What is an ***operating system****?*

1

What are two common forms of lock?

2

The specification for communication between different programme elements at the object code level is the . E.g. when function arguments are passed on the stack the ABI specifies in which order they should go.

3

In C, what is an array?

4

refers to a set of action that have to happen together. In a single processor system, disabling interrupts is enough to ensure atomicity. But in a multiple processor system, a lock is needed.

5

What is a canary?

6

What is a cache? Why are there multiple cache levels?

7

What is a ***Container****?*

8

A layer of software which lies between application(s) programme(s) and the hardware. In primitive systems, its just an abstraction layer. But in more complex systems, it provides more services. This requires a degree of resource management.

- Protection: some security, in some cases prevent unauthorised access.
- Multi-tasking
- Multi-user: requires further security functionality.

Example services available

1

- Mutex: one-at-a-time permission mechanism. Consider it a binary semaphore.
- Semaphores: allows a set number of threads into a critical section. A semaphore will have an initialized value. While a thread is in a critical section the value is reduced. If the semaphore is 0, a thread may block.

Two forms of lock

2

The specification for communication between different programme elements at the object code level is the Application binary interface (ABI). E.g. when function arguments are passed on the stack the ABI specifies in which order they should go.

3

Pointers to locations in memory. The required memory is allocated when they are declared.

4

Atomicity refers to a set of action that have to happen together. In a single processor system, disabling interrupts is enough to ensure atomicity. But in a multiple processor system, a lock is needed.

5

A random value that may be placed near the buffer and can be checked for the correct value before the function return to detect a buffer overflow.

6

A small, fast memory that is close to the processor, used to increase performance. High performance machines have multiple caches, going frm a level one cache upwards, lower level caches are smaller but faster. The OS scheduler keeps track of any caches that need to be flushed.

7

A virtual computer that exists within a host system. Its a form of virtualisation that may be used to contain some users and processes to using certain files or devices. Code/Hardware may be shared but it appears as its own virtual machine.

8

What is a context, and how is the switching of it managed?

9

A situation where progress cannot be made is called a ______.

10

When a cache is about to be flushed the ______ is checked, if the cache is clean it can be dropped, else the entry must be copied down the memory hierarchy.

11

What is a DMA controller? How does DMA work?

12

How does Dynamic Memory Allocation alleviate the problem of fragmentation?

13

The OS is capable of generating ______ but most come from hardware and are then usually handled by the OS. They act as a call to the processor, causing it to call a subroutine, the call also raises the privilege of the processor so that it can deal with the hardware.

14

Give some examples of exceptions?

15

A ______ is a persistent and large data store that all processes can access. Access to the ______ is done via device drivers that are part of the O.S.

16

A situation where progress cannot be made is called a deadlock.

10

An executing process has a ***context*** *and a PCB which is used to hold information. Context switching is managed by the scheduler. All information held in hardware will be stored to the PCB. Old caches may need to be flushed.*

9

A simple processor capable of moving data from one place to another and counting how much it has done. Direct memory access is a technique for letting I/O devices interact with memory via a simple controller rather than via the CPU, feeing it up. The CPU will set up the data transfer, telling the DMA controller the device, memory address and amount to transfer. It will raise an interrupt when it is complete.

12

When a cache is about to be flushed the dirty bit is checked, if the cache is clean it can be dropped, else the entry must be copied down the memory hierarchy.

11

The OS is capable of generating exceptions but most come from hardware and are then usually handled by the OS. They act as a call to the processor, causing it to call a subroutine, the call also raises the privilege of the processor so that it can deal with the hardware.

14

By allocating fixed size block but this can be wasteful or inconvenient when the programmer wants a larger block.

13

A Filing System is a persistent and large data store that all processes can access. Access to the Filing System is done via device drivers that are part of the O.S.

16

- *Reset: used to start a system in a well defined way*
- *System call*
- *Emulator trap*
- *Memory fault*
- *Interrupt*
- *Privilege violation*

Example exceptions

15

A typical computer might have a number of ”levels” of storage. The machine’s view is usually something like:

17

What are the four layers of the I/O stack?

18

It is the role of the to intermediate between peripheral devices and application software as well providing as sensible abstraction of the IO resources.

19

are a form of hardware level exception that is handled by a privileged function before returning to the code that was originally running.

20

What does an Interrupt Service Routine (ISR) do?

