothing happens usually that that is what is could happen p enter irq is some kind of a pointer that is sitting there. If it is null nothing happens on entry of the entry in to the interrupt nothing happens. And when you exit nothing happens, nothing know execute no extra code is executed.

But if own take control of the what happens when interrupt happens. In it exit interrupt gives, then I can provide a function pointer, right? Saying execute this code when you enter the thing exit or you execute, and then I am also doing here. So, I am saying p enter irqs and enter irq. So, this is what the color is going to call, but now I am defining you to enter irq and enter irq is the basically here, I am defining them here. What it means is that on entry of whenever an entry whenever interrupt handler is starts execute, you first execute this part before it does it is process, does it processing and it sorry, it does it is processing whatever minimal it does and then it a cause this function right.

So, similarly when it once you leave it does some processing and cross this. So, again you have to look at it very closely, I sort of might have made it slightly set things slightly in the different order than actually should happen normally what will happen is on interrupt entry you do some processing, then you call this function and interrupt exit you call this p enter exit, p enter leave sorry p leave irq and then do the think that the system does.

So, anyway all I am trying to say is that there are functions that are available; that can be executed on every entry of their of the interrupt. And there is a function that is available that can be equal to every time you leave it, and now I am defining it also. So, this module is going to define what has to be done. There is also something called cli here all these things are old stuff current Linux kernels will not support these things. You have to be careful everything keeps changing.

So, this basically says shut off all interrupts all over the system. Because this is a sensitive thing I am trying to get control about what should happen in interrupts, and if interrupt happens while this is going on there can be some mess for example, it could be that interrupts happens just in between. So, there could be some messy situation is developing which you do not have any control over. So, I am saying here shut off all interrupts across all the process.

Once I have safely said it properly then only what is that the cli will not return back until any interrupt that is already happening at the time this call was calling made right. They had to be completed before cli will come here. Anything that is happening at the time when cli is called back in other cpus all the interrupts have been completed. Then only the cli will return therefore, in a completely clean state. So, you can do these things without too much headache you do it properly again sti also will saying it is enable interruption now.

So now all interrupts are now enabled. So, this basically what init module does similarly there is a cleanup module. So, that will essentially say that if you have been recording so far stop recording. So, recording is a variable in the driver you define it here. And basically, this says that whether the recording is going on; that means, I am a tracing the interrupts.

So, when I am leaving in case it is still on. So, this cleanup can actually happen asynchronously, why because your route is sitting there, and again says shut down the device. So, you already said the recording going on, and for some reason I have to shut up I probably I said shut the system completely right. So, I have to clean up the module, I am not done the proper job. Normally what would happen is that I take care of this properly and recording it I stop it etcetera, but sometimes use the super is the administrator can without telling me can shut it also.

So, that is what is being that is. So, basically, I am trying to see if somebodys, when the recording is going on somebody says clean up, I want to do is. So, you will see that there are some routines like ints open ints release. These things do nothing because the pseudo device driver nothing is do in this case sometimes nothing else.

You can also unregister basically saying that I do not need this device any longer, please you can go ahead use uses major number for somebody else. Usually the major numbers are limited because I mentioned it can only be 8-bit quantity. So, you might have 256 devices, a large system 256 devices may not be sufficient. That is why you might want to use it. As I mentioned devices are access by user mode programs by accessing special device files.

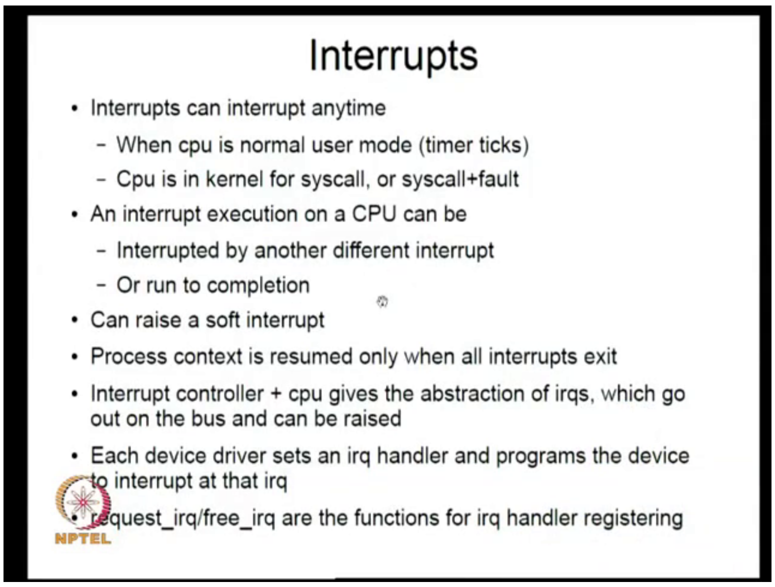
As a majors like slash dev ints, there is a and I can open it by user no programs of course, I have said bit maybe need to have privileges. User mode programs access special device that, I know it says that they are special device files. If you look at it is slash devi ints is that what is called a special file it is not a regular file. And there is a metadata that is kept for each of those things, and the metadata will basically called inode snode sometimes called snode, that will have information saying it is a special device file it is not a regular file.

