Quizlet

Operating Systems -Questions

Study online at quizlet.com/_14qgkv

1. **All modern operating systems worth talking about* are?**	Interrupt drive, multitasking operating systems
2. **Almost all data on disks are... ?**	files
3. **Almost all other devices in a modern operating system are viewed in one of two ways... ?**	A memory space or a file
4. **Almost all other devices in a modern operating system are viewed in one of two ways. 1. A memory space, 2. As a file. What is the biggest exception to this?**	Network Interfaces are the biggest exception Because everything needs either somewhere its coming from or somewhere is going to and a file does not suffice.

5. **constant int SIZE = 8**
shared data buffer[SIZE]
shared int in = 0
shared int out = 0
-->semaphore s = true<--
producer_thread {
data d
while true {
while ((in+1)%SIZE) == out {
do nothing
}
-->wait(s)<--
d = produce_item()
buffer[in] = d
in = in + 1
in = in % SIZE
-->signal(s)<--
}
}
	consumer_thread {
	data d
	while true {
	while in == out {
	do nothing
	}
	wait(s)
	data d = buffer[out]
	consume_item(d)
	out = out + 1
	out = out % SIZE
	signal(s)
	}
	}

6. **Each device has a ....?**	Device Driver
7. **Each device has a device driver that does 1 of 4 things ... it either ...**	• Maps it to a file • Maps it to memory • Maps it to another core OS service (like networking) • Gives it a unique interface (very rare in a modern OS)
8. **Each process has a process control block with at least the following ...**	• State - generally new, ready, waiting, running or terminated • Program counter - we talked about this before • Register set - what the process thinks the state of all the other registers currently is • Scheduling information - the process's priority, and anything else the OS uses to decide how to schedule it • Memory-management information - we'll get to this later • Accounting information - who's responsible for this process, how much time it's taken, etc. • I/O status information - what files and devices this process has open, what it's waiting on, etc.
9. **Each thread needs to have it's own ...**	• Program counter • Register set • Call stack (whether in kernel space or user space)

10. **For our purposes what is accounting information?**	• Accounting information - who's responsible for this process, how much time it's taken, etc.
11. **How does a computer context switch?**	• Use an interrupt to return control to the OS • Copy the frozen process's current state - particularly the registers, including the program counter - into its PCB • Change the frozen process's state from running to ready • Select the new process • Copy the new process's state from its PCB into the registers • Change the new process's state from ready to running • Return control to the new process at its current program counter
12. **How does Java implement threads ?**	Java does it's own thing entirely • Any class can be made a thread by implementing the Runnable interface
13. **How does Windows functions deal with parents and children processes ?**	The same as Unix like systems, but accepts ten different parameters to account for every possible permutation in one function call
14. **How do you switch a process ?**	Return control of cpu to scheduler with a timer interrupt Change the state save the registers Select the next process load the new registers change the state pc
15. **In order for programs to run what part of memory must they be located in?**	Main memory
16. **In terms of message passing involving sending and receiving what are 3 types of addressing ?.**	• Symmetric send(P, message) message = receive(P) • Asymmetric send(P, message) receive(var message, var sender) • Mailbox (but how do the rules work?) send(M, message) message = receive(M)
17. **In terms of multitasking, how do we deal with scheduling and what is the technical term for this ?**	we need to freeze processes in the middle and give other processes their turn. The technical term for this is context switching.
18. **In the threading model, processes are allowed to have any number of ....**	Program Counters, up until now Each process has had a single program counter that represents the next instruction it's going to execute
19. **Linux uses threads true or false ?**	True, but fundamentally false • Linux doesn't actually have threads • It just has processes with different levels of sharing
20. **Modern OS systems need what 4 main things ?**	• Process Management • Memory Management • I/O Management • Security
21. **Operating system on behalf of user programs must do the kernel mode stuff. How does this happen?**	Interrupts! (Properly, these are really traps), OS receives the interrupt, services it, and returns control to the process.
22. **Originally Large Systems were used for Centralizing very scarce computing they are not quite dead yet, who uses them?**	SS Administration, dude cracking you pass-code, NASA,

