QUESTION BANK

Name of the Department : M.E - COMPUTER SCIENCE AND ENGINEERING

Subject Code & Name : CP5153 OPERATING SYSTEM INTERNALS

Year & Semester : I & I

UNIT I INTRODUCTION

1. List the major objectives of the operating system.

- ✓ Interact with the hardware components, servicing all low-level programmable elements included in the hardware platform.
- ✓ Provide an execution environment to the applications that run on the computer system (the so-called user programs).

2. Define multiuser system.

In a multiuser system, each user has a private space on the machine. Typically, he owns some quota of the disk space to store files, receives private mail messages, and so on. The operating system must ensure that the private portion of a user space is visible only to its owner. In particular, it must ensure that no user can exploit a system application for the purpose of violating the private space of another user.

3. What is a root or super user?

The UNIX operating system has a special user called root or super user. The system administrator must log in as root to handle user accounts, perform maintenance tasks such as system backups and program upgrades, and so on. The root user can do almost everything, because the operating system does not apply the usual protection mechanisms to her. The root user can access every file on the system and can manipulate every running user program.

4. Define process.

A process can be defined either as "an instance of a program in execution" or as the "execution context" of a running program. A process executes a single sequence of instructions in an address space. The address space is the set of memory addresses that the process is allowed to reference. Modern operating systems allow processes with multiple

execution flows that is, multiple sequences of instructions executed in the same address space.

5. Define operating system or kernel.

Each computer system includes a basic set of programs called the operating system. The most important program in the set is called the kernel. It is loaded into RAM when the system boots and contains many critical procedures that are needed for the system to operate. It can provide a wide variety of interactive experiences for the user as well as doing all the jobs the user bought the computer. The essential shape and capabilities of the system are determined by the kernel. The kernel provides key facilities to everything else on the system and determines many of the characteristics of higher software.

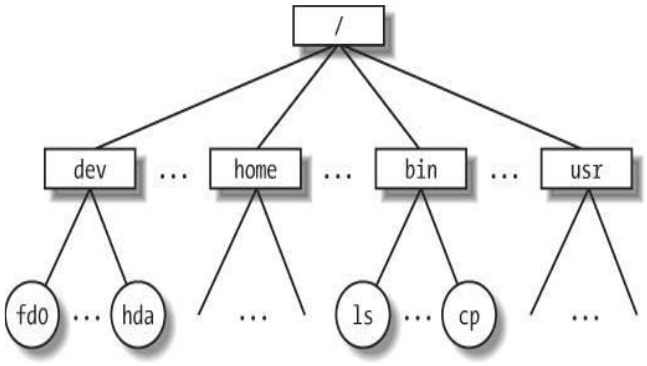
6. Differentiate monolithic and microkernel.

In monolithic kernel, each kernel layer is integrated into the whole kernelprogram and runs in Kernel Mode on behalf of the current process. Most UNIX kernels are monolithic. In contrast, microkernel operating systemsdemand a very small set of functions from the kernel, generally including a few synchronization primitives, asimple scheduler, and an inter-process communication mechanism. Several system processes that run on top ofthe microkernel implement other operating system-layer functions, like memory allocators, device drivers,and system call handlers.

7. What is a module?

A module is an object file whose code can be linked to (and unlinked from) the kernel at runtime. The object code usually consists of a set of functions that implements a file system, a device driver, or other features at the kernel's upper layer. The module does not run as a specific process. Instead, it is executed in Kernel Mode on behalf of the current process, like any other statically linked kernel function.

8. Draw the tree-structure of a UNIX file system.



9. Differentiate hard and soft links.

A filename included in a directory is called a file hard link or a link. The same file may have several links included in the same directory or in different ones, so it may have several filenames. The UNIX command $ ln p1 p2 is used to create a new hard link that has the pathname p2 for a file identified by the pathname p1.

Symbolic links or soft links are short files that contain an arbitrary pathname of another file. The pathname may refer to any file or directory located in any file system. It may even refer to a nonexistent file. The UNIX command $ ln -s p1 p2 creates a new soft link with pathname p2 that refers to pathname p1.

10. List the advantages of using modules.

- ✓ modularized approach
- ✓ Platform independence
- ✓ Frugal main memory usage
- ✓ No performance penalty

11. Define files.

A UNIX file is an information container structured as a sequence of bytes. The kernel does not interpret the contents of a file. Many programming libraries implement higher-level abstractions, such as records structured into fields and record addressing based on keys. The programs in these libraries must rely on system calls offered by the kernel.