I need in that particular inode we will have all these details. What is major number, minor number, etcetera. Again, on any other devices on open the corresponding open method is called in our case it is int ints underscore open. Because that is when you load this load this device the open method again is now essentially the pointer that is going to be there actually ints underscore open. And close also send story. Again, the important thing to notice is that open called only on the file object creation not on dups, basically only when the object is created do you call it if some other party also calls at the same time right. If it is already open you saying you just returned back saying it is open only into the fd.

So, you do not do it on every single thing. For example, you know that there is something called a to dup operication dup operation in the file right. So, you can duplicate the file descriptor. Now you are not really creating a object of fresh. So, you do not have to reread any specific operation for setting the device up or the file up. So, on duplicate duplicating fds you do not need to do anything special. So, that is why even if you call open on such fds nothing happens. Similarly close we can close something many times there is some kind of reference count kept, and in the last final call of the close when every other ref every other reference is gone right. That is one on a char will does the file final object relation.

If you have read and write you typically pass a pointer, because there is an implicit pointer. Some of you might know that when you are read and write there is an implicit pointer which says where if you have to read form or where you have to do the write. So, you have to pass this current offset, and that might have to be updated. Again, if you do another seek you have to check whether the lseek offset.

(Refer Slide Time: 24:22)



## Interrupts

- Interrupts can interrupt anytime
  - When cpu is normal user mode (timer ticks)
  - Cpu is in kernel for syscall, or syscall+fault
- An interrupt execution on a CPU can be
  - Interrupted by another different interrupt
  - Or run to completion
- Can raise a soft interrupt
- Process context is resumed only when all interrupts exit
- Interrupt controller + cpu gives the abstraction of irqs, which go out on the bus and can be raised
- Each device driver sets an irq handler and programs the device to interrupt at that irq
- request_irq/free_irq are the functions for irq handler registering

Now, let us because we are doing the interrupts example, let us just advise something you know what interrupts. Interrupts can interrupt any time, when cpus in normal user mode that is am I user program is running here browser is running for example, even if the CPU is the kernel mode, it is cause on a system call it can get interrupted because the interrupt is coming from hardware. In our pseudo device driver, we do not really have any hardware interrupts for our pseudo device driver, but our interrupts coming in from other reasons, other devices, which other drivers with other device drivers are handling.

Because on this system there will also be timer interrupts which is being handled by some other part of the kernel code, but we are talking about pseudo device pseudo device driver which we are writing, but that itself does not have any hardware interrupts coming. Here there is no hardware corresponding pseudo device. Pseudo devices do not have a corresponding hardware, whereas, real device drivers have corresponding hardware device.

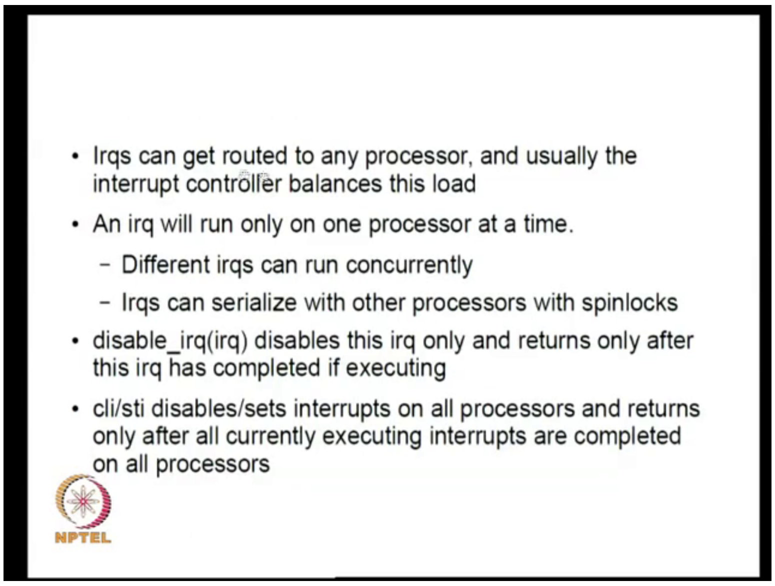
So, the in-interrupt execution on a CPU can be interrupted by another different interrupt. For example, oftentimes the tty interrupt is considered, lower low priority, while the tty interrupt is getting serviced, the disk interrupt or a timer interrupt is coming here takeover.

