From Sheldon Fan Club

Chapter 5 + 6 = 40 marks

Chapter 1 + 2 + 3 + 4 = 20 marks

Every _*TIP*_ is APPRECIATED.

Upi id –

766175840@apl

kulkarniomkar767@okaxis

Chapter 6

Omkar Kulkarni

Q1. Why there is per user or per process file descriptor table?

Ans: kernel uses the file descriptor as an index into a per-process table, so that every process has a private space of file descriptors.

File descriptor in turn indexes into a system-wide table of files opened by all processes.

Q2. Why processes in kernel mode cannot be preempted? Justify?

Ans: There are critical functions which have to run without interruption.

Q3. Bases on 13 entries of table of content for regular file, What is max size of file a file system support?

Ans: inode contains total 13 pointers (52 bytes per inode!) Assuming pointer requires 4 bytes, n = 256 • Max file size: (10 + 256 + 2562 + 2563) * 1024 = 16 GB Same max file size: 16 GB.

Q4. Which system call is used to send the signal?

Ans: The kill(pid,sig) system call is commonly used to send signals

Q5. Whether a signal handling is optional to a process, if yes/no justify? Which system call is used to make it optional?

Ans: Yes. signal(signum, function) Process can specify special action to take on receipt of certain signals with signal system call. Where signum is the signal number, function is address of the user function that process wants to invoke. If we specify function parameter as 1, future occurrences of signal is ignored by process.

Q6. Advantages and disadvantages of buffer cache.

Ans: ADVANTAGES

1. The use of buffers allows uniform disk access. It simplifies system design.
2. The system places no data alignment restrictions on user processes doing I/O. By copying data from user buffers to system buffers (and vice versa), the kernel eliminates the need for special alignment of user buffers, making user programs simpler and more portable.

3. Use of the buffer cache can reduce the amount of disk traffic, thereby increasing overall system throughput and decreasing response time.
4. The buffer algorithms help insure file system integrity.

DISADVANTAGES

1. Since the kernel does not immediately write data to the disk for a delayed write, the system is vulnerable to crashes that leave disk data in an incorrect state.
2. Use of the buffer cache requires an extra data copy when reading and writing to and from user processes. When transmitting large amounts of data, the extra copy slows down performance.

Q7. Define zombie state. Why it is designed in lifecycle of process?

Ans: a zombie process or defunct process is a process that has completed execution (via the exit system call) but still has an entry in the process table. After a child function has finished execution, it sends an exit status to its parent function. Until the parent function receives and acknowledges the message, the child function remains in a “zombie” state, meaning it has executed but not exited.

Q8. When a process terminates, the kernel performs clean-up, assigns any children of the existing process to be adopted by inti, and sends the death of a child signal to the parent process. Why? / What is orphan process? Who is parent of orphan process? Why?

Ans: When a parent process dies before a child process, the kernel knows that it's not going to get a wait call, so instead it makes these processes "orphans" and puts them under the care of init (remember mother of all processes). Init will eventually perform the wait system call for these orphans so they can die.

Q9. What is system call? Why they are designed in OS?

Ans: A system call is a programmatic way a program requests a service from the kernel, and strace is a powerful tool that allows you to trace the thin layer between user processes and the Linux kernel.

Q10. What information does wait finds when child process invokes exit without a parameter? This is, the child process call exit() instead of exit(n).

Ans: exit (status); where status is the exit code returned to the parent.

pid = wait (stat_addr);

where pid is the process ID of the zombie child, and stat_addr is the address in user space of an integer that will contain the exit status code of the child.

If process call exit without parameter, the parent process won’t get the exit status of child.

Q11. What are the advantages in kernel, when devices are treated as file?

Ans:A device file is a special type of system file that is used to communicate directly with a device. The advantage of treating all devices like files in Unix is that it provides a uniform interface for I/O on Unix systems. Programs written to manage files will also work to manage devices, and vice versa.

Q12. Which memory management technique suitable for multiuser OS?

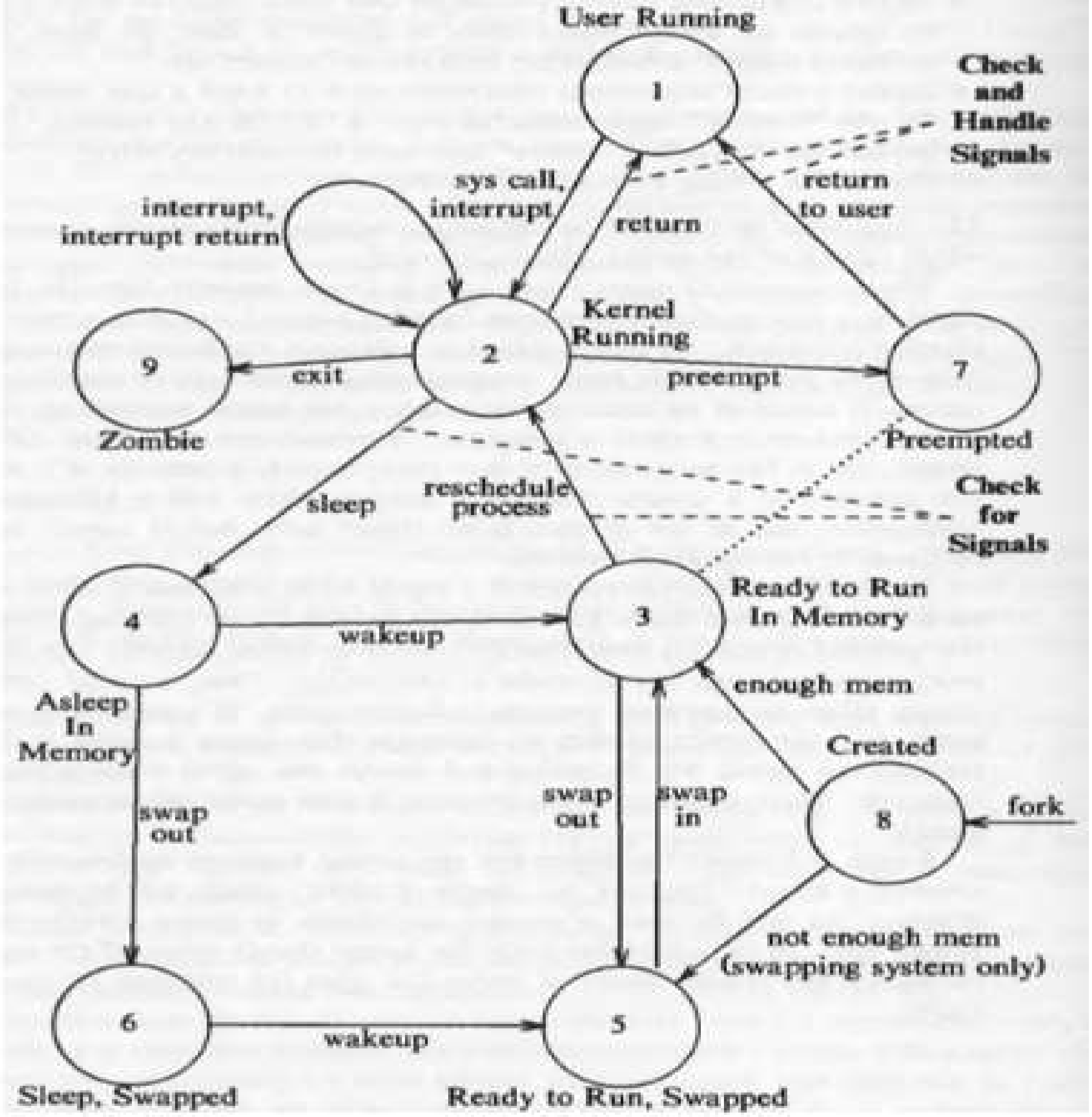
Ans: **time-sharing and batch processing**

Q13. Draw the nine state diagram and show when the signals are handled and checked? Why signals are handled and checked on those transitions only?

Ans: When a kernel or a process sends a signal to another process, a bit in the process table entry of that process is set, with respect to the type of signal received. If the process is asleep at an interruptible priority, the kernel awakens it. A process can remember different types of signals but it cannot remember how many times a signal of a particular type was received.

The kernel checks for receipt of a signal when a process about to return from kernel mode to user mode and when it enters or leaves the sleep state at a suitably low scheduling priority. The kernel handles signals only when a process returns from kernel mode to user mode. This is shown in the following figure:

If a process is running in user mode, and the kernel handles an interrupt that causes a signal to be sent to the process, the kernel will recognize and handle the signal when it returns from the interrupt. Thus, a process never executes in user mode before handling outstanding signals.



Q14. What are the functions (algorithm) carried out by exit system call(process termination). Why exit system call is designed in Kernel for process termination?

Ans: Processes on the UNIX system exit by executing the *exit* system call. When a process *exit*s, it enters the zombie state, relinquishes all of its resources, and dismantles its context except for its process table entry.

exit (status);

where **status** is the exit code returned to the parent.
The kernel may call *exit* on receiving an uncaught signal. In such cases, the value of *status* the signal number.

A system call is how a process requests a service from an operating system's kernel that it does not normally have permission to run. System calls provide the interface between a process and the operating system. Therefore, exit system call is designed in Kernel for process termination.

Q15. How many signals are there in system V UNIX? Give the correspondence between PID and set of processes/process in kill system call for sending the signal? Or What are the system calls that support the processing environment in Kernel? How Kernel uses these system calls for processing?

Ans: There are 19 signals in the System V (Release 2) UNIX system that can be classified as follows:

- Signals having to do with the termination of a process, send when a process *exit*s or when a process invokes the *signal* system call with the *death of child* parameter.
- Signals having to do with process induced exceptions such as when a process accesses an address outside its virtual address space, when it attempts to write memory that is

read-only (such as program text), or when it executes a privileged instruction or for various hardware errors.

- Signals having to do with the unrecoverable conditions during a system call, such as running out of system resources during *exec* after the original address space has been released

- Signals caused by an unexpected error condition during a system call, such as making a nonexistent system call (the process passed a system call number that does not correspond to a legal system call), writing a pipe that has no reader processes, or using an illegal "reference" value for the *lseek* system call. It would be more consistent to return an error on such system calls instead of generating a signal, but the use of signals to abort misbehaving processes is more pragmatic.

- Signals originating from a process in user mode, such as when a process wishes to receive an *alarm* signal after a period of time, or when processes send arbitrary signals to each other with the *kill* system call.

- Signals related to terminal interaction such as when a user hands up a terminal (or the "carrier" signal drops on such a line for any reason), or when a user presses the "break" or "delete" keys on a terminal keyboard.

- Signals for tracing execution of a process.

Processes use the *kill* system call to send signals.

kill (pid, signum);
where **pid** identifies the set of processes to receive the signal, and **signum** is the signal number being sent. The following list shows the correspondence between values of *pid* and sets of processes.

- If *pid* is a positive integer, the kernel sends the signal to the process with process ID *pid*.
- If *pid* is 0, the kernel sends the signal to all processes in the sender's process group.
- If *pid* is -1, the kernel sends the signal to all processes whose real user ID equals the effective user ID of the sender (real and effective user IDs are studied later). If the sending process has effective user ID of superuser, the kernel sends the signal to all processes except processes 0 and 1.
- If *pid* is a negative integer but not -1, the kernel sends the signal to all processes in the process group equal to the absolute value of *pid*.

In all cases, if the sending process does not have effective user ID of superuser, or its real or effective user ID do not match the real or effective user ID of the receiving process, *kill* fails.

Q16. Explain the system boot steps or (Start) algorithm.

Source ~ Internet {

Ans: In Iinux, there are 6 distinct stages typical booting process.

1) BIOS – BIOS stands for Basic input output system. In simple terms BIOS loads and executes the Master Boot record boot loader. Bios search for loader program present in MBR.

2) MBR- MBR stands for Master Boot Record and is it responsible for loading and executing GRUB boot loader.

3) GRUB- Grans Unified BootLoader

The GRUB splash screen is visible to user to choose the Kernel.

4) Kernel- The Kernel that was related by GRUB first mount the root file system. Then it executes init process.

5) Init- At this point system executes run level programs.