12. What are the limitations of the hard links?

- ✓ It is not possible to create hard links for directories. Doing so might transform the directory tree into a graph with cycles, thus making it impossible to locate a file according to its name.
- ✓ Links can be created only among files included in the same file system. This is a serious limitation, because modern UNIX systems may include several file systems located on different disks and/or partitions, and users may be unaware of the physical divisions between them.

13. List the file types used in UNIX.

UNIX files may have one of the following types:

- ✓ Regular file
- ✓ Directory
- ✓ Symbolic link

- ✓ Block-oriented device file
- ✓ Character-oriented device file
- ✓ Pipe and named pipe (also called FIFO)
- ✓ Sockets

14. Define inode.

All information needed by the file system to handle a file is included in a data structure called an inode. Each file has its own inode, which the file system uses to identify the file.

15. List the information's specified in the POSIX standard?

- ✓ File type
- ✓ Number of hard links associated with the file
- ✓ File length in bytes
- ✓ Device ID (i.e., an identifier of the device containing the file)
- ✓ Inode number that identifies the file within the file system
- ✓ UID of the file owner
- ✓ User group ID of the file Several timestamps that specify the inode status change time, the last access time, and the last modify time
- ✓ Access rights and file mode

16. List the file-handling system calls.

- ✓ fd = open(path, flag, mode)
- ✓ newoffset = lseek(fd, offset, whence);
- ✓ nread = read(fd, buf, count);
- ✓ res = close(fd);
- ✓ res = rename(oldpath, newpath);
- ✓ res = unlink(pathname);

17. What are the characteristics of the kernel threads?

UNIX systems include a few privileged processes called kernel threads with the following characteristics:

- ✓ They run in Kernel Mode in the kernel address space.
- ✓ They do not interact with users, and thus do not require terminal devices.
- ✓ They are usually created during system startup and remain alive until the system is shut down.

18. How to activate kernel routines?

Kernel routines can be activated in several ways:

- ✓ A process invokes a system call: The CPU executing the process signals an exception, which is an unusual condition such as an invalidinstruction. The kernel handles the exception on behalf of the process that caused it.
- ✓ A peripheral device issues an interrupt signal to the CPU to notify it of an event such as a request forattention, a status change, or the completion of an I/O operation. Each interrupt signal is dealt by akernel program called an interrupt handler. Because peripheral devices operate asynchronously withrespect to the CPU, interrupts occur at unpredictable times.
- ✓ A kernel thread is executed. Because it runs in Kernel Mode, the corresponding program must beconsidered part of the kernel.

19. List the information's available in the processor register.

When the kernel stops the execution of a process, it saves the current contents of several processor registers in the process descriptor. These include:

- ✓ The program counter (PC) and stack pointer (SP) registers
- ✓ The general purpose registers
- ✓ The floating point registers
- ✓ The processor control registers (Processor Status Word) containing information about the CPU state
- ✓ The memory management registers used to keep track of the RAM accessed by the process

20. Define reentrant kernels.

All UNIX kernels are reentrant. This means that several processes may be executing in Kernel Mode at the same time.

21. What do you mean by process address space?

Each process runs in its private address space. A process running in User Mode refers to private stack, data,and code areas. When running in Kernel Mode, the process addresses the kernel data and code areas and usesanother private stack.

22. Define Semaphores.

A semaphore is simply a counter associated with a data structure. It is checked by all kernelthreads before they try to access the data structure. Each semaphore may be viewed as an object composed of;

- ✓ An integer variable
- ✓ A list of waiting processes
- ✓ Two atomic methods: down( ) and up( )

23. What is a spin locks?

A spin lock is very similar to a semaphore,but it has no process list; when a process finds the lock closed by another process, it "spins" aroundrepeatedly, executing a tight instruction loop until the lock becomes open.

24. What is the use of signals in UNIX?

UNIX signals provide a mechanism for notifying processes of system events. Each event has its own signalnumber, which is usually referred to by a symbolic constant such as SIGTERM. There are two kinds of systemevents:

- ✓ Asynchronous notifications
- ✓ Synchronous notifications

25. List the advantages of a virtual memory.

- ✓ Several processes can be executed concurrently.
- ✓ It is possible to run applications whose memory needs are larger than the available physical memory.
- ✓ Each process is allowed to access a subset of the available physical memory.
- ✓ Processes can share a single memory image of a library or program.
- ✓ Programs can be relocatable that is, they can be placed anywhere in physical memory.
- ✓ Programmers can write machine-independent code, because they do not need to be concerned aboutphysical memory organization.

