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

Study Set for Lecture 02: Operating System Structures

Due Sep 7 at 11:59pm **Points** 10 **Questions** 13
Available Sep 1 at 12pm - Dec 8 at 8pm 3 months **Time Limit** None
Allowed Attempts Unlimited

Instructions

Review lecture notes from **lect02_Operating_System_Structures.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 Sep 1 at 2:22pm
This attempt took 51 minutes.

Question 1

Explain what a bootstrap program and an OS loader are, and what's the difference between them.

Your Answer:

Operating systems are designed to run on a variety of machines.

The Bootstrap is a process that the computer must go through to be loaded and started. It is sometimes called boot for short.

The OS loader is a intermediary software that allows users to choose from several different operating systems that are present on the boot device.

The difference between the two is Bootstrap is a program that resides in the computer that is automatically executed by the processor when turning on the computer. The OS loader controls the board upon power-up and does not rely on the Linux kernel in any way. In contrast, the bootstrap primary purpose in life is to act as the glue between a board-level bootloader and the Linux kernel. It is the bootstrap responsibility to provide a proper context for the kernel to run in, as well as perform the necessary steps to decompress and relocate the kernel binary image.

Question 2

Explain what is a difference between a boot sequence and a boot menu.

Your Answer:

After power-on self-tests (POST), the code that is called bootstrap (or boot) loader tests and initializes the hardware and then following a **boot sequence** loads an operating system.

The **boot sequence** is the order of devices listed in BIOS that the computer will look for an operating system on.

On some computers, boot sequence can be overridden during the boot by invoking a **boot menu**, and then selecting a device to boot from disregarding the boot sequence.

● for example, it can be done on Dell computers by pressing the F12 key (as used in the lab)

Question 3

List and describe each of the services provided by an operating system to users.

Your Answer:

Program execution - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)

I/O operations - A running program may require I/O, which may involve a file or an I/O device.

File-system manipulation - The file system is of particular interest. Obviously, programs need to read and write files and directories, create and delete them, search them, list file information, permission management.

Communication – Processes may exchange information, on the same computer or between computers over a network ● Communications may be via shared memory or through message passing (packets moved by the OS)

Error detection – OS needs to be constantly aware of possible errors

● May occur in the CPU and memory hardware, in I/O devices, in user program

● For each type of error, OS should take the appropriate action to ensure correct and consistent computing

● Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system 7 COMP362 Operating Systems, Prof. AJ Bieszczad OS Services for Resource Sharing

Resource allocation

● When multiple users or multiple jobs run concurrently, resources must be allocated to each of them

●Some resources such as CPU cycles, main memory, and file storage need customized allocation algorithms (e.g., dynamically depending on the state)

●Others such as I/O devices may use generalized request and release algorithms

Accounting/Logging

● To keep track of which users use how much and what kinds of computer resources

Protection and security

● The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other

●Protection is a mechanism

●i.e., involves ensuring that all access to system resources is controlled; for example, from erroneous code like runaway (“floating”) pointers

●Security is a policy

●i.e., access requires user authentication to defending resources from invalid access attempts; for example, from unauthorized access

Question 4

What is the purpose of the command interpreter (shell)? Why is a shell usually separate from the kernel?

Your Answer:

The command interpreter or the command-line interface is one of the ways a user can interface with the operating system. The command interpreter's main task is to understands and executes commands which it turns into system calls.

The kernel is the central module of an OS. Since the kernel is the core of OS it would be dangerous to have code which is prone to changes as part of the kernel. Any update or change made to the kernel need to be thought carefully.

Question 5

List and describe each of the OS services for resource sharing.

Your Answer:

Resource allocation

● When multiple users or multiple jobs run concurrently, resources must be allocated to each of them

-Some resources such as CPU cycles, main memory, and file storage need customized allocation algorithms (e.g., dynamically depending on the state)

-Others such as I/O devices may use generalized request and release algorithms

Accounting/Logging

● To keep track of which users use how much and what kinds of computer resources

Protection and security

● The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other

-Protection is a mechanism

●i.e., involves ensuring that all access to system resources is controlled; for example, from erroneous code like runaway (“floating”) pointers

-Security is a policy

●i.e., access requires user authentication to defending resources from invalid access attempts; for example, from unauthorized access

Question 6

What is a system call? What is it used for? How can it be implemented? Do application programs usually use system calls directly?

Your Answer:

In computing, a **system call** is the programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on. A system call is a way for programs to **interact with the operating system**. A computer program makes a system call when it makes a request to the operating system's kernel. System call **provides** the services of the operating system to the user programs via Application Program Interface(API). It provides an interface between a process and operating system to allow user-level processes to request services of the operating system. System calls are the only entry points into the kernel system. All programs needing resources must use system calls.

Why do you need System Calls in OS?

Following are situations which need system calls in OS:

- Reading and writing from files demand system calls.
- If a file system wants to create or delete files, system calls are required.
- System calls are used for the creation and management of new processes.
- Network connections need system calls for sending and receiving packets.
- Access to hardware devices like scanner, printer, need a system call.

Question 7

How can parameters be passed to system calls?

Your Answer:

There are three general methods used to pass parameters to the OS

●NOTE: This is a choice that the OS designers make, and not the users

●The developers of the APIs have to stick to the OS convention

1) Simplest: pass the **parameters in registers** (e.g., MS-DOS)

● In some cases, may be more parameters than registers

2) **Parameters placed, or pushed, onto the stack** by the program and popped off the stack by the operating system (e.g., NetBSD)

● This is just like a regular function call

3) **Parameters stored** in a block, or table, in memory, and address of the block passed as a parameter in a register (e.g., Linux, Solaris)

Question 8

List and describe common types of system calls.

Your Answer:

Process Control

These system calls deal with processes such as process creation, process termination etc.

File Management

These system calls are responsible for file manipulation such as creating a file, reading a file, writing into a file etc.

Device Management

These system calls are responsible for device manipulation such as reading from device buffers, writing into device buffers etc.

Information Maintenance

These system calls handle information and its transfer between the operating system and the user program.

Communication

These system calls are useful for interprocess communication. They also deal with creating and deleting a communication connection.

Some of the examples of all the above types of system calls in Windows and Unix are given as follows −

Types of System Calls	Windows	Linux
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Management	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Management	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()

Question 9

What is the purpose of system programs (utilities)?
Why are they a better solution than providing system or shell built-in functions.

Your Answer:

System programs provide a convenient environment for program development and execution.

Some are simply user interfaces to system calls; others are considerably more complex

1)File management

● Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories

2) Status information

● Some ask the system for info - date, time, amount of available memory, disk space, number of users

● Others provide detailed performance, logging, and debugging information

● Typically, these programs format and print the output to the terminal or other output devices

●the text can be piped into another program rather than sent to the terminal

● Some systems implement a system database used to store and retrieve configuration information; for example, Registry in Windows

● Others, provide utilities to manipulate per-program configuration files; for example, plists in macOS 17 COMP362 Operating Systems, Prof. AJ Bieszczad System Programs (cont.)

3)File modification

● Text editors to create and modify files

●Special commands to search contents of files or perform transformations of the text

4) Programming-language support

● Compilers, assemblers, debuggers and interpreters sometimes provided

5) Program loading and execution

●Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language

6) Communication

●Provide the mechanism for creating virtual connections among processes, users, and computer systems

●Allow users to send messages to one another’s screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another

7)Common application programs

●(e.g., date, time, text processing, pattern matching, searching, calculators, formatters, etc.)

●Some applications may become recognized as system programs as they gain popularity and are included with the distribution of the OS

●e.g., many GNU utility programs

Question 10

List and describe the categories of system programs.

Your Answer:

Some are simply user interfaces to system calls; others are considerably more complex

1)File management

● Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories

2) Status information

● Some ask the system for info - date, time, amount of available memory, disk space, number of users

● Others provide detailed performance, logging, and debugging information

● Typically, these programs format and print the output to the terminal or other output devices

●the text can be piped into another program rather than sent to the terminal

● Some systems implement a system database used to store and retrieve configuration information; for example, Registry in Windows

● Others, provide utilities to manipulate per-program configuration files; for example, plists in macOS 17 COMP362 Operating Systems, Prof. AJ Bieszczad System Programs (cont.)

3)File modification

● Text editors to create and modify files

●Special commands to search contents of files or perform transformations of the text

4) Programming-language support

● Compilers, assemblers, debuggers and interpreters sometimes provided

5) Program loading and execution

●Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language

6) Communication

●Provide the mechanism for creating virtual connections among processes, users, and computer systems

●Allow users to send messages to one another's screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another

7)Common application programs

●(e.g., date, time, text processing, pattern matching, searching, calculators, formatters, etc.)

●Some applications may become recognized as system programs as they gain popularity and are included with the distribution of the OS

●e.g., many GNU utility programs

Question 11

Describe and evaluate the microkernel approach to operating system design.

Your Answer:

Kernel is the core part of an operating system which manages system resources. It also acts like a bridge between application and hardware of the computer. It is one of the first programs loaded on start-up (after the Bootloader).

Microkernel is one of the classification of the kernel. Being a kernel it manages all system resources. But in a **microkernel,** the **user services** and **kernel services** are implemented in different address space. The user services are kept in **user address space**, and kernel services are kept under **kernel address space**, thus also reduces the size of kernel and size of operating system as well.

Advantages of Microkernel –

- The architecture of this kernel is small and isolated hence it can function better.

- Expansion of the system is easier, it is simply added in the system application without disturbing the kernel.

Microkernel Architecture –
Since kernel is the core part of the operating system, so it is meant for handling the most important services only. Thus in this architecture only the most important services are inside kernel and rest of the OS services are present inside system application program. Thus users are able to interact with those not-so important services within the system application. And the microkernel is solely responsible for the most important services of operating system they are named as follows:

- Inter process-Communication
- Memory Management
- CPU-Scheduling

Question 12

Describe and evaluate the layered architecture of operating systems.

Your Answer:

This approach breaks up the operating system into different layers.

- This allows implementers to change the inner workings, and increases modularity.
- As long as the external interface of the routines don’t change, developers have more freedom to change the inner workings of the routines.
- With the layered approach, the bottom layer is the hardware, while the highest layer is the user interface.
    - The main *advantage* is simplicity of construction and debugging.
    - The main *difficulty* is defining the various layers.
    - The main *disadvantage* is that the OS tends to be less efficient than other implementations.

Question 13

Describe and evaluate the architecture of operating system based on compiled and loadable modules.

Your Answer:

Most modern operating systems implement kernel modules

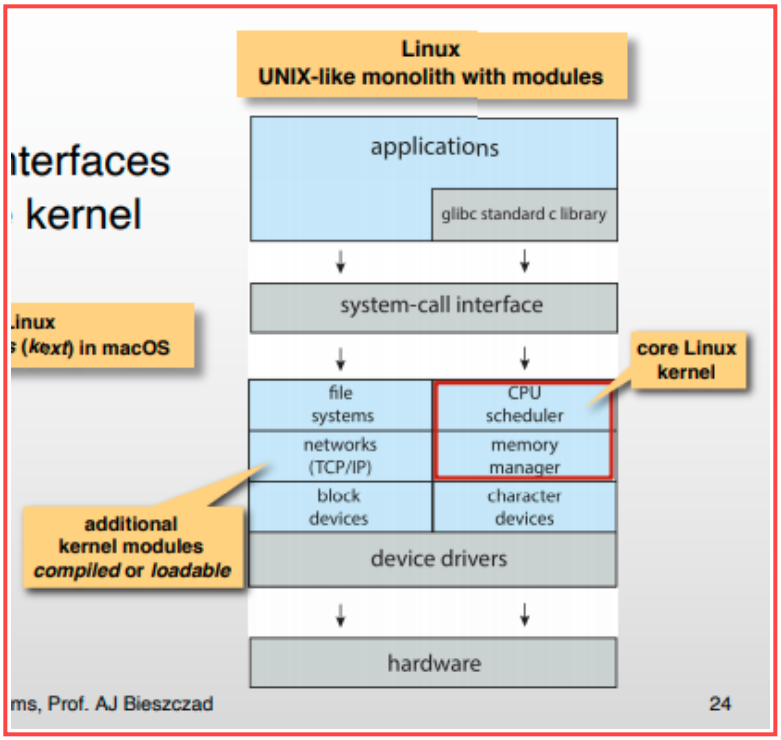
● Object-oriented approach

●Each core component is separate

●Each talks to the others over known interfaces

●Each is loadable as needed within the kernel

● Overall, like layers but more flexible



The basic goal remains however the same: keep what is loaded at boot-time minimal while still allowing the kernel to perform more complex functions. The basics of modular kernel are very close to what we find in implementation of *plugins* in applications or *dynamic libraries* in general.