21

How does priority affect the ISR?

22

Most of a processes context will be held in and each process will have some which it can use to store variables and code.

23

The is the core of an OS and resides in protected space and is accessed by the user via system calls.

24

- User-level processing - application code, application is responsible for formatting.
- Device-independent system calls - forces different devices to follow similar interface conventions and helps applications interface with hardware.
- Device driver - translates standard commands to communicate wit the hardware into the specific required sequences.
- Interrupt handlers - handle the normal servicing of the a device using ISRs.

The 4 layers of the I/O stack

18

- Processor registers
- Primary memory - chiefly RAM
- Secondary storage - typically disks although other exists
- Network - not really part of the computer but a source of data nevertheless

Storage "levels"

17

Interrupts are a form of hardware level exception that is handled by a privileged function before returning to the code that was originally running.

20

It is the role of the O.S. to intermediate between peripheral devices and application software as well providing as sensible abstraction of the IO resources.

19

Interrupts of a lower priority are disabled so that only higher priority interrupts can interrupt the one currently being serviced. ISR code is privileged and should not be accessible by the user for security reasons.

22

It is called when an interrupt is handled, borrowing the processor, some data, such as register state, has to be saved but not the entire process context. They are generally kept quite quick and lightweight, as interrupts are often urgent urgent and low latency is important.

21

The kernel is the core of an OS and resides in protected space and is accessed by the user via system calls.

24

Most of a processes context will be held in memory and each process will have some address space which it can use to store variables and code.

23

What is the kernel responsible for?

25

What are the 3 types of kernel and briefly describe them?

26

MidTermQ: What is the difference between virtual and physical memory addresses? Which (if any) is the application programmer able to manipulate?

27

MidTermQ: How does a system call differ from an application method/function call?

28

MidTermQ: A is a background process, typically owned by root. Examples abound: page, cron, logging...

29

MidTermQ: If a page is pinned, what effect does this have (if any) on the page eviction policy? Give two different reasons for pinning a page.

30

MidTermQ: What are the three main states a process can be in?

31

MidTermQ: How can a state change?

32

- Monolithic: the entire operating system runs as a single program in kernel mode.
- Layered Systems: layers selected so each only uses functions, operations services of lower layers. Lower layers (kernel) contain most fundamental functions to manage resources.
- Microkernels: keep only minimal functionality in the OS.

3 types of kernel

26

Implementing mechanisms, e.g. how a scheduler switches process.

25

It changes mode to enter privileged code, having a fixed set of entry addresses in protected space. It is normally slower than a simple function call.

28

Physical memory addresses refers to a specific location in memory, whereas virtual memory is a meaningless address that needs to be mapped to a physical address using a page table. A programmer will manipulate the virtual address space.

27

A pinned page is not considered for page eviction; it is retained in RAM. A page may be pinned if the code or data it contains is likely to be wanted rapidly and unpredictably – e.g. interrupt handlers. If the page is currently subject to a DMA transfer it must be kept in RAM even if that process is not running in software.

30

MidTermQ: A daemon is a background process, typically owned by root. Examples abound: page, cron, logging...

29

- Running: can become stuck (e.g. on I/O) and change to blocked; can be preempted and change to ready.
- Ready: will be changed to running if chosen by the scheduler.
- Blocked: will be changed to ready (or, possibly, running) when unblocked (e.g. by a system call or interrupt).

Ways in which states can change.

32

Ready, running and blocked.

31

MidTermQ: In a virtual memory system, under what circumstances might a page fault occur?

33

MidTermQ: What does the operating system need to do to classify and process a page fault?

34

MidTermQ: Give an example of a hard real-time computer system.

35

MidTermQ: What is the basic difference in process scheduling in a hard real-time computer compared to a Windows or Unix system? What is the role of the process' priority in each case?

36

MidTermQ: A real-time system has three processes, at high, medium and low priority. The high and the low priority processes share a data structure which is protected by a mutex. Describe how it might be possible to reach a circumstance where the medium priority process is delaying the running of the high-priority process.

37

MidTermQ: What is the main difference between a process and a thread? In what way(s) are they similar?

38

MidTermQ: Why is thread-switching within a process usually much quicker than process-switching? Outline all the differences which you believe may have a significant impact on the time penalty for switching processes in a system using virtual memory and note which (if any) are unnecessary when thread switching.

