Chapter 5 Concurrency: Mutual Exclusion and Synchronization

Quizlet

Study online at quizlet.com/_2q596z

1.	**atomic operations**	group of operations that are uninterruptible
2.	**basic requirement for the execution of concurrent processes**	the ability to enforce mutual exclusion
3.	**binary semaphore**	Semaphore that has two values: 0 to 1 (busy or free) initial value always free (1)
4.	**blocking**	keeping a process from being run. i.e. if one of the process's critical resources is already in use by another thread
5.	**Buddy System**	a memory allocation algorithm that divides memory into partitions to try to satisfy a memory request as suitably as possible
6.	**busy waiting**	When the CPU is utilized by a process waiting for the critical section, thus there no computation progress.
7.	**Circular wait**	The processes in the system form a circular list or chain where each process in the list is waiting for a resource held by the next process in the list.
8.	**Compaction**	the process of moving allocated objects together, and leaving empty space together.
9.	**Concurrency**	Concurrency encompasses a host of design issues, including - communication among processes - sharing of and competing for resources (such as memory, files, and I/O access) - synchronization of the activities of multiple processes - allocation of processor time to processes.
10.	**Concurrent processes**	Processes that run almost simultaneously
11.	**Consumable Resource**	A resource that disappears after being used
12.	**coroutine**	A type of procedure in which the called procedure resumes from the point of its last RESUME command, rather than at the beginning of the method. It can pass control back to the calling procedure using the RESUME command. Traditionally, there is a master/slave relationship between the called and calling procedure. The calling procedure may execute a call from any point in the procedure; the called procedure is begun at its entry point and returns to the calling procedure at the point of call. The coroutine exhibits a more symmetric relationship. As each call is made, execution takes up from the last active point in the called procedure. p. 246
13.	**critical resource**	a nonsharable resource
14.	**critical section**	piece of code that only one thread can execute at a time This is due to the section of code requiring a critical resource(s) to run.
15.	**Deadlock**	occurs when two or more threads are waiting for an event that can only be generated by these same thread -not starvation, but imply starvation
16.	**Deadlock Avoidance**	-The system dynamically considers every request and decides whether it is safe to grant it at this point -The system requires additional information regarding the overall potential use of each resource for each process. -Allows more concurrency.

17.	**Deadlock Detection**	the system checks allocation against resource availability for all possible allocation sequences to determine if the system is in deadlocked state
18.	**Deadlock Prevention**	Elimination of "Mutual Exclusion" Condition The mutual exclusion condition must hold for non-sharable resources. That is, several processes cannot simultaneously share a single resource. This condition is difficult to eliminate because some resources, such as the tap drive and printer, are inherently non-shareable. Note that shareable resources like read-only-file do not require mutually exclusive access and thus cannot be involved in deadlock. Elimination of "Hold and Wait" Condition There are two possibilities for elimination of the second condition. The first alternative is that a process request be granted all of the resources it needs at once, prior to execution. The second alternative is to disallow a process from requesting resources whenever it has previously allocated resources. This strategy requires that all of the resources a process will need must be requested at once. The system must grant resources on "all or none" basis. If the complete set of resources needed by a process is not currently available, then the process must wait until the complete set is available. While the process waits, however, it may not hold any resources. Thus the "wait for" condition is denied and deadlocks simply cannot occur. This strategy can lead to serious waste of resources. For example, a program requiring ten tap drives must request and receive all ten derives before it begins executing. If the program needs only one tap drive to begin execution and then does not need the remaining tap drives for several hours. Then substantial computer resources (9 tape drives) will sit idle for several hours. This strategy can cause indefinite postponement (starvation). Since not all the required resources may become available at once. Elimination of "No-preemption" Condition The nonpreemption condition can be alleviated by forcing a process waiting for a resource that cannot immediately be allocated to relinquish all of its currently held resources, so that other processes may use them to finish. Suppose a system does allow processes to hold resources while requesting additional resources. Consider what happens when a request cannot be satisfied. A process holds resources a second process may need in order to proceed while second process may hold the resources needed by the first process. This is a deadlock. This strategy require that when a process that is holding some resources is denied a request for additional resources. The process must release its held resources and, if necessary, request them again together with additional resources. Implementation of this strategy denies the "no-preemptive" condition effectively. High Cost When a process release resources the process may lose all its work to that point. One serious consequence of this strategy is the possibility of indefinite postponement (starvation). A process might be held off indefinitely as it repeatedly requests and releases the same resources. Elimination of "Circular Wait" Condition The last condition, the circular wait, can be denied by imposing a total ordering on all of the resource types and than forcing, all processes to request the resources in order (increasing or decreasing). This strategy impose a total ordering of all resources types, and to require that each process requests resources in a numerical order (increasing or decreasing) of enumeration. With this rule, the resource allocation graph can never have a cycle. For example, provide a global numbering of all the resources, as shown
19.	**Dynamic Partitioning**	Allocating exactly the right amount of storage in memory that a process requires.
20.	**External Fragmentation**	When enough total memory space exists to satisfy a request, but it is not contiguous; storage is fragmented into a large number of small holes in order to be used for a process. Example: https://www.youtube.com/watch?v=BzV5purl6Ls
21.	**Fixed Partitioning**	Partition main memory into a set of non overlapping regions called partitions. These partitions are fixed and can have a process that requires less memory or the exact amount of memory that a partition has.
22.	**Frame**	available chunk of memory