23.	**Originally Large Systems were used for Centralizing very scarce computing where they job-oriented or built for interactivity?**	interactivity-oriented
24.	**Practical cryptography is intended to do two what two things?**	• Allow you to send messages that are very hard for anybody other than your recipient to read • Allow you to sign messages in a way that is very hard for anybody else to impersonate
25.	**producer creates chunks of data for the consumer to operate on → place those chunks of data in a shared memory location. → space that producer has to store that data is limited. → needs to be delayed as appropriate to not over run that buffer. → consumer grabs what the producer produces and uses it → Consumer has to be delayed so that it waits for the producer to generate data to operate on → this is the classic synchronization problem.** **another deffinition** **• Classic example of inter-process sharing** **• A producer in one process creates chunks of data for a consumer to further operate on** **• The producer needs to be able to place those chunks of data in a shared memory location that the consumer can read them from** **• The producer needs to be delayed as appropriate to not overrun this buffer** **• The consumer needs to be delayed as appropriate to wait for the producer to generate data for it to operate on** **• So how do we make this happen? --> next Power Point we explain this ! Threads**	What is the classic producer-consumer problem ?
26.	**Sending and receiving can be either ...**	blocking or nonblocking - including several queued variations
27.	**Should you allow programs to control when to use kernel mode or only allow the OS to do it?**	Only allow the OS to do it!
28.	**signal (semaphore s) {** **s = true** **}** **wait (semaphore s) {** **while s == false {** **} do nothing** **}** **s = false** **}**	semaphore: A counter of some sort that can be tested and incremented without being interrupted

29. **What are 3 example of Virtual computers ?**	•Emulation is running code for another kind of computer • Interpretation is running code for a theoretical computer To note, calling Java "interpreted" is a stretch • Virtualization is running a computer on a computer
30. **What are 3 examples of Networked computers?**	• Client-Server Systems: Networked systems that allow a client computer to ask a server to do something on its behalf • Peer-to-Peer Systems: Actually a lot like client-server computing, except at any time, any computer can be a client, a server or both • Cloud Computing: Broadly constructed client-server systems running over the commodity Internet or private networks
31. **What are 3 fundamental benefits of threads ?**	• Responsiveness - Threads can allow for an application to be more responsive without having to logically interrupt its own work • Resource sharing - Threads can implement parallelism without the need for message passing or shared memory • Scalability - Exploitation of multiple processors • ...and with almost all computers now having multi-core processors, this is important
32. **What are 4 examples of physical computers ?**	• Large Systems: Initially required to centralize very scarce computing resources • Desktop/Laptop Computing: What most people think of as "operating systems" at this point •Mobile Computing: Laptops and tablets • Real-Time Operating Systems: When things have to be done now. Popular for control systems. You really don't want your X-Ray machine to decide to garbage collect at random times
33. **What are 4 methods of Authentication?**	Passwords, Pass phrases, Tokens (Two-Factor), Biometrics
34. **What are 5 fundamental issues with threads ?**	• Identifying tasks - dividing up the work in terms of operations • Data splitting - dividing up the work in terms of data • Balancing - making sure every thread gets enough work • Dependency - managing dependent work (remember Producer/Consumer?) • Debugging - debugging multithreaded processes is much harder
35. **What are Client-Server Systems ?**	Neworked systems that allow a client computer to ask a server to do something on its behalf
36. **What are Peer-to-Peer Systems?**	a lot like client-server computing, except at any time, any computer can be a client, a server or both Computers have to be able to discover each other, either via a central directory or exchanging information over the network
37. **What are some example of Client-Server Systems?**	Web servers, Database Servers, Game Servers
38. **What are some examples of Cloud computing ?**	file servers, computation servers, virtual machines, etc.
39. **What are some examples of Emulation?**	PlayStation 2 exclusive video game could be played on a PC using an emulator. Many printers, for example, are designed to emulate Hewlett-Packard LaserJet printers because so much software is written for HP printers.