26. What are the possible ways that the portion of RAM in the virtual memory system is used?

- ✓ To satisfy kernel requests for buffers, descriptors, and other dynamic kernel data structures

- ✓ To satisfy process requests for generic memory areas and for memory mapping of files
- ✓ To get better performance from disks and other buffered devices by means of caches

27. Define kernel memory allocator.

The Kernel Memory Allocator (KMA) is a subsystem that tries to satisfy the requests for memory areas fromall parts of the system. Some of these requests come from other kernel subsystems needing memory for kerneluse, and some requests come via system calls from user programs to increase their processes' address spaces.

28. List the features of a KMA.

- ✓ It must be fast.
- ✓ It should minimize the amount of wasted memory.
- ✓ It should try to reduce the memory fragmentation problem.
- ✓ It should be able to cooperate with the other memory management subsystems to borrow and releasepage frames from them.

29. List the algorithms used by KMA's.

- ✓ Resource map allocator
- ✓ Power-of-two free lists
- ✓ McKusick-Karels allocator
- ✓ Buddy system
- ✓ Mach's Zone allocator
- ✓ Dynix allocator
- ✓ Solaris 's Slab allocator

30. What is the use of device drivers?

The kernel interacts with I/O devices by means of device drivers. Device drivers are included in the kerneland consist of data structures and functions that control one or more devices, such as hard disks, keyboards,mouses, monitors, network interfaces, and devices connected to an SCSI bus. Each driver interacts with theremaining part of the kernel through a specific interface.

UNIT II PROCESSES

1. Define Process?

The term "process" is often used with several different meanings.A process is an instance of a program in execution. It is the collection of data structures that fully describes how far the execution of the program has progressed

2. What is a process state?

As its name implies, the state field of the process descriptor describes what is currently happening to the process. It consists of an array of flags, each of which describes a possible process state.

3. How to identify a process?

The strict one-to-one correspondence between the process andprocess descriptor makes the 32-bit addressof the task_struct structure a useful means for the kernel to identify processes. These addresses are referred to as process descriptor pointers. Most of the references to processes that the kernel makes are through process descriptor pointers.

4. What is the process list?

The process list is a list that links together all existing process descriptors. Each task_struct structure includes a tasks field of type list head whose previous and next fields point respectively to the previous and to the next task_struct element.

5. What kind of relationships may exist among processes?

Processes created by a program have a parent/child relationship. When a process creates multiple children, these children have sibling relationships. Several fields must be introduced in a process descriptor to represent these relationships.

6. How the processes are organized?

The run queue lists group all processes in a TASK_RUNNING state. When it comes to grouping processes in other states, the various states call for different types of treatment, with Linux opting for one of the choices shown in the following list.Processes in a TASK_STOPPED, EXIT_ZOMBIE, or EXIT_DEAD state are not linked in specific lists. There is no need to group processes in any of these three states, because stopped,

zombie, and dead processes are accessed only via PID or via linked lists of the child processes for a particular parent.

7. What is the use of handling wait queues?

A new wait queue head may be defined by using the DECLARE_WAIT_QUEUE_HEAD(name) macro, which statically declares a new wait queue head variable called name and initializes its lock and task_list fields. The init_waitqueue_head ( ) function may be used to initialize a wait queuehead variable that was allocated dynamically.

8. Define process resource limit.

Each process has an associated set of resource limits, which specify the amount of system resources it can use. These limits keep a user from overwhelming the system (it's CPU, disk space, and so on).

9. How to create a lightweight process in Linux?

Lightweight processes are created in Linux by using a functionnamed ***clone( ),*** which uses the following parameters:

i. **fn:** Specifies a function to be executed by the new process.
ii. **arg**: Points to data passed to the fn( ) function.
iii. **flags**:Miscellaneous information.
iv. **child_stack:** Specifies the User Mode stack pointer to be assigned to the esp register of the child process.
v. **Tls:** Specifies the address of a data structure that defines a Thread Local Storage segment for the new lightweight process.
vi. **Ptid:** Specifies the address of a User Mode variable of the parent process that will hold the PID of the new lightweight process.
vii. **Ctid:** Specifies the address of a User Mode variable of the new lightweight process that will hold the PID of such process.

