USN										

CMRIT
CMR INSTITUTE OF TECHNOLOGY, BENGALURU.
ACCREDITED WITH A++ GRADE BY NAAC

Internal Assessment Test 2 – December 2024

Sub:	**OPERATING SYSTEMS**	Sub Code:	**BCS303**	Branch:	**AIML/CSEAI ML**		
Date:	**17.12.24**	Duration:	**90 minutes**	Max Marks:	**50**	Sem/Sec:	**III -A, B, C**	**IPCC**

	Answer any FIVE FULL Questions	**MARKS**	**CO**	**RBT**
1.	Discuss in detail about contiguous memory allocation with illustration?	10M	CO4	L2
2	Consider the following page reference stream: 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1. How many page faults would occur for LRU and FIFO replacement algorithms assuming 3-page frames? Which one of the two is more efficient?	10M	CO4	L3
3	Explain briefly the various operations performed on files.	10M	CO5	L2
4	What is a deadlock? Illustrate the necessary conditions an operating system must satisfy if a deadlock has to occur?	10M	CO3	L2
5	Discuss briefly about demand paging in memory management schemes.	10M	CO4	L2
6	Given the memory partitions of 100k, 500k, 200k,300k and 600k apply first fit, best fit and worst fit algorithms to place 212k, 417k, 112k and 426k.	10M	CO4	L3

CI **CCI** **HOD**

——-----——-----------------------------All the Best---

Internal Assessment Test 2 – December 2024 Scheme & Solutions									
Sub:	**OPERATING SYSTEMS**					Sub Code:	**BCS303**	Branch:	**AIML/CSEAIML**
Date:	**17.12.24**	Duration:	**90 minutes**	Max Marks:	**50**	Sem/Sec:	**III -A, B, C**		**IPCC**

	Answer any FIVE FULL Questions	**MARKS**	**CO**	**RBT**
	Discuss in detail about contiguous memory allocation with illustration?	10M	CO 4	L2

Memory Management--------3M
Address Reallocation diagram & Explanation-----3M
Memory Allocation------2M
Fragmentation-----2M

The main memory must accommodate both the operating system and the various user processes. Therefore we need to allocate the parts of the main memory in the most efficient way possible.
• Memory is usually divided into 2 partitions: One for the resident OS. One for the user processes.
• Each process is contained in a single contiguous section of memory.
1. Memory Mapping and Protection
• Memory-protection means protecting OS from user-process and protecting user processes from one another.
• Memory-protection is done using
Relocation-register: contains the value of the smallest physical-address.
Limit-register: contains the range of logical-addresses.
• Each logical-address must be less than the limit-register.
• The MMU maps the logical-address dynamically by adding the value in the relocation register.
This mapped-address is sent to memory
• When the CPU scheduler selects a process for execution, the dispatcher loads the relocation and limit-registers with the correct values.
• Because every address generated by the CPU is checked against these registers, we can protect the OS from the running-process.
• The relocation-register scheme provides an effective way to allow the OS size to change dynamically.
• Transient OS code: Code that comes & goes as needed to save memory-space and overhead for unnecessary swapping.