6) Run level program-A runlevel is an operating state that is preset on the Linux-based system. }

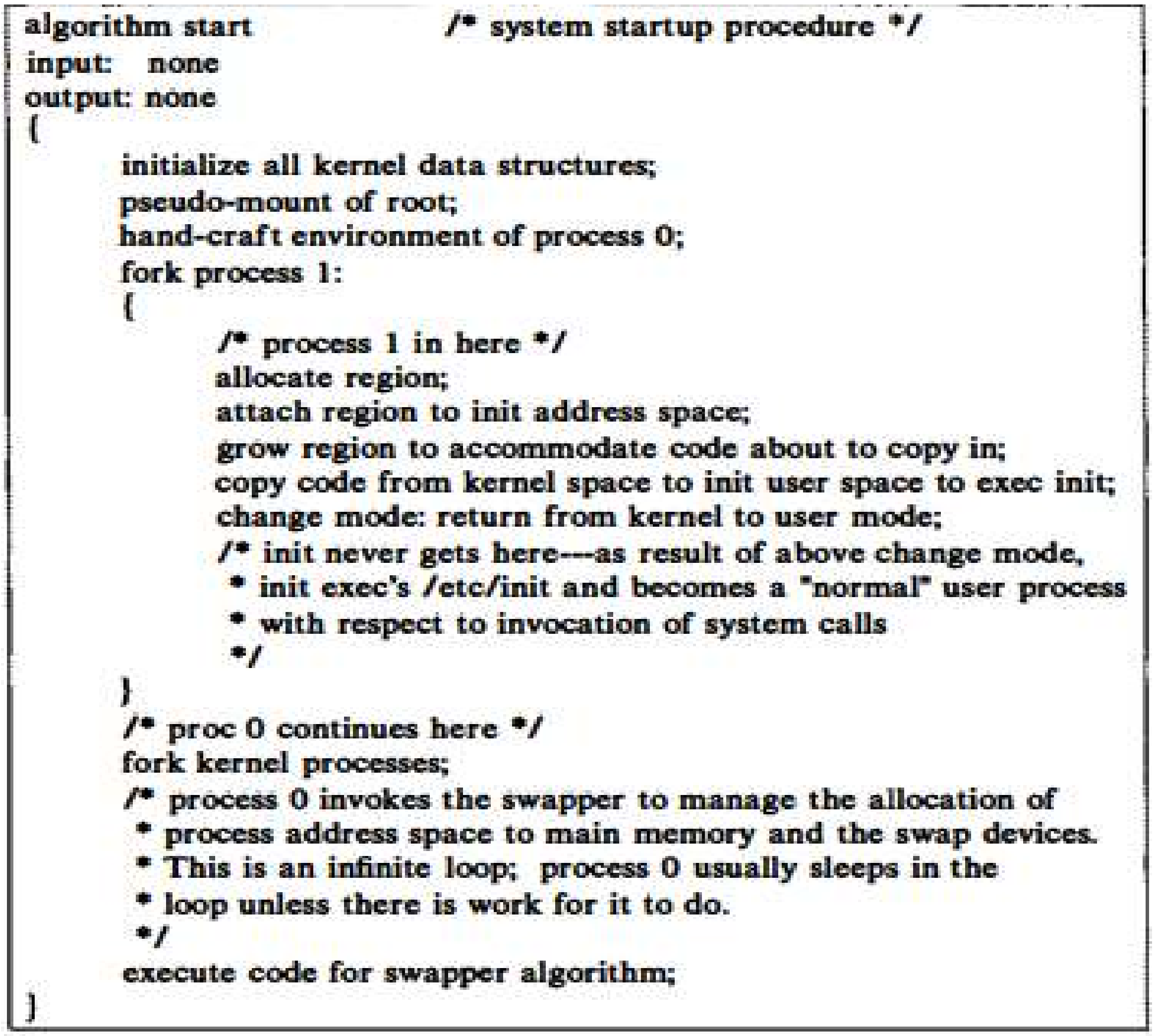
Source ~ Maurice Bach {

Boot procedures vary according to machine type, but the goal is common to all machines: to get a copy of the operating system into machine memory and to start executing it. This is usually done in a series of stages; hence the name bootstrap.

1. The bootstrap procedure eventually reads the boot block (block 0) of a disk, and loads it into memory. The program contained in the boot block loads the kernel from the file system (from the file "/unix".

2. After the kernel is loaded in memory, the boot program transfers control to the start address of the kernel, and the kernel starts running. The kernel initializes its internal data structures.

3. After completing the initialization phase, it mounts the root file system onto root ("/") and fashions the environment for process 0, creating a u



```
algorithm start              /* system startup procedure */
input:   none
output: none
{
        initialize all kernel data structures;
        pseudo-mount of root;
        hand-craft environment of process 0;
        fork process 1:
        {
                /* process 1 in here */
                allocate region;
                attach region to init address space;
                grow region to accommodate code about to copy in;
                copy code from kernel space to init user space to exec init;
                change mode: return from kernel to user mode;
                /* init never gets here---as result of above change mode,
                 * init exec's /etc/init and becomes a "normal" user process
                 * with respect to invocation of system calls
                 */
        }
        /* proc 0 continues here */
        fork kernel processes;
        /* process 0 invokes the swapper to manage the allocation of
         * process address space to main memory and the swap devices.
         * This is an infinite loop;  process 0 usually sleeps in the
         * loop unless there is work for it to do.
         */
        execute code for swapper algorithm;
}
```

area

Figure 7.30. Algorithm for Booting the System

,

initializing slot 0 in the process table and making root the current directory of process 0.

4. When the environment of process 0 is set up, the system is running as process 0. Process 0 forks, invoking the fork algorithm directly from the kernel, because it is executing in kernel mode.

5. The new process, process 1, running in kernel mode, creates its user-level context by allocating a data region and attaching it

to its address space. It grows the region to its proper size and copies code (described shortly) from the kernel address space to the new region.

6. The text for process 1, copied from the kernel, consists of a call to the exec system call to execute the program "/etc/init". Process 1 calls exec and executes the program in the normal fashion. Process 1 is commonly called init because it is responsible for initialization of new processes. }

Q17. What is the difference between fork and vfork? What sequence operation fork does on calling? Or With example explain, how fork system call is different from vfork system calls.

Ans:

	fork	vfork
1	In fork() system call, child and parent process have separate memory space.	While in vfork() system call, child and parent process share same address space.
2	The child process and parent process gets executed simultaneously.	Once child process is executed then parent process starts its execution.
3	The fork() system call uses	While vfork() system call

	copy-on-write as an alternative.	does not use copy-on-write.
4	Child process does not suspend parent process execution in fork() system call.	Child process suspends parent process execution in vfork() system call.
5	Page of one process is not affected by page of other process.	Page of one process is affected by page of other process.
6	There is wastage of address space.	There is no wastage of address space.
7	If child process alters page in address space, it is invisible to parent process.	If child process alters page in address space, it is visible to parent process.

The steps followed by the kernel for *fork* are:

1. It creates a new entry in the process table.
2. It assigns a unique ID to the newly created process.
3. It makes a logical copy of the regions of the parent process. If a regions can be shared, only its reference

count is incremented instead of making a physical copy of the region.

4. The reference counts of file table entries and inodes of the process are increased.

5. It turned the child process ID to the parent and 0 to the child.

Example:

in progress…..

Q18. Why there is need of wait system call between parent and child?

Ans: A call to wait() blocks the calling process until one of its child processes exits or a signal is received. After child process terminates, parent continues its execution after wait system call instruction. In the case of a terminated child, performing a wait allows the system to release the resources associated with the child; if a wait is not performed, then the terminated child remains in a "zombie" state.

Q19. What happen to situation where parent itself to die before the child dies and what Kernel do with the such process?

Ans: If parent itself dies before the child, the kernel disconnects the parent from the process tree by making process 1 (init) adopt all its child processes. That is, process 1 becomes the legal parent of all live children that the exiting process had created. If any of the children are zombie, the exiting process sends init a "death of child" signal so that init can remove them from the process table.

Q20. When a process terminates, the Kernel performs clean-up, assigns any children of the exiting process to be adopted by init, and sends the death of child signal to the parent process. Why? In what state init process is?

Ans: Otherwise, zombie processes would eventually fill the maximum allowed slots of the process table.

Init process is in interruptible sleep (waiting for an event to complete).

Q21. What is an orphan process? Who is parent of orphan process? Why?

Ans: If parent process itself dies before the child process, then child process becomes orphan. the kernel disconnects the parent from the process tree by making process 1 (init) adopt all its child processes. Otherwise, zombie processes would eventually fill the maximum allowed slots of the process table.

Q22. Enlist the system calls used in process control.

Ans:**wait() :**

In some systems, a process may wait for another process to complete its execution. This happens when a parent process creates a child process and the execution of the parent process is suspended until the child process executes. The suspending of the parent process occurs with a wait() system call. When the child process completes execution, the control is returned back to the parent process.

exec() :

This system call runs an executable file in the context of an already running process. It replaces the previous executable file. This is known as an overlay. The original process identifier remains since a new process is not created but data, heap, stack etc. of the process are replaced by the new process.

fork() :

Processes use the fork() system call to create processes that are a copy of themselves. This is one of the major methods of process creation in operating systems. When a parent process creates a child process and the execution of the parent process is suspended until the child process executes. When the child process completes execution, the control is returned back to the parent process.

exit() :

The exit() system call is used by a program to terminate its execution. In a multi-threaded environment, this means that the thread execution is complete. The operating system reclaims resources that were used by the process after the exit() system call.

kill() :

The kill() system call is used by the operating system to send a termination signal to a process that urges the process to exit. However, kill system call does not necessary mean killing the process and can have various meanings.

in progress…..

Q23. State various functions of clock interrupt handler.

Ans: The clock handler adjusts the priorities of all processes in user mode at 1 second intervals (on System V) and causes the kernel to go through the scheduling algorithm to prevent a process from monopolizing use of the CPU.

The clock may interrupt a process several times during its time quantum; at every clock interrupt, the clock handler increments a field in the process table that records the recent CPU usage of the process. Once a second, the clock handler also adjusts the recent CPU usage of each process according to a decay function, decay(CPU) = CPU/2; on System V. When it recomputes recent CPU usage, the clock handler also recalculates the priority of every process in the "preempted but ready-to-run" state according to the formula priority=("recent CPU usage"/2) + (base level user priority) where "base level user priority" is the threshold priority between kernel and user mode described above.

Q24. What is the difference between user mode and Kernel mode and when process moving from user mode to kernel mode.

Ans : In kernel mode, the program has direct and unrestricted access to system resources. In user mode, the application program executes and starts out. In user mode, a single process fails if an interrupt occurs. Kernel mode is also known as the master mode, privileged mode, or system mode.

The transition from user mode to kernel mode occurs when the application requests the help of operating system or an interrupt or a system call occurs. The mode bit is set to 1 in the user mode. It is changed from 1 to 0 when switching from user mode to kernel mode

Q25. What is the stat system call? Enlist and explain the various fields shown by stat system call?

Ans: The system calls stat and /stat allow processes to query the status of files.

The syntax for the system calls is 5.11 stat (pathname, statbuffer);   where pathname is a file name and statbuffer is the address of a data structure in the user process that will contain the status information of the file on completion of the call.

Stat returns information such as the file type, file owner, access permissions, file size, number of links, inode number, and file access times.

Q26. Enlist the function of system administrator.

Ans: in process……………..

Chapter 5

Bhushan Jadhav

Q1. Why kernel data Structure are Static?

Ans: 1.Most kernel data structures occupy fixed-size tables rather than dynamically allocated space.

2. The advantage of this approach is that the kernel code is simple.

3. during operation of the system, the kernel should run out of entries for a data structure, it cannot allocate space for new entries dynamically but must report an error to the requesting user.

4.the kernel is configured so that it it is unlikely to run out of

table space, the extra table space may be wasted because it cannot be used for other purposes.

5.Nevertheless, the simplicity of the kernel allgorithms has generally

been considered more important than the need to squeeze out every last byte of

main memory.

6.Algorithms typically use simple loops to find free table entries, a

method that is easier to understand and sometimes more efficient than more

complicated allocation schemes.