10. What is the use of do_fork( ) function?

The do_fork( ) function, which handles the clone( ), fork( ), and vfork( ) system calls, acts onthe following parameters:

i. **clone_flags:** Same as the flags parameter of clone( )
ii. **stack_start:** Same as the child_stack parameter of clone( )

iii. **regs:** Pointer to the values of the general purpose registers saved into the Kernel Mode stack when switching from User Mode to Kernel Mode.

iv. **stack_size:** Unused (always set to 0)

v. **parent_tidptr, child_tidptr:** Same as the corresponding ptid and ctid parameters of clone()

11. How to create a kernel thread?

The kernel_thread( ) function creates a new kernel thread. It receives as parameters the address of thekernel function to be executed (fn), the argument to be passed to that function (arg), and a set of clone flags(flags). The function essentially invokes do_fork( ) as follows:

do_fork(flags|CLONE_VM|CLONE_UNTRACED, 0, pregs, 0, NULL, NULL);

12. Define process 0.

The ancestor of all processes, called process 0, the idle process, or, for historical reasons, the swapper process,is a kernel thread created from scratch during the initialization phase of Linux. Thisancestor process uses the following statically allocated data structures:

i. A process descriptor stored in the init_task variable, which is initialized by the INIT_TASK macro.

ii. A thread_info descriptor and a Kernel Mode stack stored in the init_thread_union variable and initialized by the INIT_THREAD_INFO macro.

13. How to terminate the process?

In Linux there are two system calls that terminate a User Mode application:The ***exit_group( )*** system call, which terminates a full thread group, that is, a wholemultithreaded application. The main kernel function that implements this system call is calleddo_group_exit( ). This is the system call that should be invoked by the exit() C libraryfunction.

The ***_exit( )*** system call, which terminates a single process, regardless of any other process in thethread group of the victim. The main kernel function that

implements this system call is calleddo_exit( ). This is the system call invoked, for instance, by the pthread_exit( ) function ofthe LinuxThreads library.

14. What is the difference between do_group_exit( ) and do_exit( ) function?

The do_group_exit( ) function kills all processes belonging to the thread group of current.All process terminations are handled by the do_exit( ) function, which removes most references to theterminating process from kernel data structures.

UNIT III FILE SYSTEM

1. Define virtual file system.

To put a wide range of information in the kernel to represent many different types of file systems.There is a ***field or function*** to support each operation provided by all real filesystems supported by Linux. For each read, write, or other function called, the ***kernel substitutes the actual function*** that supports a native Linux filesystem.

2. **List the standard UNIX file types.**
   i. Regular files
   ii. Directories
   iii. Symbolic links
   iv. Device files
   v. Pipes
3. **List the file systems supported by the VFS.**
   i. Disk-based file systems
   ii. Network file systems
   iii. Special file systems

4. What is a disk-based file system?

It ***manages memory space*** available in a local disk or in some other device that imitate a disk. Example: USB flash drive. Some of the well-known disk-based file systems supported by the VFS are:

i. File systems for Linux
ii. File systems for UNIX variants

iii. Microsoft file systems

iv. ISO9660 CD-ROM file system and Universal Disk Format (UDF) DVD file system.

v. Other proprietary file systems such as IBM's OS/2 (HPFS), Apple's Macintosh (HFS), Amiga's Fast File system (AFFS), and Acorn Disk Filing System (ADFS).

vi. Additional journaling file systems originating in systems other than Linux such as IBM's JFS and SGI's XFS.

5. Define network file systems.

These allow ***easy access to files*** included in file systems belonging to other networked computers. Some well-known network file systems supported by the VFS are ***NFS, Coda, AFS*** (Andrew file system), ***CIFS*** (Common Internet File System used in Microsoft Windows), and ***NCP*** (Novell's NetWare Core Protocol).

6. Define special file systems.

A special file system presents non-file elements of an operating system as files so they can be acted on using file system APIs. This is most commonly done in Unix-like operating systems, but devices are given file names in some non-Unix-like operating systems as well.

7. List the objects included in the common file model

The common file model consists of the following object types:

i. **The superblock object**: Stores information concerning a mounted file system. For disk-based file systems, this object usually corresponds to a ***file system control block*** stored on disk.

ii. **The inode object:** Stores general information about a specific file. Each inode object is associated with an inode number, which uniquely identifies the file within the filesystem.

iii. **The file object:** Stores information about the interaction between an open file and a process. This information exists only in kernel memory during the period when a process has the file open.

iv. **The dentry object:** Stores information about the linking of a directory entry (that is, a particular name of the file) with the corresponding file.

8. What is the use of VFS Data Structures?

Each VFS object is stored in a suitable data structure,which includes both the object attributes and a pointer to atable of object methods. The kernel may dynamicallymodify the methods of the object and, hence, it may installspecialized behavior for the object.