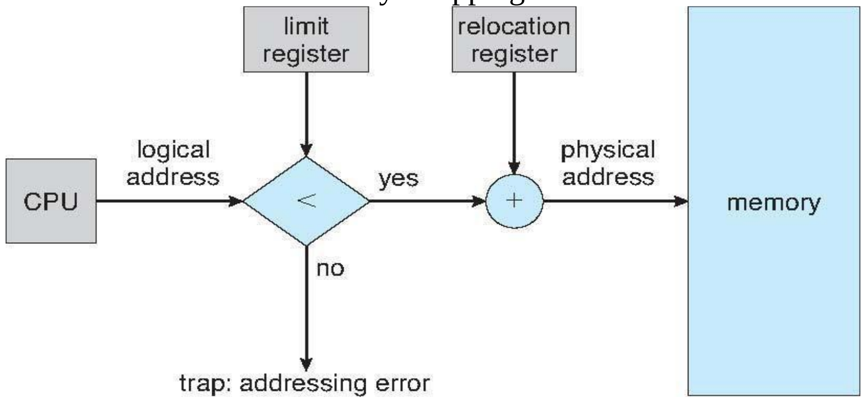
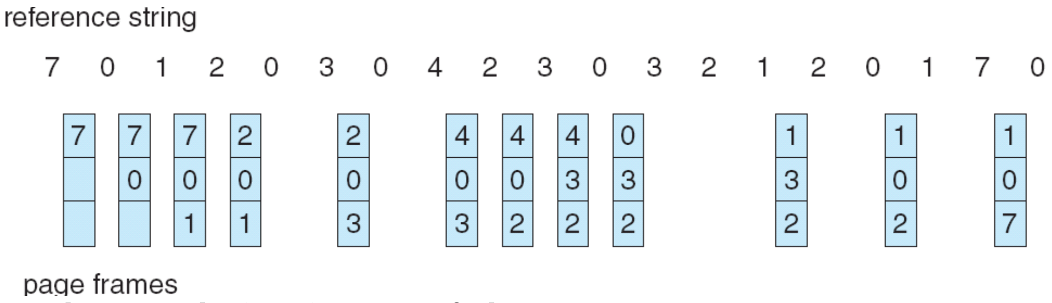
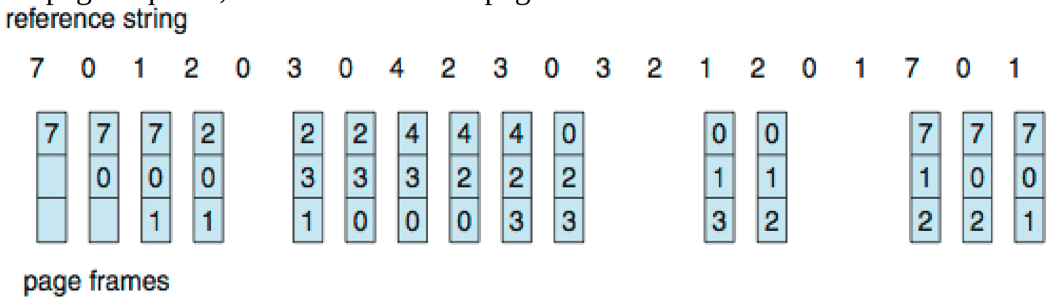


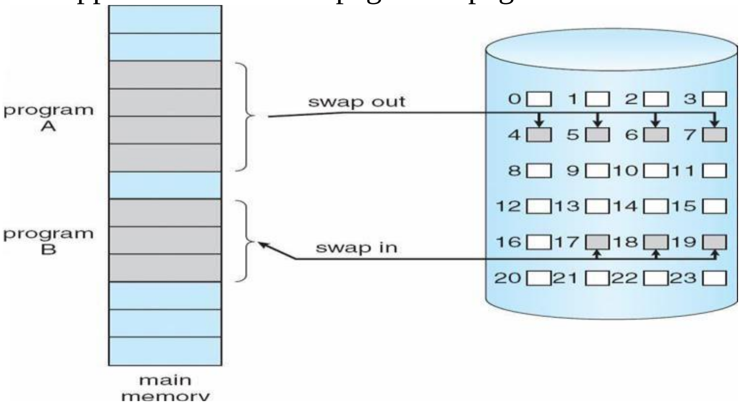
Figure: Hardware support for relocation and limit-registers
2. Memory Allocation
Two types of memory partitioning are:
1. Fixed-sized partitioning

