	b)	What are the disadvantages of single contiguous memory allocation? Explain.	L2	CO1	5 M
		OR			
9	a)	Consider a disk queue with following requests for I/O to blocks on cylinders 30,70,115,130,110,80,20,25(Assume disk head is at 90) Draw FCFS and SSTF scheduling and also determine how many times the disk head changes its direction for each of the above mentioned scheduling techniques.	L3	CO3	5 M
	b)	What is virtual memory? Discuss the benefits of virtual memory techniques.	L2	CO1	5 M
		UNIT-V			
10	a)	Explain different File Attributes and File Operations.	L2	CO1	5 M
	b)	Explain the three allocation methods in file system implementation. Illustrate with proper diagram.	L3	CO4	5 M
		OR			
11	a)	Explain the concept of file sharing. What are the criteria to be followed in systems which implement file sharing?	L2	CO4	5 M
	b)	Explain about domains of protection.	L2	CO1	5 M

PVP 23

Code: 23CS3401, 23IT3401

II B.Tech - II Semester – Regular Examinations - MAY 2025

OPERATING SYSTEMS
(Common for CSE, IT)

Duration: 3 hours Max. Marks: 70

Note: 1. This question paper contains two Parts A and B.
2. Part-A contains 10 short answer questions. Each Question carries 2 Marks.
3. Part-B contains 5 essay questions with an internal choice from each unit. Each Question carries 10 marks.
4. All parts of Question paper must be answered in one place.

BL – Blooms Level CO – Course Outcome

PART – A

		BL	CO
1.a)	Explain the role of system calls in an operating system.	L2	CO1
1.b)	Differentiate between Free and Open-Source Operating Systems.	L2	CO1
1.c)	Explain the purpose of process scheduling in an operating system.	L2	CO1
1.d)	Describe the role of multithreading in improving system performance.	L2	CO1
1.e)	Explain the critical section problem with an example.	L2	CO1
1.f)	Explain how Peterson's solution helps in solving the critical section problem.	L2	CO1
1.g)	Describe how virtual memory helps in efficient memory utilization.	L2	CO1
1.h)	Differentiate between paging and segmentation.	L2	CO1

1.i)	Describe the role of directory structures in file management.	L2	CO1
1.j)	Explain different file access methods with examples.	L2	CO1

PART – B

			BL	CO	Max. Marks
		UNIT-I			
2	a)	Describe various operating system services with examples.	L2	CO1	6 M
	b)	Demonstrate the role of system programs in OS functionality.	L3	CO1	4 M
		OR			
3	a)	Compare different computing environments and their impact on OS design.	L2	CO1	5 M
	b)	Examine the significance of system calls in process management.	L4	CO1	5 M
		UNIT-II			
4	a)	Explain different states of a process with a process state diagram.	L2	CO1	5 M
	b)	What is Round Robin Scheduling? Illustrate with an example. Can it be useful for a single user system? If yes, then explain. If no, then why not?	L3	CO2	5 M
		OR			

5	a)	Compare preemptive and non-preemptive scheduling with examples.	L2	CO1	5 M
	b)	Compare multiple processor scheduling techniques and their impact on performance.	L4	CO4	5 M
		UNIT-III			
6	a)	How semaphores can be used to deal with n-process critical section problem? Explain.	L3	CO2	5 M
	b)	Explain Banker's deadlock-avoidance algorithm with an illustration.	L2	CO3	5 M
		OR			
7	a)	Explain Deadlock Detection scheme for Several Instances of a resource Type.	L3	CO3	5 M
	b)	What is semaphore? Why it is important? Suggest the solution for bounded buffer problem with semaphores.	L2	CO1	5 M
		UNIT-IV			
8	a)	Consider the following reference string 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1. Assume there are three frames. Apply LRU replacement algorithm to the reference string above and find out how many page faults are produced. Illustrate the LRU page replacement algorithm in detail and also two feasible implementations of the LRU algorithm.	L3	CO3	5 M

II B.Tech – II Semester – Regular Examination – May -2025

Operating Systems (23CS3401/23IT3401)

Short Scheme

Part-A

1. a) Role of System Calls in an Operating System (2M)

1.b) Free-Source Operating System vs Open-Source Operating System (2M)

1.c) Purpose of process scheduling in an operating system (2M)

1.d) Role of multithreading in improving system performance (2M)

1.e) Critical Section Problem (1M)

Examples (1M)

1.f) Peterson's Solution (2M)

1.g) Virtual Memory (2M)

1.h) Paging vs Segmentation (2M)

1.i) Directory Structures in file management (2M)

1.j) File Access Methods (2M)

PART-B

2.a) Operating Systems Services (6M)

2.b) Role of System Programs is OS Functionality (4M)

3.a) Different Computing Environments (5M)

3.b) Significance of System Calls in process Management (5M)

4.a) Process (5M)

4.b) Round Robin Scheduling (5M)

5.a) Preemptive vs Non-preemptive Scheduling (5M)

5.b) Multiple Processor Scheduling (5M)

6.a) Semaphores (5M)

6.b) Banker's Deadlock-avoidance Algorithm (5M)

7.a) Deadlock Detection (5M)

7.b) Semaphores (5M)

8.a) LRU Page Replacement Algorithm (5M)

8.b) Disadvantages of Single Contiguous Memory Allocation (5M)

9.a) FCFS(2.5M) SSTF(2.5M)

9.b) Virtual Memory (3M) Benefits (2M)

10.a) File Attributes (2M)

File Operations (3M)

10. b) Allocation methods (5M)

11.a) File Sharing (5M)

11. b) Domain of protection (5M)

II B.Tech – II Semester – Regular Examination – May -2025
Operating Systems (23CS3401/23IT3401)
Long Scheme
Part-A

1. a) Role of System Calls in an Operating System (2M)

System calls act as the interface between user applications and the operating system kernel, allowing user programs to request services from the OS.

Key Roles of System Calls

1. Process Control
2. File Management
3. Device Management
4. Information Maintenance
5. Communication
6. Protection and Security

1.b) Free-Source Operating System vs Open-Source Operating System (2M)

Free-Source Operating System	Open-Source Operating System
Users have the freedom to use, study, modify and distribute the software	Open-source means the source code is available for inspection, modification, and distribution, usually und an OSI-approved license.
Often copyleft: derivatives must also be free.	May allow permissive licenses: derivatives can be proprietary.