So, sometimes what happens is that I have multiple devices, disk devices, and multiple disk devices when one interrupt is being handled, another disk drive also interrupts now. And these guys are at the same privilege. So, what you will do in that cases you will first from the first interrupted completion before we did the second (Refer Time: 26:19) that is what you say it can get interrupted or you complete the current thing whatever is happening; generally, in the next time because they will be same level.

Can raise a soft interrupt that happens server system calls process context is resumed only when all interrupts exit that is you go back to user space. Basically, the interrupt controller and CPU interrupt controller basically manage the interrupts and the CPU gives abstraction of irqs. And these are the things that the kernel actually deals with. Because these 2 things are hardware related things.

Each device driver sets in irq handler, and programs the device to interrupt at that irq. So, when the interrupt comes in you are supposed to execute that interrupt request handling. And this request irq free irq are the functions for registering this irq handler, and we are not using any of this in our driver. Because we do not have any real hardware devices that are interrupting us in our pseudo device driver we do not have this thing here.

(Refer Slide Time: 27:34)



- Irqs can get routed to any processor, and usually the interrupt controller balances this load
- An irq will run only on one processor at a time.
  - Different irqs can run concurrently
  - Irqs can serialize with other processors with spinlocks
- disable_irq(irq) disables this irq only and returns only after this irq has completed if executing
- cli/sti disables/sets interrupts on all processors and returns only after all currently executing interrupts are completed on all processors

Irqs can get routed any processor. And usually the interrupt controller balances. This load which can in most Linux systems in the past was not done, but in the recent past. You will find that the interrupts load is handled by all the cpus. So, it gets ported, but an irq will only run only on one processor at a time.

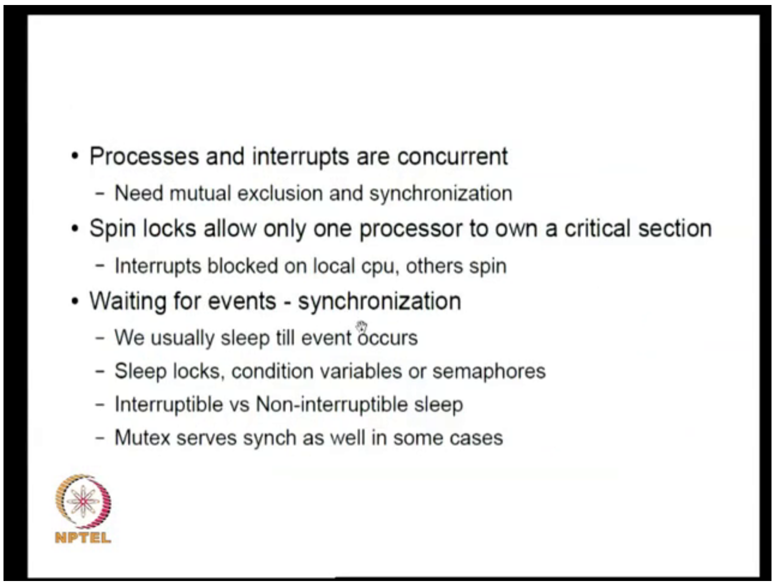
But you can have different interest request processing going on different codes at the same time. And if they are touching the same (Refer Time: 28:15) structure then the better synchronize among themselves. Irqs can serialize with other processes with spinlocks.

Normally interrupts request handling routines cannot sleep. Because it interferes with proper functioning of the system because once you sleep, there possibility is that some code is buggy and you forever sleep. That is possible. The many other reasons also latency etcetera. Other reasons, usually do not allow interrupt request handling because that is happening the highest priority. So, it is happening the highest priority if that guy is going to sleep or something for some reason. The whole system essentially is in a delicate situation now.

So, normally you can if you have to use any serialization with respect to access to a particular data structure where to do it is to spinlock. That is you keep you do not basically the idea is that each of these irqs are short running programs, short running code piece of code. And they will being good citizens and they usually do their job and run out quickly. So, you can actually afford to spinlock. You are using a bit CPU a lot, but since you are doing it for very short period (Refer Time: 29:36) matter that is a (Refer Time: 29:37).

So, there are things like cli sti already mentioned, you disables and sets interrupts on all processors returns only after all currently executing interrupts are completed on all process. There also a disable irq only and returns only after this irq has completed executing.

