Q.1 a. What is kernel? Differentiate the micro kernel from the macro kernel.

Answer:

a. Kernel is the core and essential part of the operating system. It provides basic service for all essential parts of operating system**.**

Micro kernel : run services those are minimal for OS performance. In this kernel All other operations are performed by processor

Macro kernel: it is a combination of micro and monolithic kernel(having the whole OS code as single executable image).

b. Explain the concurrency of process execution in a single processor environment and parallelism in shared-memory multiprocessor environment.

Answer:

Concurrency and Parallelism

In a multithreaded process on a single processor, the processor can switch execution resources between threads, resulting in concurrent execution.

Concurrency indicates that more than one thread is making progress, but the threads are not actually running simultaneously. The switching between threads happens quickly enough that the threads might appear to run simultaneously.

In the same multithreaded process in a shared-memory multiprocessor environment, each thread in the process can run concurrently on a separate processor, resulting in **parallel execution**, which is true simultaneous execution.

c. Why synchronization hardware is not a feasible solution in the Multi processor environment? And which is the proper alternate?

Answer:

The critical section problem could be solved easily in a **single-processor environment**, by disallowing interrupts to occur while a shared variable or resource is being modified

In this manner, we could be sure that the current sequence of instructions would be allowed to execute in order without pre-emption. Unfortunately, this solution is not feasible in a multiprocessor environment

Disabling interrupt on a multiprocessor environment can **be time consuming** as the message is passed to all the processors

This message transmission lag, delays entry of threads into critical section and the system efficiency decreases

The alternate solution to this is to use **MutexLock**

d. What are the different strategies involved in address binding of instructions and data to memory addresses?

Answer:

Address binding of instructions and data to memory addresses

It can happen at three different stages

Compile time: If memory location known a priori, absolute code can be generated

Needs to recompile the code if starting location changes

Load time: if memory location is not known at compile time, relocatable code has to be generated **Execution time:** Binding delayed until run time if the process can be moved during its execution from one memory segment to another

Need hardware support for address maps (e.g., base and limit registers)

e. In the layered approach to file system which organizes storage on disk drives, what are the roles of logical file system and file organization module?

Answer:

The roles of **logical file system, file organization module**

The **file organization module**

knows about files and their logical blocks, and how they map to physical blocks on the disk.

In addition to translating from logical to physical blocks, the file organization module also maintains the list of free blocks, and allocates free blocks to files as needed.

The **logical file system**

deals with all of the meta data associated with a file (UID, GID, mode, dates, etc), i.e. everything about the file except the data itself.

This level manages the directory structure and the mapping of file names to ***file control blocks, FCBs***, which contain all of the meta data as well as block number information for finding the data on the disk.

f. Write down the definition of a distributed system and explain the naming and transparency of the distributed file system.

Answer:

The roles of **logical file system, file organization module**

The **file organization module**

knows about files and their logical blocks, and how they map to physical blocks on the disk.

In addition to translating from logical to physical blocks, the file organization module also maintains the list of free blocks, and allocates free blocks to files as needed.

The **logical file system**

deals with all of the meta data associated with a file (UID, GID, mode, dates, etc), i.e. everything about the file except the data itself.

This level manages the directory structure and the mapping of file names to ***file control blocks, FCBs***, which contain all of the meta data as well as block number information for finding the data on the disk.

g. What is meant by language based protection in systems? (7×4)

Answer:

Language based protection

Specification of protection in a programming language allows the high-level description of policies for the allocation and use of resources.

Language implementation can provide software for protection enforcement when automatic hardware-supported checking is unavailable.

Interpret protection specifications to generate calls on whatever protection system is provided by the hardware and the operating system.

Q.2 Explain how the thread creation differs from the process creation and which is costlier? (4)

Answer:

When a new thread is created it shares

its code section, data section and operating system resources like open files with other threads. But it is allocated its own stack, register set and a program counter.

The creation of a new process differs from that of a thread mainly in the fact that all the shared resources of a thread are needed explicitly for each process.

So though two processes may be running the same piece of code they need to have their own copy of the code in the main memory to be able to run.

Two processes also do not share other resources with each other. This makes the creation of a new process very costly compared to that of a new thread.

b. Briefly state how the process synchronization happens in Windows XP. (3)

Answer:

Synchronization in Windows

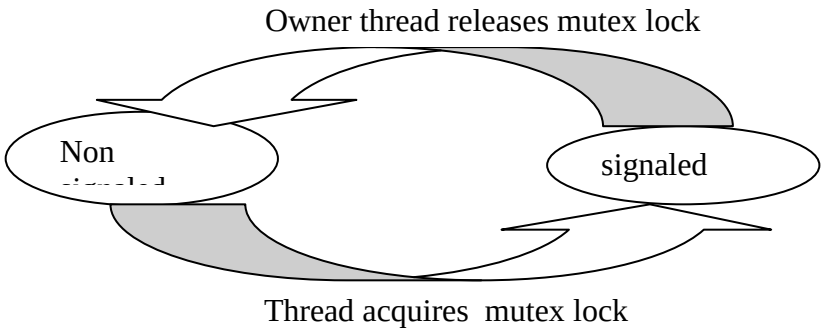


Fig: Mutex dispatcher object

c. Compare the logical and physical address spaces (4)