1.c) Purpose of process scheduling in an operating system (2M)

Purposes of Process Scheduling

1. Efficient CPU Utilization: Minimizes CPU idle time by selecting ready processes to run.
2. Throughput: Maximizes the number of processes completed per unit time.
3. Turnaround Time: Aims to minimize the time taken from process submission to completion.
4. Waiting Time: Tries to reduce the time processes spend in the ready queue.

1.d) Role of multithreading in improving system performance (2M)

Multithreading is a technique where a single process is divided into multiple threads that run concurrently. These threads share the same memory space but can execute different parts of a program simultaneously.

Benefits of Multithreading

1. Improved CPU utilization
2. Efficient Resource Sharing
3. Increased Throughput
4. Scalability on Multi-core Systems

1.e) Critical Section Problem (2M)

A critical section is a part of a program where the process accesses shared resources. To ensure correctness, only one process should execute its critical section at any given time.

Examples: Bounded-Buffer Problem, Readers-Writers Problem, The Dining-Philosophers Problem.

1.f) Peterson's Solution (2M)

Peterson's solution states that when one process is executing its critical section then the other process executes the rest of the code and vice versa.

Peterson solution requires two shared data items:

1) turn: indicates whose turn it is to enter into the critical section. If turn == i , then process i is allowed into their critical section.

2) flag: indicates when a process wants to enter into critical section. When process i wants to enter their critical section, it sets flag[i] to true.

1.g) Virtual Memory (2M)

Virtual memory is a memory management technique that allows an operating system (OS) to use disk space as an extension of RAM, creating the illusion of a large, contiguous block of memory for each process

1. Enables execution of Larger Program
2. Efficient use of RAM
3. Prevents Fragmentation.

1.h) Paging vs Segmentation (2M)

Paging	**Segmentation**
Divides memory into fixed-size blocks called pages (logical) and frames (physical).	Divides memory into variable-sized segments based on logical divisions in the program (e.g., code, data, stack).
Suffers from internal fragmentation	May suffer from external fragmentation
Uses a page table	Uses a segment table

1.i) Directory Structures in file management (2M)

In an operating system, directory structures play a crucial role in organizing, managing, and accessing files efficiently.

1. Organization of Files
2. Access control and security
3. Efficient File location and Access
4. Supports File Operations

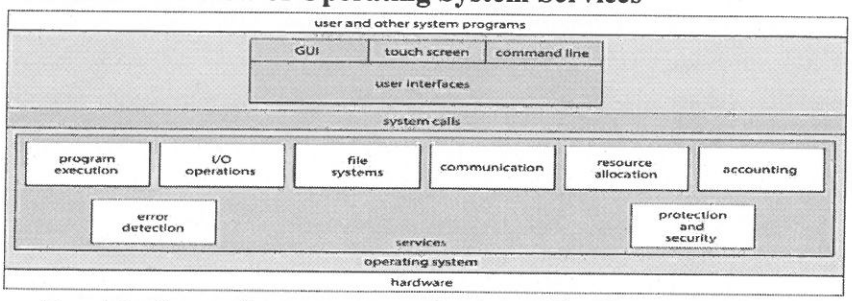
1.j) File Access Methods (2M)

1. Sequential Access: Information in the file is processed in order, one record after the other.
2. Direct access is also called relative access. Here records can read/write randomly without any order. The direct access method is based on a disk model of a file, because disks allow random access to any file block.
3. Indexed Sequential File Access: Records are organized in sequence based on a key field.

PART-B

2. a) Operating Systems Services (6M)

Operating systems provide an environment for execution of programs and services to programs and users. **A View of Operating System Services**



- One set of operating-system services provides functions that are helpful to the user:
  - **User interface** - Almost all operating systems have a user interface (UI).
    - ✓ Varies between Command-Line (CLI), Graphics User Interface (GUI), touch-screen
  - **Program execution** - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
  - **I/O operations** - A running program may require I/O, which may involve a file or an I/O device
  - **File-system manipulation** - The file system is of particular interest. Programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.
  - **Communications** – Processes may exchange information, on the same computer or between computers over a network
    - ✓ Communications may be via shared memory or through message passing (packets moved by the OS)
  - **Error detection** – OS needs to be constantly aware of possible errors
    - ✓ May occur in the CPU and memory hardware, in I/O devices, in user program
    - ✓ For each type of error, OS should take the appropriate action to ensure correct and consistent computing
    - ✓ Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system

2.b) Role of System Programs is OS Functionality (4M)

System programs, also known as system utilities provide a convenient environment for program development and execution. They can be divided into:

- **File management**: These programs create, delete, copy, rename, print, dump, list and generally manipulate files and directories.
- **Status information:** Some ask the system for info - date, time, amount of available memory, disk space, and number of users. Others provide detailed performance, logging, and debugging information.
- **Programming language support:** Compilers, assemblers, debuggers, and interpreters for common programming languages are often provided with the operating system.
- **Program loading and execution:** Once a program is assembled or compiled, it must be loaded into memory to be executed.
- **Communications:** These programs provide the mechanism for creating virtual connections among processes, users, and computer systems.

3.a) Different Computing Environments (5M)

The different computing environments are-

1. **Traditional Computing**

   The current trend is toward providing more ways to access these computing environments. Web technologies are stretching the boundaries of traditional computing. Companies establish portals, which provide web accessibility to their internal servers. Network computers are essentially terminals that understand web-based computing. Handheld computers can synchronize with PCs to allow very portable use of company information.

2. **Mobile Computing**

   Mobile Computing is possible through Handheld smartphones, tablets, etc.

The Main functional differences between mobile phones and a "traditional" laptop are :

- Mobiles support Extra feature – more OS features (GPS(Global Positioning System) – for finding locations, gyroscope- for orientation,sliding,tilting etc)
- Allows new types of apps like augmented reality
- Use IEEE 802.11 wireless, or cellular data networks for connectivity

3. **Distributed Systems**

- Collection of separate, possibly heterogeneous, systems networked together
- Network is a communications path, TCP/IP most common
  - ✓ Local Area Network (LAN)
  - ✓ Wide Area Network (WAN)
  - ✓ Metropolitan Area Network (MAN)
  - ✓ Personal Area Network (PAN)
- Network Operating System provides features between systems across network
  - Communication scheme allows systems to exchange messages
  - Different computers communicate closely enough to provide the Illusion that only single operating system controls the network.---- Distributed operating system

4. **Client-Server Computing**

Designers shifted away from centralized system architecture to - terminals connected to centralized systems. As a result, many of today's systems act as server systems to satisfy requests generated by client systems. This form of specialized distributed system, called client-server system.