Data Structure for my own OS

1.process Management : Doubly Linked List, Process table

2.File Management : Tree, user file descriptor, File table, Inode table

3.Buffer cache Management : Free list, Hash queue

4.region Management : Stack, Region table, Per process region table

Q2.What are the system level context of the process static and dynamic parts

Ans: Kernel data structures that relate to the process, called as system context

The system level context has a "static part" and a "dynamic part". A process has one static part throughout its lifetime. But it can have a variable number of dynamic parts.

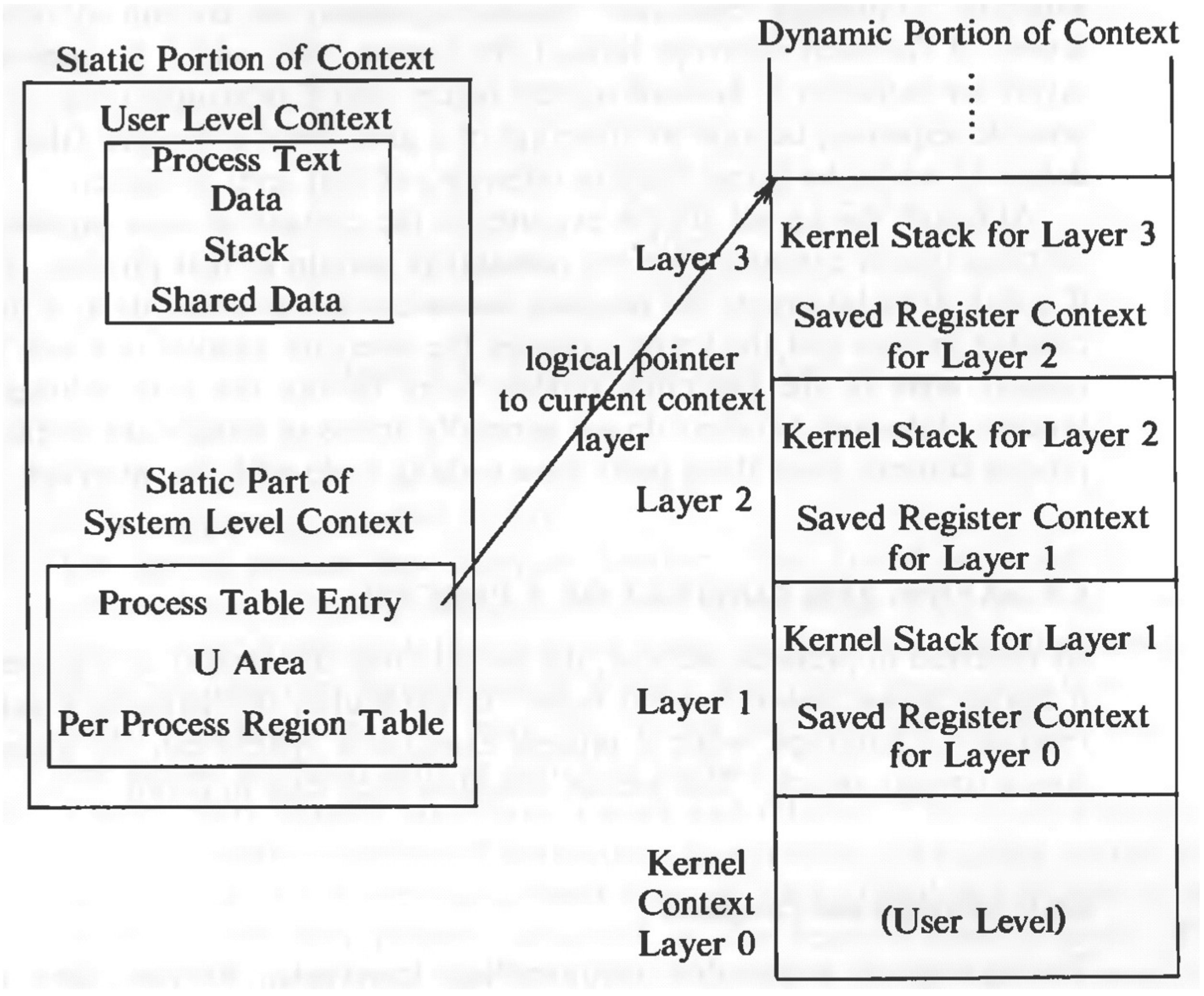
The static part consists of the following components:

- The process table entry
- The u-area
- Pregion entries, region tables and page tables.

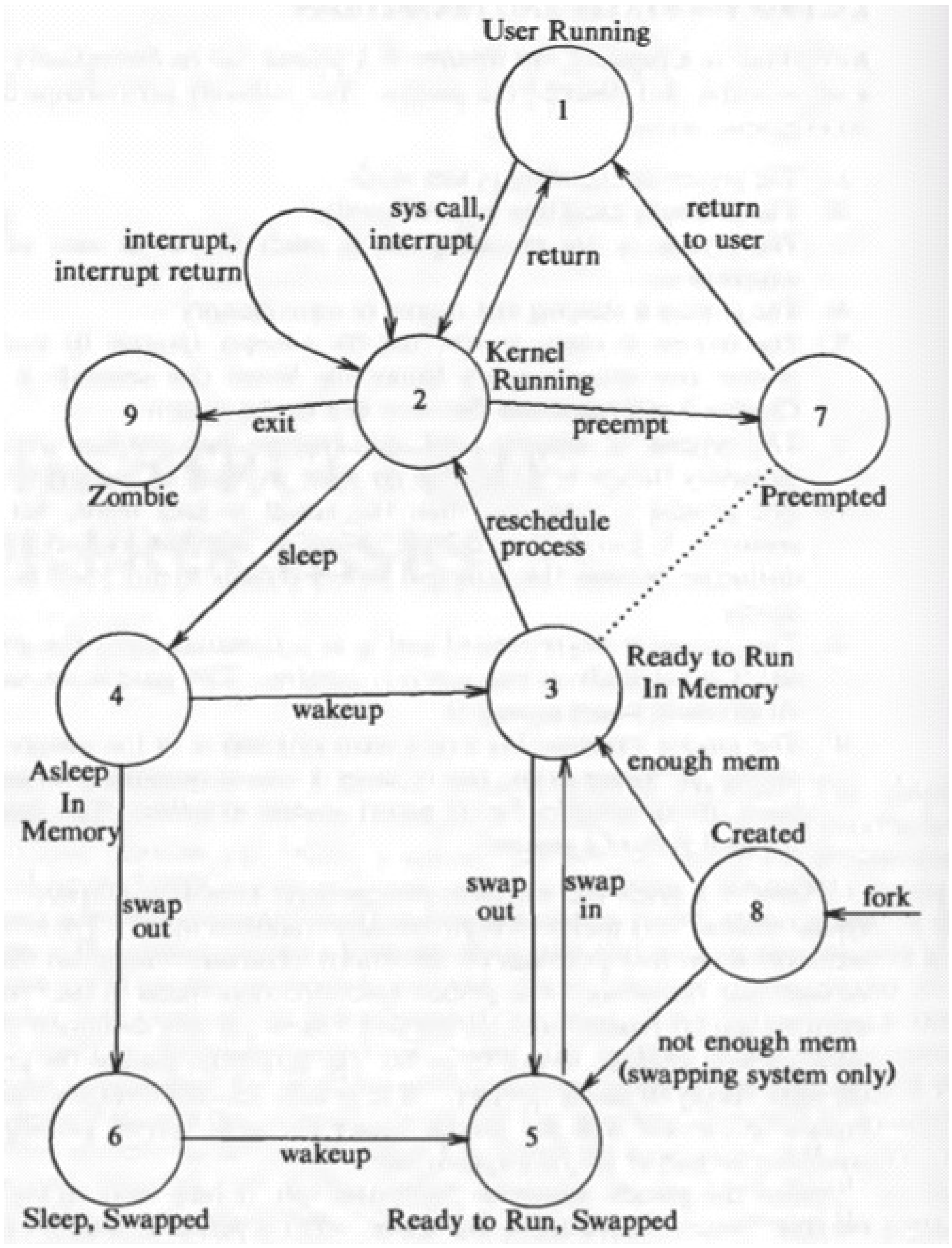
The dynamic part consists of the following components:

- The kernel stack contains the stack frames the kernel functions. Even if all processes share the kernel text and data, kernel stack needs to be different for all processes as every process might be in a different state depending on the system calls it executes. The pointer to the kernel stack is usually stored in the u-area but it differs according to system implementations. The kernel stack is empty when the process executes in user mode

- The dynamic part of the system level context consists of a set of layers, visualized as a last-in-first-out stack. Each system-level context layer contains information necessary to recover the previous layer, including register context of the previous layer.



Q3.state Transition Diagram:



Q4. Difference between named and unnamed pipes

Ans:

Unnamed pipe:

- These are created by the shell automatically.
- They exists in the kernel.
- They can not be accesses by any process, including the process that creates it.
- They are opened at the time of creation only.
- They are unidirectional.

Named Pipe:(also called FIFO, First In FIrst Out)

- They are created programatically using the command mkfifo.
- They exists in the file system with a given file name.
- They can be viewed and accessed by any two un-related processes. ls cmd shows “p” in the permission bits for a named pipe.
- They are not opened while creation.
- They are Bi-directinoal.

- A process writing a named pipe blocks until there is a process that reads that data.
- Broken pipe error occurs when the writing process closes the named pipe while another reading process reads it

Q5.Block which helps to find Max number of files /directories user can create Block which gives info about state of file system

Ans: Super block

Q6.Why user file descriptor table is allocated per process?

Ans: please check once allows sharing of files among process maintains state of file and user's access to it

Q7.Why various fields of u-area are required to access during process execution?

Ans : Because kernel allocates space for u-area only when creating a process

Q8.Kernel is non-preemptive .Dies it mean mutual exclusion is there on kernel .if yes/no justify

Ans: Yes, kernel maintains consistency of data structures thereby avoiding mutual exclusion problem making sure that critical sections of code are executed by at most one process at a time

Q9. Difference Between interrupt and Exception

Ans

Interrupt:

1.The UNIX system allows devices such as 1/0 peripherals or the system clock to

interrupt the CPU asynchronously. On receipt of the interrupt, the kernel saves its

current context , determines the cause of the interrupt, and services the interrupt.

2.After the kernel services the interrupt, it restores its interrupted context and proceeds as if

nothing had happened. The hardware usually prioritizes devices according to the order that

interrupts should be handled: When the kernel services an interrupt, it blocks out lower priority interrupts but services higher priority interrupts

3. interrupts are caused by events that are external to a process.

Exception :

1.An exception condition refers to unexpected events caused by a process, such as

addressing illegal memory, executing privileged instructions, dividing by zero, and

so on.

2. Exceptions happen "in the middle" of the execution of an instruction, and the system

attempts to restart the instruction after handling the exception.

Q10. With diagram, show that how many direct, single indirect, double indirect and triple indirect blocks are required for file size of 3,50,000 Bytes. (1 logical block-2K bytes, Block number address: a 64 bit (8byte) integer) What is maximum size of file a file system support? What is maximum number of files a file system can contain?