Answer:

Logical Versus Physical Address Space

The address generated by the CPU is a **logical address**, whereas the address actually seen by the memory hardware is a **physical address**

Addresses bound at compile time or load time have identical logical and physical addresses

Addresses created at execution time, however, have different logical and physical addresses In this case the logical address is also known as a **virtual address**,

The set of all logical addresses used by a program composes the **logical address space**, and the set of all corresponding physical addresses composes the **physical address space**

d. Briefly explain the role of Memory Management Unit with a simple schematic that shows dynamic relocation using a relocation register. (7)

Answer:

The run time mapping of logical to physical addresses is handled by the **memory-management unit, MMU.** The MMU can take on many forms. One of the simplest is a modification of the base-register scheme

The base register is termed as **relocation register**, whose value is added to every memory request at the hardware level.user programs never see physical addresses.

User programs work entirely in logical address space, and any memory references or manipulations are done using purely logical addresses. Only when the address gets sent to the physical memory chips is the physical memory address generated.

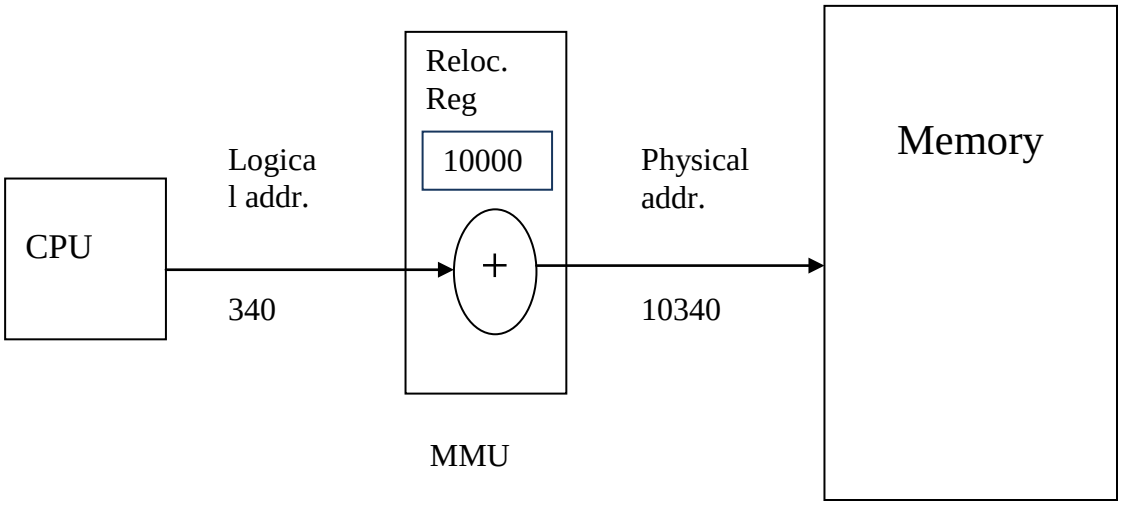


Fig: Dynamic relocation using a relocation register

Q.3 a. Explain the Peterson's solution to the critical section problem. (9)

Answer:

Peterson's solution to critical section problem

Peterson's Solution is a classic software-based solution to the critical section problem. it illustrates a number of important concepts.

Peterson's solution is based on two processes, Pi and Pj, which alternate between their critical sections and remainder sections.

Peterson's solution requires two shared data items:

int turn - Indicates whose turn it is to enter into the critical section. If turn = = i, then process i is allowed into their critical section.

Boolean flag[2] - Indicates when a process wants to enter into their critical section. When process i wants to enter their critical section, it sets flag[i] to true.

Do{

```
Flag[i]=TRUE;

Turn=j;

While(Flag[j]==TRUE  &&  Turn == j);

Critical section;

Flag[i]=FALSE;

Remainder section;

}While(TRUE);
```

From the code, the entry and exit sections are enclosed in boxes.

In the entry section, process i first raises a flag indicating a desire to enter the critical section.

Then turn is set to j to allow the other process to enter their critical section if process j so desires.

The while loop is a busy loop (notice the semicolon at the end), which makes process i wait as long as process j has the turn and wants to enter the critical section.

Process i lowers the flag[i] in the exit section, allowing process j to continue if it has been waiting.

b. Describe the necessary requirements a solution to the critical section problem must satisfy. (9)

Answer:

To prove that the solution is correct, the conditions listed below have to be examined:

Mutual exclusion - If one process is executing their critical section when the other wishes to do so, the second process will become blocked by the flag of the first process. If both processes attempt to enter at the same time, the last process to execute "turn = j" will be blocked.

Progress - Each process can only be blocked at the while if the other process wants to use the critical section (flag[j] = = true), and it is the other process's turn to use the critical section (turn = = j).

If both of those conditions are true, then the other process (j) will be allowed to enter the critical section, and upon exiting the critical section, will set flag[j] to false, releasing process i.