5. **Cloud Computing**

- Delivers computing, storage, even apps as a service across a network
- Logical extension of virtualization because it uses virtualization as the base for it functionality.
- Many types
  - ✓ Public cloud – available via Internet to anyone willing to pay
  - ✓ Private cloud – run by a company for the company's own use
  - ✓ Hybrid cloud – includes both public and private cloud components
  - ✓ Software as a Service (SaaS) – one or more applications available via the Internet (i.e., word processor)
  - ✓ Platform as a Service (PaaS) – software stack ready for application use via the Internet (i.e., a database server)
  - ✓ Infrastructure as a Service (IaaS) – servers or storage available over Internet (i.e., storage available for backup use)

3.b) Significance of System Calls in process Management (5M)

- System calls can be grouped into six major categories: process control, file manipulation, device manipulation, information maintenance, communications, and protection.

Process Control

- If a system call is made to terminate the currently running program abnormally, or if the program runs into a problem and causes an error trap, a dump of memory is sometimes taken and an error message generated.
- The dump is written to disk and may be examined by a debugger – errors, or bugs – to determine the cause of the problem.
- Process control do the following:
  - create process, terminate process
  - end (halt program execution normally) , abort (halt program execution annormally)
  - load and execute programs
  - get process attributes, set process attributes
  - wait for time to finish execution of currently executing jobs
  - wait event, signal event
  - allocate and free memory
  - Locks for managing access to shared data between processes

4.a) Process (5M)

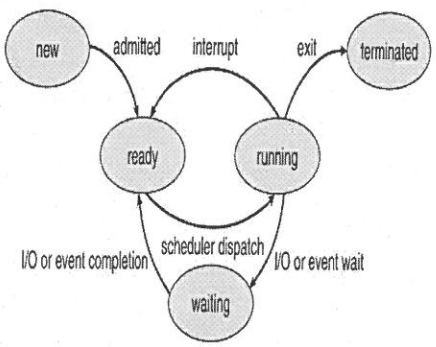
- A process is the unit of work in a modern time-sharing system.
- An operating system executes a variety of programs that run as a process.

Process – a program in execution; process execution must progress in sequential fashion. No parallel execution of instructions of a single process

- Multiple parts of a process:
  - ✓ The program code, also called text section
  - ✓ Stack containing temporary data
  - ✓ Data section containing global variables
  - ✓ Heap containing memory dynamically allocated during run time
- Program is passive entity stored on disk (executable file): process is active
  - Program becomes process when an executable file is loaded into memory
- Execution of program started via GUI mouse clicks, command line entry of its name, etc.
- One program can be several processes
  - Consider multiple users executing the same program.

Process State

As a process executes, it changes state
New: The process is being created
Ready: The process is waiting to be assigned to a processor
Running: Instructions are being executed
Waiting: The process is waiting for some event to occur
Terminated: The process has finished execution



4.b) Round Robin Scheduling (5M)

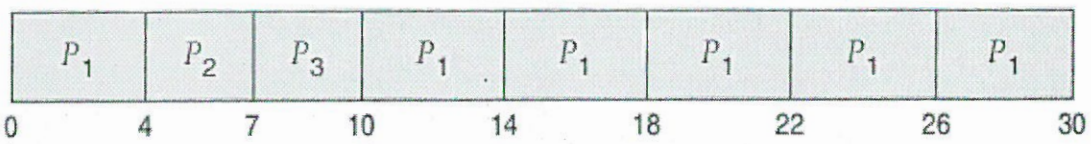
- The round-robin (RR) scheduling algorithm is designed especially for time sharing systems. Each process gets a small unit of the CPU time called the time slices or time quantum, which is usually 10- 100 milliseconds.
- The ready queue is treated as a circular queue. The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum. If a process CPU burst exceeds 1 time quantum, that process is preempted and is put back in the ready queue. The round robin (RR) scheduling algorithm is preemptive scheduling algorithm.
- After the time quantum elapsed and the process has not finished it's execution the timer sets an interrupt to the operating system. A context switch will be executed, and the process will be put at the tail of the ready queue. The CPU scheduler will then select the next process in the ready queue. It is a preemptive scheduling algorithm.
- If there are n processes in the ready queue and the time quantum is q, then each process gets 1/n of the CPU time in chunks of at most q time units at once. No process waits more than (n-1)q time units.
- The average waiting time under the RR policy is often long.

Example : Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds:

Process	Burst Time
P_1	24
P_2	3
P_3	3

If we use a time quantum of 4 milliseconds, then process P1 gets the first 4 milliseconds. Since it requires another 20 milliseconds, it is preempted after the first time quantum, and the CPU is given to the next process in the queue, process P2 . Process P2 does not need 4 milliseconds, so it quits before its time quantum expires. The CPU is then given to the next process, process P3 . Once each process has received 1 time quantum, the CPU is returned to process P1 for an additional time quantum.

The resulting RR schedule is as follows:



Once each process has received 1 time quantum,
The CPU is returned to process P1 for an additional time quantum.
So, The waiting time for process P1 = (10-4) = 6 milliseconds
The waiting time for process P2 = 4 milliseconds
The waiting time for process P3 = 7 milliseconds
Thus, the average waiting time = (6 + 4 + 7)/3 = 17/3 = 5.66 milliseconds.