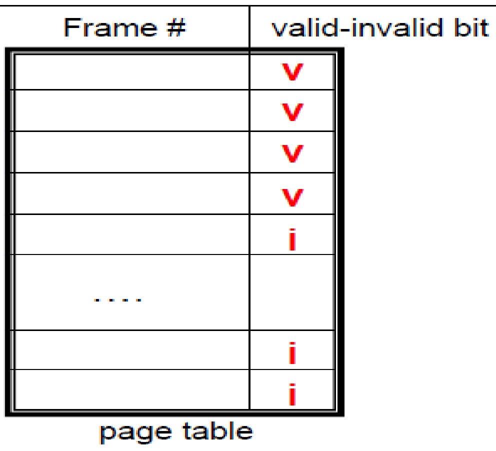
2. Variable-sized partitioning
1. Fixed-sized Partitioning
• The memory is divided into fixed-sized partitions.
• Each partition may contain exactly one process.
• The degree of multiprogramming is bound by the number of partitions.
• When a partition is free, a process is selected from the input queue and loaded into the free partition.
• When the process terminates, the partition becomes available for another process.
2. Variable-sized Partitioning
• The OS keeps a table indicating which parts of memory are available and which parts are occupied.
• A hole is a block of available memory. Normally, memory contains a set of holes of various sizes.
• Initially, all memory is available for user-processes and considered one large hole.
When a process arrives, the process is allocated memory from a large hole.
• If we find the hole, we allocate only as much memory as is needed and keep the
remaining memory available to satisfy future requests.
Three strategies used to select a free hole from the set of available holes:
1. First Fit: Allocate the first hole that is big enough. Searching can start either at the beginning of the set of holes or at the location where the previous first-fit search ended.
2. Best Fit: Allocate the smallest hole that is big enough. We must search the entire list, unless the list is ordered by size. This strategy produces the smallest leftover hole.
3. Fragmentation
Two types of memory fragmentation:
1. Internal fragmentation
2. External fragmentation
1. Internal Fragmentation
• The general approach is to break the physical-memory into fixed-sized blocks and allocate memory in units based on block size.
• The allocated-memory to a process may be slightly larger than the requested-memory.
• The difference between requested-memory and allocated-memory is called internal
fragmentation i.e. Unused memory that is internal to a partition.
2. External Fragmentation
• External fragmentation occurs when there is enough total memory-space to satisfy a request but the available-spaces are not contiguous. (i.e. storage is fragmented into a large number of small holes).
• Both the first-fit and best-fit strategies for memory-allocation suffer from external
fragmentation.
• Statistical analysis of first-fit reveals that given N allocated blocks, another 0.5 N blocks will be lost to fragmentation. This property is known as the 50-percent rule.
Two solutions to external fragmentation:
• Compaction: The goal is to shuffle the memory-contents to place all free memory
together in one large hole. Compaction is possible only if relocation is dynamic and done at execution-time

	• Permit the logical-address space of the processes to be non-contiguous. This allows a process to be allocated physical-memory wherever such memory is available. Two techniques achieve this solution: 1) Paging and 2) Segmentation. 3. Worst Fit: Allocate the largest hole. Again, we must search the entire list, unless it is sorted by size. This strategy produces the largest leftover hole. First-fit and best fit are better than worst fit in terms of decreasing time and storage utilization.			
2	**Consider the following page reference stream: 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1. How many page faults would occur for LRU and FIFO replacement algorithms assuming 3-page frames? Which one of the two is more efficient?** **LRU-------5M** **FIFO----4M** **Conclusion—1M** Least Recently Used (LRU) Algorithm • The *LRU (Least Recently Used)* algorithm, predicts 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. • Some view LRU as analogous to OPT, but here we look backwards in time instead of forwards. The main problem to how to implement LRU is the LRU requires additional h/w assistance.  reference string 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 page frames Total Page Faults (LRU): 12 page faults FIFO Algorithm: • This is the simplest page replacement algorithm. A FIFO replacement algorithm associates each page the time when that page was brought into memory. • When a Page is to be replaced the oldest one is selected. • We replace the queue at the head of the queue. When a page is brought into memory, we insert it at the tail of the queue. • In the following example, a reference string is given and there are 3 free frames. There are 20 page requests, which results in 15 page faults  reference string 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1 page frames Example: Consider the following references string with frames initially empty.	10M	CO 4	L3