23. **general semaphores (counting semaphores)**	These semaphores take more numbers than a binary semaphore: s > 0 = free s == 0 = used, no waiting s < 0 = used, process waiting Thus, general semaphores can utilize a FIFO queue, and binary semaphores cannot
24. **Give examples of reusable and consumable resources.**	Examples of reusable resources include processors, I/O channels, main and secondary memory, devices, and data structures such as files, databases, and semaphores. Examples of consumable resources are interrupts, signals, messages, and information in I/O buffers.
25. **Hold and Wait**	a process is currently holding at least one resource and requesting additional resources which are being held by other processes
26. **In a fixed-partitioning scheme, what are the advantages of using unequal-size partitions?**	• The number of partitions specified at system generation time limits the number of active (not suspended) processes in the system. • Because partition sizes are preset at system generation time, small jobs will not utilize partition space efficiently. In an environment where the main storage requirement of all jobs is known beforehand, this may be reasonable, but in most cases, it is an inefficient technique.
27. **Internal Fragmentation**	wasted space within a partition
28. **List the requirements for Mutual Exclusion**	1. Mutual exclusion: only one process is allowed into its critical section 2. A process that halts in its noncritical section must do so without interfering with other processes 3. It must not be possible for a process requiring access to a critical section to be delayed indefinitely: no deadlock or starvation. 4. When no process is in a critical section, any process that requests entry to its critical section must be permitted to enter without delay. 5. No assumptions are made about relative process speeds or number of processors. 6. A process remains inside its critical section for a finite time only.
29. **List the requirements for mutual exclusion.**	1. Mutual exclusion must be enforced: Only one process at a time is allowed into its critical section, among all processes that have critical sections for the same resource or shared object 2. A process that halts in its noncritical section must do so without interfering with other processes. 3. It must not be possible for a process requiring access to a critical section to be delayed indefinitely: no deadlock or starvation. 4. When no process is in a critical section, any process that requests entry to its critical section must be permitted to enter without delay. 5. No assumptions are made about relative process speeds or number of processors. 6. A process remains inside its critical section for a finite time only.
30. **Loading**	The compiling of processes by the OS
31. **locks**	one process hold the lock at a time, executes the critical section, releases the lock provide mutual exclusion to shared data using: acquire and release always acquire lock before accessing shared data always release the lock after finishing with shared data lock is initially free
32. **Logical Address**	The numerical value for an address (represented by page number and the offset)

