Study Set for Lecture 01: Overview of Operating Systems

Due Aug 31 at 11:59pm **Points** 10 **Questions** 14
Available Aug 25 at 12pm - Dec 8 at 8pm 4 months **Time Limit** None
Allowed Attempts Unlimited

Instructions

Review lecture notes from **lect01_Overview_of_Operating_Systems.pdf.**

Then answer the questions from this study set and submit them to gain access to the further part of the course.

Take the Survey Again

Correct answers are hidden.

Score for this attempt: **10** out of 10
Submitted Dec 6 at 7:40am
This attempt took 13 minutes.

Question 1

What are the three main goals of an operating system?

Your Answer:

The main goal is efficient use. The O/S makes sure the efficient use of memory, CPU and input output devices.

Operating systems help ensure user conveniences, as it can help make a user (human) interfaces like the Graphical User Interfaces.

Help execute user programs to help solve problems.

Question 2

Describe the role of an operating system as a resource allocator and as a control program.

Your Answer:

The role of an operating system as a resource allocator manages all resources and decides which resources to allocate to different programs fairly.

The role of an operating system as a control program helps execute programs to help prevent errors and improper use of the computer.

Question 3

What is a an operating system kernel?

Your Answer:

The Kernal is a program that is at the core of the computers Operating System. The main point of a Kernal is it facilitates interactions between hardware and software components.

Deals with the critical aspects of running the O/S.

Question 4

What is the purpose of interrupts? What are the differences between an exception and an interrupt? Can exceptions be generated intentionally by a user program? If so, for what purpose?

Your Answer:

Interrupts are signals sent to the CPU by external devices, normally I/O devices. They tell the CPU to stop its current activities and execute the appropriate part of the operating system.

Interrupts are important because they give the user better control over the computer.

Interrupts are outside the processor(serial ports, keyboard), and exception is inside the processor(division by zero, undefined opcode).

Exceptions can be generated internationally by a user program by a trap.

Question 5

Describe each of the following exception types:

- traps,
- faults, and
- aborts.

Your Answer:

A trap is a software-generated interrupt caused either by an error or a user request.

A **trap** is an exception that is reported immediately following the execution of the trapping instruction. Traps allow execution of a program or task to be continued without loss of program continuity. The return address for the trap handler points to the instruction to be executed after the trapping instruction.

A **fault** is an exception that can generally be corrected and that, once corrected, allows the program to be restarted with no loss of continuity. When a fault is reported, the processor restores the machine state to the state prior to the beginning of execution of the faulting instruction. The return address (saved contents of the CS and EIP registers) for the fault handler points to the faulting instruction, rather than to the instruction following the faulting instruction.

An **abort** is an exception that does not always report the precise location of the instruction causing the exception and does not allow a restart of the

program or task that caused the exception. Aborts are used to report severe errors, such as hardware errors and inconsistent or illegal values in system tables. Errors are severe.

Question 6

Describe synchronous and asynchronous communication (input / output; I/O) between the OS and a device.

Your Answer:

Synchronous communication means that two or more people exchange information in real-time.

Asynchronous communication refers to the exchange of data between two or more parties without the requirement for all the recipients to respond immediately.

Question 7

Why are caches useful? What problems do they solve? What problems do they cause? If a cache can be made as large as the device for which it is caching (for instance, a cache as large as a disk), why not make it that large and eliminate the device?

Your Answer:

Caches are useful when two or more components need to exchange data, and the components perform transfers at differing speeds. Caches solve the transfer problem by providing a buffer of intermediate speed between the components. If the fast device finds the data it needs in the cache, it need not wait for the slower device. The data in the cache must be kept consistent with the data in the components. If a component has a data value change, and the datum is also in the cache, the cache must also be updated. This is

especially a problem on multiprocessor systems where more than one process may be accessing a datum. A component may be eliminated by an equal-sized cache, but only if :

1. the cache and the component have equivalent state-saving capacity (that is, if the component retains its data when electricity is removed, the cache must retain data as well) , and
2. the cache is affordable, because faster storage tends to be more expensive.

Question 8

Describe the challenges in using cached data.

Your Answer:

.1 What is cache penetration?