Advantages

- The kernel doesn't have to load everything at boot time; it can be expanded as needed. This can decrease boot time, as some drivers won't be loaded unless the hardware they run is used (NOTE: This boot time decrease can be negligible depending on what drivers are modules, how they're loaded, etc.)
- The core kernel isn't as big
- If you need a new module, you don't have to recompile.

Disadvantages

- It may lose stability. If there is a module that does something bad, the kernel can crash, as modules should have full permissions.
- ...and therefore security is compromised. A module can do anything, so one could easily write an evil module to crash things. (Some OSs, like Linux (https://en.wikipedia.org/wiki/Linux) , only allow modules to be loaded by the root user.)
- Coding can be more difficult, as the module cannot reference kernel procedures without kernel symbols.

What does a Modular Kernel look like ?

There are several components that can be identified in virtually every modular kernel:

The core
This is the collection of features in the kernel that are absolutely mandatory regardless of whether you have modules or not.

The modules loader
This is a part of the system that will be responsible of preparing a module file so that it can be used as if it was a part of the core itself.

The kernel symbols table
This contains additional information about the core and loaded modules that the module loader needs in order to *link* a new module to the existing kernel.

The dependencies tracking
As soon as you want to *unload* some module, you'll have to know whether you can do it or not. Especially, if a module *X* has requested

symbols from module *Z*, trying to unload *Z* while *X* is present in the system is likely to cause havoc.

Modules

Every part of the system you might want (or don't want) to have.

Survey Score: **10** out of 10

Study Set for Lecture 03: Processes

Due Sep 14 at 11:59pm **Points** 10 **Questions** 7
Available Sep 8 at 12pm - Dec 8 at 8pm 3 months **Time Limit** None
Allowed Attempts Unlimited

Instructions

Review lecture notes from **lect03_Processes.pdf**.

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

Take the Quiz Again

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	51 minutes	8 out of 10

Correct answers are hidden.

Score for this attempt: **8** out of 10
Submitted Sep 8 at 2:27pm
This attempt took 51 minutes.

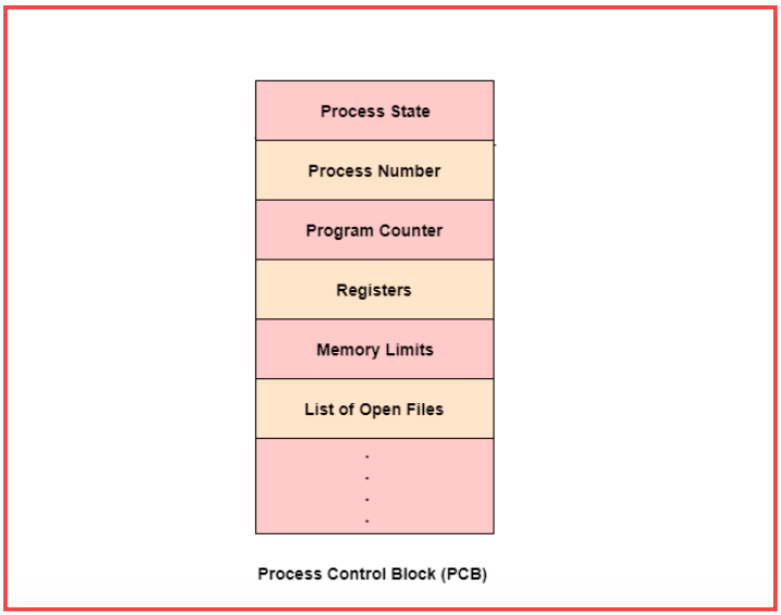
Question 1 — 0 / 0 pts

Describe what a PCB (Process Control Block) is and how it is used by the OS.
What information does it contain?

Your Answer:

Process Control Block is a data structure that contains information of the process related to it. The process control block is also known as a task control block, entry of the process table, etc.

It is very important for process management as the data structuring for processes is done in terms of the PCB. It also defines the current state of the operating system.



The following are the data items −

Process State

This specifies the process state i.e. new, ready, running, waiting or terminated.

Process Number

This shows the number of the particular process.

Program Counter

This contains the address of the next instruction that needs to be executed in the process.

Registers

This specifies the registers that are used by the process. They may include accumulators, index registers, stack pointers, general purpose registers etc.

List of Open Files

These are the different files that are associated with the process

CPU Scheduling Information

The process priority, pointers to scheduling queues etc. is the CPU scheduling information that is contained in the PCB. This may also include any other scheduling parameters.

Memory Management Information

The memory management information includes the page tables or the segment tables depending on the memory system used. It also contains the value of the base registers, limit registers etc.

I/O Status Information

This information includes the list of I/O devices used by the process, the list of files etc.

Accounting information

The time limits, account numbers, amount of CPU used, process numbers etc. are all a part of the PCB accounting information.

Location of the Process Control Block

The process control block is kept in a memory area that is protected from the normal user access. This is done because it contains important process information. Some of the operating systems place the PCB at the beginning of the kernel stack for the process as it is a safe location.

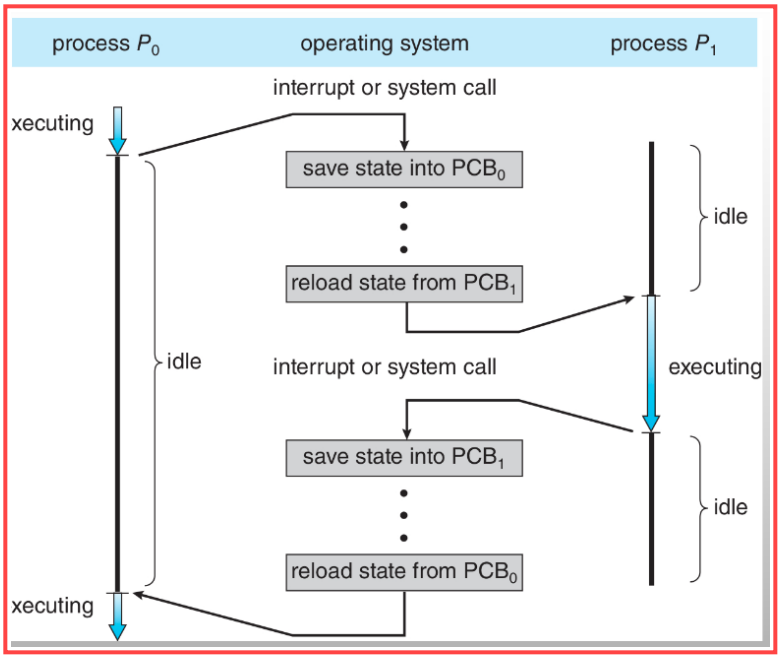
Question 2 **0 / 0 pts**

Describe how processes migrate between a variety of OS queues.

Your Answer:

Processes migrate between a variety of OS queues with context switches. When a context switch occurs , the kernel saves the context of the old

process in its PCB and loads the saved context of the new process scheduled to run.



Question 3

0 / 0 pts

Describe how the state of a process changes and under which circumstances (that is, describe the Process State Diagram as a Finite State Machine, or an FSM).

In what state process instructions are being actively executed by the CPU?

How many processes can be in running state?

Your Answer:

Throughout its life in the system, a process changes states as follows

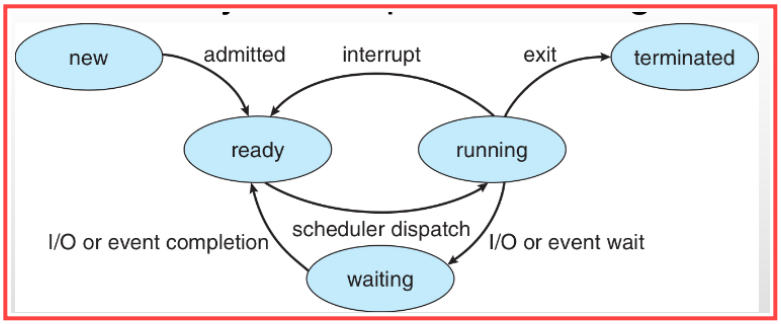
●new: The process is being created

●ready: The process is waiting to be assigned to a processor (CPU)

●running: Instructions are being executed

●waiting: The process is waiting for some event to occur

●terminated: The process has finished execution



Question 4

0 / 0 pts

Describe how process creation and termination is handled by operating systems.

Your Answer:

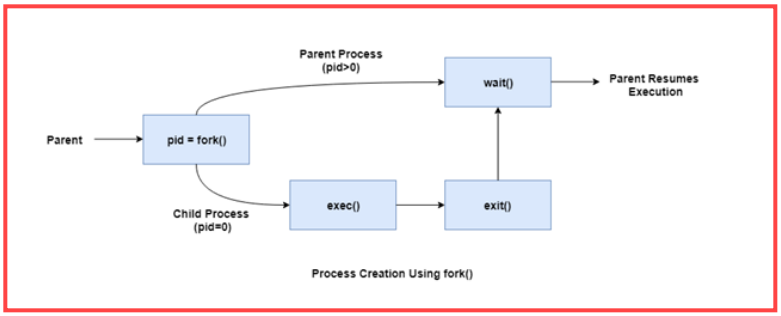
Process Creation

A process may be created in the system for different operations. Some of the events that lead to process creation are as follows −

- User request for process creation
- System Initialization
- Batch job initialization
- Execution of a process creation system call by a running process

A process may be created by another process using fork(). The creating process is called the parent process and the created process is the child process. A child process can have only one parent but a parent process may have many children. Both the parent and child processes have the same memory image, open files and environment strings. However, they

have distinct address spaces.



Process Termination

Process termination occurs when the process is terminated The exit() system call is used by most operating systems for process termination.

Some of the causes of process termination are as follows −

- A process may be terminated after its execution is naturally completed. This process leaves the processor and releases all its resources.
- A child process may be terminated if its parent process requests for its termination.
- A process can be terminated if it tries to use a resource that it is not allowed to. For example - A process can be terminated for trying to write into a read only file.
- If an I/O failure occurs for a process, it can be terminated. For example - If a process requires the printer and it is not working, then the process will be terminated.
- In most cases, if a parent process is terminated then its child processes are also terminated. This is done because the child process cannot exist without the parent process.
- If a process requires more memory than is currently available in the system, then it is terminated because of memory scarcity.

Question 5

0 / 0 pts

Describe short-term, medium-term, and long-term scheduling.
What are the differences between them?

Your Answer:

Process Scheduling handles the selection of a process for the processor on the basis of a scheduling algorithm and also the removal of a process from the processor. It is an important part of multiprogramming in operating system.

Process scheduling involves short-term scheduling, medium-term scheduling and long-term scheduling. Details about these are given as follows −

Long-Term Scheduling

- Long-term scheduling involves selecting the processes from the storage pool in the secondary memory and loading them into the ready queue in the main memory for execution. This is handled by the long-term scheduler or job scheduler.

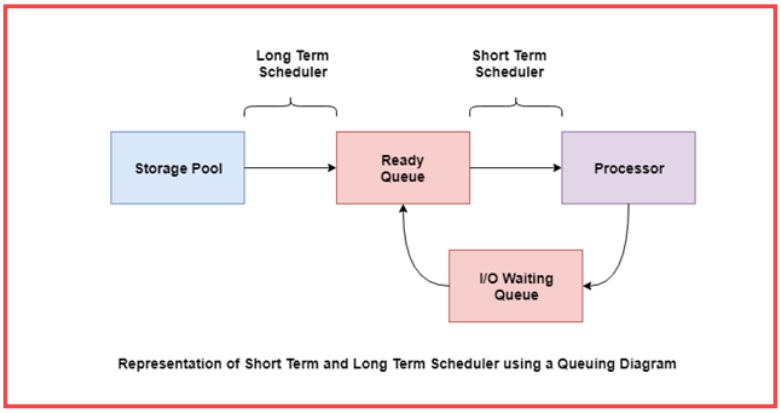
  The long-term scheduler controls the degree of multiprogramming. It must select a careful mixture of I/O bound and CPU bound processes to yield optimum system throughput. If it selects too many CPU bound processes then the I/O devices are idle and if it selects too many I/O bound processes then the processor has nothing to do.

Short-Term Scheduling

- Short-term scheduling involves selecting one of the processes from the ready queue and scheduling them for execution. This is done by the short-term scheduler. A scheduling algorithm is used to decide which process will be scheduled for execution next by the short-term scheduler.

  The short-term scheduler executes much more frequently than the long-term scheduler as a process may execute only for a few milliseconds.

