Locking and synchronization

1.On a uniprocessor machine with preemption disabled, what will happen internally when we say spin_lock()?

Ans:on a uniprocessor machine with preemption disabled.then when you apply a spin_lock then a uniprocessor will complete the critical section if the common resource was free otherwise it will keep on polling if the common resource was not free.it leads to deadlock.

2. Can I lock a spinlock in one CPU and unlock it another CPU?

Ans:No

Basically, a per-CPU variable is an array of data structures, one element per each CPU in the system.

A CPU should not access the elements of the array corresponding to the other CPUs.

3.What are the pros and cons of using per CPU variable as synchronization method?

Pros:

Freely read and modify its element without fear of race conditions.

It avoids cache line snooping and invalidations, which are costly operations.

Cons:

It can only be used when it makes sense to split the data across the CPUs of the system logically.

Do not protect against access from asynchronous functions such as interrupt handlers and deferrable functions.

Per-CPU variables are prone to race conditions caused by kernel preemption, both in uniprocessor and multiprocessor systems.

4.What is the maximum amount of time CPU can be in critical section after acquiring spinlock?

Ans:RLIMIT_CPU is the maximum amount of time CPU can be in critical section after acquiring spinlock.

5.What is the difference between binary semaphore and mutex in Linux?

6.What is the difference between down_interruptible vs down_killable in semaphore?

down_interruptible	down_killable
puts process in interruptible wait queue.	puts process in interruptible wait queue
Process can be interrupted by any signal.	Process can be interrupted by only signal number 9.

7.Can i call kmalloc(GFP_KERNEL) while holding a spinlock?

Ans:YES but we can't do because it is possible blocking call.

8.What are the lock free algorithms present in Linux kernel?

Ans:

Thread-safe access to shared data without the use of synchronization primitives such as mutexes
Possible but not practical in the absence of hardware support
Lock-free programming can produce good performance
Difficult to get right – Performance and correctness (ABA problem)
Well-established, tested, tuned implementations of common data structures are available
lock free algorithms:read lock versions in linux are lock free algorthms.

9.By disabling interrupts, will we get protection from concurrent access?
Ans:yes

Memory Management

10. What are the use of L1, L2, and L3 Caches?

Level 3 (**L3**) **cache** is specialized memory developed to improve the performance of **L1** and **L2** **L1** or **L2** can be significantly faster than **L3**, though **L3** is usually double the speed of DRAM. With multicore processors, each core can have dedicated **L1** and **L2 cache**, but they can share an **L3 cache**.

11. Difference between GFP_KERNEL and GFP_ATOMIC?

GFP_KERNEL	GFP_ATOMIC
1.normal zone(non atomic reserves)	normal zone(allocation atomic reserves)
2.normal pririoty allocation	high pririoty allocation
3.possible blocking call	possible non-blocking call
ex:kmalloc(GFP_KERNEL,4096);	ex:kmalloc(GFP_ATOMIC,4096)
4.allocating memory for kernel use	allocating memory within atomic context code
ex:driver routines(non critical)	ex:spinlock critical section,interrupt handler
5.low chance of success	high chance of success

12.Where are page tables stored?
typically, page tables are said to be stored in the kernel-owned physical memory. However page tables can get awfully big since each process have their own page table

13.How do you test whether there are memory leaks in a Linux application?
Valgrind tool helps to find memory leaks in linux application.

```
Ex:#include<stdio.h>
#include<stdlib.h>
int main()
{
int i,*ptr;
```

```
for(i=0;i<3;i++)
        malloc(10);

}
#valgrind ./a.out
==11838== LEAK SUMMARY:
==11838==    definitely lost: 30 bytes in 3 blocks
==11838==    indirectly lost: 0 bytes in 0 blocks
==11838==      possibly lost: 0 bytes in 0 blocks
==11838==    still reachable: 0 bytes in 0 blocks
==11838==         suppressed: 0 bytes in 0 blocks
==11838== Rerun with --leak-check=full to see details of leaked memory
```

14. Which file in Linux gives you information about memory zones?
file:/proc/zoneinfo

15. Difference between kmalloc and vmalloc.

Kmalloc	vmalloc
return inear address	return virtual address
linear address=page no+offset	virtual address=segment+offset
GFPmask	no GFPmask
not for storage devices	storage devices
physical is contigous	physical address is scatted
one address trasalation table all allocations	no. Of allocations equal to no of addess trans table
max allocation:4mb	32mb

16.What does malloc(0) return?
Malloc(0) will create block in heap area and it will return block starting address.
Ex:block sizes 8,16,32,64,....