The data to be queried by the business system simply exists! When the business system initiates a query, according to the above process, the query will first go to the cache, because the cache does not exist, and then go to the database for query. Since the data does not exist at all, the database also returns null. This is the cache penetration.

To sum up: the data that the business system access does not exist at all is called cache penetration.

1.2 The hazard of cache penetration

If there are massive data that does not exist in the query request, then these massive requests will fall into the database, and the database pressure will increase dramatically, which may lead to system crash. (You have to know that the most vulnerable in the current business system is IO, a little bit It will collapse under pressure, so we have to think of ways to protect it).

1.3 Why does cache penetration occur?

There are many reasons for cache penetration, which are generally as follows:

1. Malicious attacks deliberately create a large amount of non-existent data to request our services. Since these data do not exist in the cache, massive requests fall into the database, which may cause the database to crash.
2. Code logic error. This is the programmer's pot, nothing to say, must be avoided in development!

2. Cache avalanche

2.1 What is a cache avalanche?

As you can see from the above, the cache actually plays a role in protecting the database. It helps the database to withstand a large number of query requests, thus avoiding vulnerable databases.

If the cache goes down for some reason, the massive query request that was originally blocked by the cache will flock to the database like a mad dog. At this point, if the database can't withstand this huge pressure, it will collapse.

This is the cache avalanche.

3. Hotspot data set is invalid

3.1 What is the hotspot data set failure?

We usually set an expiration time for the cache. After the expiration time, the database will be deleted directly by the cache, thus ensuring the real-time performance of the data to a certain extent.

However, for some hot data with very high requests, once the valid time has passed, there will be a large number of requests falling on the database at this moment, which may cause the database to crash. The process is as follows:

If a hotspot data fails, then when there is a query request [req-1] for the data again, it will go to the database query. However, from the time the request is sent to the database to the time the data is updated into the cache, since the data is still not in the cache, the query request arriving

during this time will fall on the database, which will cause the database Enormous pressure. In addition, when these request queries are completed, the cache is updated repeatedly.

Question 9

Describe what multiprogramming is and why is it used in modern computers.

Your Answer:

multi programming is the ability of an os to execute multiple programs at the same time on single processor machine .one or more programs reside in the main memory which are ready to execute.the cpu can execute only one instruction at a time.if the currently executing process performs i/o operation or waiting for i/o then the os may interrupt that process and gives the control to other process residing in main memory.

the main aim of multi programming is to make cpu busy untill as there are processes waiting to execute.

thus no cpu time is wasted by the system waiting for i/o task to be completed.

Question 10

Explain what is meant by stating that modern operating systems are interrupt-driven.

Your Answer:

When they say that modern OS's are interrupt-driven, they are referring to the fact that the hardware generates interrupts, and software reacts to them

Question 11

How does the distinction between kernel mode and user mode function as a rudimentary form of a protection (security) system?

Your Answer:

The distinction between kernel mode and user mode provides a rudimentary form of protection in the following manner. Certain instructions could be executed only when the CPU is in kernel mode. Similarly, hardware devices could be accessed only when the program is executing in kernel mode. Control over when interrupts could be enabled or disabled is also possible only when the CPU is in kernel mode. Consequently, the CPU has very limited capability when executing in user mode, thereby enforcing protection of critical resources.

Question 12

What is a system call?

Your Answer:

System calls allow user-level processes to request services of the operating system.

In **computing** (https://en.wikipedia.org/wiki/Computing), a **system call** (commonly abbreviated to **syscall**) is the programmatic way in which a **computer program** (https://en.wikipedia.org/wiki/Computer_program) requests a service from the **kernel** (https://en.wikipedia.org/wiki/Kernel_(computing)) of the **operating system** (https://en.wikipedia.org/wiki/Operating_system) on which it is executed. In most systems, system calls can only be made from **userspace** (https://en.wikipedia.org/wiki/Userspace) processes,

while in some systems, **OS/360 and successors** (https://en.wikipedia.org/wiki/OS/360_and_successors) for example, privileged system code also issues system calls.