5.a) Preemptive vs Non-preemptive Scheduling (5M)

S. No.	Preemptive Scheduling	Non-preemptive Scheduling
1	A scheduling scheme is said to be preemptive **if and only if process switches from the running state or waiting state to the ready sate.**	A scheduling scheme is said to be non-preemptive **if a process switches from the running state to the waiting state or if a process terminates.**
2	In this scheduling, the server **before completing the current request will switch to a new request for processing.**	In this scheduling, **the server will switch to a new request only after completing the currently scheduled request.**
3	Before a request gets completed, it may have to be **scheduled for many times.**	Scheduling is performed **only after completing the previously scheduled request.**
4	Here, preempted request is **placed back into the pending requests list.**	Here, **preemptive of request will never occur.**
5	The preemptive scheduling policies include **Round Robin (RR) Scheduling with time slicing.**	The non-preemptive scheduling policies include **FCFS Scheduling.**

5.b) Multiple Processor Scheduling

Multiple-Processor Scheduling :

CPU scheduling more complex when multiple CPUs are available

- Multiprocess may be any one of the following architectures :
  - Multicore CPUs
  - Multithreaded cores
  - NUMA systems (Non Uniform Memory Access)
  - Heterogeneous multiprocessing

Approaches to implement Multiple processor scheduling

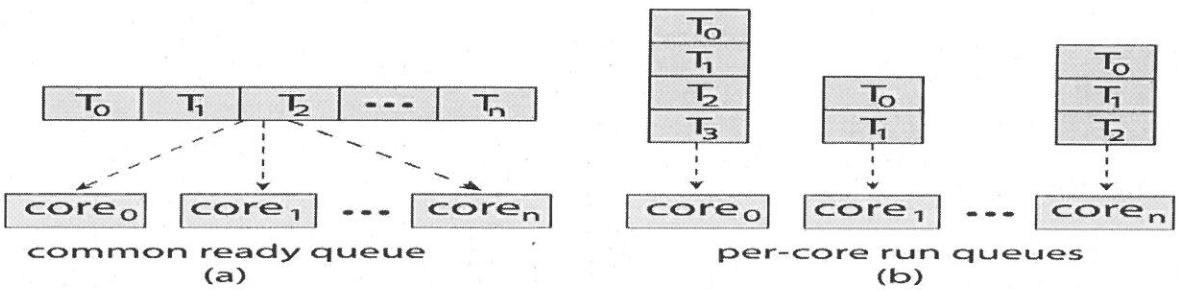
- 1. ASMP (Asymmetric Multi Processing)
- 2. SMP (Symmetric Multi Processing)

1.ASMP (Asymmetric Multi Processing)

Master Server will do all scheduling decisions, I/O Processing and other system activities. Here only one processor accesses the system data structures and reduces the need for data sharing other processors execute user code.

2. SMP (Symmetric Multi Processing)

Here each processor is self scheduling and a)All processes may be in common ready queue or b) each processor may have it's own private queue of ready processes. We must ensure that two seperatre processors do not choose to schedule the same process and that processes are not lost from the queue.



Issues with SMP Systems :

i) **Processor Affinity :**

Attempt to keep a process running on a same processor then it is said to be that the process has an Affinity for the processor on which it is currently running.

This can be divided into two types :

a) Soft Affinity : When on operating system has a policy of attempting to keep a process running on the same processor ,but not guaranteeing that it will do so.(i.e process may migrate between processors)

b) Hard Affinity: Some system provides system calls allowing a process to specify subset of processors on which it may run .

Example : Linux implements soft affinity, but also provides sched_setAffinity()
Sys call which supports hard affinity.

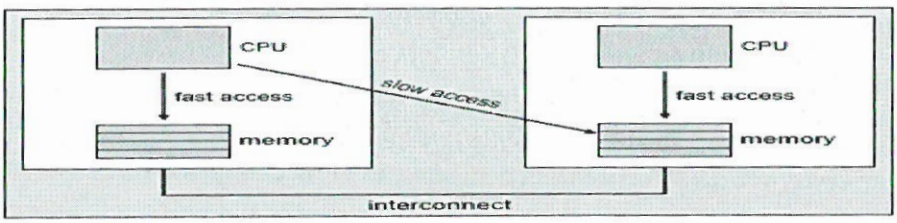
ii) **Load balancing :**

- To keep workload evenly distributed across all processors in an SMP system.
- Load balancing is important only on systems where each processor has it's own private queue of eligible processes
- On systems with common run queue load balancing is often necessary.

There are two general approaches to balance the load:

a. **Push Migration** : here a specific task periodically checks the load on each processor and if it finds an imbalance , it evenly distributes the load by moving or pushing processes from overloaded processors to idle or less busy processors.
b. **Pull Migration** : it occurs when an idle processor pulls waiting task from a busy processor

These two approaches need not mutually exclusive and often implemented in parallel



NUMA and CPU scheduling

iii) **Multicore Processors** : keep multiple processor cores on the same physical chip. Each core maintains it's architectural state and thus appears to the operating system to be a seperate physical processor.

SMP systems that use multicore processors are faster and consume less power than systems in which each processor has it's own physical chip. In general Multicore systems may complicate scheduling Issues.

6.a) Semaphores (5M)

- It is a Synchronization tool that provides more sophisticated ways (than Mutex locks) for processes to synchronize their activities.
- Semaphore is represented by S as a integer variable
- It Can only be accessed via two indivisible (atomic) operation
    - wait() and signal()
- Definition of the wait() operation

```
wait(S) {
        while (S <= 0)
        ; // busy wait
        S--;
        }
```

- Definition of the signal() operation

```
signal(S) {
        S++;
        }
```

All the modifications to the integer values of the semaphore in the wait() and signal() operations must be executed indivisibly. i.e when one process modifies the semaphore value no other process can simultaneously modify that semaphore value.

Semaphore Usage :

Types of Semaphores :

- Binary semaphore – integer value can range only between 0 and 1
  Same as a mutex lock, as they are locks that provide mutual exclusion

  Solution to the CS Problem :

  Create a semaphore "mutex" initialized to 1

```
wait(mutex);
    CS
signal(mutex);
```

- Counting semaphore – integer value can range over an unrestricted domain
- It is used to control the access to a resource that has multiple instances.

We Can implement a counting semaphore S as a binary semaphore .With semaphores we can solve various synchronization problems.

Solution to the CS Problem :

For example, consider two concurrently running processes: P1 with a statement S1 and P2 with a statementS2. Suppose we require that S2 be executed only after S1 has completed. We can implement this scheme readily by letting P1 and P2 share a common semaphore synch, initialized to 0.