	▪ The first three references (7,0,1) cases page faults and are brought into the empty frames. ▪ The next references 2 replaces page 7 because the page 7 was brought in first. x Since 0 is the next references and 0 is already in memory e has no page faults. ▪ The next references 3 results in page 0 being replaced so that the next references to 0 causer page fault. This will continue till the end of string. There are 15 faults all together. Thus, LRU is more efficient than FIFO in this case because it results in fewer page faults.			
3	**Explain briefly the various operations performed on files.** **5 file Operation- each 2 marks** File Operations A file is an abstract data type. To define a file properly, we need to consider the operations that can be performed on files. 1. Creating a file:Two steps are necessary to create a file, a) Space in the file system must be found for the file. b) An entry for the new file must be made in the directory. 2. Writing a file:To write a file, we make a system call specifying both the name of the file and the information to be written to the file. Given the name of the file, the system searches the directory to find the file's location. The system must keep a write pointer to the location in the file where the next write is to take place. The write pointer must be updated whenever a write occurs. 3. Reading a file:To read from a file, we use a system call that specifies the name of the file and where the next block of the file should be put. Again, the directory is searched for the associated entry, and 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. Because a process is usually either reading from or writing to a file, the current operation location can be kept as a per-process current file-position pointer. 4. Repositioning within a file:The directory is searched for the appropriate entry, and the current-file-position pointer is repositioned to a given value. Repositioning within a file need not involve any actual I/0. This file operation is also known as files seek. 5. Deleting a file:To delete a file, search the directory for the named file. Having found the associated directory entry, then release all file space, so that it can be reused by other files, and erase the directory entry. 6. Truncating a file:The user may want to erase the contents of a file but keep its attributes. Rather than forcing the user to delete the file and then recreate it, this	10M	CO 5	L2

	function allows all attributes to remain unchanged but lets the file be reset to length zero and its file space released.			
4	**What is a deadlock? Illustrate the necessary conditions an operating system must satisfy if a deadlock has to occur?** **Definition-2M** **Conditions and explanation—8M** A process requests resources, if the resources are not available at that time, the process enters a waiting state. Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes. This situation is called a Deadlock. SYSTEM MODEL • A system consists of a finite number of resources to be distributed among a number of competing processes. The resources are partitioned into several types, each consisting of some number of identical instances. Memory space, CPU cycles, files, and I/0 devices are examples of resource types. • A process must request a resource before using it and must release the resource after using it. A process may request as many resources as it requires carrying out its designated task. The number of resources requested may not exceed the total number of resources available in the system. Under the normal mode of operation, a process may utilize a resource in only the following sequence: 1. Request: The process requests the resource. If the request cannot be granted immediately, then the requesting process must wait until it can acquire the resource. 2. Use: The process can operate on the resource. 3. Release: The process releases the resource. A set of processes is in a deadlocked state when every process in the set is waiting for an event that can be caused only by another process in the set. The events with which we are mainly concerned here are resource acquisition and release. The resources may be either physical resources or logical resources To illustrate a deadlocked state, consider a system with three CD RW drives. Suppose each of three processes holds one of these CD RW drives. If each process now requests another drive, the three processes will be in a deadlocked state. Each is waiting for the event "CD RW is released," which can be caused only by one of the other waiting processes. This example illustrates a deadlock involving the same resource type. Deadlocks may also involve different resource types. For example, consider a system with one printer and one DVD drive. Suppose that process Pi is holding the DVD and process Pj is holding the printer. If Pi requests the printer and Pj requests the DVD drive, a deadlock occurs. DEADLOCK CHARACTERIZATION Necessary Conditions A deadlock situation can arise if the following four conditions hold simultaneously in a system: 1. Mutual exclusion: At least one resource must be held in a non-sharable mode, that is, only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released. 2. Hold and wait: A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.	10M	CO 3	L2