On **Unix** (https://en.wikipedia.org/wiki/Unix), **Unix-like** (https://en.wikipedia.org/wiki/Unix-like) and other **POSIX** (https://en.wikipedia.org/wiki/POSIX)-compliant operating systems, popular system calls are `open` (https://en.wikipedia.org/wiki/Open_(system_call)), `read` (https://en.wikipedia.org/wiki/Read_(system_call)), `write` (https://en.wikipedia.org/wiki/Write_(system_call)), `close` (https://en.wikipedia.org/wiki/Close_(system_call)), `wait` (https://en.wikipedia.org/wiki/Wait_(system_call)), `exec` (https://en.wikipedia.org/wiki/Exec_(system_call)), `fork` (https://en.wikipedia.org/wiki/Fork_(system_call)), `exit` (https://en.wikipedia.org/wiki/Exit_(system_call)), and `kill` (https://en.wikipedia.org/wiki/Kill_(system_call)). Many modern operating systems have hundreds of system calls. For example, **Linux** (https://en.wikipedia.org/wiki/Linux_kernel) and **OpenBSD** (https://en.wikipedia.org/wiki/OpenBSD) each have over 300 different calls, [2] (https://en.wikipedia.org/wiki/System_call#cite_note-2) [3] (https://en.wikipedia.org/wiki/System_call#cite_note-3) **NetBSD** (https://en.wikipedia.org/wiki/NetBSD) has close to 500,[4] (https://en.wikipedia.org/wiki/System_call#cite_note-4) **FreeBSD** (https://en.wikipedia.org/wiki/FreeBSD) has over 500,[5] (https://en.wikipedia.org/wiki/System_call#cite_note-5) Windows 7 has close to 700,[6] (https://en.wikipedia.org/wiki/System_call#cite_note-6) while **Plan 9** (https://en.wikipedia.org/wiki/Plan_9_from_Bell_Labs) has 51.[7] (https://en.wikipedia.org/wiki/System_call#cite_note-7)

Question 13

What is the difference between a program and a process?

Your Answer:

Comparison Chart

BASIS FOR COMPARISON	PROGRAM	PROCESS
Basic	Program is a set of instruction.	When a program is executed, it is known as process.
Nature	Passive	Active
Lifespan	Longer	Limited
Required resources	Program is stored on disk in some file and does not require any other resources.	Process holds resources such as CPU, memory address, disk, I/O etc.

A **Program**, in simple words, can be considered as a system activity. In batch processing system these are called executing jobs while in a real-time operating system it is called tasks or programs. A user can run multiple programs where the operating system facilitates its own internal programmed activities such as memory management using some techniques.

A **Process** is an execution of a program. It is considered as an **active entity** and realizes the actions specified in a program. Multiple processes can be related to the same program. It handles the operating system activities through **PCB (Process control Block)** which includes program counter, stack, state etc. Program counter stores the next sequence of instruction that is to be executed later.

Question 14

What is system concurrency?

Your Answer:

In **computer science** **(https://en.wikipedia.org/wiki/Computer_science)** , **concurrency** is the ability of different parts or units of a program, algorithm, or problem to be executed out-of-order or in partial order, without affecting the final outcome. This allows for parallel execution of

the concurrent units, which can significantly improve overall speed of the execution in multi-processor and multi-core systems. In more technical terms, concurrency refers to the decomposability property of a program, algorithm, or problem into order-independent or partially-ordered components or units.

Concurrent programming (https://en.wikipedia.org/wiki/Concurrent_programming) encompasses programming languages and algorithms used to implement concurrent systems. Concurrent programming is usually considered to be more general than **parallel programming** (https://en.wikipedia.org/wiki/Parallel_programming) because it can involve arbitrary and dynamic patterns of communication and interaction, whereas parallel systems generally have a predefined and well-structured communications pattern. The base goals of concurrent programming include *correctness*, *performance* and *robustness*. Concurrent systems such as **Operating systems** (https://en.wikipedia.org/wiki/Operating_system) and **Database management systems** (https://en.wikipedia.org/wiki/Database_management_system) are generally designed to operate indefinitely, including automatic recovery from failure, and not terminate unexpectedly (see **Concurrency control** (https://en.wikipedia.org/wiki/Concurrency_control)). Some concurrent systems implement a form of transparent concurrency, in which concurrent computational entities may compete for and share a single resource, but the complexities of this competition and sharing are shielded from the programmer.

Survey Score: **10** out of 10