9. What are thesystem calls handled by the VFS?

S.NO	SYSTEM CALL NAME	DESCRIPTION
1	mount( ), umount( ), umount2( )	Mount/unmount file systems
2	sysfs( )	Get file system information
3	statfs( ), fstatfs( ), statfs64( ), fstatfs64( ), ustat( )	Get file system statistics
4	chroot( ), pivot_root( )	Change root directory
5	chdir( ), fchdir( ), getcwd( )	Manipulate current directory

10. File Objects

A file object describes how a process interacts with a file it has opened. The object is created when the file isopened and consists of a file structure. The file objectshave no corresponding image on disk, and hence no "dirty" field is included in the file structure to specifythat the file object has been modified.

11. Inode Objects

All information needed by the filesystem to handle a file is included in a data structure called an inode. Afilename is a casually assigned label that can be changed, but the inode is unique to the file and remains thesame as long as the file exists. An inode object in memory consists of an inode structure.

12. List the data structures used by the Linux for a dentry cache.

i. A set of dentry objects in the in-use, unused, or negative state.

ii. A hash table to derive the dentry object associated with a given filename and a given directory quickly. As usual, if the required object is not included in the dentry cache, the search function returns a null value.

13. How to mount a generic file system?

The mount( ) system call is used to mount a generic filesystem. It's sys_mount( ) service routine actson the following parameters:

i. The pathname of a device file containing the filesystem, or NULL if it is not required.
ii. The pathname of the directory on which the file system will be mounted.
iii. The file system type, which must be the name of a registered filesystem.
iv. The mount flags
v. A pointer to a file system-dependent data structure (which may be NULL)

14. What is the use of dentry Objects?

The VFS considers each directory a file that contains a list of files and other directories. Once a directory entry is read into memory, it is transformed by the VFS into a dentry object based on the dentry structure. The kernel creates a dentry object for every component of a pathname that a process looks up. The dentry object associates the component to its corresponding inode.

15. How to mount the root file system?

Mounting the root file system is a two-stage procedure, shown in the following list:

i. The kernel mounts the special rootfs file system, which simply provides an empty directory that serves as initial mount point.
ii. The kernel mounts the real root filesystem over the empty directory.

16. How to Unmounting a Filesystem?

The umount ( ) system call is used to unmount a filesystem. The corresponding sys_umount( ) serviceroutine acts on two parameters: a filename (either a mount point directory or a block device filename) and aset of flags.

UNIT IV MEMORY MANAGEMENT

1. **Justify. Linux adopts the smaller 4 KB page frame size as the standard memory allocation unit.**

This makes things simpler for two reasons:

i. The Page Fault exceptions issued by the paging circuitry are easily interpreted. Either the page requested exists but the process is not allowed to address it, or the page does not exist.
ii. Both 4 KB and 4 MB are multiples of all disk block sizes, hence transfers of data between main memory and disks are in most cases more efficient when the smaller size is used.

2. **Page Descriptors**

The kernel must keep track of the current status of each page frame. The state information of a page frame is kept in a page descriptor. All page descriptors are stored in the mem_map array and each descriptor is 32 bytes long, the space required by mem_map is slightly less than 1%of the whole RAM.

3. **List any five fields of the node descriptor.**

TYPE	NAME	DESCRIPTION
struct zone [ ]	node_zones	Array of zone descriptors of the node
struct zonelist [ ]	node_zonelists	Array of zone list data structures used by the page allocator
int	nr_zones	Number of zones in the node
struct page *	node_mem_map	Array of page descriptors of the node
Struct bootmem_data *	bdata	Used in the kernel initialization phase

4. **Define Non-Uniform Memory Access (NUMA).**

Linux 2.6 supports the Non-Uniform Memory Access (NUMA) model, in which the access times for differentmemory locations from a given CPU may vary. The physical memory of the system is partitioned in severalnodes. The time needed by a given CPU to access pages within a single node is the same. However, this timemight not be the same for two different CPUs.

5. **List the hardware constraints of the 80 x 86 architecture that must deal by Linux kernel.**

i. The Direct Memory Access (DMA) processors for old ISA buses are able to address only the first 16 MB of RAM.

ii. In modern 32-bit computers with lots of RAM, the CPU cannot directly access all physical memory because the linear address space is too small.

6. **How the memory allocation requests can be satisfied?**

The memory allocation requests can be satisfied in two different ways. If enough free memory is available, the request can be satisfied immediately. Otherwise,

some memory reclaiming must take place, and the kernel control path that made the request is blocked until additional memory has been freed.