(Refer Slide Time: 30:02)

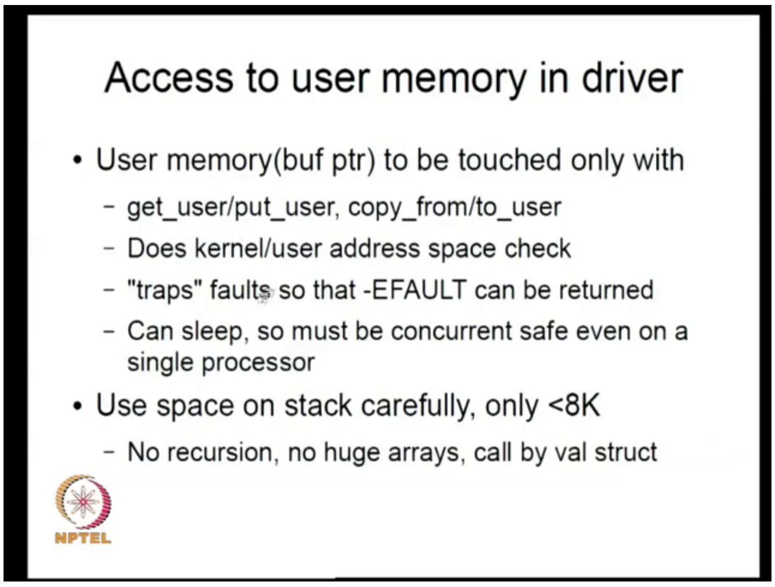


- Processes and interrupts are concurrent
  - Need mutual exclusion and synchronization
- Spin locks allow only one processor to own a critical section
  - Interrupts blocked on local cpu, others spin
- Waiting for events - synchronization
  - We usually sleep till event occurs
  - Sleep locks, condition variables or semaphores
  - Interruptible vs Non-interruptible sleep
  - Mutex serves synch as well in some cases

Other thing is in relation processes and interrupts also concurrent. So, not only interrupts, but it can interfere with regular execution. So, you need mutual exclusion. I have already mentioned spinlocks. You can also wait for events synchronization for example, there are various types of base to synchronize using sleep blocks condition variables semaphores this things sleep. So, this cannot used by interrupt handlers, but they can used by regular kernel code it is possible. You can also have interruptible and non-interruptible sleep you get into non-interruptible sleep and some very delicate situations, but the thing is if it never gets woken up that interrupts does not if that you cannot ever break the sleep all right, the system that have might have that also possible. We have to be careful about this.

I can am I going to do it is about this, but just I just want to be aware of this.

(Refer Slide Time: 31:12)



## Access to user memory in driver

- User memory(buf ptr) to be touched only with
  - get_user/put_user, copy_from/to_user
  - Does kernel/user address space check
  - "traps" faults so that -EFAULT can be returned
  - Can sleep, so must be concurrent safe even on a single processor
- Use space on stack carefully, only <8K
  - No recursion, no huge arrays, call by val struct

Again, you want to sometimes access user memory in the driver. Now it turns out when you want to access some user memory from the driver, because the driver code is running in the kernel because you installed it inside the kernel. It may refer to some data that is available in the user space. As a slight delicate operation, because the kernel code is all powerful, because it is all powerful, it can access things which normally I am I can be load access only with some care right; that means, that if I am the user gives me some invalid piece of data. It is a kernel does not check that it is invalid. And goes and accesses it then there will be a violation with respect to whether the whether with respect to the validity of the data. And the kernel can get into a complicated situation.

Now, it is now suffering an account of user giving wrong information. Typically, you want to avoid those kind of situations, that is why you need to go through specific routines. For example, there are something called copy from copy to user, copy from user means you copy there is a buffer given by the user, you delegate carefully selected look at it see if it is, and then after convincing. And so, you will that everything is fine that all the data that the user is giving is valid then you copy into kernel buffer.

Similarly, you can get user and put user. So, just further instead of comp for example, at a character level for example, we will come to this. Why do we need to use this things? Because I might ask you to I might send some parameter to the driver, let us look at one example. Here is the case where I have a device what is the device doing it is slash dev slash ints, what is you are doing? It is collecting traces of interrupts which the CPU is getting interrupted at what time etcetera.

Now, you can not I once it starts I have to tell it how long to run right. Now it can be run forever let us say. So, we want to run it for after it is collected thousand traces or one million traces. Suppose I want to give that information. I want to tell it the device that I get you started, but I want you to (Refer Time: 34:00) thousand or one million traces. How do I do it? I can write to the device, and that is what I am going to do. I am going to write to the device and said that parameter. Whether it is thousand or one million, that is what is being I am that is whast is going on here.

So, ints write is a code that basically sets the size of the buffer that has to be allocated in the kernel how many entries has to be captured, and then you start the recording process itself. All that is being done here. Let us say how does it. So, what is it doing here? This struct file star file is basically the slash dev int ints. And it is got a buffer. What is the buffer? The buffer is basically where I am giving information about how many the size of the; how many entries had to be captured.