In process P1, we insert the statements

```
S1;
signal(synch);
```

In process P2, we insert the statements

```
wait(synch);
S2;
```

Because synch is initialized to 0, P2 will execute S2 only after P1 has invoked signal(synch), which is after statement S1 has been executed.

6.b) Banker's Deadlock-avoidance Algorithm (5M)

The following Data structures are used to implement the Banker's Algorithm:
Let 'n' be the number of processes in the system and 'm' be the number of resource types.
1. Available
It is a 1-D array of size 'm' indicating the number of available resources of each type.
Available[j] = k means there are 'k' instances of resource type Rj
2. Max
It is a 2-d array of size 'n*m' that defines the maximum demand of each process in a system.
Max[i, j] = k means process Pi may request at most 'k' instances of resource type Rj.
3. Allocation
It is a 2-d array of size 'n*m' that defines the number of resources of each type currently allocated to each process.
Allocation[i, j] = k means process Pi is currently allocated 'k' instances of resource type Rj
4. Need
It is a 2-d array of size 'n*m' that indicates the remaining resource need of each process.
Need [i, j] = k means process Pi currently needs 'k' instances of resource type Rj
Need [i, j] = Max [i, j] – Allocation [i, j]
Allocation specifies the resources currently allocated to process Pi and Need specifies the additional resources that process Pi may still request to complete its task.
Banker's algorithm consists of a Safety algorithm and a Resource request algorithm.

Safety Algorithm :

The algorithm for finding out whether or not a system is in a safe state can be described as follows:
1. Let Work and Finish be vectors of length 'm' and 'n' respectively.
   Initialize: Work = Available
   Finish[i] = false; for i=1, 2, 3, 4….n
2. Find an i such that both
   2.1 Finish[i] = false
   2.2 Needi <= Work
   if no such i exists goto step (4)
3. Work = Work + Allocation[i]
   Finish[i] = true
   goto step (2)
4. if Finish [i] = true for all i
   then the system is in a safe state

Resource-Request Algorithm :

Let Requesti be the request array for process Pi. Requesti [j] = k means process Pi wants k instances of resource type Rj. When a request for resources is made by process Pi, the following actions are taken:
1. If Requesti <= Needi
Goto step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum claim.
2. If Requesti <= Available
Goto step (3); otherwise, Pi must wait, since the resources are not available.
3. Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows:
Available = Available – Requesti
Allocationi = Allocationi + Requesti
Needi = Needi– Requesti

Example: Any example

7.a) Deadlock Detection (5M)

If a system does not employ either a deadlock-prevention or a deadlock avoidance algorithm, then a deadlock situation may occur. In this environment, the system may provide:

- An algorithm that examines the state of the system to determine whether a deadlock has occurred
- An algorithm to recover from the deadlock

Several Instances of a Resource Type

A deadlock detection algorithm that is applicable to several instances of a resource type. The algorithm employs several time-varying data structures that are similar to those used in the banker's algorithm.

- Available: A vector of length m indicates the number of available resources of each type.
- Allocation: Ann x m matrix defines the number of resources of each type currently allocated to each process.
- Request: An n x m matrix indicates the current request of each process. If Request[i][j] equals k, then process P; is requesting k more instances of resource type Rj.

Algorithm:

1. Let Work and Finish be vectors of length m and n, respectively Initialize:

(a) Work = Available

(b) For i = 1,2, …, n, if Allocationi□ 0, then Finish[i] = false; otherwise, Finish[i] = true

2. Find an index isuch that both:

(a) Finish[i] == false

(b) Requesti□Work

If no such i exists, go to step 4

3. Work = Work + Allocationi

Finish[i] = true

go to step 2

4. If Finish[i] == false, for some i, 1 □i□n, then the system is in deadlock state. Moreover, if

Finish[i] == false, then Pi is deadlocked

7.b) Semaphores (5M)

- It is a Synchronization tool that provides more sophisticated ways (than Mutex locks) for processes to synchronize their activities. Semaphore is represented by S as a integer variable
- It Can only be accessed via two indivisible (atomic) operation
    - wait() and signal()

The Bonded-Buffer Problem

- Producer puts information into the buffer, consumer takes it out. The problem arise when the producer wants to put a new item in the buffer, but it is already full. The solution is for the producer has to wait until the consumer has consumed atleast one buffer.
- Similarly, if the consumer wants to remove an item from the buffer and sees that the buffer is empty, it goes to sleep until the producer puts something in the buffer and wakes it up.

Synchronization problems:

1. We must guard against attempting to write data to the buffer when the buffer is full, i.e., the producer must wait for an 'empty space'.
2. We must prevent the consumer from attempting to read data when the buffer is empty; i.e., the consumer must wait for 'data available'.
3. To provide for each of these conditions, we require to employ three semaphores. The producer and consumer processes share the following data structure:

```
int n;
semaphore mutex=1;
semaphore empty=n;
semaphore full=0;
```

- We assume that the pool consists of n buffers, each capable of holding one item. The mutex semaphore provides mutual exclusion for accesses to the buffer pool and is initialized to the value 1.
- The empty and full semaphores count the number of empty and full buffers. The semaphore empty is initialized to n; the semaphore full is initialized to 0.
- The code for the producer process is shown below:

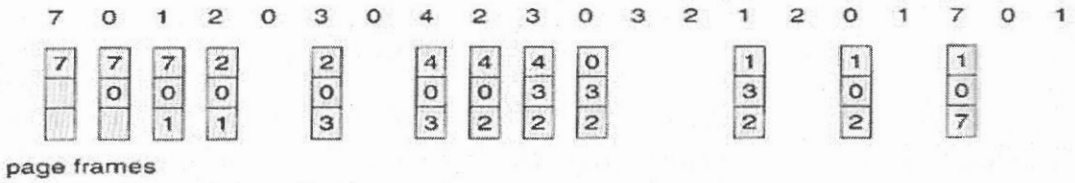
```
do {
//produce an item in next_produced
wait(empty) ;
wait(mutex) ;
// add next_produced to buffer
signal(mutex) ;
signal(full) ;
}while (TRUE);
```

- The code for the consumer process is shown below:

```
do {
wait(full) ;
wait (mutex) ;
// remove an item from buffer to next_consumed
signal(mutex) ;
signal(empty) ;
// consume the item in next_consumed
}while(TRUE);
```

8.a) LRU Page Replacement Algorithm (5M)

- The prediction behind LRU, the Least Recently Used algorithm is that the page that has not been used in the longest time is the one that will not be used again in the near future.
- LRU replacement associates with each page the time of that page's last use. When a page must be replaced, LRU chooses that page that has not been used for the longest period of time. This strategy is the optimal page-replacement algorithm looking backward in time, rather than forward.
- The following figure illustrates LRU for our sample string, yielding 12 page faults, (as compared to 15 for FIFO and 9 for OPT.)