33.	**Logical Organization**	-Programs are written in modules - Modules can be written and compiled independently - Different degrees of protection given to modules (read-only, execute-only) - Share modules among processes
34.	**Memory Management**	the functionality of an operating system which handles or manages primary memory and moves processes back and forth between main memory and disk during execution. Memory management keeps track of each and every memory location, regardless of either it is allocated to some process or it is free.
35.	**message passing**	when processes interact with one another, these requirements must be satisfied: - synchronization to enforce mutual exclusion - communication to exchange information
36.	**monitor**	-has lock and zero or more condition variables --encapsulate shared data --provide mutual exclusion --allow synchronization --allow operations on the shared data
37.	**monitors**	connects shared data to synchronization primitive
38.	**Mutex**	Similar to binary semphore, but the process that locks the mutex (sets the value to 0) must be the one to unlock it (sets the value to 1)
39.	**Mutual Exclusion**	a program object that prevents simultaneous access to a shared resource
40.	**mutual exclusion**	exactly one thread or process is doing a particular activity, or using shared resources, at a time
41.	**nonblocking**	Allowing a process to be run
42.	**Page**	a fixed-length contiguous block of virtual memory
43.	**Page Table**	the data structure used by a virtual memory system in a computer operating system to store the mapping between virtual addresses and physical addresses
44.	**Paging**	paging is one of the memory management schemes by which a computer stores and retrieves data from the secondary storage for use in main memory.
45.	**Partitioning**	Splitting up memory into chunks
46.	**Physical Address**	a binary number in the form of logical high and low states on an address bus that corresponds to a particular cell of primary storage(also called main memory), or to a particular register in a memory-mapped I/O(input/output) device
47.	**Pipe**	a technique for passing information from one program process to another
48.	**Preemption**	the act of temporarily interrupting a task being carried out by a computer system, without requiring its cooperation, and with the intention of resuming the task at a later time
49.	**producer/consumer (reader/writer) problem**	producer places data into buffer consumer takes from buffer only one may access at a time
50.	**Protection**	dictates that programs, users, and systems be given just enough privileges to perform their tasks
51.	**race condition**	Where multiple threads read and write a shared resource and the final result depends on the relative timing of their execution
52.	**Relative Address**	an address specified by indicating its distance from another address
53.	**Relocation**	the process of assigning load addresses to various parts of a program and adjusting the code and data in the program to reflect the assigned addresses
54.	**Resource Allocation Graph**	tracks which resource is held by which process and which process is waiting for a resource of a particular type

55. **resource variable**	each condition variable should have one of these must maintain it check resource before calling wait on the associated condition variable to ensure the resource really isn't available
56. **Reusable Resource**	A resource that can be used multiple times, and may lead to deadlock due to this fact.
57. **Segmentation**	an instruction operand that refers to a memory location includes a value that identifies a segment and an offset within that segment. A segment has a set of permissions, and a length, associated with it.
58. **semaphores**	an integer value used for signaling among processes - initialized to a nonnegative int value - semWait operation decrements value (sends processes to blocked queue) - semSignal operation increments value (unblocks processes)
59. **Sharing**	Sharing resources between processes dictated by the OS
60. **Spinlock**	a lock which causes a thread trying to acquire it to simply wait in a loop ("spin") while repeatedly checking if the lock is available.
61. **spin waiting (Spinlocks)**	Mutual exclusion mechanism in which a process executes in an infinite loop waiting for the value of a lock variable to indicate availability.
62. **Starvation**	the name given to the indefinite postponement of a process because it requires some resource before it can run, but the resource, though available for allocation, is never allocated to this process.
63. **starvation**	a situation in which a runnable process is overlooked indefinitely by the schedule
64. **strong semaphore**	includes the removal policy of FIFO, the process that has been blocked the longest is released first from the queue
65. **three contexts in which concurrency arises**	- multiple applications - structured applications - operating system structure p. 199 - 200
66. **weak semaphore**	a semaphore that does not specify the order in which processes are removed from the queue
67. **What are the distinctions among logical, relative, and physical addresses?**	A logical address is a reference to a memory location independent of the current assignment of data to memory; a translation must be made to a physical address before the memory access can be achieved
68. **What are the four conditions that create deadlock?**	1. Mutual exclusion. 2. Hold and wait. 3. No preemption 4. Circular wait
69. **What are the three conditions that must be present for deadlock to be possible?**	1. Mutual exclusion. Only one process may use a resource at a time. No process may access a resource unit that has been allocated to another process. 2. Hold and wait. A process may hold allocated resources while awaiting assignment of other resources. 3. No preemption. No resource can be forcibly removed from a process holding it.