So, what am I going to do here? If get user c buff right, or size less than 2 I am just checking essentially if the data is valid. If my size is less than 2 figure out size will equal to 1, or if I find that get user does not give me any characters at all this basically error condition. I am not able to get any useful, then something is wrong with it I exit get user is basically saying that somebody is giving me here this is users space buffer. And I am instead just saying get the character, I do not want to get directly. I want to go through a function called get underscore user, which is return by kernel which is returned and provided by the kernel people.

So, I am going to use a carefully written code which checks whether this particular buffer is a legitimate buffer. It is not some invalid piece of memory. Whatever and mapped memory those kind of things. I am checking all these things. After checking it I get some if you find this an error condition and whatever I exit. I would not touch it that is it.

So, again you can see it might have got one character. What is that? It is basically the normally you will notice that many routines have you can get a you can give some command line parameters. And that common line palette is s. You can say s is the size of the buffer or the number of entries I am going to say. So, you are going to run this program saying the name of the program followed by minus s followed by one million or thousand. So, what this get user is doing is getting that s followed by thousand or s followed by one million that is what. This is doing and once it has got the first character correctly that is checking. It is thus is the command line parameter switch right basically I have some options right. So, the option is it s for example. So, it is getting the first option. It is taken with s or capital S. If it is s or s means I am going to use the number entries that have to be captured as a part of disk tracing exercise. That is what it uses.

So, then basically what is going with I am going to copy the rest of it. See the buffer is the thing that the user has given. It has confirmed to itself that this buffer is legitimate buffer that is why it came here. Otherwise it will die here itself it will return efault and out if it invalid buffer etcetera. But it successfully went came here; that means, the character c was populated with one character. So, it got s. So, either it is s or it is some of the junk value, then that case it does default it just says something is wrong.

So, it got the first character it is somebody is trying to give it some size, how many entries have to be captured by the tracing. Now he is going to say copy from user from buffer plus 1 because character already the first character s is already. On character in from the buffer plus 1 and I am going to get as many as this number of I given by this in this size string size.

And I am going to put into the kernel buffer kbuf is something I defined here, I am defining 32 bit. And to make sure that I do not do any go beyond 32, I am making it min of that 1 min of size of kbuf 32 comma size minus 1. I am just always keeping sure that string sizes less than the is the minimum of size minus 1 and size of kbuf this for safety. And then I am setting kbuf string size equal to 0, but it is terminates the essentially it terminates the value which is either 10,000 or one million whatever it is means. It character because I am doing it at in character form on the console, in the command line.

So, basically what is happened is that; you will find that kbuf inside the driver has either 1 0 0 0 or 1 0 0 whatever something, and that is what I am going to do, sscanf I am going to get it as number of entries. So, the number of entries is wrong then of course, I exit. So, if the number entries is fine. Then because as a part of the write, right; I am going to do the recording, but before I do that that recording is going to be done here actually.

But before that it may be that I am in some anomalous situation, then that devices device drivers handling very careful. How you come to situation sometime may not know because that this is user sitting there the supervisors are there and how many people have access to this device we do not know. So, he is basically saying I do not know the current situation; I am going to now get myself into a clean situation. So, that is why I am going say if for some reason I am recording at this stage already I do not know how it happened. I am going to say I do not know why it is now the way I am recording I do not know why it is.

Because I was told to write and it is recording some something strange is going. I do not care what it is I am going to say. I am no longer recording that is what I am doing. But a before I say record equal to 0 I am going to clear interrupts and again now enable the interrupts. And again, I have an event buffer. What is event buffer? Basically, this is what is capturing the events. What is an event here? Every interrupt is an event. So, I am capturing some piece of information about it. I am going to have a corresponding event structure. That is event buff. And I am checking whether this event buff is already has some value. Which I didn't expect because I am supposed to doing, and setting how many interests we captured right.

So, because of some reason I do not know how it happened. I am being careful here. I am just saying whatever happened I do not care how it happened I am going to free this structure. And I want to get a clean state I will start from there, again goes from in to again. And I am going to free the thing v free is a particular call in the kernel, which is basically for freeing pageable kernel memory.

Now, in the kernel it turns out you can have unpageable kernel memory, and pageable kernel memory. Unpageable kernel memory is dot memory which cannot be page dot, because if you try to page it out when you want it may be sitting in the disk and you might get a some greatness. So, it turns out you cannot allow those kinds of situations happen.

So, there are 2 types of memory. Of course, the many more types, but at the first time being just say that there are 2 types. So, vfree basically saying that somehow some buffer evtbuf in virtual pageable memory in the kernel has been allocated which I do not know uninstall how it happened, but I am going to free it right, now that is one point of view.