Ans :

Size of block = 1K = 2^10 Bytes

Size of block number addresses = 32bits = 4Bytes

Therefore number of blocks numbers in a block = 2^10 / 4 = 256

With 10 direct block with 1K size = 10 * 1024 = 10240Bytes = 10K Bytes

With 1 indirect block = 256 * 1024 = 262144Bytes = 256 K Bytes

With 1 double indirect block = 256*256*1024 = 67108864Bytes = 64 M Bytes

With 1 triple indirect block = 256*256*256 *1024 = 2^34 Bytes = 16GBytes

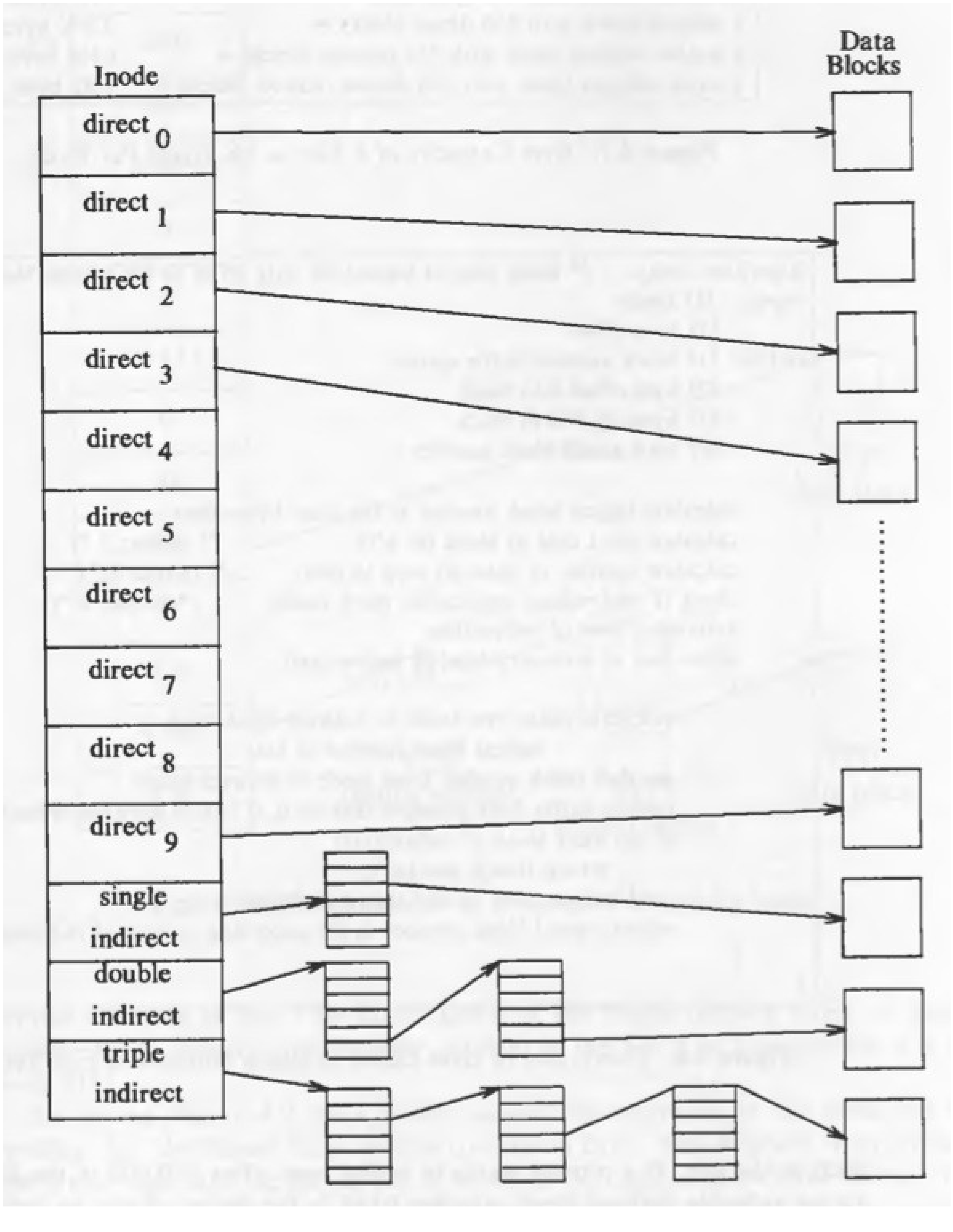
With 10direct + 1 indirect + 1 double indirect = 10240+262144+67108864 = 67381248

Since 67381248 > 350000 , we need 10 direct blocks ,1 indirect,1double indirect

AND 0 triple indirect .

Max size of file = 2^32 = 4GB

Max files = floor(10K+256K+64M+16GB) / 4GB = 4 files



Q11. Define zombie state. Why it is designed in the lifecycle of system

Ans :

The process executed the exit system call and is in the zombie state. The

process no longer exists, but it leaves a record containing an exit code and

some timing statistics for its parent process to collect. The zombie state is

the final state of a process.

To allow the parent to be guaranteed to be able to retreive exit status, accounting information, and process id for child processes so that the zombie state designed.

Q12.Which are the only two processes exist throughout in life time of the system

Ans: 1.process 0 (swapper)

2.process 1 (init)

Q13. Is the management of regions of process is similar to management of inodes in kernel if yes why is it similar to inode management

Ans : Yes , processing......

Q14. Advantages and Disadvantages of The Buffer Cache

Ans :

Advantages

1.The use of buffers allows uniform disk access

2.The system places no data alignment restrictions on user processes doing 1/0

3. Use of the buffer cache can reduce the amount of disk traffic

4. The buffer algorithms help insure file system integrity.

Disadvantages

1 .A delayed write strategy has 2 drawbacks.

2.the system is vulnerable to crashes that leave disk data in an incorrect state.

3.The size of the buffer cache would have to be huge.

4.Use of the buffer cache requires an extra data copy when reading and writing to and from user

processes

Q15.Enlist the five scenarios of getblk algorithm ,which scenarios are suffered from race condition

Ans: The algorithms for reading and writing disk blocks use the algorithm getblk to allocate buffers from the pool. There are 5 typical scenarios the kernel may follow in getblk to allocate a buffer for a disk block.

- Block is found on its hash queue and its buffer is free.
- Block could not be found on the hash queue, so a buffer from the free list is allocated.
- Block could not be found on the hash queue, and when allocating a buffer from free list, a buffer marked "delayed write" is allocated. Then the kernel must write the "delayed write" buffer to disk and allocate another buffer.
- Block could not be found on the hash queue and the free list of buffers is empty.
- Block was found on the hash queue, but its buffer is currently busy.

Q16. why System calls are required

Ans : for the commnunication between two processes system calls are required

Q17. which kernel data structure describe the state of the process

Ans: process table

Q18. What is socket ? Which System call is responsible for binding port and process

Ans : To provide common methods for interprocess communication and to allow use of

sophisticated network protocols, the BSD system provides a mechanism known as sockets.

System call is responsible for binding port and process is Bind().

Q19. What are the advantages to kernel in maintaining the U area in the system?

Ans: Kernel gets to know information about the running processes ,that is why u-area is

maintained by kernal

1.pointer to process table

2.Times(time spent by the process in user and kernal mode)

3.Arrays and signals

4.real and user effectve IDS

Q20.When attaching the region to Process ,how can the kernel check theat the region does not overlap virtual address in regions already attached to the Process?

Ans :

Q21.Suppose a process goes to sleep and the system contain no processes ready to run .What happen when the sleeping process does its context switch

Chapter 2, 3

Subhramanya Bhat

Q1)Why various fields of u-area are required to access during process execution?

The u area contains information that needs to be accessible only when the process is executing. Important fields in the u area are:

- a pointer to the process table slot of the currently executing process
- parameters of the current system call, return values and error codes

- file descriptors for all open files
- internal I/O parameters
- current directory and current root
- process and file size limits

The kernel internally uses a structure variable u which points to the u area of the currently executing process. When another process executes, the kernel rearranges its virtual address space that u refers to the u area of the new process.

Q2)Block diag of system kernel and explain responsibility of processes control and file control subsystem

The file subsystem accesses file data using a buffering mechanism that regulates dataftow between the kernel . and secondary storage devices. The buffering mechanism interacts with block 110 device drivers to initiate data transfer to and from the kernel. Device drivers are the kernel modules that control the operation of peripheral devices. Block 110 devices are random access storage devices; alternatively, their device drivers make them appear to be random access storage devices to the rest of the system. For example, a tape driver may allow the kernel to treat a tape unit as a random access storage device. The file subsystem also interacts directly with "raw" 1/0 device drivers without the intervention of a buffering mechanism. Raw devices, sometimes called character devices, Include all devices that are not block devices.

•The process control subsystem is responsible for process synchronization, interprocess communication memory management, and process scheduling. The file subsystem and the process control subsystem interact when loading a file into memory for execution, as will be seen in Chapter 7: the process subsystem reads executable files into memory before executing them

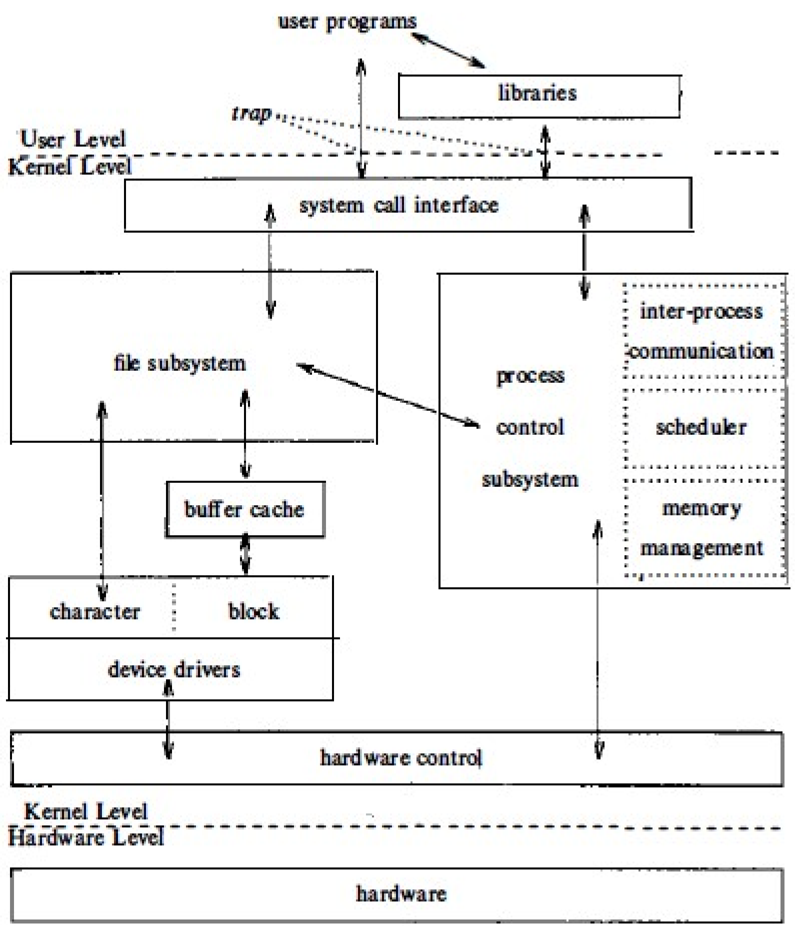
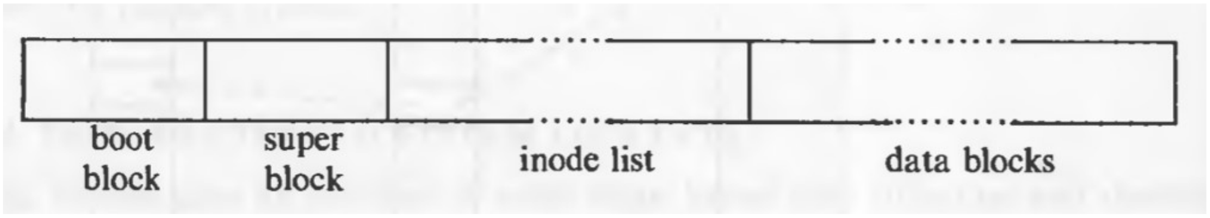


Figure 2.1. Block Diagram of the System Kernel

Q3)Blocks of file system layout

Ans:



A file system has the following structure:

- The boot block occupies the beginning of a file system, typically the first sector, and may contain the bootstrap code that is read into the machine to boot , or initialize, the operating system. Although only one boot block is needed to boot the system, every file system has a (possibly empty) boot block.
- The super block describes the state of a file system -- how large it is, how many files it can store, where to find free space on the file system, and other information.
- The inode list is a list of inodes that follows the super block in the file system. Administrators specify the size of the inode list when configuring a file system. The kernel references inodes by index into the inode list. One inode is the root inode of the file system: it is the inode by which the directory structure of the file system is accessible after execution of the mount system call.

•	The data blocks start at the end of the inode list and contain file data and administrative data. An allocated data block can belong to t one and only one file in the file system.

Q4)Block which helps to find Max number of files /directories user can create, Block which gives info about state of file system.

->super block

Q5) Why kernel data structures are static?

Most kernel data structure have fixed size rather than dynamically allocated space.

This decision was taken to make the kernel code simple. But the disadvantage is that kernel cannot be configured dynamically. If, at runtime, kernel hits the limit of some data structure, it has to throw errors. But such situations occur very rarely

Q6) Why user file descriptor table is allocated per process?

