Q.2 a. Provide a list of 14 main characteristics of LINUX (no description required) (7)

Answer:

1. Multitasking
2. Multi-user access
3. Multi-processing
4. Architecture independence
5. Demand load executables
6. Paging
7. Dynamic Cache for hard disk
8. Shared Libraries
9. Support for POSIX 1003.1
10. Various formats for executable files
11. Memory protected mode
12. Support for national keyboards and fonts
13. Different files systems
14. TCP/IP, SLIP and PPP support

b. What are the strengths and drawbacks of LINUX? (9)

Answer:

STRENGTHS: The LINUX software is developed under open and distributed conditions. "Open" means that anyone can become involved if they are able to do so. This requires LINUX activists to be able to communicate quickly, efficiently, and above all, globally. The medium for this is the internet. It is therefore no surprise that many of the developments are the product of gifted students with access to the internet at their universities and colleges. The development systems available to these students tend to be relatively modest and therefore LINUX is still the 32-bit operating system that uses the least resources without sacrificing functionality. As LINUX is distributed under the conditions of the *GNU Public Licence* [GPL], the complete source code is available to users. This allows anyone to find out how the system works, and trace and remove any bugs.

DRAWBACKS: LINUX is a "programmer system" like UNIX. Cryptic commands, configurations that are difficult to follow, and documentation that is not always comprehensible make it far from easy to use – and not only for beginners.

Q.3 a. Distinguish between the file structure and inode structure. (6)

Answer:

The inode structure describes a file. The concept *inode* is used more than once in different contexts. Both the data structure in the kernel and the data structure on the hard disk describe files (each from their own viewpoint) and are therefore inodes. Inodes contain information such as the file's owner and access rights. There is exactly one inode entry in the kernel for each file used in the system.

File structures (that is, data structures of the *struct* file type), on the other hand, contain the view of a process on these files (represented by inodes). This view on the file includes attributes, such as the mode in which the file can be used (read, write, read+write), or the current position of the next I/O operation.

b. Explain the system call nice. (6)

Answer:

The system call *nice* is a little more complicated that the system call *getuid*. It expects a number by which the static priority of the current process is to be modified as its argument.

All system calls which process arguments must test the arguments for plausibility.

```
asmlinkage int sys_nice (int increment)
{
int newpriority;
```

Note that a larger argument for sys_nice( ) indicates a lower priority. This makes the name increment for the argument of nice a bit confusing.

```
if (increment  <  0  &&  !capable (CAP_SYS_NICE))
        return  –EPERM;
```

capable( ) checks whether the current process has the right to increase its priority. This is the case with the classical UNIX systems when the process has privileges. LINUX has a concept of subdividing these privileges in a finer way.

The new priority for the process can now be calculated. Among other things, a check is made at this point to ensure that the new priority for the process is within a reasonable range.

```
Newpriority = …

if      (newpriority  <  -20)
        newpriority  =  -20;
if      (newpriority  >  19 )
        newpriority  =  19;
current -> nice  = newpriority;

return 0;
} /* sys_nice */
```

c. Describe any four important states in a process. (4)

Answer:

1. **Running** State
   The task is active and running in non-privileged User Mode. In this case the process will go through the program in a perfectly normal way. This state can only be exited via an interrupt or a system call. System calls are no more than special cases of interrupts. In either case, the processor is switched to the privileged System Mode and the appropriate interrupt routine is activated.
2. **Interrupt Routine** State
   The interrupt routines become active when the hardware signals an exception condition, which may be new characters input at the keyboard or the clock generator issuing a signal every 10 milliseconds.
3. **System Call** State
   System calls are initiated by software interrupts. A system call is able to suspend the task to wait for an event.
4. **Return from system call** State
   This state is automatically adopted after every system call and after some interrupts. At this point checks are made as to whether the scheduler needs to be called and whether there are

signals to process. The scheduler can switch the process to the 'Ready' state and activate another process.

Answer:

Created or new: When a process is created, it occupies the Created or new state

Ready and waiting: A ready or waiting process has been loaded into main memory and is waiting for CPU.

Running: A process moves into Running state when it is chosen for execution.