39

MidTermQ: What implications are there for thread and process scheduling if the computer has multiple processors (using the same physical memory) rather than a single CPU?

40

Identify the faulting address. Check validity against the process' segment definitions; permissions can be obtained and checked. If the address or the operation was illegal, a segmentation fault is indicated and the process is terminated. Else, if the operation was legitimate and the page is present but the page permissions have been artificially reduced, increase the permissions and return to the faulting operation (a minor page fault). Else, if the page is absent, set up a DMA transfer to fetch the faulting page from the disk.

34

- an illegal virtual address (e.g. corrupted pointer)
- a legal virtual address with the page not present in memory
- a privilege violation
- a legitimate page which has had its privileges temporarily reduced (for monitoring purposes)

Reasons which may cause a page fault.

33

A real-time system schedules to minimise latency, particularly preferring high-priority processes. It can assume that every process will block reasonable quickly and the processor(s) will spend some time in enforced idleness. Timeslicing is counterproductive and may not be used, or be confined to processes of equal priority. An interactive scheduler tries to balance general throughput; it will assume that any process may be running indefinitely and will timeslice running processes. All processes will progress: low-priority processes will receive a smaller share of (rather than no) processor time.

36

Any decent example will do: flight control, in-car control system, Segway balancing system etc.

35

The context which is private to a process includes the address space and the resources it is using. Threads within a process share access to these resources. Both processes and threads are independently schedulable code units. Each has some private data such as register values and a stack.

38

1) Low priority process wins the mutex
2) High priority process preempts
3) Medium priority process becomes ready
4) High priority process blocks on mutex
5) Medium priority process runs, preventing low priority process from releasing the mutex.

This is the classic priority inversion.

37

For processes, really not much as each process has an independent context. For threads, the virtual address space is shared but will be managed independently on each processor. Threads can only be switched cheaply if they use the same processor, otherwise they will be similar to processes. A kernel-based thread scheduler may take this into account.

40

Switching the processor context (registers) is common to both and takes a noticeable time. The major difference is the change of the memory view in process switching. Switching page tables is quite quick but the consequence of changing the virtual memory map is that the TLB and the (virtual) L1 cache(s) will need to be flushed. There is a penalty in writing back the dirty lines from the cache, plus an ongoing performance impact until the cache(s) and TLB have refilled for the new process.

39

A is one which must be executed atomically.

41

MidTermQ: A critical section of code is used by a single system call. What are the simplest ways of guaranteeing its correct operation in a single-tasking uniprocessor system?

42

MidTermQ: A critical section of code is used by a single system call. What are the simplest ways of guaranteeing its correct operation in a multi-tasking uniprocessor system?

43

MidTermQ: A critical section of code is used by a single system call. What are the simplest ways of guaranteeing its correct operation in a multi-tasking, shared memory multiprocessor system?

44

MidTermQ: is the keeping of copies of a subset of available data in a local, fast store. If the data is wanted in again it is available rapidly.

45

MidTermQ: What has happened if part of a cache is "dirty"?

46

MidTermQ: Give some examples of where the principle of caching is applied in computing, and how the kernel is involved (if it is).

47

MidTermQ: In the general context of caching, what is "LRU" and why is it generally effective?

48

In a single-tasking system there can be no interruptions so there is no need for anything other than the code itself.

42

A critical section is one which must be executed atomically.

41

In a multi-processor system interference from other processors must be prevented; some form of lock (e.g. a "mutex") will be needed.]

44

In a multi-tasking system, context switching needs to be avoided; this can be done by disabling interrupts over the critical section.

43

There has been a write operation and the cache area is inconsistent with the original copy; it will need saving, later.

46

MidTermQ: Caching is the keeping of copies of a subset of available data in a local, fast store. If the data is wanted in again it is available rapidly.

45

LRU is "Least Recently Used" - an algorithm which can be used to evict part of a cache contents. It selects the part which has not been used in the longest time. It works because, statistically, data which haven't been used for some time are typically not going to be used in the immediate future either.

48

- The memory data cache hierarchy. The kernel is responsible for managing cachability of different memory areas and flushing the virtual cache at context-switch time.
- Virtual memory. The kernel is responsible for trapping cache misses, choosing pages for eviction, setting up the refill transfers, etc.
- TLB. The kernel needs to flush the TLB on a context switch.
- Web cache. Not O.S. related.

Examples of caching

47