7. What is the zoned page frame allocator?

The kernel subsystem that handles the memory allocation requests for groups of contiguous page frames iscalled the zoned page frame allocator.

8. List any five fields of the zone descriptor.

TYPE	NAME	DESCRIPTION
unsigned long	free_pages	Number of free pages in the zone.
unsigned long	pages_min	Number of reserved pages of the zone
unsigned long	pages_low	Low watermark for page frame reclaiming; also used by the zone allocator as a threshold value
unsigned long	pages_high	High watermark for page frame reclaiming; also used by the zone allocator as a threshold value
unsigned long []	lowmem_reserve	Specifies how many page frames in each zone must be reserved for handling low-on-memory critical situations

9. What are the components of the zoned page frame allocator?

The component named "***zone allocator***" receives the requests for allocation and deallocation of dynamicmemory. Inside eachzone, page frames are handled by a component named "***buddy system***".

10. List the functions and macros used to request the page frames.

i. alloc_pages(gfp_mask, order)
ii. alloc_page(gfp_mask)
iii. _ _get_free_pages(gfp_mask, order)
iv. _ _get_free_page(gfp_mask)
v. get_zeroed_page(gfp_mask)
vi. _ _get_dma_pages(gfp_mask, order)

11. What are the mechanisms used by the kernel to map page frames in high memory?

The kernel uses three different mechanisms to map page frames in high memory. They are called permanent kernel mapping, temporary kernel mapping, and noncontiguous memory allocation.

12. What is a permanent kernel mapping?

Permanent kernel mappings allow the kernel to establish long-lasting mappings of high-memory page frames into the kernel address space. They use a dedicated Page Table in the master kernel page tables. The pkmap_page_table variable stores the address of this Page Table, while the LAST_PKMAP macro yields the number of entries.

13. What is a Temporary kernel mappings

Temporary kernel mappings are simpler to implement than permanent kernel mappings. They can be used inside interrupt handlers and deferrable functions, because requesting a temporary kernel mapping never blocks the current process. Every page frame in high memory can be mapped through a window in the kernel address space namely, a Page Table entry that is reserved for this purpose. The number of windows reserved for temporary kernel mappings is quite small.

14. What are the ways to avoid external fragmentation?

There are essentially two ways to avoid external fragmentation. They are

i. Use the paging circuitry to map groups of noncontiguous free page frames into intervals of contiguous linear addresses.

ii. Develop a suitable technique to keep track of the existing blocks of free contiguous page frames, avoiding as much as possible the need to split up a large free block to satisfy a request for a smaller one.

15. What is the zone allocator?

The zone allocator is the frontend of the kernel page frame allocator. This component must locate a memoryzone that includes a number of free page frames large enough to satisfy the memory request.

16. List the conditions that must be satisfied by the zone allocator.

The zone allocator must satisfy the following goals:

i. It should protect the pool of reserved page frames.

ii. It should trigger the page frame reclaiming algorithm when memory is scarce and blocking the current process is allowed. Once some page frames have been freed, the zone allocator will retry the allocation.

iii. It should preserve the small, precious ZONE_DMA memory zone, if possible. For instance, the zone allocator should be somewhat disinclined

to assign page frames in the ZONE_DMA memory zone if the request was for ZONE_NORMAL or ZONE_HIGHMEM page frames.

17. List any five fields of the kmem_cache_t descriptor.

TYPE	NAME	DESCRIPTION
Struct array_cache * []	array	Per-CPU array of pointers to local caches of free objects
unsigned int	batchcount	Number of objects to be transferred in bulk to or from the local caches
unsigned int	limit	Maximum number of free objects in the local caches. This is tunable
struct kmem_list3	lists	See next table
unsigned int	objsize	Size of the objects included in the cache

18. When the slab can be released from a Cache?

The slabs can be destroyed in two cases:

i. A timer function invoked periodically determines that there are fully unused slabs that can be released

ii. There are too many free objects in the slab cache.

19. What are the types of caches?

Caches are divided into two types: general and specific. General caches are used only by the slab allocator forits own purposes, while specific caches are used by the remaining parts of the kernel.

20. When the slab can be allocated to a Cache?

A newly created cache does not contain a slab and therefore does not contain any free objects. New slabs areassigned to a cache only when both of the following are true:

i. A request has been issued to allocate a new object.

ii. The cache does not include a free object.