Blocked: A process is blocked for some event either I/O or some Signal.

Terminated: A process may be terminated either from the Running State or by explicitly being killed

Q.4 a. What are bdflush and kupdate and how are they used? What is the advantage of the combination of bdflush and kupdate? (8)

Answer:

Bdflusg and kupdate are kernel threads that write the buffer back to the hard disk. Kupdate writes the old modified buffers back to the hard disk, including the superblock and inode information.

Kupdate writes all modified buffer blocks that have not been used since a certain time period back to the hard disk, including any superblock and inode information. The kupdate interval used under LINUX is five seconds by default. The time that kupdate waits to write a modified buffer to the disk is 30 seconds by default.

Bdflush writes the number of blocks (standard 64) provided by means of the bdflush parameter to the hard disk in an endless loop. If the total number of modified blocks is higher than a percentage (standard 30) the buffers are written back to the disk. The system call *bdflush* sets the parameters for both kernel threads during the ongoing operation.

The advantage of using the combination of bdflush and kupdate is: the number of block buffers contained in the modified buffer cache is minimized.

b. Provide a complete list of memory page flags along with the respective descriptions. (8)

Answer:

Flags	**Description**
PG_locked	The page is locked

PG_error	This flag indicates an error condition
PG_referenced	This page has been recently accessed
PG_uptodate	This page matches the hard disk contents
PG_free_after	This page should be released after an I/O operation
PG_decr_after	The counter *nr_async_pages* is decremented after reading this page
PG_swap_unlock_after	After reading from the swap space, the page should be unlocked by calling *swap_after_unlock_page* ( ) function
PG_reserved	The page is reserved

Q.5 a. Discuss how Shared Memory is used for inter process communication. (8)

Answer:

Shared Memory is an efficeint means of passing data between programs. One program will create a memory portion which other processes (if permitted) can access.

In the Solaris 2.x operating system, the most efficient way to implement shared memory applications is to rely on the mmap() function and on the system's native virtual memory facility. Solaris 2.x also supports System V shared memory, which is another way to let multiple processes attach a segment of physical memory to their virtual address spaces. When write access is allowed for more than one process, an outside protocol or mechanism such as a semaphore can be used to prevent inconsistencies and collisions.

A process creates a shared memory segment using shmget(). The original owner of a shared memory segment can assign ownership to another user with shmctl(). It can also revoke this assignment. Other processes with proper permission can perform various control functions on the shared memory segment using shmctl(). Once created, a shared segment can be attached to a process address space using shmat(). It can be detached using shmdt(). The attaching process must have the appropriate permissions for shmat(). Once attached, the process can read or write to the segment, as allowed by the permission requested in the attach operation. A shared segment can be attached multiple times by the same process. A shared memory segment is described by a control structure with a unique ID that points to an area of physical memory. The identifier of the segment is called the shmid. The structure definition for the shared memory segment control structures and prototypews can be found in <sys/shm.h>.

b. What is the purpose of socket programming? What is the advantage of using socket? Illustrate with an example. (8)

Answer:

The socket programming interface provides for communication via a network as well as locally on a single computer.

The advantage of this interface is that it allows network applications to be programmed using the long-established UNIX concept of file descriptions.

A particularly good example of this is the INET demon. The daemon waits for incoming network service requests and then calls the appropriate service program with the socket descriptor as standard input and output. For very simple services, the program called need not contain a single line of network-relevant code.

Q.6 a. Describe the two algorithms used by Ext2 file system to limit the fragmentation of files? (4)

Answer:

A. **Target Oriented allocation**.
New data blocks are searched for near a "target block". If this block is free, it is allocated. Otherwise, a free block is sought within 32 blocks of the target block, and if found, is allocated. If this fails, the block allocation routine tries to find a free block which is at least in the same block group as the target block. Only after these avenues have been exhausted are the other groups investigated.