17. What is the maximum memory that can be allocated using vmalloc?
The vmalloc upper limit is, in theory, the amount of physical RAM on the system.
Practical Max:16gb.

18. What is the maximum memory that can be allocated using kmalloc?
On both x86 and ARM, with a standard page size of 4 Kb and MAX_ORDER of 11, the kmalloc upper limit is 4 MB

19. What is the difference between VIRT, RES and SHR fields in top command?

PID – l'ID of the process(4522)

- USER – The user that is the owner of the process (root)
- PR – priority of the process (15)
- NI – The "NICE" value of the process (0)
- VIRT – virtual memory used by the process (132m)
- RES – physical memory used from the process (14m)
- SHR – shared memory of the process (3204)
- S – indicates the status of the process: S=sleep R=running Z=zombie (S)
- %CPU – This is the percentage of CPU used by this process (0.3)
- %MEM – This is the percentage of RAM used by the process (0.7)

•TIME+ –This is the total time of activity of this process (0:17.75)
•COMMAND – And this is the name of the process (bb_monitor.pl)

20. What is the system call used by malloc and free?

sbrk,brk

21.What is the maximum memory that I can allocate using malloc?

•The malloc() function reserves a block of storage of size bytes. Unlike the calloc() function,malloc()does not initialize all elements to 0. The maximum size for a non-teraspace malloc() is 16711568 bytes.(16gb)

22. What is cache coherence?

Cpu has a one extra hw circuit we can call it as a cache coherence circuit.

Cache coherence ensures instant sync between cpu caches and memory.

23. Linux intentionally leaves the first few kilobytes (or even megabytes) of each process's

virtual address space unmapped, so that attempts to dereference null pointers generatean unhandled page fault resulting in an immediate SIGSEGV, killing the process.
a) Yes b)No

ans:a
24. Virtual memory is divided into ____ a) pages b) bytes c) bits
ans:a
25. What are the advantages of using virtual memory?
No need of contigous physical memory
high memory allocations
upto 16gb
random memory allocation(like 5,67,55444.....)

26. DMA is a method of data transfer between main memory to input/output device without the need of processor.

28. What is the maximum amount of RAM you can access with 32 bit processor?
Ans:4Gb

29. When you call malloc from user space, from where it allocates memory: low mem/high mem or any other location?
Ans:for 32bit systems malloc allocates memory in high mem zone.
For 64bit systems malloc allocates in other Zone i.e Normal Zone.

30. Why use volatile memory? Why not always use nonvolatile memory?
Ans:Your computer needs a volatile memory, which temporarily stores important data or information while the computer is working.
Volatile for speed
non-volatile for long term storage
we dont use non-volatile memory in all the casess bcz in some casses we need faster access so we will some componnents as avolatile memory.

31.What does kmalloc(0) return?

One possibility is to return `NULL`
Another possibility is to return the smallest object that `kmalloc()` can manage - currently eight bytes. That is what `kmalloc()` has silently done for years.

Process Management

32.What is the difference between context switch and preemption?

Context Switch is the consequence of preemption.

Context switching means to save the state of the current running process and bringing in a new process that is supposed to run next.
Reason for a context switch could be many.
1.Current process goes for an I/O operation.
2.Current process finishes execution.
3.Current process is preempted by a higher priority process.
On the other hand, preemption means to pause the execution of a lower priority process because a higher priority process is ready for execution.

33. On a multiprocessor system, how do you find out which process is running on which processor?
Ans:mpstat -p ALL

34. How to change the priority of a process in Linux?
Ans:nice -5 a.out NI:5
nice –5 a.out NI:-5
nice -n ~~5 a.out NI:~~5
renice -n -5 ~~-p pid NI:-~~5
You can change the process priority using nice and renice utility. Nice command will launch a process with an user defined scheduling priority. Renice command will modify the scheduling priority of a running process.

