Chapter 6 - Concurrency: Deadlock and Starvation

Study online at quizlet.com/_2mdmj5

1. **atomic operations (Linux)**	- execute without interruption or interference two types - integer operations - bitmap operations
2. **conditions for POSSIBLE deadlock**	- mutual exclusion (only one process may use a resources at a time) - hold-and-wait (a process may hold allocated resources while awaiting assignment of other resources) - no preemption (no resources can be forcibly removed from a process holding it)
3. **conditions that CREATE deadlock**	- all conditions for possible deadlock - circular wait (a potential consequence of the first three - a closed chain of processes exists, such that each process hold at least one resource needed by the next process in the chain)
4. **consumable resources**	one that can be created (produced) and destroyed (consumed) - interrupts, signals, messages, information
5. **consumable resources deadlock**	deadlock occurs if the receive is blocking (i.e. the receiving process is blocked until the message is received)
6. **deadlock**	the permanent blocking of a set of processes that either compete for system resources or communicate with each other
7. **deadlock avoidance**	a decision is made dynamically whether the current resource allocation request will, if granted, potentially lead to a deadlock - requires knowledge of future process requests
8. **deadlock avoidance advantages**	- it is not necessary to preempt and rollback processes, as in deadlock detection - it is less restrictive than deadlock prevention
9. **(deadlock avoidance) process initiation denial**	do not start a process if its demands might lead to deadlock
10. **(deadlock avoidance) resource allocation denial**	do not grant an incremental resource request to a process if this allocation might lead to deadlock (state of system reflects the current allocation of resources to processes)
11. **deadlock avoidance restrictions**	- maximum resource requirement for each processes must be stated in advance - processes under consideration must be independent and with no synchronization requirements - there must be a fixed number of resources to allocate - no process may exist while holding resources
12. **deadlock condition prevention**	only practical one is no preemption, but only practical when applied to resources whose state can be saved and restored later - all others are either not practical or inefficient
13. **deadlock detection algorithms - advantages**	- it leads to early detection - the algorithm is relatively simple
14. **deadlock detection algorithms - disadvantages**	- frequent checks consume considerable processor time
15. **deadlock detection strategies**	resource requests are granted whenever possible, as opposed to deadlock prevention which limits access to resources by imposing restriction on processes
16. **deadlock prevention strategy**	design a system in such a way that the possibility of deadlock is excluded - indirect (prevent occurence of one of the three necessary conditions) - direct (prevent the occurrence of a **circular wait**)
17. **dealing with deadlock (reference, pg. 266)**	- prevention (adopt a policy that eliminates one of the conditions) - avoidance (make the appropriate dynamic choices based on the current state of resource allocation) - detection (attempt to detect the presence of deadlock at take action to recover)
18. **dining philosophers problem**	- no two can use the same fork, but two forks must be used at the same time (mutual exclusion) - no one must starve to death (deadlock and starvation) - solution: allocate enough resources at the start
19. **Linux kernel concurrency mechanism**	- barriers - spinlocks - semaphores - atomic operations
20. **memory request**	deadlock occurs if separate processes progress to requests requiring more memory than is available
21. **messages (UNIX)**	a block of bytes with an accompanying type - associated with each process is a message queue (mailbox)

22. **pipes (UNIX)** circular buffers allow two processes to communicate on the producer-consumer model (FIFO)
two types
- named
- unnamed

23. **recovery strategies**
- abort all deadlocked processes
- back up each deadlocked process to some previously defined checkpoint and restart all processes
- successively abort deadlocked processes until deadlock no longer exists
- successively preempt resources until deadlock no longer exists

24. **reusable resources** can be safely used by only one process at a time and is not depleted by that use
- processors, I/O channels, main and secondary memory, devices, data structures (files, databases, semaphores)

25. **safe state** a state in which there is at least one sequence of resource allocations to process that does not result in a deadlock

26. **semaphores (Linux)** three types
- binary
- counting
- reader-writer

27. **semaphores (UNIX)** generalization of the semWait and semSignal primitives

28. **shared memory (UNIX)** fastest form of interprocess communication
- common block of virtual memory shared by multiple processes

29. **signals (UNIX)** a software mechanism that informs a process of the occurrence of asynchronous events
- a signal is delivered by updating a field in the process table for the process to which the signal is being sent

30. **spinlocks (Linux)** most common technique for protecting a critical section in Linux
- can only be acquired by on thread at a time
- built on an integer location in memory that is checked by each thread before it enters its critical section

31. **UNIX concurrency mechanisms**
- pipes
- messages
- shared memory
- semaphores
- signals

32. **unsafe state** a state that is not safe

33. **wait functions**
- allow a thread to block its own execution
- do not return until the specified criteria have been met
- the type of wait function determines the set of criteria used

34. **when does deadlock happpen** a set of processes is deadlocked when each process in the set is blocked awaiting an event that can only be triggered by another blocked process in the set
- permanent
- no efficient solution

35. **windows 7 concurrency mechanisms** windows provides synchronization among threads as part of the object architecture important methods:
- executive dispatcher objects
- user mode critical sections
- slim reader-writer locks
- condition vars
- lock-free operations