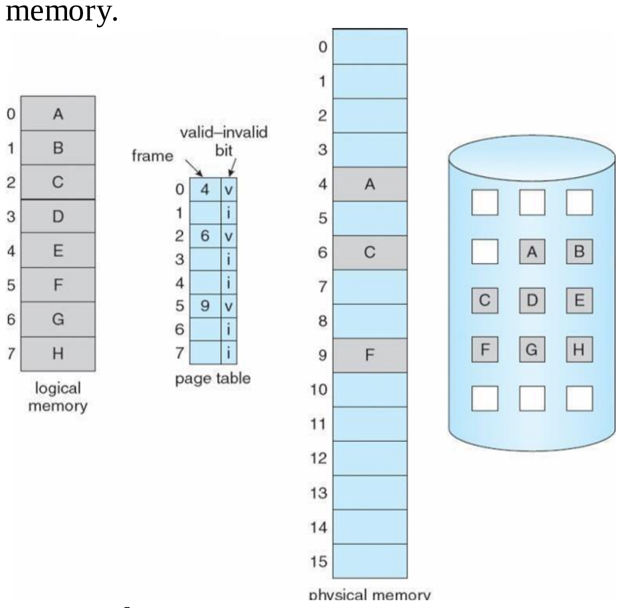
	3. No preemption: Resources cannot be preempted; that is, a resource can be released only voluntarily by the process holding it, after that process has completed its task. 4. Circular wait: A set {P0, Pl, ... , Pn} of waiting processes must exist such that Po is waiting for a resource held by P1, P1 is waiting for a resource held by P2, ... , Pn-1 is waiting for a resource held by Pn and Pn is waiting for a resource held by Po.			
5	**Discuss briefly about demand paging in memory management schemes.** **DEMAND PAGING ----------4M** **Diagram with valid and invalid bit ---3M** **Page Fault -------3M** • A demand paging is similar to paging system with swapping when we want to execute a process we swap the process the in to memory otherwise it will not be loaded in to memory. • A swapper manipulates the entire processes, where as a pager manipulates individual pages of the process. ▪ Bring a page into memory only when it is needed ▪ Less I/O needed ▪ Less memory needed ▪ Faster response More users ▪ Page is needed ⇒ reference to it ▪ invalid reference ⇒abort ▪ not-in-memory ⇒ bring to memory ▪ *Lazy swapper*– never swaps a page into memory unless page will be needed ▪ Swapper that deals with pages is a pager.  program A program B swap out swap in main memory 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 Fig: Transfer of a paged memory into continuous disk space • Basic concept: Instead of swapping the whole process the pager swaps only the necessary pages in to memory. Thus it avoids reading unused pages and decreases the swap time and amount of physical memory needed. • The valid-invalid bit scheme can be used to distinguish between the pages that are on the disk and that are in memory. ▪ With each page table entry a valid–invalid bit is associated ▪ (v ⇒ in-memory, i⇒not-in-memory) ▪ Initially valid–invalid bit is set to ion all entries ▪ Example of a page table snapshot:	10M	CO 4	L2



During address translation, if valid–invalid bit in page table entry is I ⇒ page fault.

• If the bit is valid then the page is both legal and is in memory.

• If the bit is invalid then either page is not valid or is valid but is currently on the disk. Marking a page as invalid will have no effect if the processes never access to that page. Suppose if it access the page which is marked invalid, causes a page fault trap. This may result in failure of OS to bring the desired page in to memory.