B. **Pre-allocation.**
If a free block is found, a number of the following blocks are reserved (if they are free). The number can be inserted in the *Ext2* superblock, otherwise EXT2_DEFAULT_PREALLOC_BLOCKS (8) blocks are reserved. If the file is closed, the rest of reserved blocks will be released. This also guarantees that as many data blocks as possible are collected into one cluster. Pre-allocation of blocks can be deselected by removing the definition of EXT2_PREAALOCATE from the file <linux/ext2_fs.h>.

b. Discuss about the Superblock of the Ext2 file system. (4)

Answer:

ANS: A partition is divided into a number of block groups, with each block holding a copy of Superblock. Superblock Contains the Control information on the file system, such as number of inodes and blocks. In addition the Superblock Contains information about the times of last mount operation, the last write to the Superblock and the last file system check. It also holds information about the behaviour of the file system in the event of errors, the maximum time interval to the next file system check, a mount Counter and the maximum no. of mount operations

c. Describe the structure of a directory entry in the Ext2 file system. How is an entry deleted? (8)

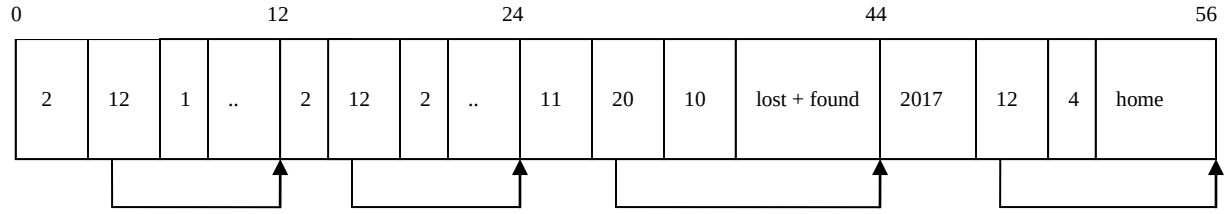
Answer:

In the ext2 file system, directories are administered using a singly linked list. Each entry in this list has the following structure.

```
Struct   ext2_dir_entry
         {
         unsigned      long    inode;      /* inode number              */
         unsigned      short   rec_len;    /* length of directory entry */
```

```
unsigned    short   name_len;       /* length of filename          */
char                name [EXT2_NAME_LEN];   /* filename        */
};
```

The field *rec_len* contains the length of the current entry, and is always rounded up to a multiple of 4. This enables the start of the next entry to be calculated. The *name_len* field holds the length of the filename. It is perfectly possible for a directory entry to be longer than is required to store the filename. **A possible structure is shown below.**



An entry is deleted by setting the inode number to zero and removing the directory entry from the linked list: that is, the previous entry is simply extended. This eliminates the need for shift operations in the directory, which might otherwise exceed the limits of the buffers. However, the 'lost space' is not wasted, but is reused when a new name is entered, either by overwriting an entry with a value of 0 or by using the additional space by removal of the link.

Q.7 a. What is a device driver and what special attention must be taken while implementing a device driver? (8)

Answer:

Device drives are a collection of routines, which write magical numbers to magical places in the hardware. So that these routines can be sensibly fitted into an OS, an interface is implemented, which can be called by the OS with definite actions to be carried out on the hardware. The OS, in addition to hardware access, takes on the task of coordinating resource distribution. The device driver runs in the memory of the kernel, and therefore has the same rights as the kernel. Therefore special care and attention should be taken when implementing a device driver, as an entry in the wrong register can have devastating consequences. The device driver should always use the resources as economically as possible, especially if it is permanently inserted in the kernel.

b. How many broad types of devices are allowed in LINUX? Describe them.(4)

Answer:

There are two basic types of device: block-oriented devices and character-oriented devices.

Block devices are those to which there is random access, which means that any block can be read or written to at will. Under LINUX, these read and write accesses are handled transparently by the cache. Random access is an absolute necessity for file systems, which means that they can only be mounted on block devices.

Character devices, on the other hand, are devices which can usually only be processed sequentially and are therefore accessed without a buffer. This class includes the commonest hardware, such as sound cards, scanners, printers and so on, even where internal operation uses blocks. These blocks, however, are sequential in nature, and cannot be accessed randomly.

c. Briefly describe four different transfer operation modes supported by the DMA controller. (4)

Answer:

Demand transfer operation mode

In this mode, the DMA controller continues transferring data until the terminal count is reached or the device deactivates the DREQ. The transfer is then suspended until the device reactivates the DREQ.