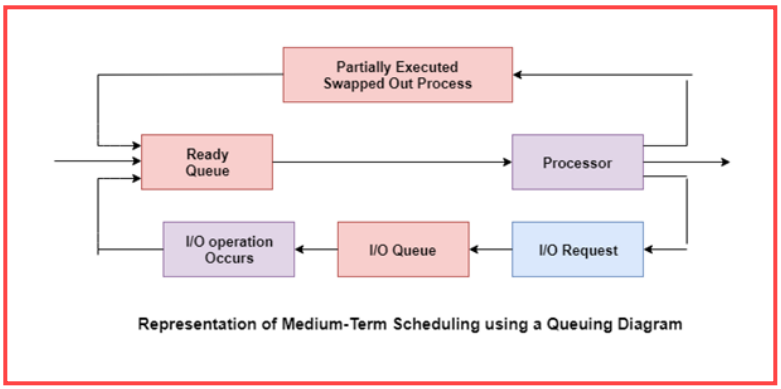
  The choices of the short term scheduler are very important. If it selects a process with a long burst time, then all the processes after that will have to wait for a long time in the ready queue. This is known as starvation and it may happen if a wrong decision is made by the short-term scheduler.



Representation of Short Term and Long Term Scheduler using a Queuing Diagram

- **Medium-Term Scheduling**
- Medium-term scheduling involves swapping out a process from main memory. The process can be swapped in later from the point it stopped executing. This can also be called as suspending and resuming the process and is done by the medium-term scheduler.

  This is helpful in reducing the degree of multiprogramming. Swapping is also useful to improve the mix of I/O bound and CPU bound processes in the memory.



Representation of Medium-Term Scheduling using a Queuing Diagram

Unanswered

Question 6

0 / 0 pts

Describe the actions taken by a kernel to context-switch between processes.

Your Answer:

Unanswered

Question 7

8 / 10 pts

I have submitted answers to all questions in this study set.

- True
- False

Quiz Score: **8** out of 10

Study Set for Lecture 04: Inter-Process Communication

Due Sep 21 at 11:05pm **Points** 10 **Questions** 7
Available Sep 15 at 12pm - Dec 8 at 8pm 3 months **Time Limit** None
Allowed Attempts Unlimited

Instructions

Download **lect04code.zip** that contains the code shown in the class, unzip it, build and run all examples.

Review lecture notes from **lect04_IPC.pdf**.

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

Take the Quiz Again

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	2,636 minutes	10 out of 10

Correct answers are hidden.

Score for this attempt: **10** out of 10
Submitted Sep 17 at 10:12am
This attempt took 2,636 minutes.

Question 1 — 0 / 0 pts

What are the main mechanisms for IPC (Inter-Process Communication)? Describe each of them.

Your Answer:

Inter process communication (IPC) is a mechanism which allows processes to communicate with each other and synchronize their actions. The communication between these processes can be seen as a method of co-operation between them. Processes can communicate with each other through both:

1. Shared Memory
2. Message passing

These are the methods in IPC:

1. **Pipes (Same Process) –**
   This allows flow of data in one direction only. Analogous to simplex systems (Keyboard). Data from the output is usually buffered until input process receives it which must have a common origin.
2. **Names Pipes (Different Processes) –**
   This is a pipe with a specific name it can be used in processes that don't have a shared common process origin. E.g. is FIFO where the details written to a pipe is first named.
3. **Message Queuing –**
   This allows messages to be passed between processes using either a single queue or several message queue. This is managed by system kernel these messages are coordinated using an API.
4. **Semaphores –**
   This is used in solving problems associated with synchronization and to avoid race condition. These are integer values which are greater than or equal to 0.
5. **Shared memory –**
   This allows the interchange of data through a defined area of memory. Semaphore values have to be obtained before data can get access to shared memory.
6. **Sockets –**
   This method is mostly used to communicate over a network between a client and a server. It allows for a standard connection which is computer and OS independent.

Question 2

0 / 0 pts

List at least four common uses of IPC (Inter-Process Communication).

Describe each of them.

Your Answer:

Pipes

Pipe is widely used for communication between two related processes. This is a half-duplex method, so the first process communicates with the second process. However, in order to achieve a full-duplex, another pipe is needed.

Message Passing:

It is a mechanism for a process to communicate and synchronize. Using message passing, the process communicates with each other without resorting to shared variables.

IPC mechanism provides two operations:

- Send (message)- message size fixed or variable
- Received (message)

Message Queues:

A message queue is a linked list of messages stored within the kernel. It is identified by a message queue identifier. This method offers communication between single or multiple processes with full-duplex capacity.

Direct Communication:

In this type of inter-process communication process, should name each other explicitly. In this method, a link is established between one pair of communicating processes, and between each pair, only one link exists.

Indirect Communication:

Indirect communication establishes like only when processes share a common mailbox each pair of processes sharing several communication links. A link can communicate with many processes. The link may be bi-directional or unidirectional.

Shared Memory:

Shared memory is a memory shared between two or more processes that are established using shared memory between all the processes. This type of memory requires to protected from each other by synchronizing access across all the processes.

FIFO:

Communication between two unrelated processes. It is a full-duplex method, which means that the first process can communicate with the second process, and the opposite can also happen.

Question 3

0 / 0 pts

Describe producer-consumer paradigm explaining the difference between using bounded and unbounded buffers in the communication link.

Your Answer:

Bounded Capacity: The queue has finite length n; thus, at most n message can reside in it. If the queue is not full when the message is sent,, the message is placed in the queue, and the sender can continue executon without waiting.
unbound capacity: the queue's length is potentially infinite; thus, any number of messages can wait in it. The sender never blocks.

In a producer-consumer paradigm, we get that with these two, bounded capacity is better for a instant messaging style, while the unbounded capacity is much better in an email type environment since the instant messaging, people generally have short conversations or the conversations dont require many words to fit in the box.

Question 4

0 / 0 pts

Describe direct and indirect Interprocess Communication.

Your Answer:

Direct Communication: Each process that wants to communicate must explicitly name the recipient or sender of the communcation.
Indirect Interprocess Communication: The messages are sent to and received from mailboxes, or ports.

In a sense, instant messaging vs email.

Question 5

0 / 0 pts

Describe blocking and non-blocking communication. What is a rendezvous?

Your Answer:

Blocking communication is done using `MPI_Send()` (https://www.mpich.org/static/docs/v3.2/www3/MPI_Send.html) and `MPI_Recv()` (https://www.mpich.org/static/docs/v3.2/www3/MPI_Recv.html). These functions do not return (i.e., they block) until the communication is finished. Simplifying somewhat, this means that the buffer passed to `MPI_Send()` can be reused, either because MPI saved it somewhere, or because it has been received by the destination.
Similarly, `MPI_Recv()` returns when the receive buffer has been filled with valid data.

In contrast, non-blocking communication is done using `MPI_Isend()` (https://www.mpich.org/static/docs/v3.2/www3/MPI_Isend.html) and `MPI_Irecv()` (https://www.mpich.org/static/docs/v3.2/www3/MPI_Irecv.html). These function return immediately (i.e., they do not block) even if the communication is not finished yet. You must call `MPI_Wait()` (https://www.mpich.org/static/docs/v3.2/www3/MPI_Wait.html) or `MPI_Test()` (https://www.mpich.org/static/docs/v3.2/www3/MPI_Test.html) to see whether the communication has finished.

Blocking communication is used when it is sufficient, since it is somewhat easier to use. Non-blocking communication is used when necessary, for example, you may call `MPI_Isend()`, do some computations, then do `MPI_Wait()`. This allows computations and communication to overlap, which generally leads to improved performance.

Question 6

0 / 0 pts

Let's a flags `byte` be a collection of 8 one-bit flags.

Explain with details how a one-bit flag that is located in the 5th bit from the right can be set and cleared in the `byte` byte.

Note that the bits are counted from 0.

Your Answer:

Setting

n = 01001001

left shift 5 = 00100000

OR

n | (left shift 5) = 01001001

00100000

= **01101001**

set the 5th bit from the right.

Clearing

So say we start with a collection of 8 one-bit flags.

n = 01101001

<<

(left shift 5) = 00100000

~(left shift 5) = 11011111

AND

n & ~(left shift 5) = 01101001

11011111

= **01001001**

cleared the 5th bit from right.

Question 7

10 / 10 pts

I have submitted answers to all questions in this study set.

- (•) True
- ( ) False

Quiz Score: **10** out of 10

Study Set for Lecture 05: Threads

Due Sep 28 at 11:59pm **Points** 10 **Questions** 8
Available Sep 22 at 12pm - Dec 8 at 8pm 3 months **Time Limit** None
Allowed Attempts Unlimited

Instructions

Here are the examples from the lecture: **lect05code.zip**. Download the file, unzip, and then compile and experiment with the examples.

Review lecture notes from **lect05_Threads.pdf**.

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

Take the Quiz Again

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	47 minutes	10 out of 10

ⓘ Correct answers are hidden.

Score for this attempt: **10** out of 10
Submitted Sep 22 at 2:44pm
This attempt took 47 minutes.

Question 1 — 0 / 0 pts

Describe with details the differences between a process and a thread? Does every process have a thread? What's the minimum and maximum threads per process?
Can a thread span multiple processes?

Your Answer:

Process
Each process provides the resources needed to **execute** a program. A process has a virtual address space, executable code, open handles to system objects, a security context, a unique process identifier, environment variables, a priority class, minimum and maximum working set sizes, and at least one thread of execution. Each process is started with a single thread, often called the primary thread, but can create additional threads from any of its threads.

Thread
A thread is an entity within a process that can be **scheduled for execution**. All threads of a process share its virtual address space and system resources. In addition, each thread maintains exception handlers, a scheduling priority, thread local storage, a unique thread identifier, and a set of structures the system will use to save the thread context until it is scheduled. The thread context includes the thread's set of machine registers, the kernel stack, a thread environment block, and a user stack in the address space of the thread's process. Threads can also have their own security context, which can be used for impersonating clients.

Does every process have a thread? What's the minimum and maximum threads per process?

Threads exist within a **process** — **every process has** at least one. **Threads** share the **process's** resources, including memory and open files.

Can a thread span multiple processes?

Multiple threads can exist within one **process**, executing concurrently and sharing resources such as memory, while different **processes** do not share these resources.

You'd prefer multiple threads over multiple processes for two reasons:

1. Inter-thread communication (sharing data etc.) is significantly simpler to program than inter-process communication.
2. Context switches between threads are faster than between processes. That is, it's quicker for the OS to stop one thread and start running another than do the same with two processes.

Example:

Applications with GUIs typically use one thread for the GUI and others for background computation. The spellchecker in MS Office, for example, is a separate thread from the one running the Office user interface. In such applications, using multiple processes instead would result in slower performance and code that's tough to write and maintain.

Question 2

0 / 0 pts

Why use threads?

Your Answer:

Light Weight:

- When compared to the cost of creating and managing a process, a thread can be created with much less operating system overhead. Managing threads requires fewer system resources than managing processes.

- For example, the following table compares timing results for the **`fork()`** subroutine and the **`pthread_create()`** subroutine. Timings reflect 50,000 process/thread creations, were performed with the `time` utility, and units are in seconds, no optimization flags.

Platform	fork()			pthread_create()		
	real	user	sys	real	user	sys
Intel 2.6 GHz Xeon E5-2670 (16 cores/node)	8.1	0.1	2.9	0.9	0.2	0.3
Intel 2.8 GHz Xeon 5660 (12 cores/node)	4.4	0.4	4.3	0.7	0.2	0.5
AMD 2.3 GHz Opteron (16 cores/node)	12.5	1.0	12.5	1.2	0.2	1.3
AMD 2.4 GHz Opteron (8 cores/node)	17.6	2.2	15.7	1.4	0.3	1.3
IBM 4.0 GHz POWER6 (8 cpus/node)	9.5	0.6	8.8	1.6	0.1	0.4
IBM 1.9 GHz POWER5 p5-575 (8 cpus/node)	64.2	30.7	27.6	1.7	0.6	1.1
IBM 1.5 GHz POWER4 (8 cpus/node)	104.5	48.6	47.2	2.1	1.0	1.5
INTEL 2.4 GHz Xeon (2 cpus/node)	54.9	1.5	20.8	1.6	0.7	0.9
INTEL 1.4 GHz Itanium2 (4 cpus/node)	54.5	1.1	22.2	2.0	1.2	0.6

Efficient Communications/Data Exchange:

- The primary motivation for considering the use of Pthreads in a high performance computing environment is to achieve optimum performance. In particular, if an application is using MPI for on-node communications, there is a potential that performance could be improved by using Pthreads instead.

- MPI libraries usually implement on-node task communication via shared memory, which involves at least one memory copy operation (process to process).