Entries in the user file descriptor tablemaintain the state of the file and the user's access to it.The user file descriptor table identifies all open files for a process.The kernel returns a file descriptor for the open and creat system calls, which is an index into the user file descriptor table. Keeping all these records of all the processes in a single global data structure is not possible hence, user file descriptor table is allocated per process.

Q7)Kernel is non-preemptive .Does it mean mutual exclusion is there on kernel .If yes/no justify.

Yes,kernel maintains consistency of data structures thereby avoiding mutual exclusion problem making sure that critical sections of code are executed by at most one process at a time

The kernel allows a context switch only when a process moves from the state "kernel running" to the state "asleep in memory". Processes running in kernel mode cannot be preempted by other processes; therefore the kernel is sometimes said to be non-preemptive.

Q8)What data structures will you use to design your own OS.

file: user file descriptor

File table

Inode table

Process mgmt:

Process table

Region mgmt:

Region table

Per process region table

Buffer mgmt:

Free list

Hash queue

Q9) What are the system calls that support processing environment in kernel? How kernel uses system calls for processing.

??
??????????????????????????

Chapter 3

Q1)Five scenarios of getblk algorithm. Which scenarios are suffered from race condition, why?

The algorithms for reading and writing disk blocks use the algorithm getblk to allocate buffers from the pool. There are 5 typical scenarios the kernel may follow in getblk to allocate a buffer for a disk block.

1. Block is found on its hash queue and its buffer is free.

2. Block could not be found on the hash queue, so a buffer from the free list is allocated.

3. Block could not be found on the hash queue, and when allocating a buffer from free list, a buffer marked "delayed

write" is allocated. Then the kernel must write the "delayed write" buffer to disk and allocate another buffer.

4. Block could not be found on the hash queue and the free list of buffers is empty.

5. Block was found on the hash queue, but its buffer is currently busy.

Race condition:

It is also possible to imagine cases where a process is starved out of accessing a

buffer In the fourth scenario, for example, if several processes sleep while waiting

for a buffer to become free, the kernel does not guarantee that they get a buffer in

the order that they requested one. A process could sleep and wake. upwhen • a

buffer becomes free, only to go to sleep again because another process got control 'of

the buffer first. Theoretically, this could go on forever, but practically, it is not •a

problem because of the many buffer& that are typically configured in "the system”.

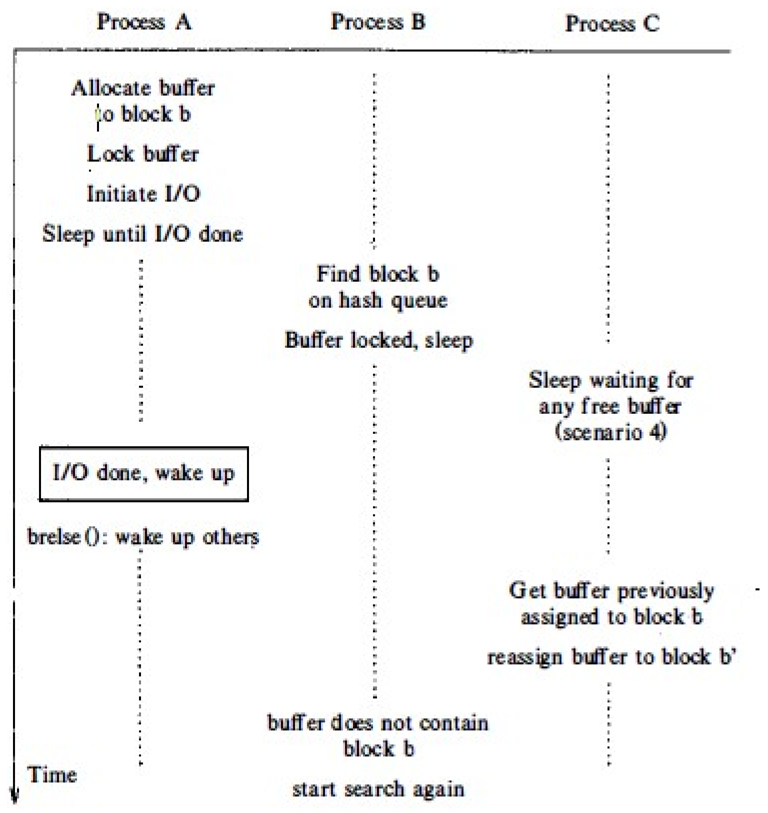

Process A
Process B
Process C
Allocate buffer to block b
Lock buffer
Initiate I/O
Sleep until I/O done
Find block b on hash queue
Buffer locked, sleep
Sleep waiting for any free buffer (scenario 4)
I/O done, wake up
brelse (): wake up others
Get buffer previously assigned to block b
reassign buffer to block b'
buffer does not contain block b
start search again
Time

Figure 3.12. Race for a Locked Buffer

Q2)In the algorithm getblk,if kernel removes a buffer from the free list, it must raise processor priority level to block out interrupts before checking free list, why?

The kernel raises the processor execution level to prevent disk interrupts while

manipulating the free list, thereby preventing corruption of the buffer pointers that

could result from a nested call to brelse . Similar bad effects could happen if an interrupt handler invoked brelse while a process was executing getblk, so the kernel

raises the processor execution level at strategic places in getblk , too.

Q3)Why free list of buffer in buffer cache is implemented like LRU if it is like MFU what will happen.

>……………………………………………………..

Q4)Enlist advantages and disadvantages of buffer cache.

Advantages:

The use of buffers allows uniform disk access, because the kernel does not need

to know the reason for the I/O.

Use of the buffer cache can reduce the amount of disk traffic, thereby increasing

overall system throughput and decreasing response time.

The buffer algorithms help insure file system integrity, because they maintain a

common, single image of disk blocks contained in the cache.

Disadvantages:

Since the kernel does not immediately write data to the disk for a delayed write, the system is

vulnerable to crashes that leave disk data in an incorrect state.

Use of the buffer cache requires an extra data copy when reading and writing to

and from user processes.

Q5)Suppose the Kernal does a delayed write of a block. What happens when another process takes that block from the hash queue, from the free list

However, it discovers that the buffer it removes from the free

list has been marked for "delayed write," so it must write the contents of the buffer

to disk before using the buffer. The kernel starts an asynchronous write to disk and

tries to allocate another buffer from the free list. When the asynchronous write

completes, the kernel releases the buffer and places it at the head of the free list

Q6)In buffer cache management why hash queue and free list needed to maintain separately which are suitable DS for Hash queue and free list

Each buffer always exists on a hash queue, but there is no significance to its

position on the queue. As stated above, no two buffers may simultaneously contain

the contents of the same disk block; therefore, every disk block in the buffer pool

exists on one and only one hash queue and only once on that queue. However, a

buffer may be on the free list as well if its status is free.

Because a buffer may be simultaneously on a hash queue and on the free list, the kernel has two ways to find

it: It searches the hash queue if it is looking for a particular buffer, and it removes

a buffer from the free list if it is looking for any free buffer.

Q7)Explain scenarios of getblk in details.

>…………………………………………………………..

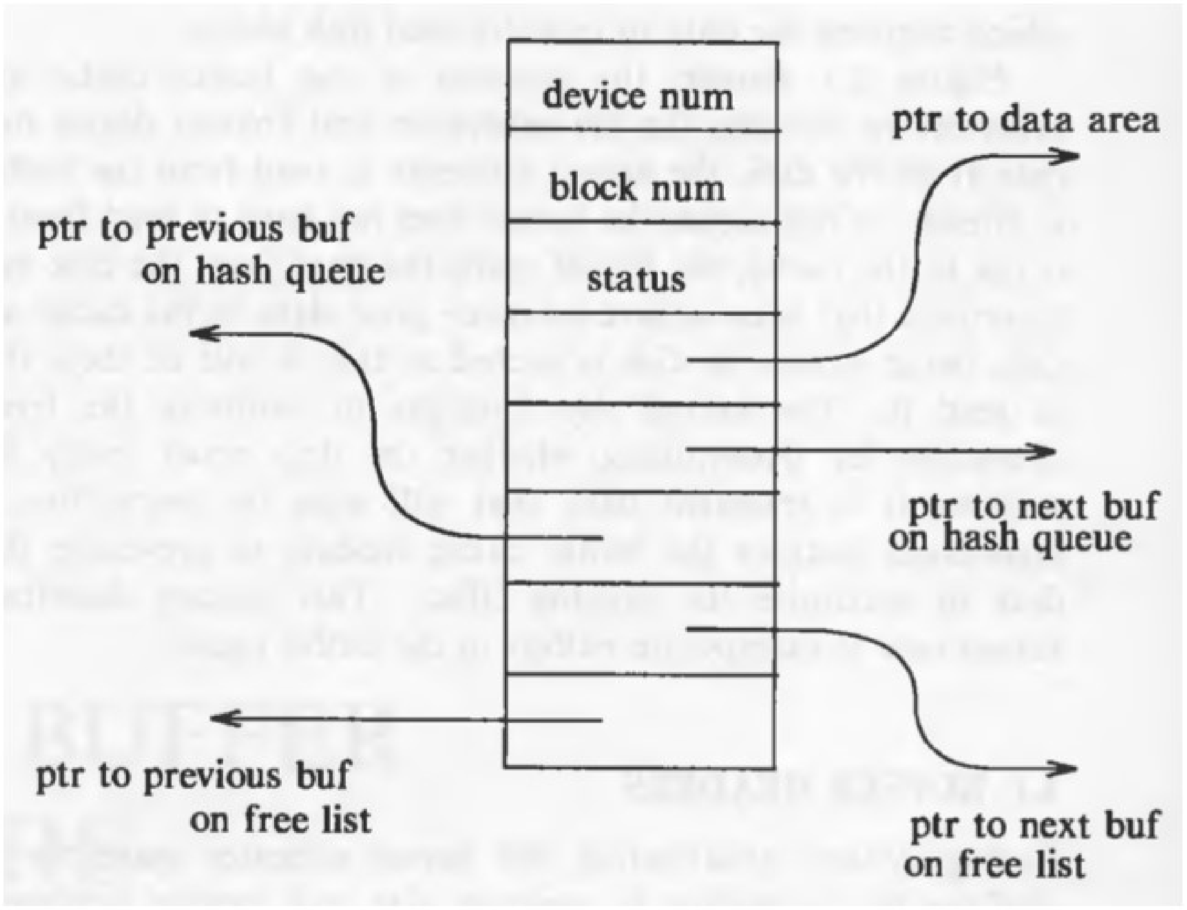
Q8) With diagram describe various fields of buffer header and structure of buffer header

During system initialization, the kernel allocates space for a number of buffers, configurable according to memory size and performance constraints.

Two parts of the buffer:

1. a memory array that contains data from the disk.
2. buffer header that identifies the buffer.

Data in a buffer corresponds to data in a logical disk block on a file system. A disk block can never map into more than one buffer at a time.


device num
block num
status
ptr to data area
ptr to previous buf on hash queue
ptr to next buf on hash queue
ptr to previous buf on free list
ptr to next buf on free list

The device number fields specifies the logical file system (not physical device) and block number block number of the data on disk. These two numbers uniquely identify the buffer. The status field summarizes the current status of the buffer. The ptr to data area is a pointer to the data area, whose size must be at least as big as the size of a disk block.

The status of a buffer is a combination of the following conditions:

- Buffer is locked / busy
- Buffer contains valid data
- Kernel must write the buffer contents to disk before reassigning the buffer; called as "delayed-write"
- Kernel is currently reading or writing the contexts of the buffer to disk
- A process is currently waiting for the buffer to become free.

The two set of pointers in the header are used for traversal of the buffer queues (doubly linked circular lists).

……………………………………………………………………………………………….

Sanket Sanatan

Q1)Write the 5 scenarios for retrieval of buffer and explain any 4th scenario in detail.

Ans: five typical scenarios the kernel may follow in getblk to allocate a buffer for a disk block.

1. The kernel finds the block on its hash queue, and its buffer is free.

2. The kernel cannot find the block on the hash queue, so it allocates a buffer from the free list.

3. The kernel cannot find the block on the hash queue and, in attempting to allocate a buffer from the free list (as in scenario 2), finds a buffer on the free list that has been marked "delayed write." The kernel must write the "delayed write" buffer to disk and allocate another buffer.