The shared variable turn assures that only one process at a time can be blocked, and the flag variable allows one process to release the other when exiting their critical section.

Bounded Waiting - As each process enters their entry section, they set the turn variable to be the other processes turn. Since no process ever sets it back to their own turn, this ensures that each process will have to let the other process go first at most one time before it becomes their turn again.

The instruction "turn = j" is **atomic,** that is it is a single machine instruction which cannot be interrupted.

Q.4 **a. Apply the FIFO Page replacement policy on the following reference string and find the number of page faults**

(i) if 3 frames are used

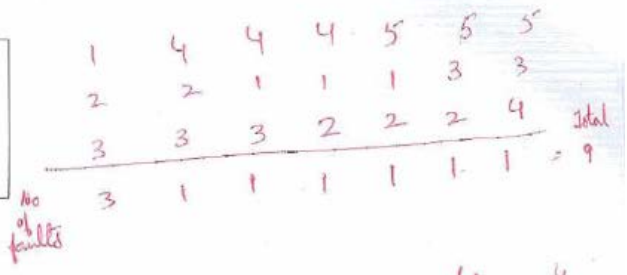
(ii) if 4 frames are used **(4+4)**

1	2	3	4	1	2	5	1	2	3	4	5

Answer:

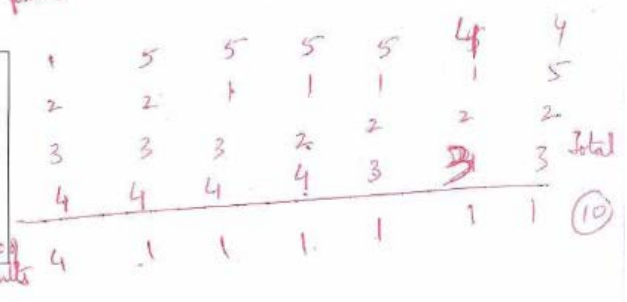
a. If 3 frames are used then

1	4	5	9 page faults
2	1	3	
3	2	4	



If 4 frames are used

1	5	4	10 page faults
2	1	5	
3	2		
4	3		



If frames are increased to 4, then number of page faults also increases, to 10.

b. Briefly explain the following criteria to compare CPU scheduling algorithms:

(i) CPU utilization
(ii) Throughput
(iii) Turnaround time
(iv) Waiting time
(v) Response time **(2x5)**

Answer:

Different CPU scheduling algorithms have different properties, and the choice of a particular algorithm may favor one class of processes over another. In choosing which algorithm to use in a particular situation, we must consider the properties of the various algorithms.

Many criteria have been suggested for comparing CPU scheduling algorithms. Which characteristics are used for comparison can make a substantial difference in which algorithm is judged to be best. The criteria include the following:

- **CPU utilization**. We want to keep the CPU as busy as possible. Conceptually, CPU utilization can range from 0 to 100 percent. In a real system, it should range from 40 percent (for a lightly loaded system) to 90 percent (for a heavily used system).
- **Throughput**. If the CPU is busy executing processes, then work is being done. One measure of work is the number of processes that are completed per time unit, called *throughput*. For long processes, this rate may be one process per hour; for short transactions, it may be 10 processes per second.
- **Turnaround time**. From the point of view of a particular process, the important criterion is how long it takes to execute that process. The interval from the time of submission of a process to the time of completion is the *turnaround time*. Turnaround time is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O.
- **Waiting time**. The CPU scheduling algorithm does not affect the amount of time during which a process executes or does I/O; it affects only the amount of time that a process spends waiting in the ready queue. *Waiting time* is the sum of the periods spent waiting in the ready queue.
- **Response time**. In an interactive system, turnaround time may not be the best criterion. Often, a process can produce some output fairly early and can continue computing new results while previous results are being output to the user. Thus, another measure is the time from the submission of a request until the first response is produced. This measure, called *response time*, is the time it takes to start responding, not the time it takes to output the response. The turnaround time is generally limited by the speed of the output device.

It is desirable to maximize CPU utilization and throughput and to minimize turnaround time, waiting time, and response time. In most cases, we optimize the average measure. However, under some circumstances, it is desirable

to optimize the minimum or maximum values rather than the average. For example, to guarantee that all users get good service, we may want to minimize the maximum response time.

Investigators have suggested that, for interactive systems (such as time-sharing systems), it is more important to minimize the *variance* in the response time than to minimize the average response time. A system with reasonable and *predictable* response time may be considered more desirable than a system that is faster on the average but is highly variable. However, little work has been done on CPU-scheduling algorithms that minimize variance.

As we discuss various CPU-scheduling algorithms in the following section, we will illustrate their operation. An accurate illustration should involve many processes, each being a sequence of several hundred CPU bursts and I/O bursts. For simplicity, though, we consider only one CPU burst (in milliseconds) per process in our examples. Our measure of comparison is the average waiting time. More elaborate evaluation mechanisms are discussed in Section 5.7.