Now, I am actually going to now I know how many entries I should provide. Because I just read the size here right. I got the size of how many entries have to be there right. On this particular right call I am being told how many entries to capture right. The something here which I didn't expect, but I am just clearly out all I am going to be in a safe situation.

Now, I am going to say I am going to allocate that many number of events. So, I can store that number of events information corresponding that. I am going to use something called vmalloc. Again, this is the example of pageable kernel, I am going to ask it to get me that amount. And that is going to be number of entries multiplied by size of this event. So, eventbuf is basically some you getting the starting address of so many entries that have been allocated in the kernel log. And this is pageable kernel memory. And it is some further reason I it is no you know it is written 0; that means, are some bad stuff has happened I do not know what it is I am going to delocalize.

Now, these are starting address, and basically, I have to start recording it from next event; which I am going to start as a starting point eventbuf. So, next event is where I always keep putting stuff. So, in the beginning it has to be at the very beginning that is what it is. So, after one event will be pointing to the next one.

Last event is where I should end. So, I have allocated this much. I am putting saying next should start from here, and then it comes to the last one stop. There that is what it is. Now once I have done all the initialization properly, according to the information I have been told to use right. Because previously I have been told to use this information, that I am going to see whatever is there in this buffer, get that number, I allocate that number of entries and that is what I have done all this here.

Once I have done this now I again I can now allow recording to start happening it. This recording equal to 0 is just because I didn't understand why it is the situation the way it is and just getting to I am just want to be seen, that is all. Here is a case where I know exactly what I am doing I said recording equal to 1 now I moved. So, the right code is basically what is the big? What it is doing is taking the some user has given a buffer. It could be wrong wrongly given by shared mistake, but you could be maliciously given also you do not know how it happened I want to sanitize it first here. All these sanitization happening here in this code and in this code once the sanitization happened I look at the option pick up that value of entries just again this is this paranoia of some kind someone there I do not know why it is there clean up. Then get the allocate the thing then you start recording a walk out. So, that basically there right code.

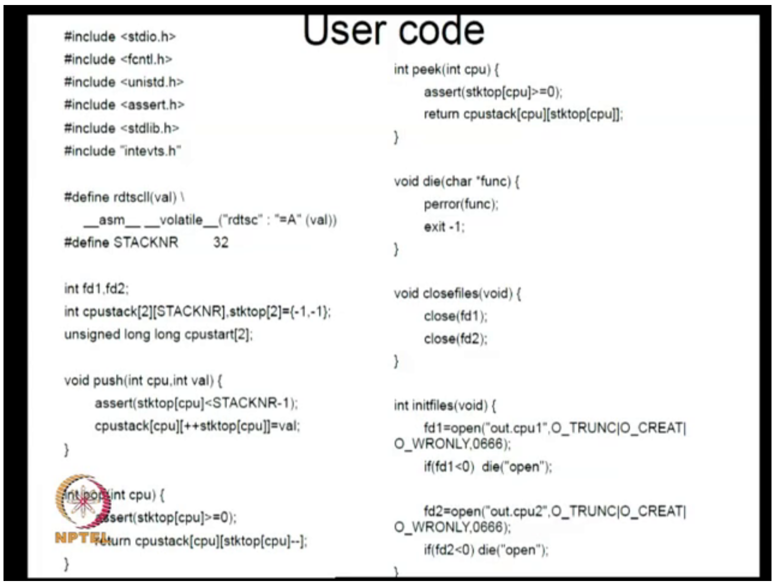
Now, what is the read call read call I should I should just mention also one more thing. So, there is also in the write call there is a offset also. I have mentioned that in all these devices there is a implicit pointer right, when you do a read and write I think all of you know that the implicit pointer gets updated.

Now, in this particular device is not used properly let us say, this is not much of used here. But all the same you are doing the implicit pointer update here that is what we (Refer Time: 46:38). So, I was talking about this part. So, basically when you use get user put user copy from etcetera, you does kernel user space address space check just want to validate. So, that is part of it.

So, yeah so, basically what we are doing in this example is; when ints module is loaded read write open close operations are device mapped to driver routines, I thing already mentioned this. First user writes to slash dev ints to set the number of trace entries using the command line and start trace recording in an internal kernel buffer. So, this what you to looked in already.

Now, what we have to do is using the command line user runs a program that reads each entry and prints each trace. So, already it has collected that amount of information right. You see the you asked to record it you set up the thing, it is already called kept in a kernel buffer it is sitting in the kernel buffer. So now, this pageable that is why I am not. So, bothered because if that particular once it has been done right, it let us say the one megabyte it will be that swapped out it is it will be certain it is not it not be using real physical memory (Refer Time: 48:03). So, after some time after I recorded all this stuff, I want to read it and print it out.