40. **What are some examples of interpreter ?**	Early versions of the Lisp programming language and Dartmouth BASIC would be examples of the first type. Perl, Python, MATLAB, and Ruby are examples of the second, while UCSD Pascal is an example of the third type. first type: parse the source code and perform its behavior directly second type: translate source code into some efficient intermediate representation and immediately execute this third type: explicitly execute stored precompiled code[1] made by a compiler which is part of the interpreter system
41. **What are some examples of message passing ?**	sql (complicated example), sockets --especially local domain sockets
42. **What are some examples of Peer-to-Peer Systems ?**	Pirate Bay, Napster
43. **What are some examples of real time computing?**	When things have to be done now: cell phone towers, X-ray machine,control systems
44. **What are some examples of Virtualization?**	For example, a computer that is running Microsoft Windows may host a virtual machine that looks like a computer with the Ubuntu Linux operating system; Ubuntu-based software can be run on the virtual machine.
45. **What are some practical uses of threads ?**	Threads are used as a simple form of parallelism (doing different parts of the same job on different processors). Image processing, Sockets, Tabs in a browser
46. **What are the 4 main functionalities of an OS with respect to disks?**	What are the 4 main functionalities of an OS with respect to disks?
47. **What are the 4 main functionalities of an OS with respect to disks?**	Input, Output and Everything Else: Almost all other devices (everything else) in a modern operating system are viewed in one of two ways. 1. A memory space, 2. As a file
48. **What are the 4 main functionality an OS must perform with respect to memory?**	• Keep track of what memory is being used, and by which processes • Decide what to move into and out of memory • Allocate and de-allocate memory as necessary • Make sure processes don't "color outside the lines" of their memory (more on this later)
49. **What are the 4 main functionality of an OS with respect to files?**	• Creating, writing, reading and deleting files • Creating, writing, reading and deleting directories to organize files • Providing functions for processes to work with files • Mapping files onto disks (really, partitions, but we'll call them disks for now)
50. **What are the 6 main functionalists of the Operating System?**	• Load programs into memory to turn them into processes • Create, delete, suspend and resume processes • Schedule processes and threads to CPUs • Assign resources to processes • Provide synchronization between processes • Provide communication between processes
51. **What are the three main functions in Unix like systems ?**	fork(), exec() and wait().
52. **What are the three main queues ?**	• job queue - contains all the processes in the system • ready queue - contains the processes that are in main memory and aren't waiting • I/O device queue - It contains the processes that are waiting on that device for each I/O device

53. **What are the two fundamental types of parallelism ?**	• Data parallelism divides up data, using multiple processors to run the same code on different data slices • Simple example: Image processing • Task parallelism divides up code, using multiple processors to run different parts of the program • Simple example: Sockets
54. **What are two great example of what Operating systems use cryptography for ?**	• Secure storage in the face of potential theft • Secure communication over potentially hostile networks
55. **What are two main modes of operation ?**	Kernel and User Mode
56. **What are two useful/fundamental mechanisms ?**	• Message-passing - avoids conflicts and works better across networks • Shared memory - is more convenient for large amounts of data
57. **What are you allowed to do in kernel mode?**	mess with hardware or other memory. It can tell you what memory does and doesn't belong to you. Only the OS can use this
58. **What are you allowed to do in user mode? (basic answer)**	Add subtract, compare, go to ...Basically assembly stuff
59. **What controls the input and output ?**	Device controllers connected to a buss
60. **What does the CPU scheduler do ?**	selects from ready processes and decides which one to run. Out of all the schedulers, this is the one we care about
61. **What does the medium term scheduler do ?**	kicks processes back out to disk. We don't really use this anymore either.
62. **What does the program counter do?**	It points at where the next instruction that's going to be executed is.
63. **What error happens when one program can write over space that another is using while the other is using it.**	Undefined behavior
64. **What interesting issue does shared memory cause ?**	We do not control who is writing an reading ... we need to sync if not one program can write over space that another is using while the other is using it → undefined behavior
65. **What is a file?**	It is a name, sequential array of bytes
66. **What is Asymmetric messaging?**	send(P, message) receive(var message, var sender) Here the receiver will accept a message from any sender. The sender can pass its own id inside the message if it wants.
67. **What is authentication?**	The process of determining that you are who you say you are, and given who you are, what you should be able to do.
68. **What is Could Computing ?**	based on utility and consumption of computing resources. Cloud computing involves deploying groups of remote servers and software networks that allow centralized data storage and online access to computer services or resources. constructed client-server systems running over the commodity Internet or private networks.
69. **What is Data parallelism ?**	• Data parallelism divides up data, using multiple processors to run the same code on different data slices • Simple example: Image processing