Q.5 a. Describe how the two-memory access problem is solved by the use of fast-look up cache called TLBs and in a simple paging system. What information is stored in a typical TLB table entry? Explain. (4)

Answer:

The implementation of page-table is done in the following way:

Page table is kept in main memory.

Page-table base register (PTBR) points to the page table.

Page-table length register (PRLR) indicates size of the page table.

In this scheme every data/instruction access requires two memory accesses. One for the page table and one for the data/instruction.

The two-memory access problem can be solved by the use of a special fast-lookup hardware cache called associative memory or translation look-aside buffers (TLBs).

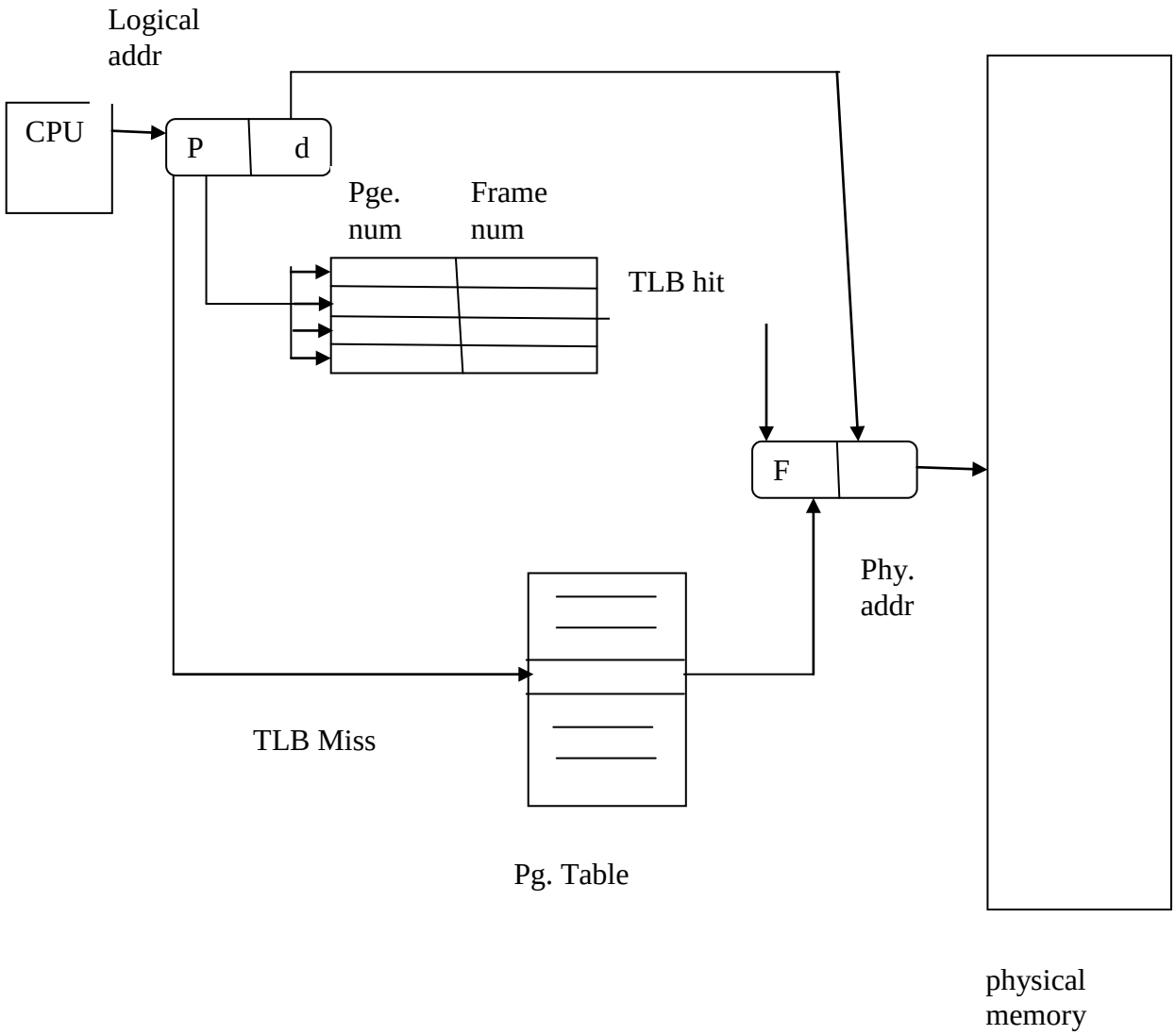
A set of associative registers is built of high-speed memory where each register consists of two parts: **a key and a value**. When the associative registers are presented with an item, it is compared with all keys simultaneously. If the item is found, the corresponding value field is the output.

A typical TLB table entry consists of page number and frame number,

when a logical address is generated by the CPU, its page number is presented to a set of associative registers that contain page number along with their corresponding frame numbers.