(Refer Slide Time: 48:14)



## User code

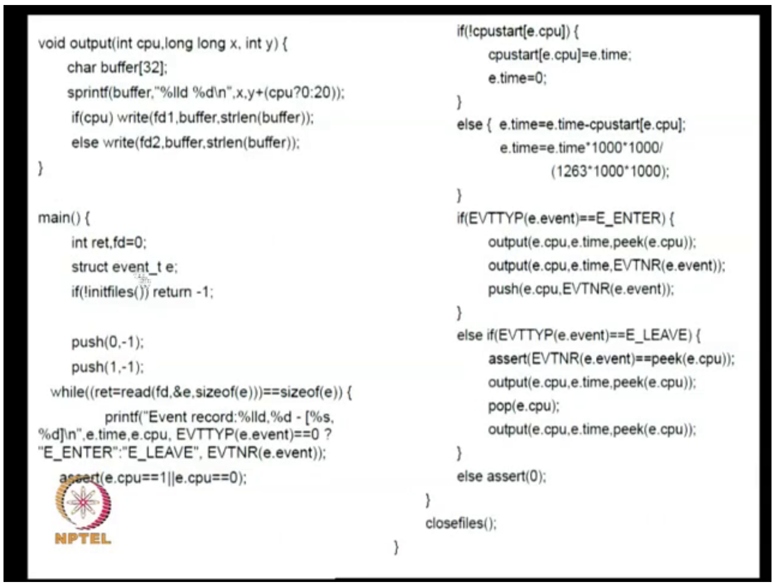
```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <assert.h>
#include <stdlib.h>
#include "intevts.h"

#define rdtscll(val) \
    __asm__ __volatile__("rdtsc" : "=A" (val))
#define STACKNR       32

int fd1,fd2;
int cpustack[2][STACKNR],stktop[2]={-1,-1};
unsigned long long cpustart[2];

void push(int cpu,int val) {
      assert(stktop[cpu]<STACKNR-1);
      cpustack[cpu][++stktop[cpu]]=val;
}

int pop(int cpu) {
      assert(stktop[cpu]>=0);
      return cpustack[cpu][stktop[cpu]--];
}
```

```
int peek(int cpu) {
      assert(stktop[cpu]>=0);
      return cpustack[cpu][stktop[cpu]];
}

void die(char *func) {
      perror(func);
      exit -1;
}

void closefiles(void) {
      close(fd1);
      close(fd2);
}

int initfiles(void) {
      fd1=open("out.cpu1",O_TRUNC|O_CREAT|
O_WRONLY,0666);
      if(fd1<0)  die("open");

      fd2=open("out.cpu2",O_TRUNC|O_CREAT|
O_WRONLY,0666);
      if(fd2<0) die("open");
}
```

So, that is a program. And this program is here that is user code. This is actual code which actually reads a stuff.

So, it is sitting in buffer right, and I now have to read it also.

(Refer Slide Time: 48:27)



```
void output(int cpu,long long x, int y) {
      char buffer[32];
      sprintf(buffer,"%lld %d\n",x,y+(cpu?0:20));
       if(cpu) write(fd1,buffer,strlen(buffer));
       else write(fd2,buffer,strlen(buffer));
}

main() {
       int ret,fd=0;
       struct event_t e;
       if(!initfiles()) return -1;

       push(0,-1);
       push(1,-1);
   while((ret=read(fd,&e,sizeof(e)))==sizeof(e)) {
             printf("Event record:%lld,%d - [%s,
%d]\n",e.time,e.cpu, EVTTYP(e.event)==0 ?
"E_ENTER":"E_LEAVE", EVTNR(e.event));
      assert(e.cpu==1||e.cpu==0);
```

```
      if(!cpustart[e.cpu]) {
            cpustart[e.cpu]=e.time;
            e.time=0;
      }
      else {  e.time=e.time-cpustart[e.cpu];
              e.time=e.time*1000*1000/
                        (1263*1000*1000);
      }
      if(EVTTYP(e.event)==E_ENTER) {
            output(e.cpu,e.time,peek(e.cpu));
            output(e.cpu,e.time,EVTNR(e.event));
            push(e.cpu,EVTNR(e.event));
      }
      else if(EVTTYP(e.event)==E_LEAVE) {
            assert(EVTNR(e.event)==peek(e.cpu));
            output(e.cpu,e.time,peek(e.cpu));
            pop(e.cpu);
            output(e.cpu,e.time,peek(e.cpu));
      }
      else assert(0);
   }
   closefiles();
}
```

So, here is the this is not driver code these use this is not. So, this is not going to loaded in the kernel, this is drive this is basically user code, and this is being run by you this driver code before had to be loaded explicitly into the kernel. This one it does not piece of code which is there exit you compile it, and you can (Refer Time: 48:56) immediately.