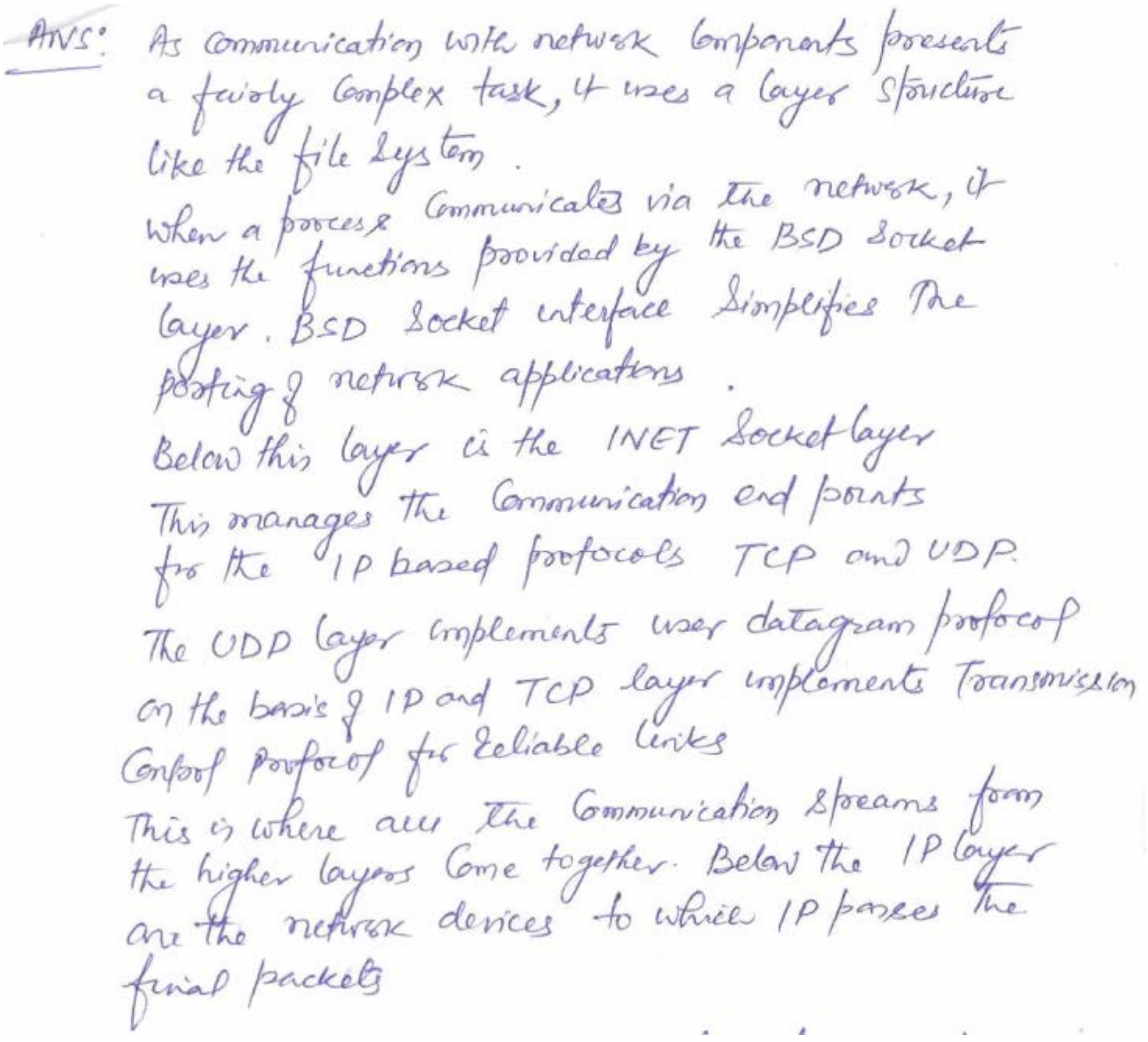
Single transfer operation mode
In this mode, the DMA controller transfers one value at a time and then returns the bus to the processor. Each further transfer must be requested by the DREQ signal or an access to the request register. This mode is used for slow devices, such as floppy disks and scanners.