If the page number is found in the associative registers, its frame number is available and is used to access memory. If the page number is not in the associated registers, a memory reference to the page table must be made. When the frame number is

obtained, it can be used to access memory and the page number along with its frame number is added to the associated register



b. Consider the following set of processes, with the length of CPU burst given in milliseconds:

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	3
P_4	1	4
P_5	5	2

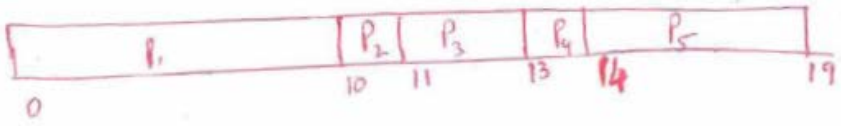
The processes are assumed to have arrived in the order P_1, P_2, P_3, P_4 and P_5 all at time 0.

(i) Draw four Gantt charts that illustrate the execution of these processes using: FCFS, SJF, nonpreemptive priority (a smaller priority number implies a higher priority) and RR (quantum = 2) (4)

(ii) What is the turnaround time of each process for each of the scheduling algorithms in part (i)? (2.5 x 4)

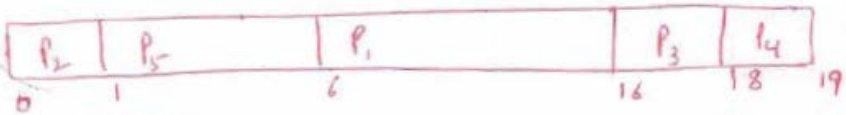
Answer:

Ans 5(b) (i) FCFS



SJF

Non-preemptive Priority



→ Gantt chart of RR on back

(ii) FCFS

Turnaround time of

$P_1 = 10$

$P_2 = 11$

$P_3 = 13$

$P_4 = 14$

$P_5 = 19$

SJF (Turnaround time of)

$P_1 = 19$

$P_2 = 1$

$P_3 = 4$

$P_4 = 2$

$P_5 = 9$

Non-preemptive priority

Turnaround time of

$P_1 = 16$

$P_2 = 1$

$P_3 = 18$

$P_4 = 19$

$P_5 = 6$

RR (quantum = 2)

Turnaround time of

$P_1 = 19$

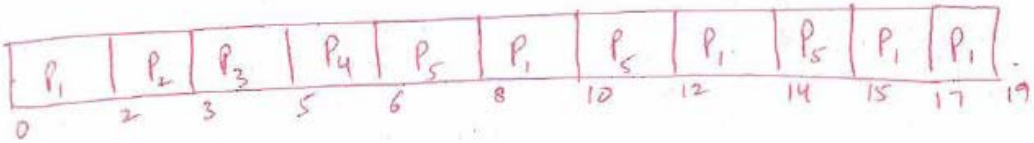
$P_2 = 3$

$P_3 = 5$

$P_4 = 6$

$P_5 = 15$

(b) (i)



Q.6 a. (i) Why the location independence plays a critical role in naming structure of DFS system?

(ii) Detail the different cache update polices such as write-through, delayed-write in distributed systems. (3+3)

Answer:

(i) Location independence

file name does not need to be changed when the file's physical storage location changes.

Better file abstraction.

Promotes sharing the storage space itself.

Separates the naming hierarchy form the storage-devices hierarchy.

(ii) Cache Update Policies

Write-through – write data through to disk as soon as they are placed on any cache. Reliable, but poor performance.

Delayed-write – modifications written to the cache and then written through to the server later. Write accesses complete quickly; some data may be overwritten before they are written back, and so need never be written at all.

Poor reliability; unwritten data will be lost whenever a user machine crashes.

Variation – scan cache at regular intervals and flush blocks that have been modified since the last scan.

Variation – write-on-close, writes data back to the server when the file is closed. Best for files that are open for long periods and frequently modified.

b. (i) Draw the schematic that shows the components of a Linux Operating System.
(ii) Brief about the Linux kernel modules. (3+3)

Answer:

(i) The components of a Linux Operating System

<table>
<tr><td>System management programs</td><td>User process</td><td>User utility programs</td><td>compilers</td></tr>
<tr><td colspan="4">System shared libraries</td></tr>
<tr><td colspan="4">Linux kernel</td></tr>
<tr><td colspan="4">Loadable kernel modules</td></tr>
</table>

(ii) Kernel Modules:

The Sections of kernel code that can be compiled, loaded, and unloaded independent of the rest of the kernel.

A kernel module may typically implement a device driver, a file system, or a networking protocol.

The module interface allows third parties to write and distribute, on their own terms, device drivers or file systems that could not be distributed under the GPL.

Kernel modules allow a Linux system to be set up with a standard, minimal kernel, without any extra device drivers built in.

Three components to Linux module support:

module management

driver registration

conflict resolution

c. Explain symmetric encryption method. (6)

Answer:

In a **symmetric encryption algorithm**, the same key is used to encrypt and to decrypt. That is, $E(k)$ can be derived from $D(k)$, and vice versa. Therefore, the secrecy of $E(k)$ must be protected to the same extent as that of $D(k)$.