4. The kernel cannot find the block on the hash queue, and the free list of buffers is empty.

5. The kernel finds the block on the hash queue, but its buffer is currently busy.

6. the fourth scenario, the kernel, acting for process A, cannot find the disk block on its hash queue, so it attempts to allocate a new buffer from the free list, as in the second scenario. However, no buffers arc available on the free list, so process A goes to sleep until another process executes algorithm brelse, freeing a buffer. When the kernel schedules process A, it must search the hash queue again for the block. It cannot allocate a buffer immediately from the free list, because it is possible that

several processes were waiting for a free buffer and that one of them allocated a newly freed buffer for the target block sought by process A. Thus, searching for the block again insures that only one buffer contains the disk block. depicts the contention between two processes for a free buffer.

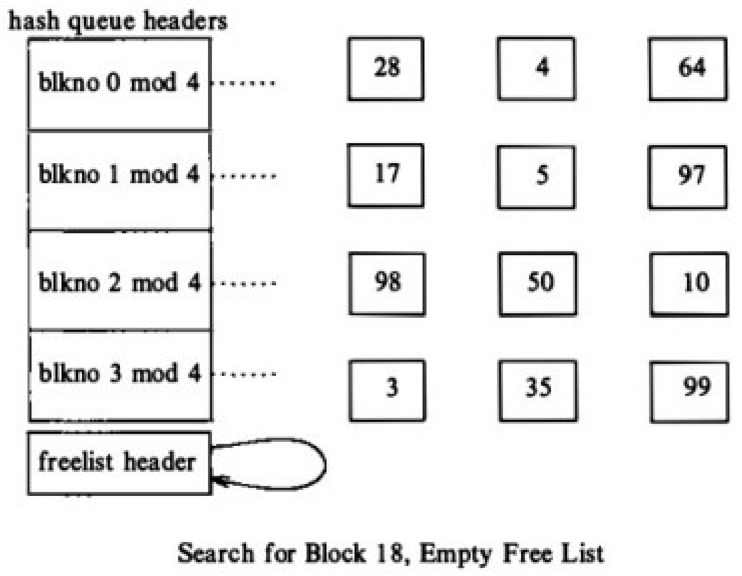


Figure 3.9. Fourth Scenario for Allocating Buffer

Q2In getblk scenarios which block suffers from race condition?why? A race condition is an undesirable situation that occurs when a device or system attempts to perform two or more operations at the same time, but because of the nature of the device or system, the operations must be done in the proper sequence to be done correctly.

Ans: In Scenario 4)Race for free buffer

in Scenario 5)Race for a locked buffer

Q3. Which system calls manipulate the inode or that maneuver through the file system?

Ans: Inodes exist in a static form on disk, and the kernel reads them into an in-core inode to manipulate them.

Q4. Which four circumstances under which kernel permits Context Switch?

Ans: The Context switching is a technique or method used by the operating

system to switch a process from one state to another to execute its function

using CPUs in the system.

1)When the kernel decides that it should execute another process,

2)he kernel does a context switch when it changes context from

process A to process B;

3)when a process moves from the state

"kernel running" to the state ••asleep in memory.

4)kernel protects its consistency by allowing a context switch

Q5 With diagram describe the various fields of buffer header and structure of buffer header

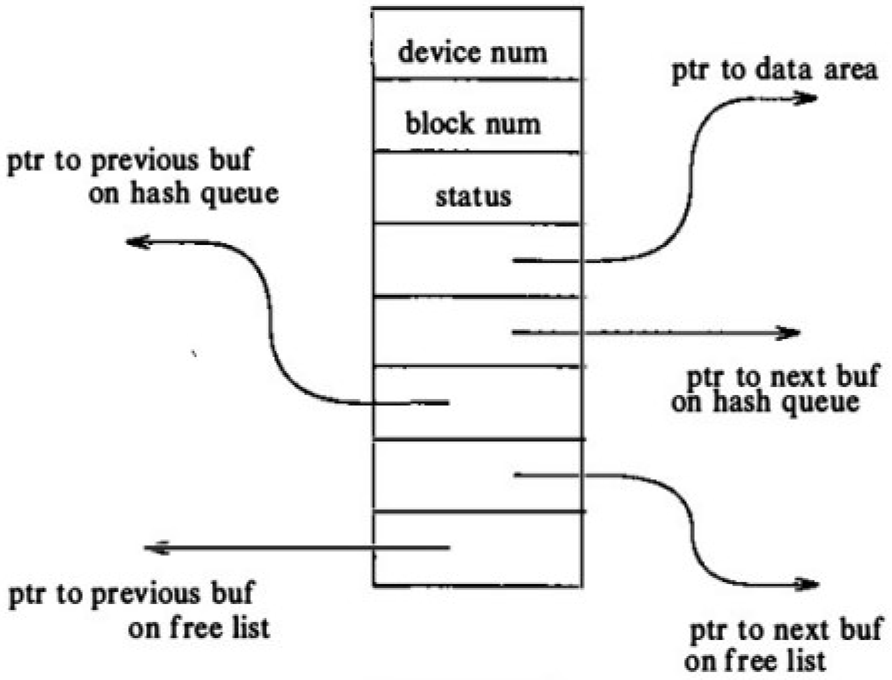

device num
block num
status
ptr to data area
ptr to previous buf on hash queue
ptr to next buf on hash queue
ptr to previous buf on free list
ptr to next buf on free list

Figure 3.1 Buffer Header

contains a device number field and a block and number field that specify the file system and block number of the data on disk and uniquely identify the buffer.

The device number is the logical file system number not a physical device (disk) unit number.

The buffer header also contains a pointer to a data array for the buffer,whose size must be at least as big as the size of a disk block, and a status field that summarizes the current status of the buffer.

The buffer header also contains two sets of pointers, used by the buffer allocation algorithms to maintain the overall structure of the buffer pool,

Q6. what is socket ?which system call is responsible for binding process and port?

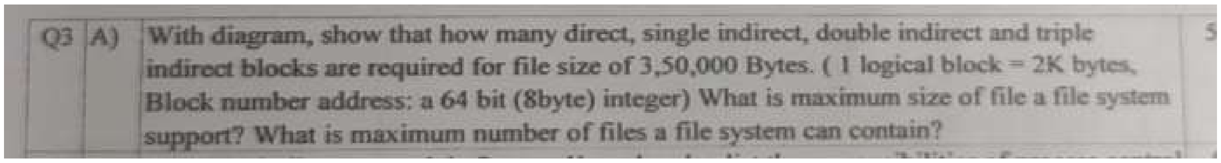
Ans: 1)The socket layer provides the interface between the system calls and the lower layers

2)Sockets are the constructs that allow processes on different machines to communicate through an underlaying network.

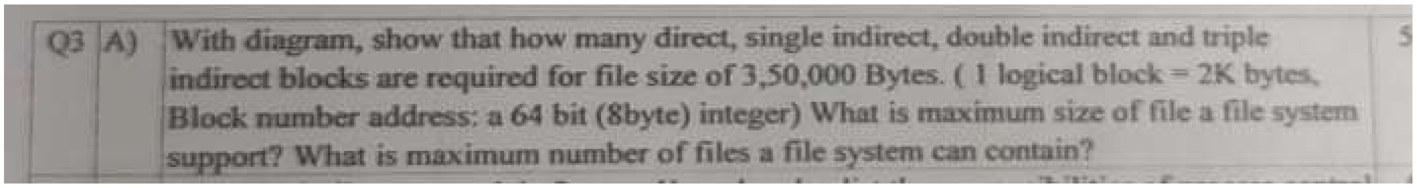
3)Bind() system call is responsible for binding processs and port.

Chapter 3, 4

Shripad Deshpande



Q3	A)	With diagram, show that how many direct, single indirect, double indirect and triple indirect blocks are required for file size of 3,50,000 Bytes. (1 logical block = 2K bytes, Block number address: a 64 bit (8byte) integer) What is maximum size of file a file system support? What is maximum number of files a file system can contain?	5



Q3	A)	With diagram, show that how many direct, single indirect, double indirect and triple indirect blocks are required for file size of 3,50,000 Bytes. (1 logical block = 2K bytes, Block number address: a 64 bit (8byte) integer) What is maximum size of file a file system support? What is maximum number of files a file system can contain?	5

4 b)

Size of block = 1K = 2^10 Bytes

Size of block number addresses = 32bits = 4Bytes

Therefore number of blocks numbers in a block = 2^10 / 4 = 256

With 10 direct block with 1K size = 10 * 1024 = 10240Bytes = 10K Bytes

With 1 indirect block = 256 * 1024 = 262144Bytes = 256 K Bytes

With 1 double indirect block = 256*256*1024 = 67108864Bytes = 64 M Bytes

With 1 triple indirect block = 256*256*256 *1024 = 2^34 Bytes = 16GBytes

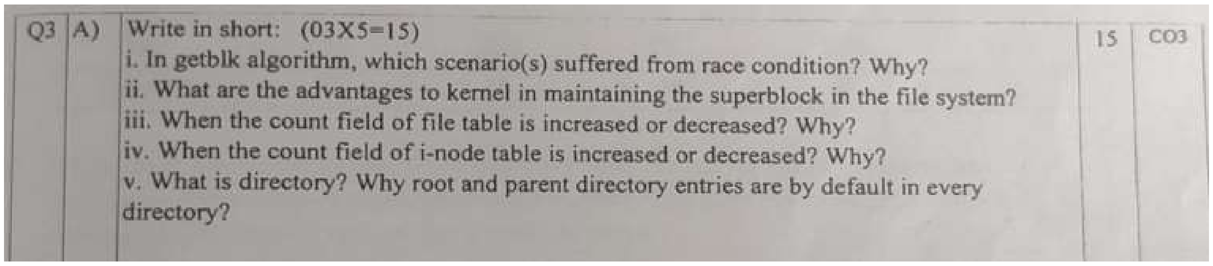
With 10direct + 1 indirect + 1 double indirect = 10240+262144+67108864 = 67381248

Since 67381248 >350000 , we need 10 direct blocks ,1 indirect,1double indirect

AND 0 triple indirect .

Max size of file = 2^32 = 4GB

Max files = floor(10K+256K+64M+16GB) / 4GB = 4 files



Q3	A)	Write in short: (03X5=15) i. In getblk algorithm, which scenario(s) suffered from race condition? Why? ii. What are the advantages to kernel in maintaining the superblock in the file system? iii. When the count field of file table is increased or decreased? Why? iv. When the count field of i-node table is increased or decreased? Why? v. What is directory? Why root and parent directory entries are by default in every directory?	15	CO3

ii)The super block consists of the following fields: ...

the size of the file system,

• the number of free blocks in the file system,

• a list of free blocks available on the fiie system,

• the index of the next

• the size of the inode list,

• the number of free inodes in the file system,

• a list_ of free inodes in the file system

iii)The count field of file table is increased when dup() system call is made.It decreases when close() sytem call is executed for a file.

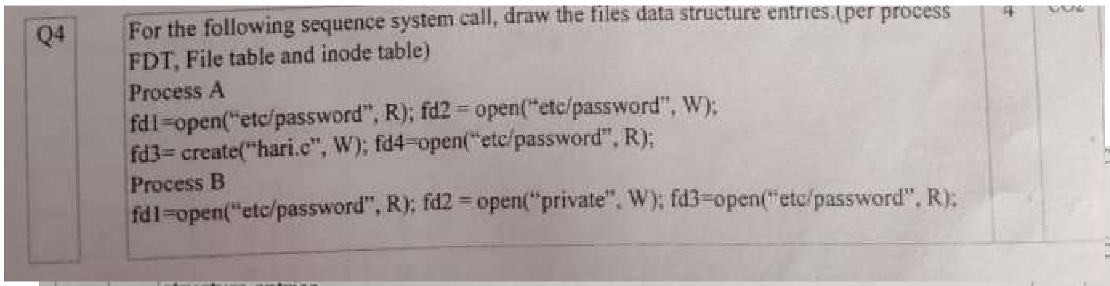
iv) The count field of inode table increases when the number of active instances of the file increase due to open or create or dup,

system call.It decreases when close() system call is executed.

v)directories are the files that give the file system its hierarchical structure; they play an important role in conversion of a file name to an inode number: A directory is a file whose data is a sequence of entries, each consisting of an inode number and the name of a file contained in the directory.