LRU page-replacement algorithm.

Step	Page	Frame Contents	Page Fault?
1	7	[7]	✓ Yes
2	0	[7, 0]	✓ Yes
3	1	[7, 0, 1]	✓ Yes
4	2	[0, 1, 2]	✓ Yes (7 out)
5	0	[0, 1, 2]	✗ No
6	3	[1, 2, 3]	✓ Yes (0 out)
7	0	[2, 3, 0]	✓ Yes (1 out)
8	4	[3, 0, 4]	✓ Yes (2 out)
9	2	[0, 4, 2]	✓ Yes (3 out)
10	3	[4, 2, 3]	✓ Yes (0 out)
11	0	[2, 3, 0]	✓ Yes (4 out)
12	3	[2, 0, 3]	✗ No
13	2	[0, 3, 2]	✗ No
14	1	[3, 2, 1]	✓ Yes (0 out)
15	2	[3, 1, 2]	✗ No
16	0	[1, 2, 0]	✓ Yes (3 out)
17	1	[2, 0, 1]	✗ No
18	7	[0, 1, 7]	✓ Yes (2 out)
19	0	[0, 1, 7]	✗ No
20	1	[0, 1, 7]	✗ No

- Total Page Faults = 12
- Total Page Hits = 8
- Final Frame Content = [0, 1, 7]

8.b) Disadvantages of Single Contiguous Memory Allocation (5M)

Single Contiguous Memory Allocation is the simplest memory management technique where entire memory (except for the OS) is allocated as one single block to a process. While simple, it has several drawbacks:

1. Poor Memory Utilization

Only one user process can reside in memory at a time. Remaining memory is wasted even if it could accommodate other smaller processes.

2. No Multiprogramming

Since only one process is allowed in memory (besides the OS), multiprogramming is not possible. This leads to low CPU utilization during I/O wait times.

3. Wastage of Memory (Internal Fragmentation)

If the process size is smaller than the available memory block, the remaining space remains unused (internal fragmentation).

4. No Process Isolation

If a program goes rogue, it can potentially overwrite the OS or crash the entire system, as there is no memory protection.

5. Inefficient for Modern Systems

Modern applications require dynamic memory, multitasking, and complex memory layouts — which this scheme cannot support.

6. Static Allocation

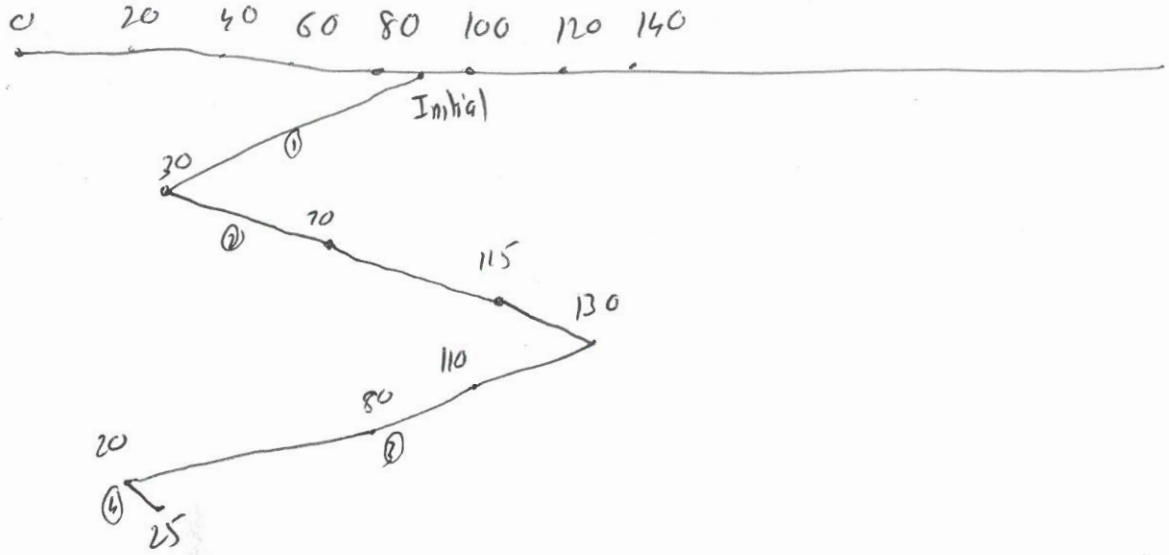
Memory is allocated at load time and cannot grow or shrink dynamically, making it inflexible.

7. Not Scalable

Not suitable for systems that run multiple concurrent applications.

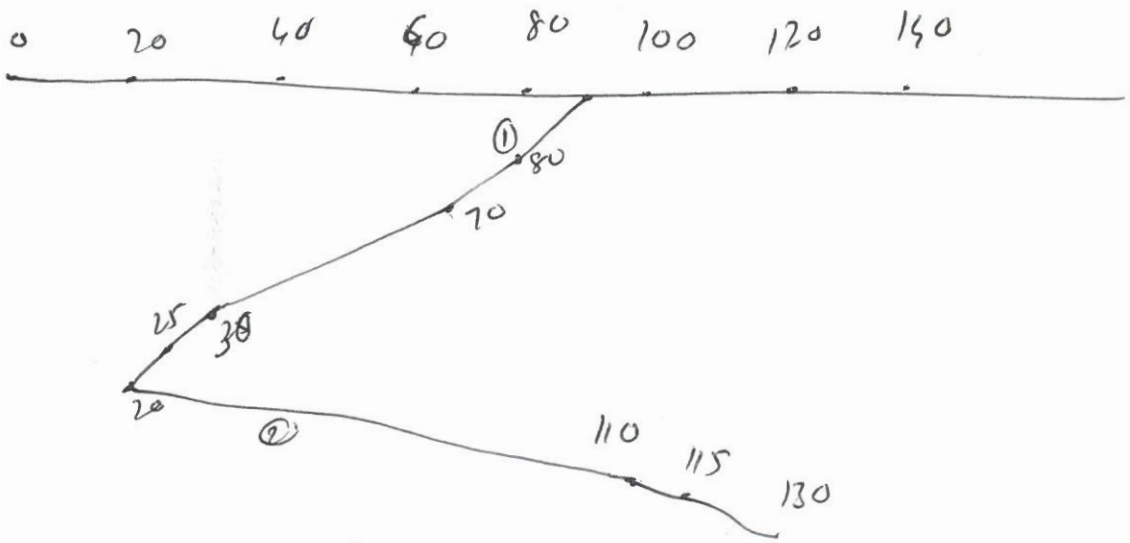
9.a) 30, 70, 115, 130, 110, 80, 20, 25

FCFS (2.5M)