For the past 20 years or so, the most commonly used symmetric encryption algorithm in the United States for civilian applications has been the **data-encryption standard (DES)** adopted by the National Institute of Standards and Technology (NIST). DES works by taking a 64-bit value and a 56-bit key and performing a series of transformations. These transformations are based on substitution and permutation operations, as is generally the case for symmetric encryption transformations. Some of the transformations are **black-box transformations**, in that their algorithms are hidden. In fact, these so-called "S-boxes" are classified by the United States government. Messages longer than 64 bits are broken into 64-bit chunks, and a shorter block is padded to fill out the block. Because DES works on a chunk of bits at a time, is a known as a **block cipher**. If the same key is used for encrypting an extended amount of data, it becomes vulnerable to attack. Consider, for example, that the same source block would result in the same ciphertext if the same key and encryption algorithm were used. Therefore, the chunks are not just encrypted but also XORed with the previous ciphertext block before encryption. This is known as **cipher-block chaining**.

DES is now considered insecure for many applications because its keys can be exhaustively searched with moderate computing resources. Rather than giving up on DES, though, NIST created a modification called **triple** DES, in which the DES algorithm is repeated three times (two encryptions and one decryption) on the same plaintext using two or three keys—for example, $c = E(k_3)(D(k_2)(E(K_1)(m)))$. When three keys are used, the effective key length is 168 bits. Triple DES is in widespread use today.

In 2001, NIST adopted a new encryption algorithm, called the **advanced encryption standard (AES)**, to replace DES. AES is another symmetric block cipher. It can use key lengths of 128, 192, and 256 bits and works on 128-bit blocks. It works by performing 10 to 14 rounds of transformations on a matrix formed from a block. Generally, the algorithm is compact and efficient.

There are several other symmetric block encryption algorithms in use today that bear mentioning. The **twofish** algorithm is fast, compact, and easy to implement. It can use a variable key length of up to 256 bits and works on 128-bit blocks. **RC5** can vary in key length, number of transformations, and block size. Because it uses only basic computational operations, it can run on a wide variety of CPUs.

RC4 is perhaps the most common stream cipher. A **stream cipher** is designed to encrypt and decrypt a stream of bytes or bits rather than a block. This is useful when the length of a communication would make a block cipher too slow. The key is input into a pseudo–random-bit generator, which is an algorithm that attempts to produce random bits. The output of the generator when fed a key is a keystream. A **keystream** is an infinite set of keys that can be used for the input plaintext stream. RC4 is used in encrypting steams of data such as in WEP, the wireless LAN protocol. It is also used in communication between web browsers and web servers, as we discuss below. Unfortunately, RC4 as used in WEP (IEEE standard 802.11) has been found to be breakable in a reasonable amount of computer time. In fact, RC4 itself has vulnerabilities.

Q.7 a. Explain indexed allocation method for blocks on disk. (10)

Answer:

Linked allocation solves the external-fragmentation and size-declaration problems of contiguous allocation. However, in the absence of a FAT, linked allocation cannot support efficient direct access, since the pointers to the blocks are scattered with the blocks themselves all over the disk and must be retrieved in order. **Indexed allocation** solves this problem by bringing all the pointers together into one location: the **index block**.

Each file has its own index block, which is an array of disk-block addresses. The i^{th} entry in the index block points to the i^{th} block of the file. The directory contains the address of the index block (Figure 11.8). To find and read the ith block, we use the pointer in the i^{th} index-block entry. This scheme is similar to the paging scheme described in Section 8.4.

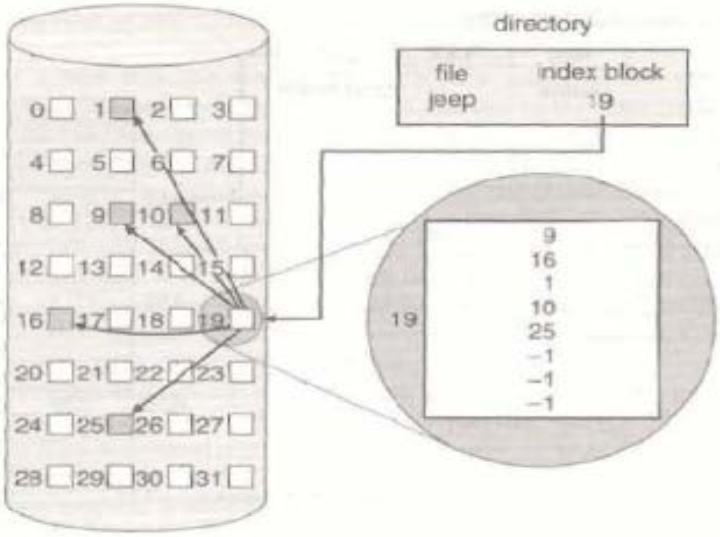


Figure 11.8 Indexed allocation of disk space.

When the file is created, all pointers in the index block are set to *nil*. When the ith block is first written, a block is obtained from the free-space manager, and its address is put in the ith index-block entry.

Indexed allocation supports direct access, without suffering from external fragmentation, because any free block on the disk can satisfy a request for more space. Indexed allocation does suffer from wasted space, however. The pointer overhead of the index block is generally greater than the pointer overhead of linked allocation. Consider a common case in which we have a file of only one or two blocks. With linked allocation, we lose the space of only one pointer per block. With indexed allocation, an entire index block must be allocated, even if only one or two pointers will be non-*nil*.