21. What is an alignment factor?

The objects managed by the slab allocator are aligned in memorythat is, they are stored in memory cellswhose initial physical addresses are multiples of a given constant, which is usually a power of 2. This constantis called the alignment factor.The largest alignment factor allowed by the slab allocator is 4,096the page frame size. This means that objectscan be aligned by referring to either their physical addresses or their linear addresses.

22. What are the types of object descriptor?

i. **External object descriptors:** Stored outside the slab, in the general cache pointed to by the slabp_cache field of the cache descriptor. The size of the memory area, and thus the particular general cache used to store object descriptors, depends on the number of objects stored in the slab (num field of the cache descriptor).

ii. **Internal object descriptors**: Stored inside the slab, right before the objects they describe.

23. **Define slab coloring.**

Objects that have the same offset within different slabs will, with a relatively high probability, end up mappedin the same cache line. The cache hardware might therefore waste memory cycles transferring two objectsfrom the same cache line back and forth to different RAM locations, while other cache lines go underutilized.The slab allocator tries to reduce this unpleasant cache behavior by a policy called slab coloring

24. How to release a Slab Object?

The ***kmem_cache_free ( )*** function releases an object previously allocated by the slab allocator to somekernel function. Its parameters are ***cachep***, the address of the cache descriptor and ***objp*** is the address of the object to be released.

25. List any five fields of the vm_struct descriptor.

Each noncontiguous memory area is associated with a descriptor of typevm_struct, whose fields are listed in the below table.

TYPE	NAME	DESCRIPTION
void *	addr	Linear address of the first memory cell of the area

unsigned long	size	Size of the area plus 4,096 (inter-area safety interval)
unsigned long	flags	Type of memory mapped by the noncontiguous memory area
struct page **	pages	Pointer to array of nr_pages pointers to page descriptors
unsigned int	nr_pages	Number of pages filled by the area

26. Define memory pools.

Memory pools are a new feature of Linux 2.6. Basically, a memory pool allows a kernel componentsuch asthe block device subsystemto allocate some dynamic memory to be used only in low-on-memoryemergencies.

27. How to allocate and reallocate a noncontiguous memory area?

The ***vmalloc( )*** function allocates a noncontiguous memory area to the kernel. The parameter ***size***denotes the size of the requested area.

The ***vfree ( )*** function releases noncontiguous memory areas created by vmalloc () or vmalloc_32(), while the ***vunmap ( )*** function releases memory areas created by vmap( ). Both functions have oneparameter that is the address of the initial linear address of the area to be released.

UNIT V PROCESS COMMUNICATION AND PROGRAM EXECUTION

1. Define pipes.

Pipes are an interprocess communication mechanism that is provided in all flavors of UNIX. A pipe is a one-way flow of data between processes: all data written by a process to the pipe is routed by the kernel to another process, which can thus read it.

2. List the advantages of using pipes instead of temporary files.

Using pipes instead of temporary files is usually more convenient due to the following reasons:

i. The shell statement is much shorter and simpler.

ii. There is no need to create temporary regular files, which must be deleted later.

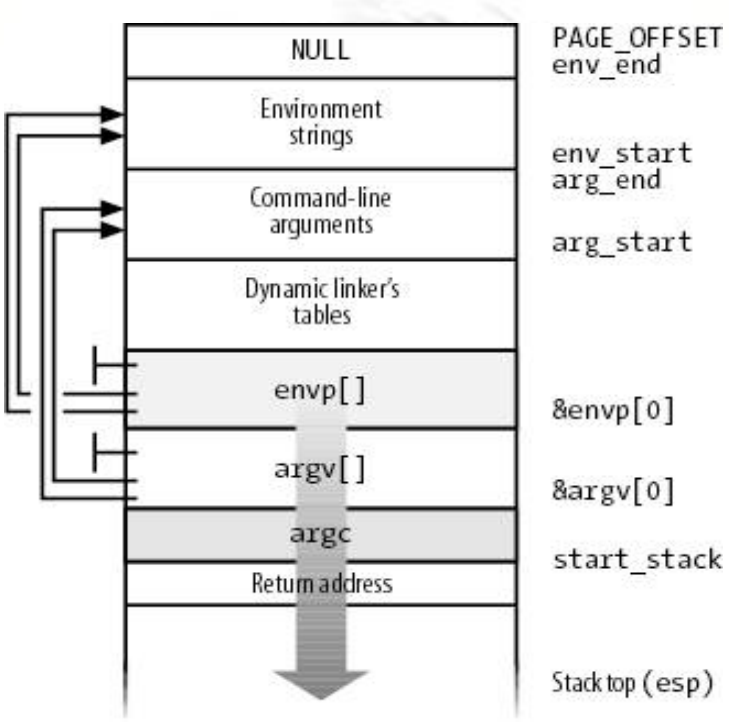
3. How the system call may block the current process?