Block transfer operation mode
In this mode, the DMA controller carries out a block transfer without relinquishing the bus. The transfer initiated by a DREQ.

Cascade operation mode
Cascading of another DMA controller: in this mode the DMA controller passes requests it receives and thus enables more than one controller to be used. By default, DMA channel 0 of the second controller (or DMA channel 4 in consecutive numbering), which is the master in the AT, is in this mode.

Q.8 a. Briefly explain the layer model of the network implementation using TCP/IP. (8)

Answer:


Ans: As communication with network components presents a fairly complex task, it uses a layer structure like the file system.
When a process communicates via the network, it uses the functions provided by the BSD socket layer. BSD socket interface simplifies the porting of network applications.
Below this layer is the INET socket layer
This manages the communication end points for the IP based protocols TCP and UDP.
The UDP layer implements user datagram protocol on the basis of IP and TCP layer implements Transmission Control Protocol for reliable links
This is where all the communication streams from the higher layers come together. Below the IP layer are the network devices to which IP passes the final packets

b. What are the differences between SLIP and PLIP? (8)

Answer:

The most significant difference between SLIP and PLIP is that one protocol uses the computer's serial interface for data transfer while the other transfers data via the parallel port. When we speak of the parallel interface here, we do not mean Ethernet pocket adapters but the "bare" interface.

While PLIP enables a very powerful link to be set up between two computers, SLIP is the simplest way of connecting a computer or a local network to the internet via a serial link (a modern connection to a telephone network). SLIP and PLIP differ from Ethernet in that they can only transfer IP packets. For simplicity, SLIP does not even use a hardware header, nor does PLIP make great demands. It simply sets the hardware address to "fd:fd" plus the IP address and then uses the Ethernet functions for the protocol header.

Q.9 a. List the eight Macros for modules along with their functions. (8)

Answer:

Macro	Functions of the macro
MODULE_AUTHOR(name)	Author of the module
MODULE_DESCRIPTION(desc)	Brief description of the module
MODULE_SUPPORTED_DEVICE(dev)	Device that is implemented by the module
MODULE_PARM(var, type)	A module parameter
MODULE_PARM_DESC(var, desc)	Brief description of the module parameter
EXPORT_SYMBOL(var)	Export the variables or functions
Module_init(func)	Defines the *func* function for the module as the *init* function
Module_exit(func)	Defines the *func* function for the module as the *cleanup* function

b. Explain using diagram the symmetric multiprocessing (SMP) system with two processors. (8)

Answer:

Figure 10.1 shows the hardware overview of a typical SMP system with two processors. Both are connected via the ICC (*Interrupt Controller Communications*) bus with one or more I/O-APICs (*Advanced Programmable Interrupt Controller*). Pentium processors have their own integrated local APIC. These local APICs, together with the I/O-APICs, constitute a unit which deals with the distribution of incoming interrupts.

One processor is preferred by the BIOS. This one is called the boot processor (BSP) and is used for system initialization. All other processors are called application processors (AP) and are initially halted by the BIOS. The MP specification defines a configuration structure which is filled in by the BIOS and informs the operating system about the existing MP system. The BIOS initially forwards all interrupts to the boot processor, so that single-processor operating systems see no difference and only run on the BSP.

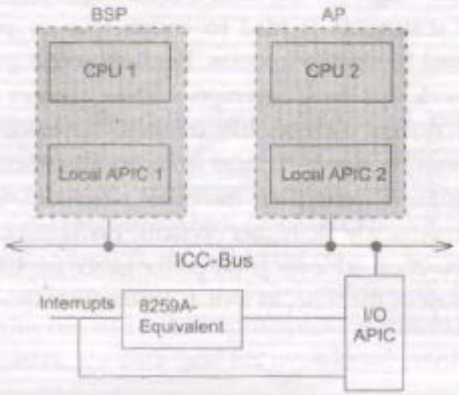


Figure 10.1: A typical SMP system with two processors.

TEXT BOOK

I. Linux Kernel Internals, M. Beck, H. Bome, et al, Pearson Education, Second Edition, 2001