- For Pthreads there is no intermediate memory copy required because threads share the same address space within a single process. There is no data transfer, per se. It can be as efficient as simply passing a pointer.

- In the worst case scenario, Pthread communications become more of a cache-to-CPU or memory-to-CPU bandwidth issue. These speeds are much higher than MPI shared memory communications.

- For example: some local comparisons, past and present, are shown below

Platform	MPI Shared Memory Bandwidth (GB/sec)	Pthreads Worst Case Memory-to-CPU Bandwidth (GB/sec)
Intel 2.6 GHz Xeon E5-2670	4.5	51.2
Intel 2.8 GHz Xeon 5660	5.6	32
AMD 2.3 GHz Opteron	1.8	5.3
AMD 2.4 GHz Opteron	1.2	5.3
IBM 1.9 GHz POWER5 p5-575	4.1	16
IBM 1.5 GHz POWER4	2.1	4
Intel 2.4 GHz Xeon	0.3	4.3
Intel 1.4 GHz Itanium 2	1.8	6.4

Other Common Reasons:

- Threaded applications offer potential performance gains and practical advantages over non-threaded applications in several other ways:
  - Overlapping CPU work with I/O: For example, a program may have sections where it is performing a long I/O operation. While one thread is waiting for an I/O system call to complete, CPU intensive work can be performed by other threads.
  - Priority/real-time scheduling: tasks which are more important can be scheduled to supersede or interrupt lower priority tasks.
  - Asynchronous event handling: tasks which service events of indeterminate frequency and duration can be interleaved. For example, a web server can both transfer data from previous requests and manage the arrival of new requests.

- A perfect example is the typical web browser, where many interleaved tasks can be happening at the same time, and where tasks can vary in priority.

Question 3

0 / 0 pts

Which of the following components of program state are shared across threads in a multithreaded process, and which ones are not?

- Register values

- Heap memory (dynamic allocation)
- Global variables
- Stack memory(call stack)

Explain with details why some elements on the list are shared while others cannot be shared.

Your Answer:

Multi threaded process share :

Heap memory

Global variables

Each thread has separate set of :

Register values

Stack memory

Question 4 **0 / 0 pts**

What are the differences between user-level threads and kernel-level threads?
Discuss the ways in which a user thread library can be implemented using kernel threads?

Your Answer:

User - Level Threads

The user-level threads are implemented by users and the kernel is not aware of the existence of these threads. It handles them as if they were single-threaded processes. User-level threads are small and much faster than kernel level threads. They are represented by a program counter(PC), stack, registers and a small process control block. Also, there is no kernel involvement in synchronization for user-level threads.

Advantages of User-Level Threads

Some of the advantages of user-level threads are as follows −

- User-level threads are easier and faster to create than kernel-level threads. They can also be more easily managed.
- User-level threads can be run on any operating system.
- There are no kernel mode privileges required for thread switching in user-level threads.

Disadvantages of User-Level Threads

Some of the disadvantages of user-level threads are as follows −

- Multithreaded applications in user-level threads cannot use multiprocessing to their advantage.
- The entire process is blocked if one user-level thread performs blocking operation.

Kernel-Level Threads

Kernel-level threads are handled by the operating system directly and the thread management is done by the kernel. The context information for the process as well as the process threads is all managed by the kernel. Because of this, kernel-level threads are slower than user-level threads.

Advantages of Kernel-Level Threads

Some of the advantages of kernel-level threads are as follows −

- Multiple threads of the same process can be scheduled on different processors in kernel-level threads.
- The kernel routines can also be multithreaded.
- If a kernel-level thread is blocked, another thread of the same process can be scheduled by the kernel.

Disadvantages of Kernel-Level Threads

Some of the disadvantages of kernel-level threads are as follows −

- A mode switch to kernel mode is required to transfer control from one thread to another in a process.
- Kernel-level threads are slower to create as well as manage as compared to user-level threads.

User threads are implemented with kernel threads by mapping user threads to kernel threads using one of the multi-threaded models:

- many-to-one
- one-to-one
- many-to-many

Question 5

0 / 0 pts

What is a signal? Discuss the issues with signals targeted at multithreaded processes. What options do operating system designers have? How is the problem solved by POSIX?

Your Answer:

Signals are software interrupts sent to a program to indicate that an important event has occurred. The events can vary from user requests to illegal memory access errors. Some signals, such as the interrupt signal, indicate that a user has asked the program to do something that is not in the usual flow of control.

Signals are similar to **interrupts** (https://en.wikipedia.org/wiki/Interrupt), the difference being that interrupts are mediated by the processor and handled by the **kernel** (https://en.wikipedia.org/wiki/Kernel_(operating_system)) while signals are mediated by the kernel (possibly via system calls) and handled by processes. The kernel may pass an interrupt as a signal to the process that caused it (typical examples are **SIGSEGV** (https://en.wikipedia.org/wiki/SIGSEGV), **SIGBUS** (https://en.wikipedia.org/wiki/SIGBUS), **SIGILL** (https://en.wikipedia.org/wiki/Signal_(IPC)#SIGILL) and **SIGFPE** (https://en.wikipedia.org/wiki/Signal_(IPC)#SIGFPE)).

There is a issue when mixing signals and threads. Operating system designers have avoided this by mask signals before the threads are created.

In POSIX, signals target process rather than a thread. aarbitrary thread is selected to handle the signal.

Question 6

0 / 0 pts

What will happen if a Linux multi-threaded process is cloned with fork()?

What will happen if a Linux multi-threaded process morphs into another program using one of the exec() functions?

Your Answer:

The **fork( )** system call creates an exact duplicate of the address space from which it is called, resulting in two address spaces executing the same code. Problems can occur if the forking address space has multiple threads executing at the time of the **fork( )**. When multithreading is a result of library invocation, threads are not necessarily aware of each other's presence, purpose, actions, and so on. Suppose that one of the other threads (any thread other than the one doing the **fork( )**) has the job of deducting money from your checking account. Clearly, you do not want this to happen twice as a result of some other thread's decision to call **fork( )**.

One solution to the problem of calling **fork( )** in a multithreaded environment exists. (Note that this method will not work for server application code or any other application code that is invoked by a callback from a library.) Before an application performs a **fork( )** followed by something other than **exec( )**, it must cancel all of the other threads. After it joins the canceled threads, it can safely **fork( )** because it is the only thread in existence. This means that libraries that create threads must establish cancel handlers that propagate the cancel to the created threads and join them. The application should save enough state so that the threads can be recreated and restarted after the **fork( )** processing completes.

Does fork() duplicate only the calling thread or all threads?

- some systems have two versions of fork()

● in Linux, only the calling thread is cloned

● i.e., the process context with the program counter, stack, and registers specific to the executing thread

What about exec()?

● usually, it replaces whole process

● including overwriting the threads

● the new process will have only one (main) thread

Question 7

0 / 0 pts

Explain what a deferred thread cancellation is and how it enhances program reliability.

Your Answer:

Terminating a thread before it has completed is called **Thread cancellation**. For an example, if multiple threads are concurrently searching through a database and one thread returns the result, the remaining threads might be canceled. Another situation might be occurred when a user presses a button on a web browser that stops a web page from loading any further. Often, using several threads a web page loads — each image is loaded in a separate thread. When the stop button is pressed by a user on the browser, all threads loading the page are canceled. A thread which is to be cancelled is often referred to as the target thread. Cancellation of a target thread may occur in two different cases −

- **Asynchronous cancellation** − One thread terminates immediately the target thread.
- **Deferred cancellation** − The target thread checks periodically whether it should terminate, allowing it an opportunity to terminate itself in an orderly fashion.

The definition above is the main reason why it enhances program reliability

Question 8

10 / 10 pts

I have submitted answers to all questions in this study set.

- (selected) True
- False

Quiz Score: **10** out of 10

Study Set for Lecture 06: CPU Scheduling

Due Oct 5 at 11:59pm **Points** 10 **Questions** 15
Available Sep 29 at 12pm - Dec 8 at 8pm 2 months **Time Limit** None
Allowed Attempts Unlimited

Instructions

Review lecture notes from **lect06_CPU_Scheduling.pdf**. The archive of the accompanying code is in **lect06code.zip**.

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

Take the Quiz Again

Attempt History

	Attempt	Time	Score
KEPT	Attempt 2	99 minutes	10 out of 10
LATEST	Attempt 2	99 minutes	10 out of 10
	Attempt 1	1,586 minutes	0 out of 10 *

* Some questions not yet graded

ⓘ Correct answers are hidden.

Score for this attempt: **10** out of 10
Submitted Oct 4 at 7:25pm
This attempt took 99 minutes.

Question 1 — 0 / 0 pts

Explain what a CPU burst is and how it is used by CPU scheduler.

Next, compute a series of predictions for CPU bursts for a process with the following sequence of CPU bursts:

```
10, 4, 6, 2, 10, 8, 2, 8, 10, 4, 6
```

Assume alpha = 0.75
Start with tau_0 = 8

Show all intermediate details of your computation.

NOTE:

Assume that you are receiving each of the burst values in the sequence one at a time.

So, the first value estimate is computed as:

```
tau_1 = 0.75 * 10 + (1 - 0.75) * 8
```

Your Answer:

A CPU burst is the time that a process has control of the CPU, and the time is how long it has the CPU to complete. The scheduler uses the burst to determine which process should go when, depending on what scheduling it's using.

tau(n+1) - predicted value for the (n+1)th CPU burst

t(n) - actual length of n-th CPU burst

tau(n) - previous average

0 <= alpha <= 1 average coefficient

FORMULA:

taun(n+1) = (alpha)(t(n)) + (1-(alpha))(tau(n))

tau1 = (.75*10) + ((.25)8) = 7.5 + 2 = 9.5

tau2 = (.75*4) + ((.25)9.5) = 3 + 2.375 = 5.375

tau3 = (.75*6) + ((.25)5.375) = 4.5 + 1.34375 = 5.84375

tau4 = (.75*2) + ((.25)5.84375) = 1.5 + 1.4609375 = 2.9609375

tau5 = (.75*10) + ((.25)2.9609375) = 7.5 + 0.740234375 = 8.240234375

tau6 = (.75*8) + ((.25)8.240234375) = 6 + 2.060058594 = 8.060058594

tau7 = (.75*2) + ((.25)8.060058594) = 1.5 + 2.015014648 = 3.515014648

tau8 = (.75*8) + ((.25)3.515014648) = 6 + 0.878753662 = 6.878753662

tau9 = (.75*10) + ((.25)6.878753662) = 7.5 + 1.719688416 = 9.219688416

tau10 = (.75*4) + ((.25)9.219688416) = 3 + 2.304922104 = 5.304922104

tau11 = (.75*6) + ((.25)5.304922104) = 4.5 + 1.326230526 = 5.826230526

Question 2

0 / 0 pts

Prove that in exponential averaging the further a value is in the past, the less impact it has on the average value.

Your Answer:

In exponential averaging the further the value the less impact it has since both alpha and (1-alpha) are less than or equal to 1, therefore each successive term has less weight than its predecessor, because each one divides to be smaller and smaller.

Question 3

0 / 0 pts

List and describe scheduling criteria for scheduling algorithms.

Your Answer:

Different **CPU scheduling algorithms (https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/)**

have different properties and choice of a particular algorithm depends on the various factors. Many criteria have been suggested for comparing CPU scheduling algorithms.

The criteria include the following:

1. **CPU utilisation –**
   The main objective of any CPU scheduling algorithm is to keep the CPU as busy as possible. Theoretically CPU utilisation can range from 0 to 100 but in a real time system it varies from 40 to 90 percent depending on the load upon the system.
2. **Throughout –**
   A measure of the work done by CPU is the number of processes being executed and completed per unit time. This is called throughput. The throughput may vary depending upon the length or duration of processes.
3. **Turnaround time –**
   For a particular process, an important criteria is how long it takes to execute that process. The time elapsed from the time of submission of a process to the time of completion is known as turnaround time. Turn-around time is the sum of times spent waiting to get into memory, waiting in ready queue, executing in CPU and waiting for I/O.
4. **Waiting time –**
   A scheduling algorithm does not affect the time required to complete the process once it starts execution. It only affects the waiting time of a process i.e. time spent by a process waiting in the ready queue.
5. **Response time –**
   In an interactive system turn-around time is not the best criteria. A process may produce some output fairly early and continue computing new results while previous results are being output to user. Thus another criteria is the time taken from submission of process of request until the first response is produced. This measure is called response time.