UNIX System V restricts component nameS to a maximum of 14 characters; with a 2 byte entry for the inode number, the size of a directory entry is 16 bytes.

Every directory contains die file names dot and dot-dot ("." and " .. ") whose inode numbers are those of the directory and its parent directory, respectively.So that we can get the root of the file system and can traverse back to the parent directory.



Q4 For the following sequence system call, draw the files data structure entries.(per process FDT, File table and inode table) 4

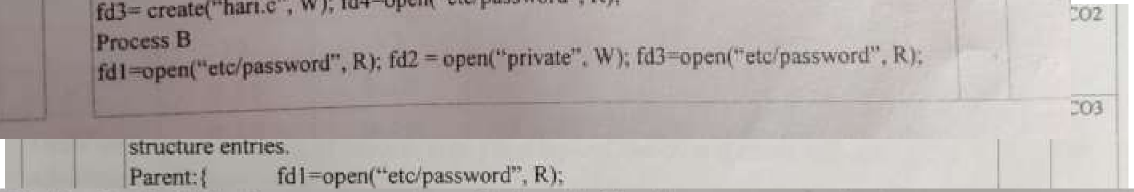
Process A

fd1=open("etc/password", R); fd2 = open("etc/password", W);

fd3= create("hari.c", W); fd4=open("etc/password", R);

Process B

fd1=open("etc/password", R); fd2 = open("private", W); fd3=open("etc/password", R);

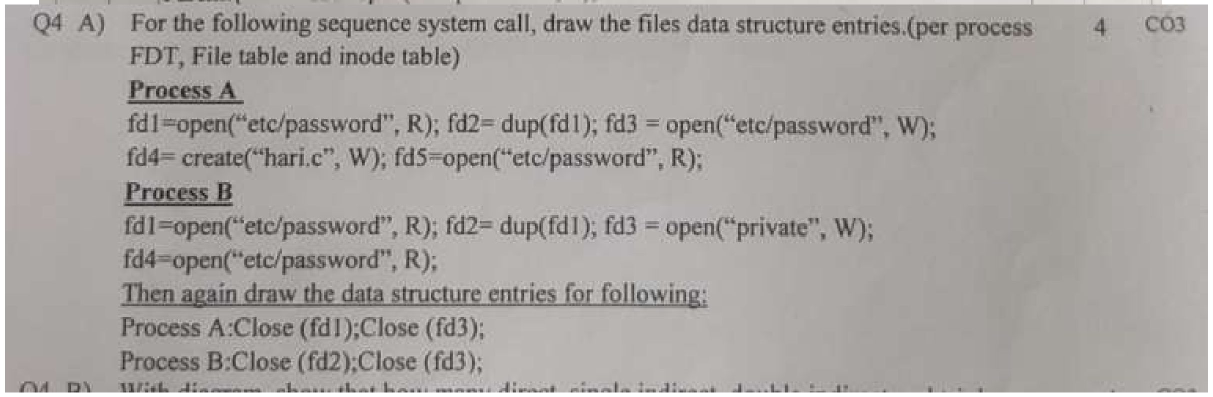


CO2

CO3

structure entries.

Parent:{ fd1=open("etc/password", R);



Q4 A) For the following sequence system call, draw the files data structure entries.(per process FDT, File table and inode table) 4 CO3

Process A

fd1=open("etc/password", R); fd2= dup(fd1); fd3 = open("etc/password", W);

fd4= create("hari.c", W); fd5=open("etc/password", R);

Process B

fd1=open("etc/password", R); fd2= dup(fd1); fd3 = open("private", W);

fd4=open("etc/password", R);

Then again draw the data structure entries for following:

Process A:Close (fd1);Close (fd3);

Process B:Close (fd2);Close (fd3);

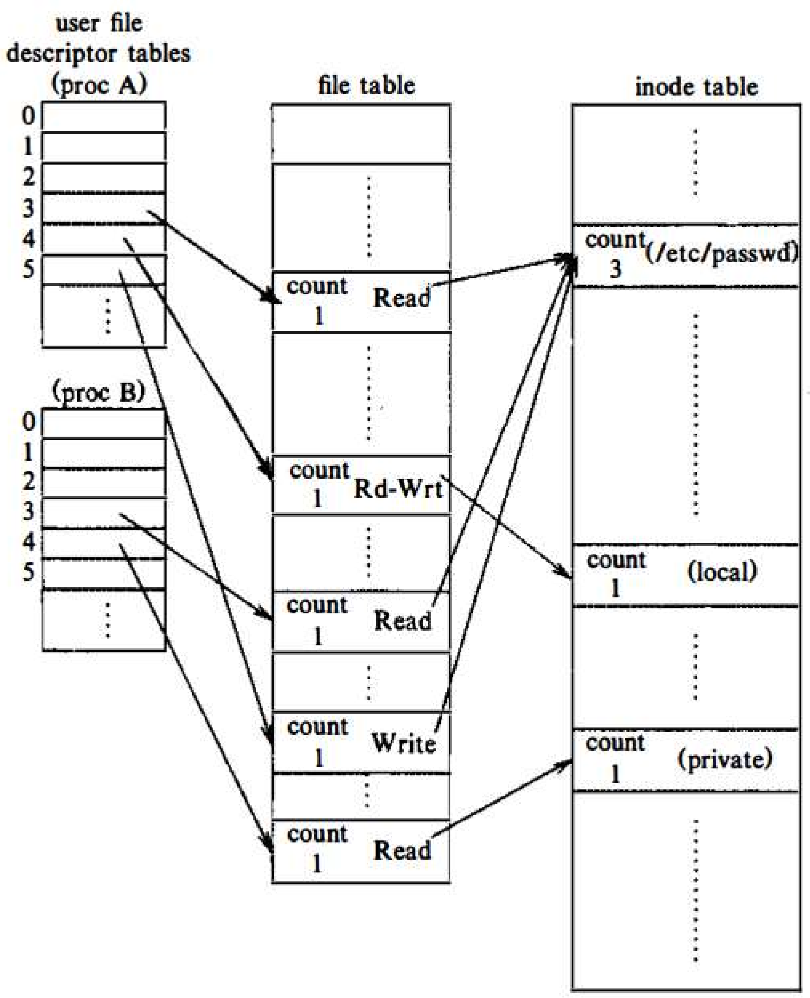


Figure 5.4. Data Structures after Two Processes Open Files

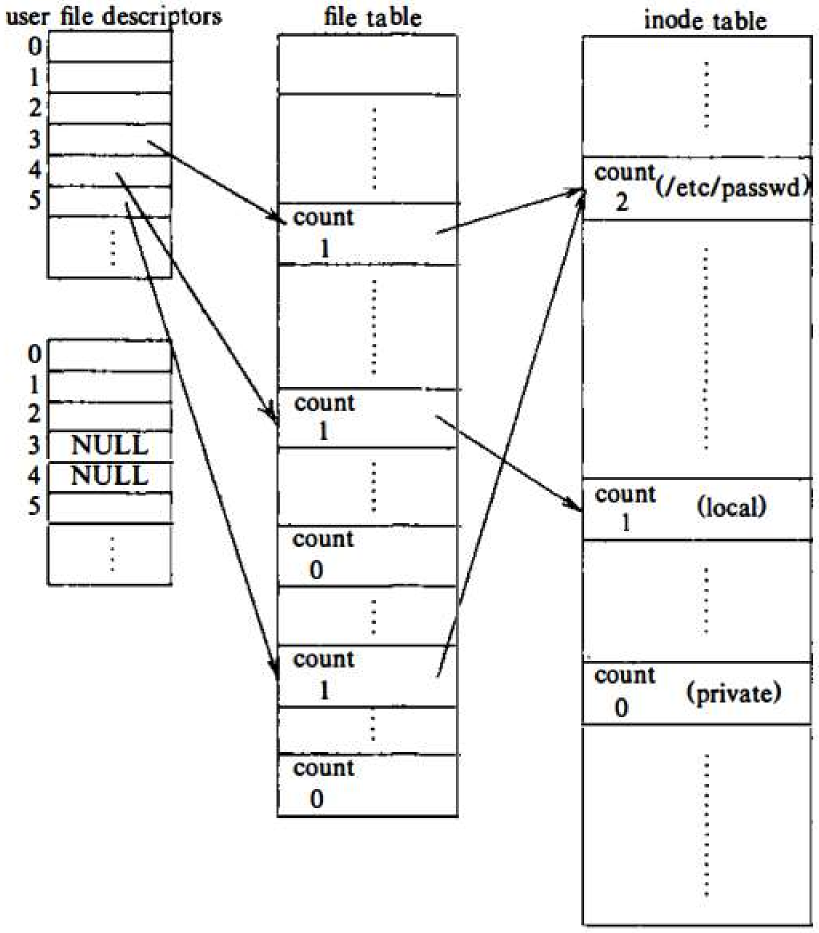


Figure 5.11. Tables after Closing a File

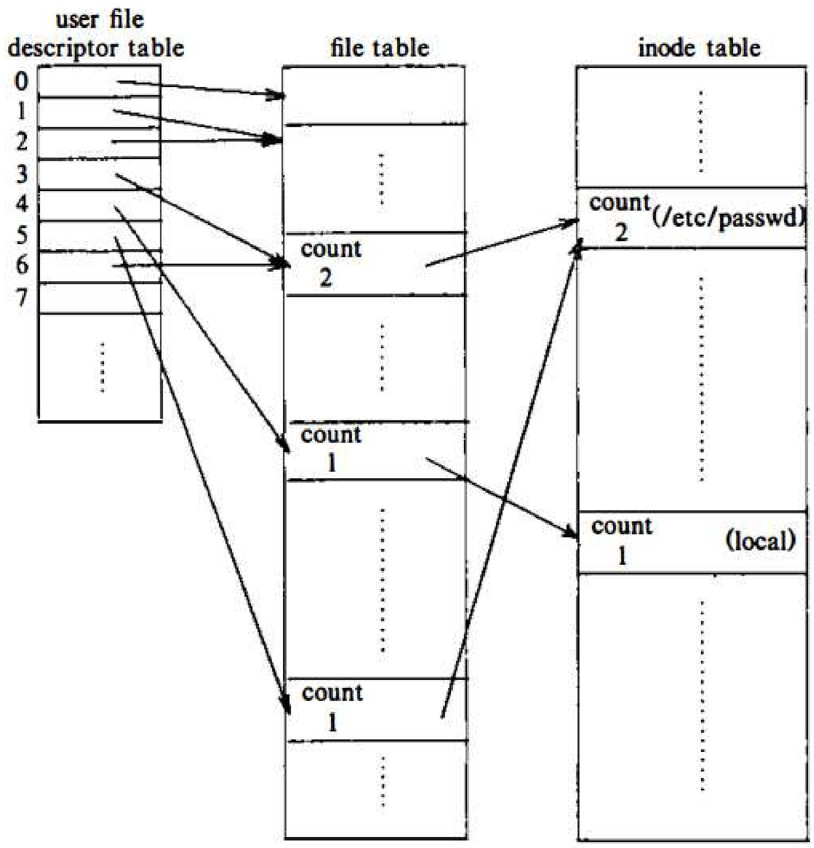
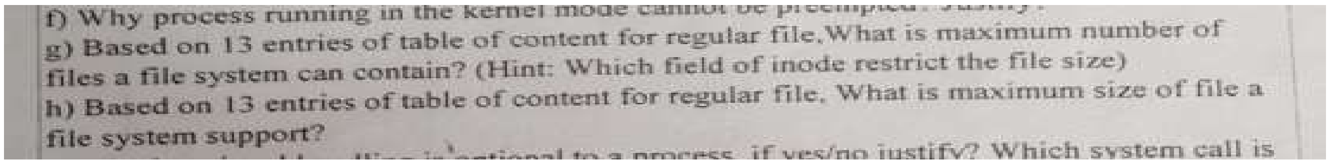


Figure 5.20. Data Structures after Dup



f) Why process running in the kernel mode cannot be preempted? Justify?

g) Based on 13 entries of table of content for regular file.What is maximum number of files a file system can contain? (Hint: Which field of inode restrict the file size)