The system call might block the current process in two cases:

i. The pipe buffer is empty when the system call starts.

ii. The pipe buffer does not include all requested bytes, and a writing process was previously put to sleep while waiting for space in the buffer.

4. Draw the bottom location of the user mode stack.



5. How to create a pipe?

The pipe( ) system call is serviced by the sys_pipe( ) function, which in turninvokes the do_pipe() function. To create a new pipe, do_pipe( ) performs the following operations:

i. Invokes the get_pipe_inode( ) function, which allocates and initializes an inode object for the pipe in the pipefs filesystem.

ii. Allocates a file object and a file descriptor for the write channel of the pipe, sets the flag field of the file object to O_WRONLY, and initializes the f_op field with the address of the write_ pipe_fops table.

iii. Allocates a dentry object and uses it to link the two file objects and the inode object then inserts the new inode in the pipefs special filesystem.

iv. Returns the two file descriptors to the User Mode process.

6. Define libraries.

Each high-level source code file is transformed through several steps into an object file, which contains themachine code of the assembly language instructions corresponding to the high-level instructions. An objectfile cannot be executed, because it does not contain the linear address that corresponds to each reference to aname of a global symbol external to the source code file, such as functions in libraries or other source code files of the same program.

7. What is an uninitialized data segment (bss)?

It contains the uninitialized datathat is, all global variables whose initial values are not stored in theexecutable file (because the program sets the values before referencing them); it is historically calleda bss segment.

8. What is an execution tracing?

Execution tracing is a technique that allows a program to monitor the execution of another program. Thetraced program can be executed step by step, until a signal is received, or until a system call is invoked.Execution tracing is widely used by debuggers, together with other techniques such as the insertion ofbreakpoints in the debugged program and runtime access to its variables.

9. Define executable formats.

The standard Linux executable format is named Executable and Linking Format (ELF). It was developed byUnix System Laboratories and is now the most widely used format in the UNIX world. Several well-knownUNIX operating systems, such as System V Release 4 and Sun's Solaris 2, have adopted ELF as their mainexecutable format.Older Linux versions supported another format named Assembler OUTput Format (a.out).

10. Define exec Functions.

UNIX systems provide a family of functions that replace the executioncontext of a process with a new context described by an executablefile. The names of these functions start with the prefix exec, followed by one or two letters. Therefore, a generic function in the family isusually referred to as an exec function.

11. List the exec functions used in UNIX system.

FUNCTION NAME	PATH SEARCH	COMMAND-LINE ARGUMENTS	ENVIRONMENT ARRAY
execl( )	No	List	No
execlp( )	Yes	List	No
execle( )	No	List	Yes
execv( )	No	Array	No
execvp( )	Yes	Array	No
execve( )	No	Array	Yes

PART – B

1. Explain in detail about basic operating system concepts with necessary diagrams.
2. Explain in detail about UNIX file systems with necessary comments.
3. Write short notes on processes, lightweight processes and threads.
4. Explain in detail about process descriptor with necessary examples and diagrams.
5. Explain in detail about memory management and device drivers used in UNIX.
6. Explain in detail about UNIX kernels with its architecture diagram.
7. Explain in detail about process creation with necessary system calls.
8. Explain in detail about the role of virtual file system with necessary diagrams.
9. Explain in detail about VFS data structures with necessary system calls.
10. Write short note on File system types and pathname lookup.
11. How to lock the files in UNIX. Explain in detail.
12. How to destroy the process in LINUX. Explain in detail.
13. List and explain the various system calls used in UNIX file handling.
14. Explain in detail about implementation of VFS system calls.
15. Explain in detail about page descriptors with necessary system calls.

16. Write short notes on the pool of reserved page frames and kernel mappings of high-memory page frames.
17. Explain in detail about memory zones with necessary system calls.
18. Explain in detail about the zoned page frame allocator with necessary examples and diagrams.
19. Explain in detail about the buddy system algorithm.
20. Explain in detail about execution domains with necessary diagrams and system calls.
21. Explain in detail about executable formats.
22. Explain in detail about non-uniform memory access (NUMA) with necessary system calls.
23. Write short notes on the zone allocator.
24. Explain in detail about executable files.
25. Explain in detail about process communication with necessary diagrams and system calls.