Disk Head Changed direction 4 times

SSTF (2.5M)



Disk Head Changes direction 2 times

27
16

9.b) Virtual Memory (3M)

- Virtual Memory is a space where large programs can store themselves in form of pages while their execution and only the required pages or portions of processes are loaded into the main memory. This technique is useful as large virtual memory is provided for user programs when a very small physical memory is there.
- In real scenarios, most processes never need all their pages at once, for following reasons :
  - Error handling code is not needed unless that specific error occurs, some of which are quite rare.
  - Arrays are often over-sized for worst-case scenarios, and only a small fraction of the arrays are actually used in practice.
  - Certain features of certain programs are rarely used.

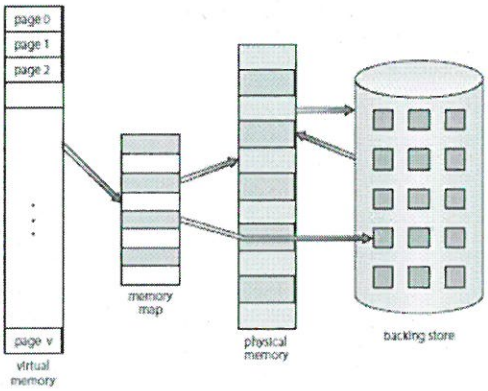


Figure 10.1 Diagram showing virtual memory that is larger than physical memory.

- Virtual memory involves the separation of logical memory as perceived by users from physical memory. This separation allows an extremely large virtual memory to be provided for programmers when only a smaller physical memory is available.
- Virtual memory makes the task of programming much easier, because the programmer non longer needs to worry about the amount of physical memory available.
- Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory

Benefits of having virtual memory (2M)

1) Large programs can be written, as virtual space available is huge compared to physical memory.
2) Less I/O required, leads to faster and easy swapping of processes.
3) More physical memory available, as programs are stored on virtual memory, so they occupy very less space on actual physical memory.

10.a) File Attributes (2M)

A file's attributes vary from one operating system to another but typically consist of these:

Name – The symbolic file name is the only information kept in human-readable form

Identifier – The unique tag (number) identifies file within file system

Type – This information is needed for systems that support different types

Location – This information is a pointer to file location on device

Size – The current size of the file

Protection – Access-control information determines who can do reading, writing, executing

Time, date, and user identification – This information may be kept for creation, last modification, and last use..

- The information about all files is kept in the directory structure, which also resides on secondary storage.
- Typically, a directory entry consists of the file's name and its unique identifier. The identifier in turn locates the other file attributes.

File Operations (3M)

- A file is an abstract data type. To define a file properly, we need to consider the operations that can be performed on files.
- Six basic operations comprise the minimal set of required file operations.

Creating a file: Two steps are necessary to create a file. First, space in the file system must be found for the file. Second, an entry for the new file must be made in the directory.

Writing a file: To write a file, we have to know two things. One is name of the file name and second is the information or data to be written on the file, the system searches the entire given location for the file. If the file is found, the system must keep a write pointer to the location in the file where the next write is to take place.

Reading a file: To read a file, first of all we searches the directories for the file. If the file is found, the system needs to keep a read pointer to the location in the file where the next read is to take place. Once the read has taken place, the read pointer is updated.

Repositioning within a file: The directory is searched for the appropriate entry and the current-file-position pointer is repositioned to a given value. This operation is also called **file seek.**

Deleting a file: To delete a file, first of all search the directory for the named file, then released the file space and erase the directory entry.

Truncating a file: To truncate a file, remove the file contents only but, attributes are as it is.

- Several pieces of data are needed to manage open files:

Open-file table: contains information about all open files

File pointer: pointer to last read/write location, per process that has the file open

File-open count: counter of number of times a file is open – to allow removal of data from open-file table when last process closes it

Access rights: Each process opens a file in an access mode. This information is stored on the per-process table so the operating system can allow or deny subsequent I/O requests.

File Locking

- Provided by some operating systems and file systems
  - Similar to reader-writer locks
  - **Shared lock** similar to reader lock – several processes can acquire concurrently
  - **Exclusive lock** similar to writer lock

10. b) Allocation methods (5M)

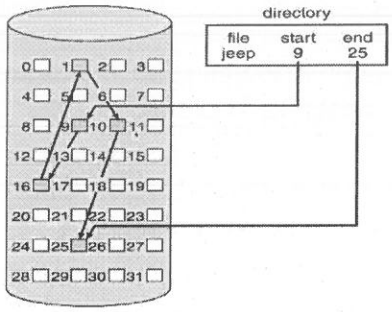
- The direct-access nature of disks gives us flexibility in the implementation of files. In almost every case, many files are stored on the same disk.
- The main problem is how to allocate space to these files so that disk space is utilized effectively and files can be accessed quickly.
- An allocation method refers to how disk blocks are allocated for files:
- Three major methods of allocating disk space are in wide use:

1) Contiguous Allocation

- Contiguous allocation requires that each file occupy a set of contiguous blocks of disks.
- Contiguous allocation of a file is defined by the address of the first block and length (in block units) of the file. If the file is n blocks long and starts at location b, then it occupies blocks b, b + 1, b + 2, ..., b + n − 1.
- The directory entry for each file indicates the address of the starting block and the length of the area allocated for this file.
- Accessing a file that has been allocated contiguously is easy.
  - For sequential access, the file system remembers the address of the last block referenced and, when necessary, reads the next block.
  - For direct access to block i of a file that starts at block b, we can immediately access block b + i.
  - Thus, both sequential and direct access can be supported by contiguous allocation.
- Problems with contiguous allocation include
  - Finding space for a new file on disk
  - Determining how much space is needed for a file.
  - Suffers from external fragmentation.