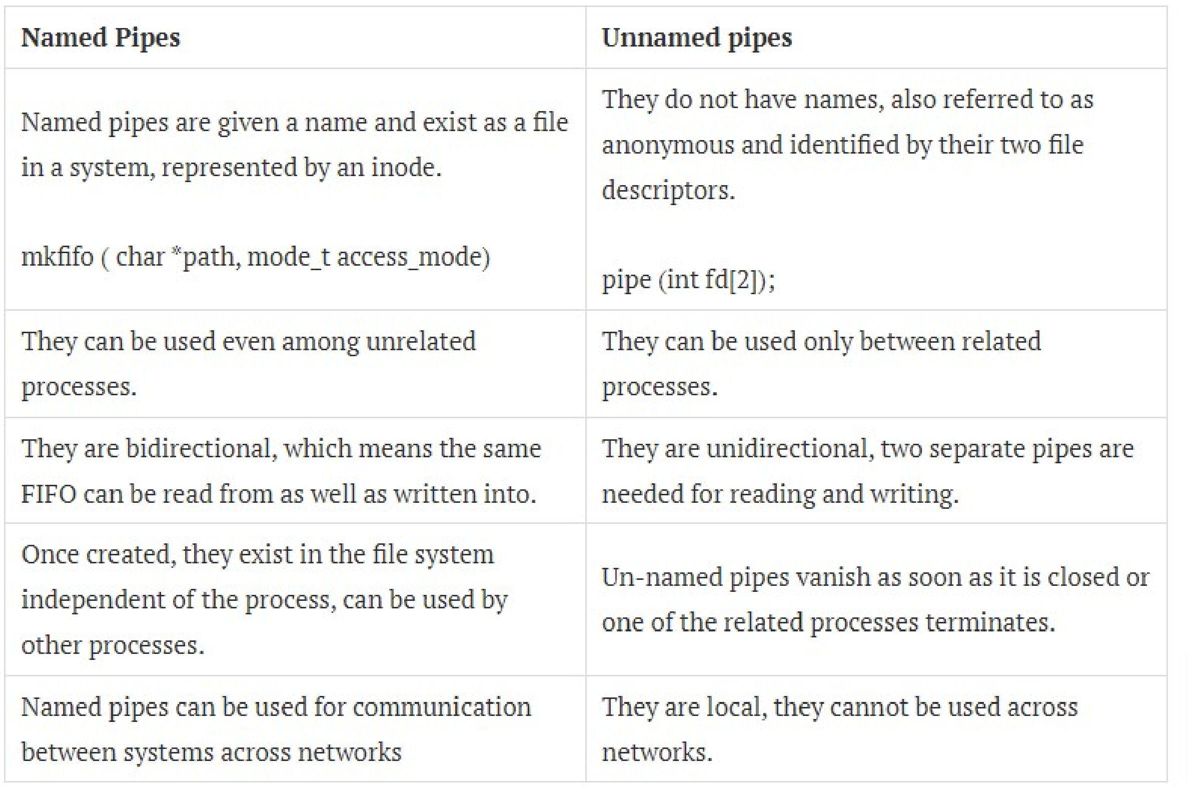
h) Based on 13 entries of table of content for regular file. What is maximum size of file a file system support?

[illegible] is optional to a process, if yes/no justify? Which system call is

g) ……………………………….

h)The maximum number of bytes that could be held in a file is calculated at well over 16 gigabytes, using 10 direct blocks and 1 indirect, I double indirect, and 1 triple indirect block in the inode. Given that the file size field in the inode is 32 bits, the size of a file is effectively limited to 4 gigabytes

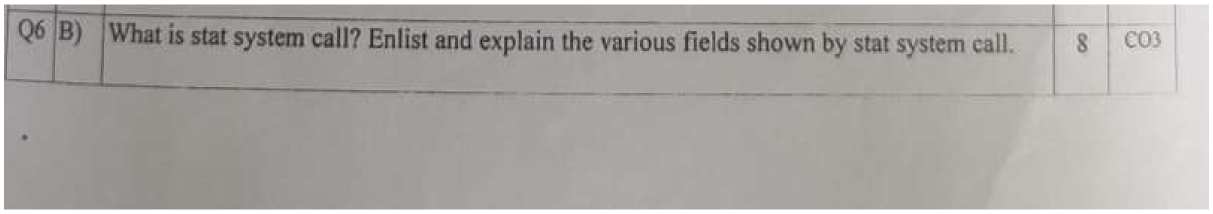
Que: Difference between named and unnamed pipe.



Named Pipes	Unnamed pipes
Named pipes are given a name and exist as a file in a system, represented by an inode. mkfifo (char *path, mode_t access_mode)	They do not have names, also referred to as anonymous and identified by their two file descriptors. pipe (int fd[2]);
They can be used even among unrelated processes.	They can be used only between related processes.
They are bidirectional, which means the same FIFO can be read from as well as written into.	They are unidirectional, two separate pipes are needed for reading and writing.
Once created, they exist in the file system independent of the process, can be used by other processes.	Un-named pipes vanish as soon as it is closed or one of the related processes terminates.
Named pipes can be used for communication between systems across networks	They are local, they cannot be used across networks.

named pipe vs. unnamed pipe

- open system call / pipe system call
- process access file permission can be given like file/default
- pipe call access/ process descendant access
- used for communication between a child and its parent process/ two unnamed process
- Permanent / transient
- handles one-way or two-way communication between two unrelated processes./handles one-way communication.
- Size 64kb /direct block size i.e10kb: depend on os



Q6	B)	What is stat system call? Enlist and explain the various fields shown by stat system call.	8	CO3

The system calls stat and fstat allow processes to query the status of files, returning information such as the file type, file owner, access permissions, file size, number of links, inode number, and file access times. The syntax for the system calls is

stat(pathname, statbuffer);

fstat(fd; statbuffer);

where pathname is a file name,

fd is a file descriptor returned by a previous open call,

and statbuffer is the address of a data structure in the user process that will contain the status information of the file on completion of the call.

The system calls simply write the fields of the inode into statbuffer

Chapter 2,3

Chetan Pukale

Q1 In the algorithm getblk, if kernel removes a buffer from free list, it must raise the process priority level to block out interrupts before checking free list. Why?

Ans:-Because handling the interrupt could corrupt the pointers

Q2)What are the system calls that support the processing environment in kernel? How kernel uses these system calls for processing?

Ans: System call supported by processing unit:-

1)fork() 2)exec() 3)brk() 4)exit() 5)wait() 6)signal() 7)kill() 8)signal()

fork : create a new process

exit : terminate process execution

wait : allow parent process to synchronize its execution with the exit of a child process

exec : invoke a new program

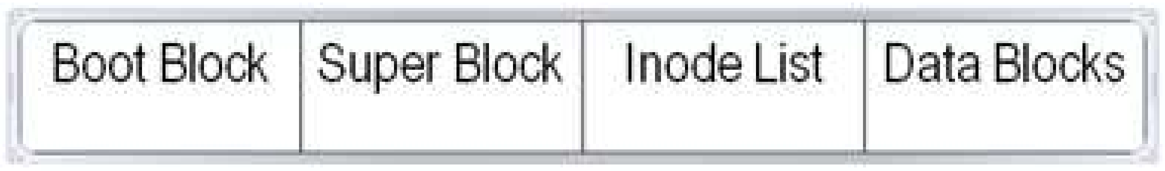
brk : allocate more memory dynamically

signal : inform asynchronous event

Q3-Draw and elaborate the blocks of file system layout. Which block help to find maximum number files/directory user can create? Which block gives information about state of file system?

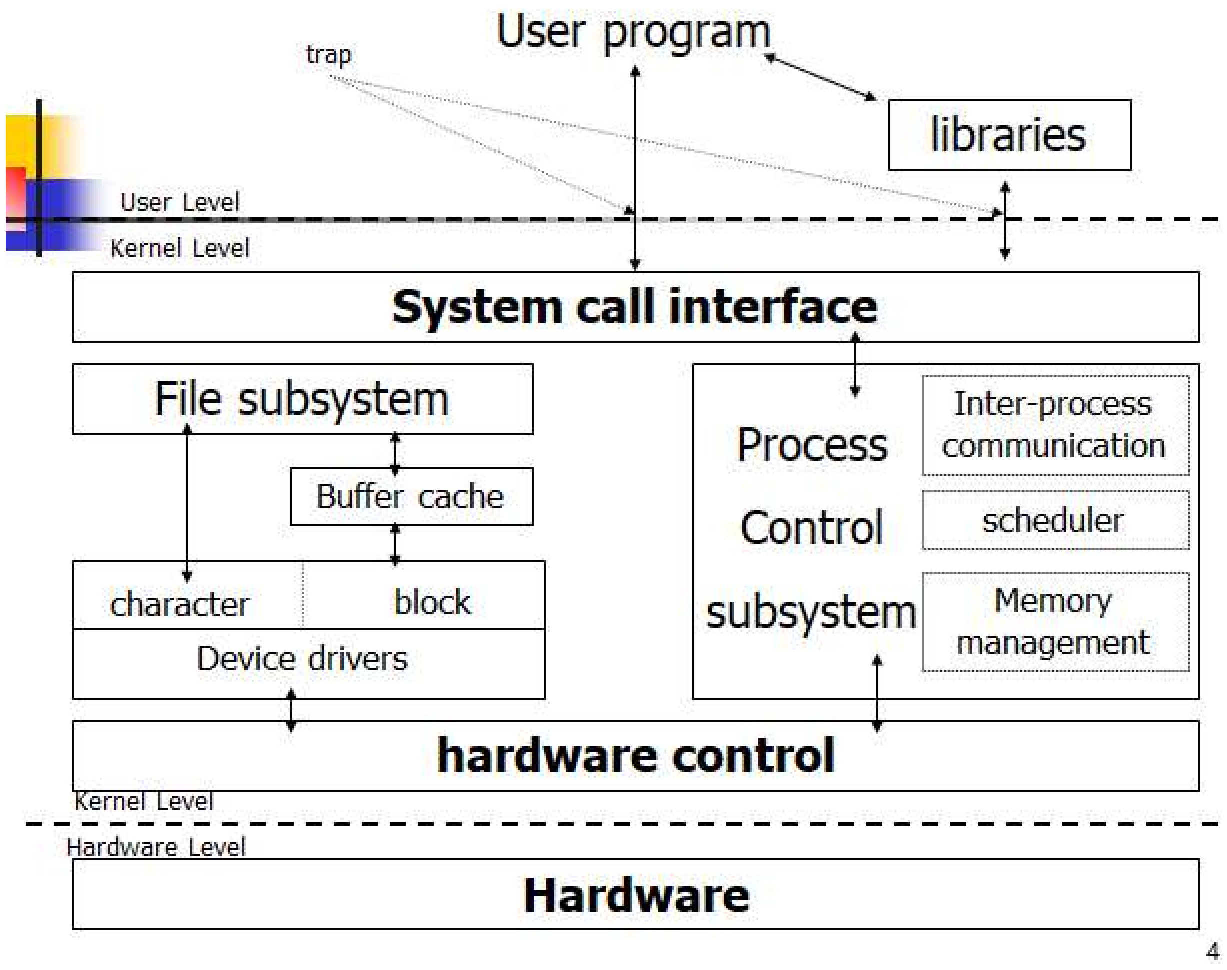
Ans: Superblock is responsible for how many files user can write

Superblock represent state of table.



Q4. With block diagram explain System Kernel and enlist the responsibilities of process control subsystem and file subsystem.

Ans:



The process control subsystem is responsible for process synchronization, interprocess communication, memory management, and process scheduling. The file subsystem and the process control subsystem interact when loading a file into memory for execution, as will be seen in Chapter 7: the process subsystem reads executable files into memory before executing them.

Q 5. Which kernel data structure describes the state of a process?

Ans: Process table entry and U area

Q6. Suppose the kernel does a delayed wrote of a block. What happen when another process takes that block from its hash queue? From the free list?

Ans: :-ii)It will copy all data from block to disk asynchronously and will allocates another buffer from free list.

Q7. What are advantages to kernel in maintaining the superblock in the file system?

Ans: Describes the state of a file system

Describes the size of the file systemi.e.How many files it can store