The process scheduling priority range is from -20 to 19. We call this as nice value.
A nice value of -20 represents highest priority, and a nice value of 19 represent least priority for a process.

By default when a process starts, it gets the default priority of 0.

35. How can I find out the Count of number of times a process has been preempted in Linux?
Ans:cat /proc/pid/status
Ex:cat /*proc*/16/status
voluntary_ctxt_switches: 3
nonvoluntary_ctxt_switches: 0

36.What happens internally during context switch in Linux kernel?
Ans:
A `context switch` occurs when the kernel transfers control of the CPU from an executing process to another that is ready to run. The kernel first saves the `context` of the process. The context is the set of CPU register values and other data that describes the process' state. The kernel then loads the context of the new process which then starts to execute.
When the process that was taken off the CPU next runs, it resumes from the point at which it was taken off the CPU. This is possible because the saved context includes the instruction pointer. This indicates the point in the executable code that the CPU had reached when the context switch occurred.

37. What is buffer/cache?

A **buffer** is just a container to hold data for a short period of time when more comes in at any given time than a consumer can use / process. It's a first-in, first-out situation - the data comes in, might be buffered, and goes out in the same order it came in, after a while.

A **cache** is a storage for speeding up certain operations. Things get put in a cache, and should be retrieved from it multiple times, over and over again. There's no "flowing through the cache" kind of mechanism - data doesn't come in and go out in the same order - but it's just a holding container. The order might be anything, really - items are addressed via a key, they don't "flow through" but they are "put in" and stay there (until they're thrown out because of not being used, or because the system goes down).

A buffer is just like a drum it can hold data and flush it out ... On the other hand Cache is used to make your operations faster.

38. Difference between orphan and zombie process.

Orphan process ->no parent(calling process is no more or terminated)

zombie process->dead process,execution over,resource not released,exit value is not given to parent

39.What is the use of swapper process in Linux?

Swapper process id is 0.

it will create 1.init(parent of user space) 2.kthreadd(parent of kernel space)

40.How to kill the process which is in TASK_UNINTERRUPTIBLE state?

Ans:reboot the system.

41. What is load average in Linux?

Ans:uptime,top,mpstat -P ALL
Load average is the number of jobs in the run queue (state R) or waiting for disk I/O (state D) averaged over 1, 5, and 15 minutes.

Single Core system – if load average is 1.00 it means that system is fully utilized and if there will be more tasks incoming they will be queue-up and wait for execution.

•**Single Core system** – if load average is 2.00 it means that System is already utilized and some tasks are already queued-up and waiting for execution.

•**Multi core system (4 cores)** – if load average is 1.00 it means that system uses 1/4 of his CPU capabilities, one task is actively running and there are still 3 cores at 'idle' stage.

•**Multi core system (4 cores)** – if load average is 4.00 it means that system uses all 4 cores and it indicate that system is fully utilized.

42. What is resident memory in process?

•Resident memory, labelled RES: How much physical memory, how much RAM, your process is using. RES is the important number.

Virtual memory, labelled VIRT: How much memory your process thinks it's using. Usually much bigger than RES, thanks to the Linux kernel's clever memory management.

```
#include <stdio.h> #include <stdlib.h> void fill(unsigned char* addr, size_t amount) { unsigned long i; for (i = 0; i < amount; i++) { *(addr + i) = 42; } } int main(int argc, char **argv) { unsigned char *result; char input; size_t s = 1<<30; result = malloc(s); printf("Addr: %p\n", result); //fill(result, s); scanf("%c", &input); return 0; }
```

43. What is the use of Procedure-linking table (PLT) while application is starting up?
Ans:

44. Difference between fork and vfork?

BASIS FOR COMPARISON	FORK()	VFORK()
Basic	Child process and parent process has separate address spaces.	Child process and parent process shares the same address space.
Execution	Parent and child process execute simultaneously.	Parent process remains suspended till child process completes its execution.
Modification	If the child process alters any page in the address space, it is invisible to the parent	If child process alters any page in the address space, it is visible to the

BASIS FOR COMPARISON	FORK()	VFORK()
	process as the address space are separate.	parent process as they share the same address space.
Copy-on-write	fork() uses copy-on-write as an alternative where the parent and child shares same pages until any one of them modifies the shared page.	vfork() does not use copy-on-write.