2) Linked Allocation

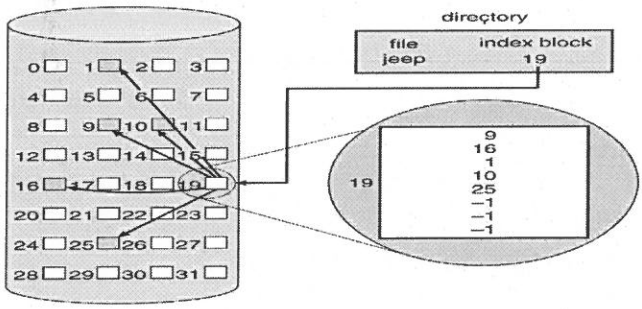
- Linked allocation solves all problems of contiguous allocation.
- Each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file.
- For example, a file of five blocks might start at block 9 and continue at block 16, then block 1, then block 10, and finally block 25.



- Each block contains a pointer to the next block. These pointers are not made available to the user. Thus, if each block is 512 bytes in size, and a disk address requires 4 bytes, then the user sees blocks of 508 bytes.
- To create a new file, we simply create a new entry in the directory. With linked allocation, each directory entry has a pointer to the first disk block of the file. This pointer is initialized to null to signify an empty file. The size field is also set to 0.
- A write to the file causes the free-space management system to find a free block, and this new block is written to and is linked to the end of the file.
- To read a file we simply read blocks by following the pointers from block to block.
- There is no external fragmentation with linked allocation, and any free block on the free-space list can be used to satisfy a request.

3) Indexed Allocation

- Indexed allocation brings 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. To find and read the i^{th} block, we use the pointer in the i^{th} block is first written, a block is obtained from the free-space manager, and its address is put in the i^{th} 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.

11.a) File Sharing (5M)

- Once multiple users are allowed to share files, the challenge is to extend sharing to multiple file systems, including remote file systems.

Multiple Users

- Given a directory structure that allows files to be shared by users, the system must mediate the file sharing. The system can either allow a user to access the files of other users by default or require that a user specifically grant access to the files.
- Most systems have evolved to use the concepts of file (or directory) owner (or user) and group. The owner is the user who can change attributes and grant access and who has the most control over the file. The group attribute defines a subset of users who can share access to the file.
- The owner and group IDs of a given file are stored with the other file attributes.
- When a user request an operation on a file, the user ID can be compared with the owner attribute to determine if the requesting user is the owner of the file.
- The result indicates which permissions are applicable. The system then applies those permissions to the requested operation and allows or denies it.

1. Remote File Systems

- Through the evolution of network and file technology, remote file-sharing methods have changed.
- The first implemented method involves manually transferring files between machines via programs like ftp. The second major method uses a distributed file system (DFS) in which remote directories are visible from a local machine. In some ways, the third method, the World Wide Web, is a revision to the first.

2. The Client-Server Model

- Remote file systems allow a computer to mount one or more file systems from one or more remote machines. In this case, the machine containing the files is server, and the machine seeking access to the files is the client.
- Generally the server declares that a resource is available to clients and specifies exactly which resource and exactly which clients.
- A server can serve multiple clients, and a client can use multiple servers, depending on the implementation details of a given client – server facility.

3. Distributed Information Systems

- To make client-server systems easier to manage, distributed information systems, also known as distributed naming services, provide unified access to the information needed for remote computing.
- The domain name system (DNS) provides host-name-to-network-address translations for the entire Internet. Other distributed information systems provide user name/password/user ID/group ID space for a distributed facility.

11. b) Domain of protection (5M)

Domain Structure

- A process operates within a **protection domain**, which specifies the resources that the process may access.
- Each domain defines a set of objects and the types of operations that may be invoked on each object. The ability to execute an operation on an object is an **access right**.
- A domain is a collection of access rights, each of which is an ordered pair
  <object-name, rights-set>
- For example, if domain D has the access right <file F, {read,write}>, then a process executing in domain D can both read and write file F.
- Domain may share access rights. For example, figure below, we have three domains: D_1,D_2 and D_3. The access right <O_4, {print}> is shared by D_2 and D_3, implying that a process executing in either of these two domains can print object O_4.

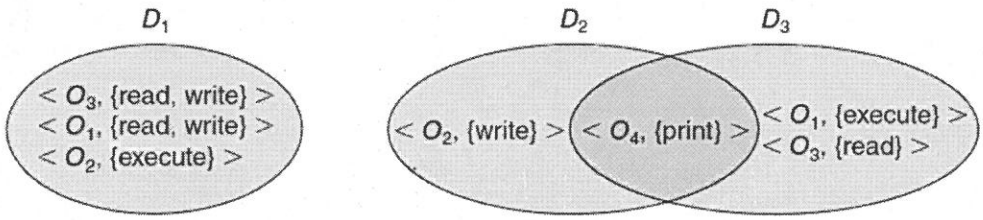


Figure 17.4 System with three protection domains.

- The association between a process and a domain may be either static, if the set of resources available to the process is fixed throughout the process's lifetime, or dynamic.
- If the association is dynamic, a mechanism is available to allow domain switching, enabling the process to switch from one domain to another.
- We may also want to allow the content of a domain to be changed. If we cannot change the content of a domain, we can provide the same effect by creating a new domain with the changed content and switching to that new domain when we want to change the domain content.
- A domain can be realized in a variety of ways:
  - Each ***user*** may be a domain. In this case, the set of objects that can be accessed depends on the identity of the user. Domain switching occurs when the user is changed—generally when one user logs out and another user logs in.
  - Each ***process*** may be a domain. In this case, the set of objects that can be accessed depends on the identity of the process. Domain switching occurs when one process sends a message to another process and then waits for a response.
  - Each ***procedure*** may be a domain. In this case, the set of objects that can be accessed corresponds to the local variables defined within the procedure. Domain switching occurs when a procedure call is made.