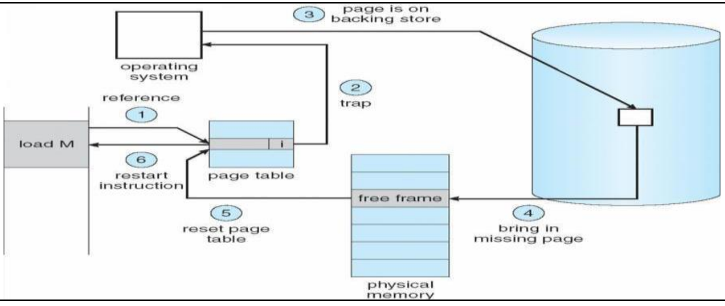


Page Fault

If a page is needed that was not originally loaded up, then a *page fault trap* is generated.

Steps in Handling a Page Fault

1. The memory address requested is first checked, to make sure it was a valid memory request.
2. If the reference is to an invalid page, the process is terminated. Otherwise, if the page is not present in memory, it must be paged in.
3. A free frame is located, possibly from a free-frame list.
4. A disk operation is scheduled to bring in the necessary page from disk.
5. After the page is loaded to memory, the process's page table is updated with the new frame number, and the invalid bit is changed to indicate that this is now a valid page reference.
6. The instruction that caused the page fault must now be restarted from the beginning.

	 3 page is on backing store operating system 2 trap reference 1 load M i 6 restart instruction page table free frame 5 reset page table 4 bring in missing page physical memory			
6	**Given the memory partitions of 100k, 500k, 200k,300k and 600k apply first fit, best fit and worst fit algorithms to place 212k, 417k, 112k and 426k.** **First Fit------3M** **Best Fit------3M** **Worst Fit---4M** 1. First Fit Algorithm The First Fit algorithm allocates the first partition that is large enough to hold the requested memory. Process: 1. 212k: The first partition large enough is 500k (since 100k and 200k are too small). Remaining partitions: 100k, 200k, 300k, 600k After allocation: 500k - 212k = 288k 2. 417k: The first partition large enough is 600k. Remaining partitions: 100k, 200k, 300k, 288k After allocation: 600k - 417k = 183k 3. 112k: The first partition large enough is 200k. Remaining partitions: 100k, 288k, 300k, 183k After allocation: 200k - 112k = 88k 4. 426k: The first partition large enough is 300k, but it's not big enough for 426k. The next available partition is 288k, which is too small. Thus, 426k cannot be allocated. 2. Best Fit Algorithm The Best Fit algorithm places the request in the smallest partition that can accommodate it, leaving the least leftover space. Process: 1. 212k: The best fit is 300k (since it's the smallest partition that can accommodate 212k). Remaining partitions: 100k, 500k, 200k, 600k After allocation: 300k - 212k = 88k 2. 417k: The best fit is 500k (the smallest partition that can accommodate 417k). Remaining partitions: 100k, 200k, 88k, 600k After allocation: 500k - 417k = 83k 3. 112k: The best fit is 200k (the smallest partition that can accommodate 112k). Remaining partitions: 100k, 88k, 83k, 600k After allocation: 200k - 112k = 88k 4. 426k: The best fit is 600k (the smallest partition that can accommodate 426k). Remaining partitions: 100k, 88k, 83k, 88k After allocation: 600k - 426k = 174k 3. Worst Fit Algorithm The Worst Fit algorithm places the request in the largest partition, leaving the largest possible leftover space.	10M	CO 4	L3

	Process: 1. 212k: The worst fit is 600k (the largest partition). Remaining partitions: 100k, 500k, 200k, 300k After allocation: 600k - 212k = 388k 2. 417k: The worst fit is 500k (the largest remaining partition). Remaining partitions: 100k, 200k, 300k, 388k After allocation: 500k - 417k = 83k 3. 112k: The worst fit is 388k (the largest remaining partition). Remaining partitions: 100k, 200k, 300k, 83k After allocation: 388k - 112k = 276k 4. 426k: The worst fit is 300k, but it's not large enough. The next available partition is 276k, which is also too small. Thus, 426k cannot be allocated.			