45. What is kernel preemption?

Ans:No Forced Preemption

The context switch is done only when we return from the kernel. Lets take an example :

- We have 2 threads – one with high real time priority(50) and the other with low RT priority(30)
- The high priority thread went to sleep for 3 seconds
- The low priority threads calls a kernel code that last for 5 seconds
- After 5 seconds the low priority thread returns from the kernel
- The high priority thread will wake up (2 seconds late)

Preemptible Kernel

In this configuration the context switch is done on time also in the kernel, means if we run the above test we will see the high priority thread waking up after 3 seconds:

46. What is process kernel stack and process user stack?

In a Linux system, every user process has 2 stacks, a user stack and a dedicated kernel stack for the process. The user stack resides in the user address space (first 3GB in 32-bit x86 arch.) and the kernel stack resides in the kernel address space (3GB-4GB in 32bit-x86) of theprocess.
When a user process needs to execute some privileged instruction (a system call) it traps to kernel mode and the kernel executes it on behalf of the user process.This execution takes place on the process' kernel stack.

47. What is difference between background process and daemon?
The difference between running a program/process as a daemon and forking it to the background using the ampersand is basically related to ownership. ... While on the other hand, forking a program/process to the background means that you can at any time call it back to the foreground and/or kill it.

48. Which state the process is in when executing the below line and waiting for input?
scanf("%d", &val);
a) TASK_INTERRUPTIBLE b) TASK_UNINTERRUPTIBL

ans:a;

Interrupts

49. Which hardware is responsible for generating timer interrupts in Linux kernel?
Ans:the PIT is the piece of hardware responsible for issuing this periodic interrupt,called timer interrupt.
PIT:programmable interrupt timer.

50. How to direct interrupt to a particular cpu in Linux kernel?

Ans:#cat /*proc/irq/32/smp_affinity*
f;
The default value for smp_affinity is `f`, meaning that the IRQ can be serviced on any of the CPUs in the system. Setting this value to `1`, as follows, means that only CPU 0 can service this interrupt

#echo 1>/proc/irq/32/smp_affinity
#cat *proc/irq/32/smp_affinity*
1
Note:

On systems that support interrupt steering, modifying the `smp_affinity` of an IRQ sets up the hardware so that the decision to service an interrupt with a particular CPU is made at the hardware level, with no intervention from the kernel.

51. Interrupt handler in Linux kernel run with current interrupt line disabled on all processors.
a)True b) False
ans:a

52. What is the difference between request_irq and request_threaded_irq?
Ans:

request_irq :
top half and bottom half cant run in parallel.

In general, SoftIRQs are preferred for bottom-half processing that could finish consistently in few 100 microseconds (well within a jiffy).

request_threaded_irq:
you can set affinity of hard-IRQ for one CPU, while setting affinity of the associated kthread to another CPU, thus allowing top half and bottom half to run in paralle**l.**

Threaded IRQs are preferred if interrupt processing can take consistently long periods of time (exceeding a jiffy in most cases), for example packet processing in Gigabit network cards or similar bulk data-handling on any other device.

Threaded IRQ handlers are preferred for bottom-half processing that would spill over half a jiffy consistently (e.g., more than 500 microseconds if CONFIG_HZ is set to 1000).

53. What are the advantages of disabling interrupts?
Ans:
By disabling interrupts the CPU will be unable to switch processes. This guarantees that the process can use the shared variable without another process accessing it.

54. What happens when two interrupts arrive at the same time in Linux?

When two interrupt requests are raised at the same time, and both are unmasked, a given processor can only respond to one of them. Historically, this is dealt with by some piece of hardware (perhaps called an "interrupt controller") which multiplexes the interrupt lines and makes a decision about which interrupt gets through, based on some priority scheme (perhaps programmable, or else fixed). Two interrupts cannot be serviced at the same time: at least some key portion of the actions of servicing an interrupt is necessarily serialized: the portion when the CPU acknowledges the interrupt and dispatches a handler. At some point, the CPU can be made ready to handle the other interrupt which has remained pending.

55. Does Linux kernel supports interrupt nesting?
Ans:yes

56. APIC vs PIC in x86.