And what is it doing? It is first initializing something basically it is opening some files. And so, that it is it is got 0 information. That is why it is in case it was there before, I find it multiple times in the past, for some reason it is got some data I want to thank it I want to.

So, basically the idea is that this file out CPU 1. Word, it is got all the trace events of CPU 1. Out CPU it is what it is all the trace events of CPU 2. What all the interrupts came there. And in the beginning after at the after this particular code in this particular file should have 0 information 0 bytes. This also should have 0 bytes. Because I am not at written it some stuff is already there in the kernel buffer sitting there, but not read it in I am not gotten into my file system write it is string in memory. Still virtual memory that is what I have done here. And it is trying to see this particular piece of code is checking for correctness. So, it has got 2 stacks CPU stack 2 it is assuming 2 cpus they say this 2 here.

So, the thing is that reason why you has got a stack is because, when I am doing one interrupt another interrupt can coming it is getting stacked. It is trying to keep track off it is checking whether the when interrupt coming in and exiting for example, they are properly bracketed, that is what is trying to do. That is why it is the 0 is basically if you look at push, int CPU int val, it saying for assert by it basically it is you can say what is doing. CPU stacks 0 equal to val it is basically putting a minus 1 for the stack correspondence CPU first CPU setting the value minus. So, is the bottom of the stack the stack has got a bottom got minus 1 into bottom. That is on your step same thing with corresponding to the CPU 2 that also is got minus 1.

Now, you read if you read from the entries that have been already saved. So, basically all these are the entries, and you got the entries and you are doing some checkings here. Let us not basically you want to see that if you read something asking you to read one unit of that it should be the same size that we read. Because size of es basically the event size basically that information about the one particular interrupt. You are just checking that whatever we read the first entry should be actually has the size of that event.

This is a assert checking base kind of things. And basically, what we are going to do is; I think you will have to look at some of this code in some more detail, but basically what this does is. It is if there is no there is no time corresponding to a CPU right now I am setting. The time to be e time because if basically, I started at some point and how is it collecting information? It is collecting the information by if you look at it, see when I am doing a read what is it doing. It is using I am not let me see some not done this part let us let us look at a moment.

So, I think you will have to get into some details about, what data was collected how it was collected right. So, as a mentioned now we did the write it was allocated buffers we start the recording right. Now the interrupts start happening. So, when interrupt happens, what happens the interrupt handler would have called p enter or p exit. Normally it is empty nothing happens, but in our case because we loaded this particular driver. We are given it a specific thing p enter and p are going to be enter irq enter leave irq. This routines were specifically done for this particular pseudo device driver. And we are defined enter irq of this and leave irq of this; that means, that on every interrupt entry or exit these things will be executed.

Let us see what is doing here. So, there are lot of details, but let us just look at constant. This one this something called rdts call this is basically read time stamp call. In the most architectures there is something called a time stamp register. And this is updated every instruction. So, for example, if your system is running at 2 gigahertz right and every instruction takes we will just simplified we will say it is this time stamp thing is increment every clock cycle, let us just say that this to be it is simple.

So, when you call this read ts call, it is recording the what is there in the time stamp counter. And look at it this happens to be actually a this particularly routine. Now unix is Linux specifically is very good at mixing a single language code, and c code. Because if you says gcc, and gcc has got very good support for mixing or some other language and regular c code. This is try a tricky business basically because you know that there is something called register allocation that happens. And if you arbitrarily put c code and assembly code together, they might be using the same registers. And all those things you have to get all those things consistently across.

So, there is some support in this is it do all those things correctly start delicate business, but they know how to do it they have done it. So, this this is also another example. It basically is tells you that there is in this value right, next event this pointer time right. What this assembly language code is basically saying you get read that particular value from rdtsc and put it. This basically the instruction and this instruction that is there in the assembler, and this basically is saying this is a parameter that has to be updated, and then parameter that is going to updated is this. This may be allocated a register whatever that something gcc also doing handling it. So, what it means is that, once this thing is executed next event arrow time will have the value of read the read time. The real it is going to have the time stamp counter. In it has read it rdts is the read part basically reside it.

So, that is what the so, basically what is happening here is that you get the time, and then you create a structure which basically says the because enter irq, that that event is actually entry into the interrupt, and then you are going to say that this event happened on this CPU, because this enter irq has if you look at this p enter irq it has got which irq and which CPU actually came as parameters. The same parameter over here and that those things are being recorded here with CPU, and now it is saying that you know that the buffer was here this big and you started from here, and you have to go to the next entry in them from the next update. That is what it is being done here.

We will look at the rest of it in the next class.