70.	**What are three contexts in which concurrency arises?**	• Multiple applications: Multiprogramming was invented to allow processing time to be dynamically shared among a number of active applications. • Structured applications: As an extension of the principles of modular design and structured programming, some applications can be effectively programmed as a set of concurrent processes. • Operating system structure: The same structuring advantages apply to systems programs, and we have seen that operating systems are themselves often implemented as a set of processes or threads.
71.	**What conditions are generally associated with the readers/writers problem?**	1. Any number of readers may simultaneously read the file. 2. Only one writer at a time may write to the file. 3. If a writer is writing to the file, no reader may read it
72.	**What is a monitor?**	A programming language construct that encapsulates variables, access procedures, and initialization code within an abstract data type. The monitor's variable may only be accessed via its access procedures and only one process may be actively accessing the monitor at any one time. The access procedures are critical sections . A monitor may have a queue of processes that are waiting to access it
73.	**What is a monitor?**	The monitor is a programming-language construct that provides equivalent functionality to that of semaphores and that is easier to control.
74.	**What is the basic requirement for the execution of concurrent processes?**	The basic requirement for support of concurrent processes is the ability to enforce mutual exclusion; that is, the ability to exclude all other processes from a course of action while one process is granted that ability.
75.	**What is the difference between a page and a frame?**	Page: Chunk of a process Frame: Chunk of memory
76.	**What is the difference between a page and a segment?**	The difference, compared to dynamic partitioning, is that with segmentation a program may occupy more than one partition, and these partitions need not be contiguous. Whereas paging is invisible to the programmer, segmentation is usually visible and is provided as a convenience for organizing programs and data.
77.	**What is the difference between binary and general semaphores?**	A concept related to the binary semaphore is the mutex. A key difference between the two is that the process that locks the mutex (sets the value to zero) must be the one to unlock it (sets the value to 1). In contrast, it is possible for one process to lock a binary semaphore and for another to unlock it
78.	**What is the difference between binary and general semaphores?**	Binary semphores take only the values 0 and 1, blocked and unblocked General semaphores take more numbers: s > 0 = free s == 0 = used, no waiting s < 0 = used, process waiting Thus, general semaphores can utilize a FIFO queue, and binary semaphores cannot
79.	**What is the difference between internal and external fragmentation?**	Phenomenon, in which there is wasted space internal to a partition due to the fact that the block of data loaded is smaller than the partition, is referred to as internal fragmentation.
80.	**What is the difference between strong and weak semaphores?**	Strong semaphores implement a FIFO queue. The process that has been waiting the longest is the one to access the program next Weak semaphores do not specify the order in which processes are removed from the queue.
81.	**What is the difference between strong and weak semaphores?**	Strong semaphore specifies in which order processes are removed from the waiting queue (such as FIFO) Weak semaphore does not specify this.

82.	**What is the distinction between blocking and nonblocking with respect to messages?**	• Blocking send, blocking receive: Both the sender and receiver are blocked until the message is delivered; this is sometimes referred to as a rendezvous. This combination allows for tight synchronization between processes. • Nonblocking send, blocking receive: Although the sender may continue on, the receiver is blocked until the requested message arrives.This is probably the most useful combination. It allows a process to send one or more messages to a variety of destinations as quickly as possible. A process that must receive a message before it can do useful work needs to be blocked until such a message arrives. An example is a server process that exists to provide a service or resource to other processes. • Nonblocking send, nonblocking receive: Neither party is required to wait.
83.	**What is the distinction between competing processes and cooperating processes?**	Competing processes - compete for resources. For example, two independent applications may both want to access the same disk or file or printer. The OS must regulate these accesses. Cooperating Processes - Share resources. May or may not be aware of each other. Some processes are designed to cooperate together (jointly) on the same activity and share resources. They may also be aware of each other by process id.
84.	**What operations can be performed on a semaphore?**	1. A semaphore may be initialized to a nonnegative integer value. 2. The semWait operation decrements the semaphore value. If the value becomes negative, then the process executing the semWait is blocked. Otherwise, the process continues execution. 3. The semSignal operation increments the semaphore value. If the resulting value is less than or equal to zero, then a process blocked by a semWait operation, if any, is unblocked.
85.	**What operations can be performed on a semaphore?**	Initialize Increment = semSignal = unblocking process Decrement = semWait = blocking process
86.	**Why is the capability to relocate processes desirable?**	In a multiprogramming system, the available main memory is generally shared among a number of processes. Typically, it is not possible for the programmer to know in advance which other programs will be resident in main memory at the time of execution of his or her program. In addition, we would like to be able to swap active processes in and out of main memory to maximize processor utilization by providing a large pool of ready processes to execute. Once a program has been swapped out to disk, it would be quite limiting to declare that when it is next swapped back in, it must be placed in the same main memory region as before. Instead, we may need to relocate the process to a different area of memory.