This point raises the question of how large the index block should be. Every file must have an index block, so we want the index block to be as small as possible. If the index block is too small, however, it will not be able to hold enough pointers for a large file, and a mechanism will have to be available to deal with this issue. Mechanisms for this purpose include the following:

- **Linked scheme**. An index block is normally one disk block. Thus, it can be read and written directly by itself. To allow for large files, we can link together several index blocks. For example, an index block might contain a small header giving the name of the file and a set of the first 100 disk-block addresses. The next address (the last word in the index block) is *nil* (for a small file) or is a pointer to another index block (for a large file).
- **Multilevel index**. A variant of the linked representation is to use a first-level index block to point to a set of second-level index blocks, which in turn point to the file blocks. To access a block, the operating system uses the first-level index to find a second-level index block and then uses that

block to find the desired data block. This approach could be continued to a third or fourth level, depending on the desired maximum file size. With 4,096-byte blocks, we could store 1,024 4-byte pointers in an index block. Two levels of indexes allow 1,048,576 data blocks and a file size of up to 4 GB.

- **Combined scheme**. Another alternative, used in the UFS, is to keep the first, say, 15 pointers of the index block in the file's inode. The first 12 of these pointers point to **direct blocks**; that is, they contain addresses of blocks that contain data of the file. Thus, the data for small files (of no more than 12 blocks) do not need a separate index block. If the block size is 4 KB, then up to 48 KB of data can be accessed directly. The next three pointers point to **indirect blocks**. The first points to a **single indirect block**, which is an index block containing not data but the addresses of blocks that do contain data. The second points to a **double indirect block**, which contains the address of a block that contains the addresses of blocks that contain pointers to the actual data blocks. The last pointer contains the address of a **triple indirect block**. Under this method, the number of blocks that can be allocated to a file exceeds the amount of space addressable by the 4-byte file pointers used by many operating systems. A 32-bit file pointer reaches only 2^{32} bytes, or 4 GB. Many UNIX implementations, including Solaris and IBM's AIX, now support up to 64-bit file pointers. Pointers of this size allow files and file systems to be terabytes in size. A UNIX inode is shown in Figure 11.9.

Indexed-allocation schemes suffer from some of the same performance problems as does linked allocation. Specifically, the index blocks can be cached in memory, but the data blocks may be spread all over a volume.

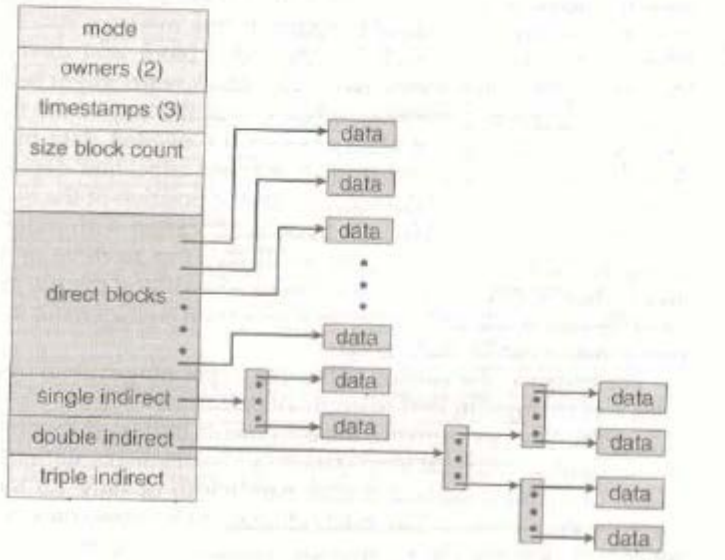


Figure 11.9 The UNIX inode.

b. Explain the main characteristics of

(i) real time systems (ii) multimedia systems (4+4)

Answer:

In this section, we discuss the features necessary for designing an operating system that supports real-time processes. Before we begin, though, let's consider what is typically *not* needed for a real-time system. We begin by examining several features provided in many of the operating systems discussed so far in this text, including Linux, UNIX, and the various versions of Windows. These systems typically provide support for the following:

- A variety of peripheral devices such as graphical displays, CD, and DVD drives
- Protection and security mechanisms
- Multiple users

Supporting these features often results in a sophisticated—and large—kernel. For example, Windows XP has over forty million lines of source code. In contrast, a typical real-time operating system usually has a very simple design, often written in thousands rather than millions of lines of source code. We would not expect these simple systems to include the features listed above.