70.	**What is Emulation ?**	running code for another kind of computer. In computing, an emulator is hardware or software or both that duplicates (or emulates) the functions of one computer system (the guest) in another computer system (the host), different from the first one, so that the emulated behavior closely resembles the behavior of the real system (the guest).
71.	**What is exec() ?**	• exec() can then be used, after running fork(), to run a different process • As the parent, you can either wait() for the child, or keep right on going
72.	**What is fork() ?**	fork() creates a new copy of the current process as a child • If you're the child, you get back 0 • Otherwise, you get back the child's process ID This is for Unix-Like systems !
73.	**What is interpretation ?**	running code for a theoretical computer In computer science, an interpreter is a computer program that directly executes, i.e. performs, instructions written in a programming or scripting language, without previously compiling them into a machine language program. An interpreter generally uses one of the following strategies for program execution:
74.	**What is Mailbox messaging ?**	• Mailbox send(M, message) message = receive(M)
75.	**What is pipe message passing ?**	Ordinary pipes can only communicate between parent and child processes Named pipes, once created, can communicate between any number of processes Use OS dependent naming conventions
76.	**What is scheduling information?**	• Scheduling information - the process's priority, and anything else the OS uses to decide how to schedule it
77.	**What is sending and receiving using blocking ?**	it sends force a process to wait until there is space in the send queue receives - force a process to wait until data are available
78.	**What is sending and receiving using non-blocking ?**	sends - fail (not the same thing as producing an error) if there isn't space in the send queue receives- fail if no data are available
79.	**What is Socket message passing ?**	Sockets are the fundamental method of communication used by TCP/IP networking. Sockets are similar to pipes, but are implemented using the IP address-and-port scheme rather than OSdependent naming conventions
80.	**What is state?**	• State - generally: new - being created, ready - can run right now, waiting - waiting on any input or output, running - if it is actually executing on the cpu right now terminated - process control block hasn't been reaped by the ...
81.	**What is Symmetric addressing?**	send(P, message) message = receive(P) One process/thread is the sender and another is the receive
82.	**What is Task parallelism ?**	• Task parallelism divides up code, using multiple processors to run different parts of the program • Simple example: Sockets
83.	**What is the Difference between root in unix and administrator in windows?**	One is named root and the other is named administrator, that's basically it
84.	**What is the largest storage device the CPU can directly address?**	Main memory

85.	**What is the Long Term Scheduler ?**	Loads entire batch-scheduled processes from disk • If practical can get a good mix of CPU-bound and I/O-bound processes going • Sadly, however, they're useless for fully-interactive operating systems
86.	**What is the main purpose of dual mode operation ?**	We don't want processes to be able to access hardware or mess with other processes.
87.	**What is the parent process of all other processes ?**	By definition all processes are children of the Boot Strap program.
88.	**What is the register set?**	• Register set - what the process thinks the state of all the other registers currently is. Entire register set to be stored in the process control block.... whats in the process control block is not necessarily whts in the registers right now.
89.	**What is Virtualization ?**	running a computer on a computer refers to the act of creating a virtual (rather than actual) version of something, including but not limited to a virtual computer hardware platform, operating system (OS), storage device, or computer network resources. Ubuntu on Windows for example.
90.	**What loads the operating system ?**	A boot program
91.	**What's a bad example of privilege escalation?**	setuid. It's a security hole anytime you use it. Book says it's a good example.
92.	**What's an example of system-level attacks take place against badly designed authorization systems?**	privilege escalation attack --> network intrusion that takes advantage of design flaws to grant the attacker elevated access to the network and its associated data and applications.
93.	**Whats the difference between authorization and authentication?**	Authentication verifies who you are but authorization verifies what you can do
94.	**What three components is a computer made of ?**	I/O, CPU, Memory
95.	**• What were Large Systems originally used for?** **• Built around time-sharing; job-oriented more than interactivity-oriented** **• Decreasingly common, but not dead yet**	Initially required to centralize very scarce computing resources. Decreasingly common, but not dead yet
96.	**When does a program become a process?**	When it is loaded into memory and started!
97.	**When is a process independent ?**	if, apart from being managed by the OS, it doesn't affect and isn't affected by any other processes including sharing data or requiring an IPC mechanism. Otherwise it's cooperating.
98.	**Where do programs become processes ?**	Main Memory
99.	**Which mechanism is better across networks and avoids conflicts ?**	Message passing
100.	**Which mechanism is more convenient for large amounts of data ?**	Shared memory
101.	**Who decides when it's a good time for a process to start ?**	Scheduler
102.	**Who has current control of the program counter and registers ?**	Which ever process is running, but all the others have their own copies ready to be loaded back in.
103.	**Who is allowed to use kernel mode ?**	the OS

104. **Who loads Programs into memory in order to turn them into processes?**	The OS
105. **Why is Producer-consumer with blocking message passing trivial ?**	Basic consumer receiver problem we don't talk producer consumer in message passing because message passing does all the work for you.
106. **Windows uses threads true or false ?**	True and True! • Windows, by contrast of Linux, implements threads directly • CreateThread() function analogous to CreateProcess()