But why don't real-time systems provide these features, which are crucial to standard desktop and server systems? There are several reasons, but three are most prominent. First, because most real-time systems serve a single purpose, they simply do not require many of the features found in a desktop PC. Consider a digital wristwatch: It obviously has no need to support a disk drive or DVD, let alone virtual memory. Furthermore, a typical real-time system does not include the notion of a user: The system simply supports a small number of tasks, which often await input from hardware devices (sensors, vision identification, and so forth). Second, the features supported by standard desktop operating systems are impossible to provide without fast processors and large amounts of memory. Both of these are unavailable in real-time systems due to space constraints, as explained earlier. In addition, many real-time systems lack sufficient space to support peripheral disk drives or graphical displays, although some systems may support file systems using nonvolatile memory (NVRAM). Third, supporting features common in standard desktop computing environments would greatly increase the cost of real-time systems, which could make such systems economically impractical.

Additional considerations apply when considering virtual memory in a real-time system. Providing virtual memory features as described in Chapter 9 require the system include a memory management unit (MMU) for translating logical to physical addresses. However, MMUs typically increase the cost and power consumption of the system. In addition, the time required to

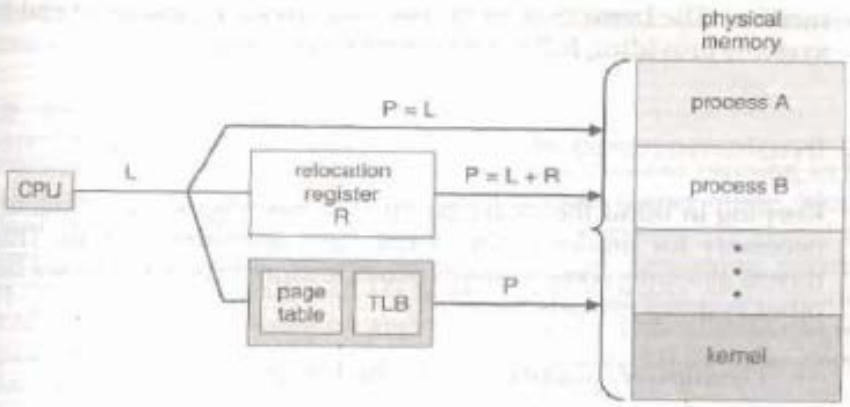


Figure 19.2 Address translation in real-time systems.

translate logical addresses to physical addresses—especially in the case of a translation look-aside buffer (TLB) miss—may be prohibitive in a hard real-time environment. In the following we examine several appraoches for translating addresses in real-time systems.

Figure 19.2 illustrates three different strategies for managing address translation available to designers of real-time operating systems. In this scenario, the CPU generates logical address L that must be mapped to physical address P. The first approach is to bypass logical addresses and have the CPU generate physical addresses directly. This technique—known as **real-addressing mode**—does not employ virtual memory techniques and is effectively stating that P equals L. One problem with real-addressing mode is the absence of memory protection between processes. Real-addressing mode may also require that programmers specify the physical location where their programs are loaded into memory. However, the benefit of this approach is that the system is quite fast, as no time is spent on address translation. Real-addressing mode is quite common in embedded systems with hard real-time constraints. In fact, some real-time operating systems running on microprocessors containing an MMU actually disable the MMU to gain the performance benefit of referencing physical addresses directly.

A second strategy for translating addresses is to use an approach similar to the dynamic relocation register shown in Figure 8.4. In this scenario, a relocation register R is set to the memory location where a program is loaded. The physical address P is generated by adding the contents of the relocation register R to L. Some real-time systems configure the MMU to perform this way. The obvious benefit of this strategy is that the MMU can easily translate logical addresses to physical addresses using $P = L + R$. However, this system still suffers from a lack of memory protection between processes.

The last approach is for the real-time system to provide full virtual memory functionality as described in Chapter 9. In this instance, address translation takes place via page tables and a translation look-aside buffer, or TLB. In addition to allowing a program to be loaded at any memory location, this strategy also provides memory protection between processes. For systems without attached disk drives, demand paging and swapping may not be possible. However, systems may provide such features using NVRAM flash memory. The LynxOS and OnCore Systems are examples of real-time operating systems providing full support for virtual memory.

The demands of multimedia systems are unlike the demands of traditional applications. In general, multimedia systems may have the following characteristics:

1. Multimedia files can be quite large. For example, a 100-minute MPEG-1 video file requires approximately 1.125 GB of storage space; 100 minutes of high-definition television (HDTV) requires approximately 15 GB of storage. A server storing hundreds or thousands of digital video files may thus require several terabytes of storage.
2. Continuous media may require very high data rates. Consider digital video, in which a frame of color video is displayed at a resolution of 800×600. If we use 24 bits to represent the color of each pixel (which allows us to have 2^{24}, or roughly 16 million, different colors), a single frame requires $800 \times 600 \times 24 = 11,520,000$ bits of data. If the frames are displayed at a rate of 30 frames per second, a bandwidth in excess of 345 Mbps is required.
3. Multimedia applications are sensitive to timing delays during playback. Once a continuous-media file is delivered to a client, delivery must continue at a certain rate during playback of the media; otherwise, the listener or viewer will be subjected to pauses during the presentation.

TEXT BOOK

I. Operating System Principles, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne(2009), Johnwiley & Sons (Asia) Pte Ltd
II. Modern Operating Systems" Andre S Tanenbaum(2009) Pearson Education