There are various CPU Scheduling algorithms such as-

- **First Come First Served (FCFS) (https://www.google.com/amp/s/www.geeksforgeeks.org/program-for-fcfs-cpu-scheduling-set-1/amp/)**
- **Shortest Job First (SJF) (https://www.google.com/amp/s/www.geeksforgeeks.org/program-for-shortest-job-first-or-sjf-cpu-scheduling-set-1-non-preemptive/amp/)**

- Longest Job First (LJF) (https://www.google.com/amp/s/www.geeksforgeeks.org/longest-job-first-ljf-cpu-scheduling-algorithm/amp/)
- Priority Scheduling (https://www.google.com/amp/s/www.geeksforgeeks.org/program-for-priority-cpu-scheduling-set-1/amp/)
- Round Robin (RR) (https://www.google.com/amp/s/www.geeksforgeeks.org/program-round-robin-scheduling-set-1/amp/)
- Shortest Remaining Time First (SRTF) (https://www.google.com/amp/s/www.geeksforgeeks.org/introduction-of-shortest-remaining-time-first-srtf-algorithm/amp/)
- Longest Remaining Time First (LRTF) (https://www.google.com/amp/s/www.geeksforgeeks.org/longest-remaining-time-first-lrtf-cpu-scheduling-algorithm/amp/)

Question 4

0 / 0 pts

Evaluate CPU scheduling algorithm based entirely on process priorities.

Are there any potential pitfalls? If yes, how can the algorithm be changed to address them?

Your Answer:

The pitfall of scheduling based entirely on priority is that low priority processes may never execute.

The solution is to use "aging". This means as time progresses the priority of the process increases.

Each process is given a priority number (int) which corresponds to how how important it is to the cpu. The lower the number means the higher the priority.

Question 5

0 / 0 pts

Describe with details the algorithm of using a roulette wheel algorithm to select processes from a number of queues with different priorities.

Your Answer:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main() {
srandom(time(NULL));
double r = ((double)random())/RAND_MAX;
printf("r = %lf\n", r);
if (r < 0.1)
printf("Hit 1st slice\n");
else if (r < 0.25)
printf("Hit 2nd slice\n");
else if (r < 0.5)
printf("Hit 3rd slice\n");
else if (r < 0.65)
printf("Hit 4th slice\n");
else if (r < 0.85)
printf("Hit 5th slice\n");
else
printf("Hit 6th slice\n");
}
```

This example above will split the percentages in slices (like a wheel). This will help us change the probability of each slice that will be hit when we spin the wheel or throw a dart etc.

Question 6

0 / 0 pts

List and describe with details at least three approaches to selecting processes from priority-based multi-level queues.

Your Answer:

- There are several different criteria to consider when trying to select the "best" scheduling algorithm for a particular situation and environment, including:
  - **CPU utilization** - Ideally the CPU would be busy 100% of the time, so as to waste 0 CPU cycles. On a real system CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)
  - **Throughput** - Number of processes completed per unit time. May range from 10 / second to 1 / hour depending on the specific processes.
  - **Turnaround time** - Time required for a particular process to complete, from submission time to completion. (Wall clock time.)
  - **Waiting time** - How much time processes spend in the ready queue waiting their turn to get on the CPU.
    - (**Load average** - The average number of processes sitting in the ready queue waiting their turn to get into the CPU. Reported in 1-minute, 5-minute, and 15-minute averages by "uptime" and "who".)
  - **Response time** - The time taken in an interactive program from the issuance of a command to the ***commence*** of a response to that command.
- In general one wants to optimize the average value of a criteria (Maximize CPU utilization and throughput, and minimize all the others.) However some times one wants to do something different, such as to minimize the maximum response time.
- Sometimes it is most desirable to minimize the ***variance*** of a criteria than the actual value. I.e. users are more accepting of a consistent predictable system than an inconsistent one, even if it is a little bit slower.

Question 7

0 / 0 pts

Design a CPU scheduler that supports "real-time" (RT) and "regular" (OTHER) processes. The RT processes should be preferred over OTHER.

Explain all details of the architecture and the algorithms that will achieve that objective.

Your Answer:

Real-time (“soft” R-T)
- FIFO, RR, DEADLINE (experimental)
- FIFO – no time slice
- RR – time slice
- DEADLINE – complicated; see man page
- highest priority process always runs first
- “r-t” processes assigned static priorities 1 to 99

Question 8

0 / 0 pts

Design and describe with details a priority-based scheduling algorithm that uses a 4-level feedback queue.

The algorithm should address process entrance (i.e., the starting queue), process promotion and demotion, queue selection, and process selection from the selected queue.

Your Answer:

In **computer science** (https://en.wikipedia.org/wiki/Computer_science), a **multilevel feedback queue** is a **scheduling** (https://en.wikipedia.org/wiki/Scheduling_(computing)) algorithm. Solaris 2.6 Time-Sharing (TS) scheduler implements this algorithm.[1] (https://en.wikipedia.org/wiki/Multilevel_feedback_queue#cite_note-1) The MacOS and Microsoft Windows schedulers can both be regarded as examples of the broader class of multilevel feedback queue schedulers.[2] (https://en.wikipedia.org/wiki/Multilevel_feedback_queue#cite_note-2) This scheduling algorithm is intended to meet the following design requirements for **multimode systems** (https://en.wikipedia.org/w/index.php?title=Multimode_systems&action=edit&redlink=1):

1. Give preference to short jobs.
2. Give preference to **I/O bound** (https://en.wikipedia.org/wiki/I/O_bound) processes.

3. Separate processes into categories based on their need for the processor.

Three queues:

● Q0 – time quantum 8 ms

● Q1 – time quantum 16 ms

● Q2 – anything longer than 24 ms

Scheduling:

● Processes in Q2 considered only if Q0 and Q1 empty

● Processes in Q1 considered only if Q0 empty ● A new job enters queue Q0 which is served FCFS. When it gains CPU, job receives 8 milliseconds. If it does not finish in 8 milliseconds, job is moved to queue Q1.

● At Q1 job is again scheduled FCFS and receives 16 additional milliseconds.

● Only if Q0 is empty!

● If it still does not complete, it is preempted and moved to queue Q2.

Question 9

0 / 0 pts

Consider the following task workload pattern:

```
<Process> <CPU Burst Time>
   P1            7
   P2           10
   P3            3
   P4            8
   P5           15
   P6            2
   P7            4
```

Assuming the FCFS scheduling policy, compute the average waiting time. Use the following format to illustrate the details of your computation:

```
T01: P1(3)
T04: P3(6)
T10: P6(5)
T15: P4(...)
...
```

Your Answer:

T01: p1(7)

T7: P2(10)

T17: P3(3)

T20: P4(8)

T28: P5(15)

T43: P6(2)

T45: P7(4)

T49: done

Question 10

0 / 0 pts

Consider the following task workload pattern:

```
<Process><CPU Burst Time><Arrival Time>
   P1            7              1
   P2           10              3
   P3            1              5
   P4            8             11
   P5           15             14
   P6            2             15
   P7            2             21
```

Assuming the SRTF scheduling policy, compute the average waiting time. Use the following format to illustrate the details of your computation:

```
T01: P1(3)
T04: P3(6)
T10: P6(5)
T15: P4(...)
...
```

Your Answer:

T01: P1(7)

T03: P1(5) Q: P2(10)

T05: P3(1) Q: P1(3) P2(10)

T06: P1(3) Q: P2(10)

T09: P2(10)

T11: P2(8) Q: P4(8)

T14: P2(5) Q: P4(8) P5(15)

T15: P6(2) Q: P2(4) P4(8) P5(15)

T17: P2(4) Q: P4(8) P5(15)

T21: P7(2) Q: P4(8) P5(15)

T23: P4(8) Q:P5(15)

T31: P5(15)

T46: done

Shortest-Job-First (SJF) Scheduling

● Associate with each process the length of its next CPU burst.

● Use CPU bursts to schedule the process with the shortest time

Two schemes:

non-preemptive -

● once CPU given to the process it cannot be interrupted until completes its CPU burst

preemptive-

● if a new process arrives with CPU burst length less than remaining time of current executing process, preempt

● this scheme is know as the Shortest-Remaining-Time-First (SRTF)

Question 11

0 / 0 pts

Assume that the a scheduler uses a round robin scheme with a quantum time of 3.

Assuming that only the following processes compete for CPU, compute the average turnaround time for time quanta 1, 2, 3, 4, 5, and 6:

```
<Process><CPU Burst Time>
   P1            6
   P2            3
   P3            1
   P4            7
```

1. Show the detailed timeline of scheduling events for each time quantum value.
2. Show an ordered list of the average turnaround times along with their corresponding time quanta.

Your Answer:

T0 : idle

T01: P1(6)

T02 : P1(5) Q: P2(3)

T03: P1(4) Q: P2(3) P3(1)

T04: P2(3) Q: P3(1) P4(7) P1(3)

T05: P2(2) Q: P3(1) P4(7) P1(3)

T06: P2(1) Q: P3(1) P4(7) P1(3)

T07: P3(1) Q: P4(7) P1(3)

T08: P4(7) Q:P1(3)

T09: P4(6) Q:P1(3)

T010: P4(5) Q:P1(3)

T011: P1(3) Q:P4(4)

T012: P1(2) Q:P4(4)

T013: P1(1) Q:P4(4)

T014: P4(4)

T015: P4(3)

T016: P4(2)

T017: P4(1)

T018: done

Each process gets a small unit of CPU time (time quantum)

● usually 10-100 milliseconds

● After this time has elapsed, the process is preempted (interrupted) and added to the end of the ready queue.

● If there are n processes in the ready queue and the time quantum is q, then each process gets 1/n of the CPU time in chunks of at most q time units at once. No process waits more than (n - 1)q time units.

Performance:

● q large ⇒ FIFO

● q small ⇒ better system response

- however, q must be still relatively large with respect to context switch (otherwise overhead is too high) -- see next slides

● There might be another algorithm used for selection of the process that will execute in the next time quantum

- the ready queue can be a FIFO or a priority queue

Question 12

0 / 0 pts

Consider the following task workload pattern:

```
<Process><CPU Burst Time><Arrival Time>
   P1            7              1
   P2           10              3
   P3            1              5
   P4            8             11
   P5           15             14
   P6            2             15
   P7            2             21
```

Assuming the SJF scheduling policy, compute the average waiting time. Use the following format to illustrate the details of your computation:

```
T01: P1(3)
T04: P3(6)
T10: P6(5)
T15: P4(...)
...
```

Your Answer:

T01: P1(7)

T03: P1(5) Q: P2(10)

T05: P1(3) Q: P2(10) P3(1)

T08: P3(1) Q: P2(10)

T09: P2(10)

T11: P2(8) Q: P4(8)

T14: P2(5) Q: P4(8) P5(15)

T15: P2(4) Q:P4(8) P5(15) P6(2)

T19: P6(2) Q:P4(8) P5(15)

T21: P7(2) Q:P4(8) P5(15)

T23: P4(8) Q:P5(15)

T33: P5(15)

T48: done

Question 13

0 / 0 pts

Explain how Completely Fair Queueing (CFQ) algorithm is used to implement Completely Fair Scheduler (CFS).

Your Answer:

CFQ is one of the input/output scheduler for the Linux kernel and is the current default scheduler in the Linux kernel.

What is kernel ?
Kernel is the central part of an operating system. It manages the operation between the hardware and the software. Every operating system has a kernel, for example the Linux kernel.

Completely Fair Queuing (CFQ) scheduler:
The Completely Fair Queuing (CFQ) scheduler is the I/O scheduler. The CFQ scheduler maintains a scalable per-process requests submitted by the processes into the number of per-process I/O queue and then allocate time for each of the queues to access the resource. This is done on the basis of the I/O priority of the given process.
In case of asynchronous requests, all the requests from all the processes are batched together into fewer queues according to their process's I/O priority.

The CFQ scheduler divides the processes into 3 separates class:

1. Real time (highest priority)
2. Best effort
3. Idle (lowest priority)

The real-time and best-effort scheduling classes are further subdivided into eight-eight I/O priorities.These priorities are 0 to 7. Zero(0) being the highest and 7 the lowest and 4 is the default priority within the class.

Process in the real time class are always performed before processes in best effort class, which is always performed before processes in the idle class. This means that due to the higher priority of the real time class, rest both class have to starve of the processor time. Processes are assigned to the best effort class by default.

When there is no other I/O pending in the system then the idle scheduling class are only serviced. Thus, it is very important to only set the I/O scheduling class of a process to idle if I/O from the process is not at all required for making further progress.

Question 14

0 / 0 pts

Describe and compare approaches to evaluate scheduling algorithms.

Your Answer:

Algorithm Evaluation

● Deterministic modeling

- takes a particular predetermined workload and defines the performance of each algorithm for that workload

● Queueing models

- Queueing Theory - Mathematical analytical models using probabilistic distributions of job arrivals, CPU bursts, etc.

● Simulation

- simulators using probabilistic distributions or trace tapes (captures of real data)

● Implementation

- measuring performance of a live system

Question 15

10 / 10 pts

I have submitted answers to all questions in this study set.

- ◉ True

False

Quiz Score: **10** out of 10

Study Set for Lecture 07: Process Synchronization

Due Oct 12 at 11:59pm **Points** 10 **Questions** 13
Available Oct 6 at 12pm - Dec 8 at 8pm 2 months **Time Limit** None
Allowed Attempts Unlimited

Instructions

Review lecture notes from **lect07_Process_Synchronization.pdf**. The archive of the accompanying code is in **lect07code.zip**.

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

Take the Quiz Again

Attempt History

	Attempt	Time	Score
KEPT	Attempt 2	25 minutes	0 out of 10
LATEST	Attempt 2	25 minutes	0 out of 10
	Attempt 1	2,867 minutes	0 out of 10 *

* Some questions not yet graded

ⓘ Correct answers are hidden.

Score for this attempt: **0** out of 10
Submitted Oct 13 at 9:41am
This attempt took 25 minutes.

Question 1 — 0 / 0 pts

In the lecture, we said that the implementation of semaphores constitutes a critical section problem itself? Explain why.

Why using spinlocks is not a big problem in this implementation?

Your Answer:

The **implementation of semaphores is a critical section** problem because it must guarantee that no two processes can execute wait() and signal() on the same semaphore at the same time.

A semaphore is an integer variable with two indivisible (atomic) standard operations to modify it. Yes it still suffers from the busy-waiting problem. Yes it is possible to implement semaphores without busy-waiting. Instead, it just implements block and wake up. Block places the process invoking the operation on the appropriate waiting queue, and then block. Wake up removes one of the processes in the waiting queue and place it in the ready queue.

Using **spinlocks** is not a problem because the implementation of wait() and signal() is short so there is little busy-waiting if the critical section is rarely occupied

Question 2

0 / 0 pts

In the lecture notes, we analyze step-by-step two possible scenarios for scheduling processes that use Peterson's solution to the synchronization problem. Using similar analysis, analyze two potential scenarios for executing at least two processes that utilize an atomic operation of testing and setting variables for entering their critical sections. Show all the steps in your analysis of the scenarios.

Your Answer:

Scenario 1

flag[1] = TRUE;

turn = 2;

while (flag[2] && turn == 2) ;

// critical section

flag[1] = FALSE;

flag[1] = TRUE;

turn = 2;

while (flag[2] && turn == 2) ;

flag[2] = TRUE;

turn = 1;

while (flag[1] && turn == 1) ;

// critical section

flag[2] = FALSE;

flag[2] = TRUE;

turn = 1;

while (flag[1] && turn == 1) ;

// critical section

Scenario 2

flag[1] = TRUE;

turn = 2;

while (flag[2] && turn == 2) ;

// critical section

flag[1] = FALSE;

flag[2] = TRUE;

turn = 1;

while (flag[1] && turn == 1) ;

// critical section

flag[2] = FALSE;

Question 3

0 / 0 pts

In the lecture, we said that the solution of the Dining Philosophers problem with semaphores suffers from a potential starvation problem (a deadlock); explain why. Provide a detailed step-by-step example that leads to a deadlock.

Your Answer:

Deadlock is two or more processes are waiting indefinitely for an event that can be caused by only one of the waiting processes. If all of the philosophers get hungry at the same time, they will all reach for the chopstick on the left and because there are no chopsticks left on the table, they all can't eat because they all need another chopstick.

A deadlock example could involve 4 cars crossing an intersection, and if there is no proper signaling and they all go at the same time, none of them will be able to move any further

Question 4

0 / 0 pts

Define what a monitor is.

Explain what problems related to semaphores led to the introduction of monitors?

Why is using monitors safer than using semaphores?

Your Answer:

A monitor is a high-level abstraction that provides a convenient and effective high-level language mechanism for process synchronization, and only one process may be active in the monitor at a time. If a semaphore is used incorrectly, then it can result in timing errors that are hard to detect, as they only occur in certain sequences and don't always occur. The monitor type also declares the variables whose values define the state of an instance of that type, along with the bodies of functions that operate on

those variables. Thus, a function defined within a monitor can access only those variables declared locally within the monitor and its formal parameters.

Question 5

0 / 0 pts

Define what a spinlock is.

Under what conditions using spinlocks might be acceptable.

Your Answer:

A spinlock is when a process is in it's critical section and another process tries to enter theirs, the process loops continuously in the entry code, it's called that because the process is "spinning" while waiting for the lock to become available.

On single processor systems, the thread holding a lock cannot be running while another thread is testing the lock, because only one thread/process can be running at a time. Therefore the thread will continue to spin and waste CPU cycles until its time slice end

They are often used in multiprocessor systems because some processors can spin-lock while others continue to execute. It can spin while waiting for the lock

Question 6

0 / 0 pts

What is a semaphore? Does it suffer from the busy-waiting problem? Is it possible to implement it so that it does not?

Explain how a semaphore can be used to solve the critical section problem. Please note that that requires a proof that using a semaphore indeed is the basis for a solution to the critical section problem.

Your Answer:

A semaphore is an integer variable with two indivisible (atomic) standard operations to modify it. It does suffer from the busy-waiting problem. It can be implemented so it doesn't by using a waiting queue with each semaphore so it now has a value (of type integer; like before) and a pointer to the waiting queue, and with 2 operations, block (place in waiting queue) and wakeup (move to ready queue).

A solution to the critical section problem using semaphores is using a mutex to enforce mutual exclusion so that only one process is in it's mutual exclusion at a time, this is accomplished by using wait() before a critical section and then signal() once the critical section ends

Question 7

0 / 0 pts

Explain what a race condition is.

Provide two examples with a detailed step-by-step justification for including them.

Your Answer:
A race condition is when there is concurrent access to shared data and the final outcome depends upon the order of execution that is controlled by the OS and may vary over time

Whoever gets up first gets to use the bathroom first, they both want to shower but the order of execution says that whoever wakes up first gets to use it first

Driving to a stopsign, and the one who gets there first drives through it first. Whoever arrives first can use it thanks to order of execution

Question 8

0 / 0 pts

Explain with details what a critical section is.

Provide two real-life examples of problems that are similar to the critical section problem and explain why do you think they fall to the same category.

Your Answer:

A critical section is a section of code prone to race condition (where shared data is accessed), no other critical section is allowed to run at the same time as another one. (CO-OPERATION)

If you and a roommate are both out of milk and you both go to get some without consulting each other, you then have too much milk. It's similar because if you cooperated with your roommate, only one of you would get the milk and you wouldn't have too much.

If you need to leave a car at a mechanic and both you and your spouse both decide to take the car that needs repairing to the mechanic, but since you both went in the same car, there's no car to take you two home. If you two cooperated, then you would know to take another car so you can go back home

Question 9

0 / 0 pts

Provide a formal proof that blocking interrupts can be used as a solution for the critical section problem on a single-processor computer.

Would it work on a multiprocessor machine? Justify your answer.

Your Answer:

When a process is ready to enter into its critical section it sets a flag to disable interrupts. This ensures that no other process can be scheduled

and ensures mutual exclusion re-enabling the interrupts after the critical section can ensure progress and bounded waiting. On multiprocessor machines, the flag for disabling interrupts would have to be relayed to all processors which is costly

Question 10

0 / 0 pts

Provide a proof that the two-process Peterson's solution satisfies all requirements for a correct solution to the critical section problem.

HINT: Analyze all possible scheduling cases for two processes.

Your Answer:

Requirements for a solution:
Mutual Exclusion: if process Pi is executing in its critical section, then no other processes can be executing in their critical sections
Progress: if no process is executing in its critical section and there exist some processes that which to enter their critical section, then the section of the processes that will enter the critical section next cannot be postponed indefinitely
Bounded Waiting: a bound just exit the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

Solutions:
Mutual Solution: for a two process system, P0 and P1, they can never be in their critical section at the same time since no state can satisfy both turn = 0 and turn = 1
Progress: The peterson algorithm satisfies this condition since the OS selects P0 first, P0 will set the flag[0] condition to false after it exits its critical section therefore allowing P1 to break its busy wait every time and will access its critical section
Bounded waiting: Petersons algorithm satisfies this condition since the wait for one process to access its critical section will never be longer than one turn. For example, once P0 will give priority to P1 once it exits its critical section by setting flag[0] to false and then break P1's loop.

Question 11

0 / 0 pts

What requirements does a solution to the critical section problem need to satisfy?
Provide a commentary for each.

Your Answer:

Mutual exclusion, if a process is executing in it's critical section, no other processes can be in it's critical section at the same time.

Progress, if no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely.

Bounded Waiting, a bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

Question 12

0 / 0 pts

In the lecture notes, we analyze step-by-step two possible cases for scheduling processes that use Peterson's solution to the synchronization problem. Using similar analysis, analyze two potential scenarios for executing at least two processes that utilize an atomic operation of comparing and swapping two variables for entering their critical sections. Show all the steps in your analysis of the scenarios.

Your Answer:

Scenario 1

```
flag[1] = TRUE;
turn = 2;
while (flag[2] && turn == 2) ;
// critical section
flag[1] = FALSE;
flag[1] = TRUE;
turn = 2;
while (flag[2] && turn == 2) ;
```

```
flag[2] = TRUE;
turn = 1;
while (flag[1] && turn == 1) ;
// critical section
flag[2] = FALSE;
flag[2] = TRUE;
turn = 1;
while (flag[1] && turn == 1) ;
// critical section
```

Scenario 2

```
flag[1] = TRUE;
turn = 2;
while (flag[2] && turn == 2) ;
// critical section
flag[1] = FALSE;
```

```
flag[2] = TRUE;
```

```
turn = 1;
while (flag[1] && turn == 1) ;
// critical section
flag[2] = FALSE;
```

Question 13

10 / 10 pts

I have submitted answers to all questions in this study set.

- (●) True
- ( ) False

Quiz Score: **0** out of 10

Study Set for Lecture 08: Deadlocks

Due Oct 19 at 11:59pm **Points** 10 **Questions** 12 **Available** after Oct 13 at 12pm
Time Limit None **Allowed Attempts** Unlimited

Instructions

Review lecture notes from **lect08_Deadlocks.pdf**.

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

Take the Quiz Again

Attempt History

	Attempt	Time	Score
KEPT	Attempt 3	2 minutes	0 out of 10
LATEST	Attempt 3	2 minutes	0 out of 10
	Attempt 2	9 minutes	0 out of 10 *
	Attempt 1	2,867 minutes	0 out of 10 *

* Some questions not yet graded

ⓘ Correct answers are hidden.

Score for this attempt: **0** out of 10
Submitted Oct 21 at 8:58am
This attempt took 2 minutes.

Question 1 — 0 / 0 pts

What are the conditions necessary for a deadlock to occur?

Do they all need to be satisfied? Will just one be sufficient?

Support your answer with details and examples.

Your Answer:

Deadlock:

- The computer system uses may types of resource which are then used by various processes to carry out their individual functions.
- But problem is that the amount of resources available is limited and many process needs to use it.
- A set of process is said to be in a deadlocked state when every process in the set is waiting for an event that can be caused only by another process in the set. The event can

be resource acquisition, resource release etc. The resource can be physical (printers, memory space) or logical (semaphores, files)

Mutual Exclusion
only one process at a time can use a resource
Hold and wait
a process holding at least one resource is waiting to aquire additional resources held by other processes.
No preemption
a resource can be release only voluntarily by the process holding it after that process has completed its task
Circular wait
There exists a set {P0,P1,...,Pn} of waiting processes such that P0 is waiting for a resource that is held by P1,P1 is waiting for a resource that is held by P2, ..., Pn-1 is waiting for a resource that is held by Pn, and Pn is waiting for a resource that is held by P0.

Question 2

0 / 0 pts

Define what a resource allocation graph is.

Let there will be a cycle in a resource allocation graph. Can you tell if there is a deadlock?

Support your answer with all necessary details and an example.

Your Answer:

A resource allocation graph is a set of vertices and edges such that the vertices are partitioned into a set of processes and a set of all resource types in the system.

If there's a cycle and all instances of all resources in the set are taken up by the processes in the cycle, then there's a deadlock.

However, a cycle doesn't necessarily mean there's a deadlock, cause there might be extra instances of resources that are being used by processes that aren't in the cycle, which will be returned after the processes exit, resolving the deadlock.

Question 3

0 / 0 pts

Define deadlock prevention and deadlock avoidance.

Explain what is the difference between the two.

Your Answer:

Deadlock prevention - engineer the system in a way that eliminates deadlocks completely

Deadlock avoidance - Manage system so it avoids dangerous states, such as ones that may lead to deadlock

Deadlock prevention - implements strict rules that restrict user programs, which are often expensive on the overhead of the system. Deadlock avoidance is often less expensive and less strict but is a bit riskier, so not often used in mission-critical systems.

Question 4

0 / 0 pts

Discuss the use of resource numbering scheme to prevent deadlocks. Describe how the effectiveness of the scheme can be proven formally.

Then, illustrate the theory by designing a deadlock prevention algorithm based on resource numbering for the the Dining Philosophers problem.

Your Answer:

The resource numbering scheme gives each resource a number which indicates which process can access it. A process can request a resource only if the number of it's currently held resource has a lower number than the one it's requesting. It can be proven by contradiction, where we assume that we got a deadlock, which means that a process got a hold of a resource of a lower number than the one it currently was holding since it needs to be in a cycle for there to be a deadlock, which would have been impossible given that the process can't do that with those rules.

If you consider that each chopstick is given a number from 0-5, the fifth philosopher couldn't grab the 0 chopstick if they already held the 5th chopstick, thus no deadlock will occur because the circular wait condition won't hold.

Question 5

0 / 0 pts

The following is a resource allocation graph:

```
V={P1, P2, P3, P4, P5, R1, R2, R3, R4}
E={(R1, P1), (R2, P2), (R3, P5), (R4, P4), (P1, R2), (P2, R3), (P5, R4), (P4, R1), (P3, R1), (P3, R3)}
```

Using these data create a an adjacency table for the graph. Then, write a pseudocode for the algorithm that uses the adjacency table to detect whether there is a deadlock in the system.

Your Answer:

R1-> P1
P1-> R2
R2->P2
R3->P5
P5-> R4
R4->P4
P4->R1
P3->R3
P3->R1

There will be a deadlock in this system because P3 is requesting resources from R1 and R3 where P4 is asking for resources from R1 and P2 is asking resources from R3, this causes an issue because P3 would have to wait until R3 and R1 are released to be able to do something.

Question 6

0 / 0 pts

Consider the following state of the system:

Process	Allocation	Max	Available
	A B C	A B C	A B C
P0	0 1 0	7 5 3	3 3 2
P1	2 0 0	3 2 2	
P2	3 0 2	9 0 2	
P3	2 1 1	2 2 2	
P4	0 0 2	4 3 3	

Prove that the sequence <P_1, P_3, P_4, P_2, P_0> satisfies the safety criteria.

Use the following Google Sheets **Baker's Algorithm Task Template (https://docs.google.com/spreadsheets/d/1Td2hK8dYLoDk4O2TOpfF5HjMW9H4ahELMuxF6O7LOcI/template/previe** to perform the analysis. When you open the template, you need to click on the "Use Template" button to create your own version. After you are done, select "Download" from the "File" menu, then choose PDF, and then submit the downloaded PDF file as your answer.

Banker's Algorithm Problem 1.pdf
(https://cilearn.csuci.edu/files/2199969/download)

Question 7

0 / 0 pts

Consider the following state of the system:

Process	Allocation	Max	Available
	A B C	A B C	A B C
P0	0 1 0	7 5 3	3 3 2
P1	2 0 0	3 2 2	
P2	3 0 2	9 0 2	
P3	2 1 1	2 2 2	
P4	0 0 2	4 3 3	

Prove that a request for (3, 3, 0) cannot be granted to P_4.

Use the following Google Sheets **Baker's Algorithm Task Template** **(https://docs.google.com/spreadsheets/d/1Td2hK8dYLoDk4O2TOpfF5HjMW9H4ahELMuxF6O7LOcI/template/previe** to perform the analysis. When you open the template, you need to click on the "Use Template" button to create your own version. After you are done, select "Download" from the "File" menu, then choose PDF, and then submit the downloaded PDF file as your answer.

Banker's Algorithm Problem 1.pdf
(https://cilearn.csuci.edu/files/2199970/download)

Question 8

0 / 0 pts

Consider the following state of the system:

Process	Allocation	Max	Available
	A B C	A B C	A B C
P0	0 1 0	7 5 3	3 3 2
P1	2 0 0	3 2 2	
P2	3 0 2	9 0 2	
P3	2 1 1	2 2 2	
P4	0 0 2	4 3 3	

Prove that granting a request (0, 2, 0) to P_0 will end up in a safe state.

Use the following Google Sheets **Baker's Algorithm Task Template** **(https://docs.google.com/spreadsheets/d/1Td2hK8dYLoDk4O2TOpfF5HjMW9H4ahELMuxF6O7LOcI/template/previe** to perform the analysis. When you open the template, you need to click on the "Use Template" button to create your own version. After you are done, select "Download" from the "File" menu, then choose PDF, and then submit the downloaded PDF file as your answer.

Banker's Algorithm Problem 1.pdf
(https://cilearn.csuci.edu/files/2199971/download)

Question 9

0 / 0 pts

Invent a deadlock avoidance scheme based on the Banker's Algorithm that can be used to solve the Dining Philosophers problem. Provide an example that illustrates how the scheme works.

Provide a formal proof that the implementation indeed prevents deadlocks.

Do not submit code; rather, plain English description of the algorithm and why it will work.

Your Answer:

Given the need of each philosopher and the availible number of chopsticks, use work and finish to determing if allocating a number of chopsticks to a particular philosopher would lead to an unsafe state, if it doesn't, continue, else, stop that particular philosopher from grabbing chopsticks so that others can successfully eat without a deadlock.

If each philosopher tries to grab two chopsticks, the algorithm will eventually detect that one philosopher is trying to grab 2 chopsticks when only 1 is available, so it will block that philosopher until more chopsticks are put down by the other philosophers.

Question 10

0 / 0 pts

The system that you are designing cannot afford the overhead of deadlock prevention and deadlock avoidance. What is needed to implement deadlock detection scheme?

Discuss the options available in dealing with deadlocked systems?

Your Answer:

In order to implement a deadlock detection scheme, you can either use a wait-for graph, which is a resource-allocation graph in which the nodes are processes and the edge indicates that one process is waiting for another process's resource to be released. Either that or you can use the bankers algorithm to detect a deadlock instead of avoid one.

You could try to reclaim a process that's deadlocked by using a rollback, which returns the process to some previous safe state, but that could be expensive. The only other solution is termination, but you could improve that method by strategically choosing which process to terminate with the least amount of collateral.

Question 11

0 / 0 pts

Explain with details and examples why resource allocation graph cannot be used for deadlock avoidance in systems that have resources with multiple instances.

Your Answer:

A cycle in a resource allocation graph doesn't always mean that there's a deadlock. Especially if there are multiple instances of resources, where there could be a deadlock if there is a cycle, but we can't tell. For example, if there is a cycle between two resources with both having two instances, but the second instance of each resource is being allocated by separate processes that aren't a part of the cycle, then those processes will eventually free the resources they used, breaking the deadlock. If the two processes were a part of the cycle it would result in a deadlock.

Question 12

10 / 10 pts

I have submitted answers to all questions in this study set.

- (•) True
- ( ) False

Quiz Score: **0** out of 10

Study Set for Lecture 09: Main Memory

Due Oct 26 at 11:59pm **Points** 10 **Questions** 18
Available Oct 20 at 12pm - Dec 8 at 8pm about 2 months **Time Limit** None
Allowed Attempts Unlimited

Instructions

Review lecture notes from **lect09_Main_Memory.pdf**.

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

Take the Quiz Again

Attempt History

	Attempt	Time	Score
KEPT	Attempt 4	18 minutes	10 out of 10 *
LATEST	Attempt 4	18 minutes	10 out of 10 *
	Attempt 3	25 minutes	10 out of 10 *
	Attempt 2	4,342 minutes	10 out of 10
	Attempt 1	less than 1 minute	0 out of 10 *

* Some questions not yet graded

ⓘ Correct answers are hidden.

Score for this attempt: **10** out of 10 *
Submitted Dec 7 at 8:27pm
This attempt took 18 minutes.

Question 1 Not yet graded / 0 pts

When can the binding of instructions and data to memory addresses occur?

For each binding type, explain the details of the binding process and the implications.

Your Answer:

Binding of Instructions and Data to Memory:

Address (of both, instructions and data) from the program on the disk must be translated into an address in the memory

The binding can happen at three different stages :

● Compile time:
- if memory location is known a priori, then the compiler can generate absolute code
- code requires recompilation if the starting location changes
● Load time:
- if memory location is not known at compile time, code must be relocatable
-- binding is done when loading program
-- program must stay in memory
● Execution time:
- binding delayed until run time
-- the process must be able to be moved during its execution from one memory location to another.

Question 2

Not yet graded / 0 pts

What are the challenges of load time and execution time binding?

What are the mechanisms that support relocatable code?
Describe the process of loading code into any part of memory.

Your Answer:

Load time blinding: the challenges are that the program must remain in memory and that the code must relocatable. For execution time binding, the challenge is that the process must be portable from one memory segment to another

The mechanisms that supportable relocatable code are "base" and "limit" registers. The base gives the beginning of the space for the process. The limit is the scope of the base (base + limit). the code is located into memory within one of these scopes via one of the allocation approaches.

Question 3

Not yet graded / 0 pts

You have 10 programs that use a certain library. Explain the process of linking the programs with

- a static version of the library.
- a dynamic version of the library.

Your Answer:

Static

Becomes a part of the program after linking

The code within the library is cloned to every program that uses it

Dynamic

Small piece of code called the stub is used to locate the correct memory-resident library routine

The stub then switches itself with the address of the routine and then executes the routine

The OS then checks if the routine is in the process’ memory address

A static library becomes part of the program after linking. It is cloned to every program that uses it. Static libraries use a ".a" extension. For dynamic version, a stub is used to locate the appropriate memory resident library routine. The stub replaces itself with the address of the routine, and executes the routine

.

Static libraries, while reusable in multiple programs, are locked into a program at compile time. **Dynamic,** or **shared libraries** on the other hand, exist as separate files outside of the executable file.

The downside of using a static library is that it's code is locked into the final executable file and cannot be modified without a re-compile. In contrast, a dynamic library can be modified without a need to re-compile.

What does this mean in practical terms? Well, imagine you're a developer who has issued an application to thousands of users. When you want to make a few updates to the app, would you rather have to re-issue the entire program, or would you rather just issue updates in the form of modified libraries? The answer depends on the downsides your application can afford. If you have a lot of files, multiple copies of a static library means an increase in the executable file's size. If, however, the benefits of execution time outweigh the need to save space, the static library is the way to go.

Question 4

Not yet graded / 0 pts

Assume that the hit ratio in the MMU that performs paging with an associative memory (translation look-aside buffer (TLB)) is 90%. The access to main memory is 50 nanoseconds, and the lookup time for the TLB is 10 nanoseconds.

Compute what is the effective access time of this system.
Show the result and explain how did you obtain it.

Your Answer:

EAT = Effective Access Time

TLB = Translation Look-aside Buffer

EAT = memory access time + (lookup time * hit ratio) + (memory access time + lookup time) (1- hit ratio)

EAT = (50 nanoseconds) + (10 nanoseconds*.9) + (50 nanoseconds + 10 nanoseconds) (1-.9)

EAT = (50 nanoseconds) + (9 nanoseconds) + (60 nanoseconds) (.1)

EAT = (59 nanoseconds) + (6 nanoseconds)

EAT = 65 nanoseconds

ALSO

Effective memory access time(EMAT) : TLB is used to reduce effective memory access time as it is a high speed associative cache.
EMAT = h*(c+m) + (1-h)*(c+2m)
where, h = hit ratio of TLB
m = Memory access time
c = TLB access time

h = 0.9;

m = 50 ns;

c = 10;

EMAT = 0.9 * (10 + 50) + (1 - 0.9) * (10 + 2(50)) = 65 ns

Question 5

Not yet graded / 0 pts

You have a system with 1 MB RAM. The system uses 4 kB pages. Your system loads 5 programs with the following lengths:

```
167852 bytes
209376 bytes
32866 bytes
254871 bytes
128527 bytes
```

Calculate the scope of internal fragmentation after all programs are loaded into the memory.
Show the result and explain how did you compute it.

Your Answer:

Page size is 4 * 1024 = 4096 bytes/page

167852/4096 = 41 frames with 0.1 left
209376/4096 = 52 frames with 0.9 left
32866/4096 = 9 frames with 0.98 left
254871/4096 = 63 frames with 0.8 left
128527/4096 = 32 frames with 0.6 left

Total scope of fragmentation is (0.1+0.9+0.98+0.8+0.6) (4096) = 3.38*4096 = 13844.48

ALSO

1MB = 1000000 bytes

4KB = 4000 bytes

frameNum = RAM / frameSize -> framNum = 250, each is 4KB frames = pi / 4000

wastedSpace = pi - 4000(frames - 1)

fragmenta!on = ∑ wastedSpacei

Programs = (p1, p2, ..., p5)

p1 = 167852 bytes, 42 frames, wasted space 3852 bytes

p2 = 209376 bytes, 53 frames, wasted space 1376 bytes

p3 = 32866 bytes, 9 frames, wasted space 866 bytes p4 = 254871 bytes, 64 frames, wasted space 2871 bytes

p5 = 128527 bytes, 33 frames, wasted space 527 bytes

fragmenta!on = 3852 + 1376 +866 + 2871 + 527 = 9492 bytes

Question 6

Not yet graded / 0 pts

Consider the following requests (Rn) for memory allocation (A) and deallocation (D) from a memory pool of size 20:

```
R1 A 6
R2 A 3
R3 A 5
R4 A 2
R1 D
R5 A 4
R4 D
R6 A 1
R7 A 2
```

Compute explicitly (i.e., show all steps) the external fragmentation assuming the **best-fit** allocation policy.

Use the following notation:

```
[xxxxxxxxxxxxxxxxxxx]
[111111xxxxxxxxxxxxx]
[111111222xxxxxxxxxx]
```

which shows the initial state and then the two following allocations.

Your Answer:

Best-Fit

Allocatte the smallest hole that is big enough; must search entire list, unless ordered by size

- Produces the smallest leftover hole

[xxxxxxxxxxxxxxxxxxx]
[111111xxxxxxxxxxxxx]

[111111222xxxxxxxxxxx]
[11111122233333xxxxxx]
[1111112223333344xxxx]
[xxxxxx2223333344xxxx]
[xxxxxx22233333445555]
[xxxxxx22233333xx5555]
[xxxxxx222333336x5555]
[77xxxx222333336x5555]

Question 7

Not yet graded / 0 pts

Consider the following requests (Rn) for memory allocation (A) and deallocation (D) from a memory pool of size 20:

```
R1 A 6
R2 A 3
R3 A 5
R4 A 2
R1 D
R5 A 4
R4 D
R6 A 1
R7 A 2
```

Compute explicitly (i.e., show all steps) the external fragmentation assuming the **worst-fit** allocation policy.

Use the following notation:

```
[xxxxxxxxxxxxxxxxxxxx]
[111111xxxxxxxxxxxxxx]
[111111222xxxxxxxxxx]
```

which shows the initial state and then two following allocations.

Your Answer:

Worst Fit

Allocate the largest hole; must also search entire list

- produces the largest leftover hole

[xxxxxxxxxxxxxxxxxxxx]
[111111xxxxxxxxxxxxxx]
[111111222xxxxxxxxxxx]
[11111122233333xxxxxx]
[1111112223333344xxxx]
[xxxxxx2223333344xxxx]
[5555xx2223333344xxxx]
[5555xx22233333xxxxxx]
[5555xx222333336xxxxx]
[5555xx22233333677xxx]

Question 8

Not yet graded / 0 pts

Consider the following requests (Rn) for memory allocation (A) and deallocation (D) from a memory pool of size 20:

```
R1 A 6
R2 A 3
R3 A 5
R4 A 2
R1 D
R5 A 4
R4 D
R6 A 1
R7 A 2
```

Compute explicitly (i.e., show all steps) the external fragmentation assuming the **first-fit** allocation policy.

Use the following notation:

```
[xxxxxxxxxxxxxxxxxxxx]
[111111xxxxxxxxxxxxxx]
[111111222xxxxxxxxxxx]
```

which shows the initial state and then the two following allocations.

Your Answer:

First fit

Allocate the first hole that is big enough

[xxxxxxxxxxxxxxxxxxxx]

[111111xxxxxxxxxxxxxx]

[111111222xxxxxxxxxxx]

[11111122233333xxxxxx]

[1111112223333344xxxx]

[xxxxxx2223333344xxxx]

[5555xx2223333344xxxx]

[5555xx22233333xxxxxx]

[55556x22233333xxxxxx]

[55556x2223333377xxxx]

Question 9

Not yet graded / 0 pts

What is the difference between a physical address and a logical address? How is the logical address mapped to its physical counterpart? What is the lifetime of such binding?

Your Answer:

1. The basic difference between Logical and physical address is that Logical address is generated by CPU in perspective of a program whereas the physical address is a location that exists in the memory unit.
2. Logical Address Space is the set of all logical addresses generated by CPU for a program whereas the set of all physical address mapped to corresponding logical addresses is called Physical Address Space.
3. The logical address does not exist physically in the memory whereas physical address is a location in the memory that can be accessed physically.
4. Identical logical addresses are generated by Compile-time and Load time address binding methods whereas they differs from each other in

run-time address binding method. Please refer **this (https://www.geeksforgeeks.org/memory-management-mapping-virtual-address-physical-addresses/)** for details.

5. The logical address is generated by the CPU while the program is running whereas the physical address is computed by the Memory Management Unit (MMU).

The logical address is mapped to its physical counterpart by the memory management unit (MMU), which is a hardware device that maps logical to a physical address. The binding lives as long as the physical address still holds the corresponding data.

Question 10

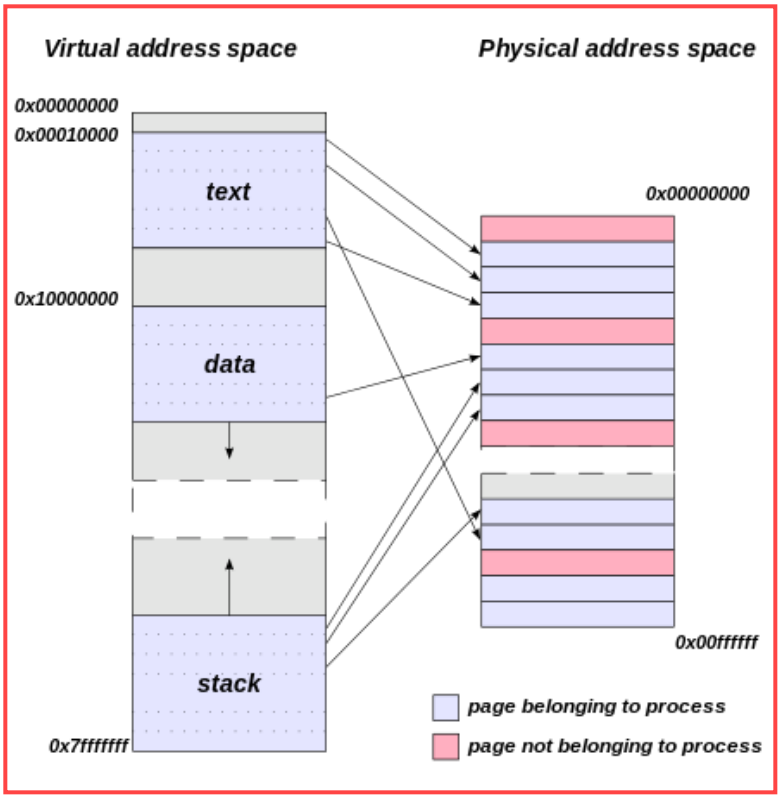
Not yet graded / 0 pts

Explain how paging alleviates the problems inherent to contiguous allocation of memory to programs.
What is a page table?
What is the difference between a page and a frame?
What is a free frame list?

Your Answer:

It reduces fragmentaion from copying whole code of process, paging allows for the program to be kept in logical memory. The logical "page" is the same size as the physical "frame". When needed, the process is allocated an available frame.

A page table is a table used to translate a logical address to a physical address in a set of frames.



A page is a block of logical memory, where a frame is a block of physical memory.

A free frame list is a list of frame positions that aren't allocated to a process and are free to be allocated to by a new program.

Question 11

Not yet graded / 0 pts

What would be the maximum size of a page table for a system with 48-bit addressing assuming a 1 kb frame size and assuming that 4 bytes are needed for every entry.

Describe each of the strategies to keep the size of the page table in check?

Your Answer:

2^48 = number of addresses

2^10 = frame size

(2^48) / (2^10) = 2^38 pages

2^4*2^38 = 2^42bytes per page = 2^12GB?

2^38 pages * 2^2 bytes / page = 2^40 bytes?

Hierarchal Pages

Slide 23

Break up the logical address space into multiple page tables

In other words, divide the page table into its own pages

Only some of the pages in the page table need to be in memory, which means it takes up less space

Hashed Page Tables

Slide 26

Common in 32-bit and up systems

Virtual page number is hashed into a page table

This page table contains a chain of elements hashing to the same location

Virtual page numbers are compared with the elements in order to find a match

If a match is found, the physical frame that belongs to the page number is extracted

Inverted Page Tables

Slide 27

One table for all processes

One entry for each real page of memory

Each entry consists of the virtual address of the page stored in that real memory location(p) as well as the info about who owns the page (pid)

The base is the offset of the entry in the entry table

Decreases space but increases the time it takes to search the table when a page is referenced

Question 12

Not yet graded / 0 pts

Consider a computer system that uses 5-bit addressing scheme with 3 bits for page numbers and two bits for offsets. Let the logical memory of some process be a contiguous sequence of letters from `a` to `z` (26 letters, only lower case).

Show the binary version of the logical address of letter `m` assuming that the following is the content of the page table: `[4, 1, 0, 5, 2, 7, 6, 3]`?

What is the physical address of the letter `m`? Show both the binary and the decimal versions.

Show the result and explain how did you derive it.

Your Answer:

Number of pages = 2^3 = 8

Number of addresses/page = 2^2 = 4

m is the 13th letter in the alphabet and we have 4 addresses per page

Therefore, m is located at the 3rd page (4th entry in the page table) because 13/4 = 3

The binary address is 10100

101 = 5, which is the virtual address space

00 = 0, which is where in the virtual address space m is

Physical address = (page number * page size) + offset = (5 * 4) + 0 =20

OR

we know that physical/logical address comprises of

page no bits	offset bits

since the logical address page number is of 3 bit. so **Page Table** assuming physical address page number is represented by 4-bit, will look like

Logical Address (page no)	**Physical address (page no)**
000	**4 (0100)**
001	1 (0001)
010	0 (0000)
011	5 (0101)
100	2 (0010)
101	7 (0111)
110	9 (1001)
111	11 (1011)

given the offset is of 2 bit which means each of the page contain $2^2 = 4$ letters.

since '**m'** is the 13^{th} alphabet of english and memory stored is contiguous.

so ceil(13/4) = 4 hence **m** will be present in 4^{th} page. i.e. p = 011

13%4 =1 , so '**m'** will be the first value inside 4^{th} page, i.e. offset = 00

logical Adress of m is given by: 01100

for physical adress ;

by page table, fourth entry 011 → 5 and offset = 00

so 6 bit physical address of '**m'** = physical page + number offset

= 0101 00

In decimal physical address = 20

Question 13

Not yet graded / 0 pts

The CPU executing a process generates `(5,0,4)` as a logical address in a system that uses an inverted page table to compute physical addresses. Let the following be the inverted page table: `[(2,1),(5,2),(7,3),(5,0),(6,4),...]`.

Assuming that there are 16 frames in the memory and page size is 8 construct the binary version of the physical address corresponding to this logical address.

Show the result and explain how did you derive it.

Your Answer:

Page number = 3, (5,0) is at page index 3

Page size = 8

4 = 4th spot in the page

24+4 = 28

1 1 1 0 0

index that matches is 3. So, 3 * 8 + 4 = 28.

Question 14

Not yet graded / 0 pts

Let the logical memory of some process be a contiguous sequence of letters from `a` to `z` (26 letters, only lower case).

Consider a computer system that uses 8-bit addressing scheme with six bits for two-level page indexing with the first level taking four bits, the second taking two, and the offsets another two.

Assuming that the following is the content of the outer page table:

```
[7, -, 0, 5, -, -, -, 2, -, 9, -, 11, 13, -, 1, -]
```

and that the content of the third (counting from zero) frame is:

```
[8, 12, 10, 4]
```

and the content of the sixth (again, counting from zero) frame is:

```
[3, 15, 6, 14]
```

show the binary version of the logical address of letter c knowing that the physical address of c is 18. Show the result and explain how did you derive it.

HINT: Draw a diagram showing the address, the two page tables, and the memory frames. Then, follow the links backwards starting with the physical location of the letter c.

Your Answer:

If the physical address of c is 18, then the physical address of a is 16

The page number of a-d is 16 (physical address) / 4 (number of bits/page) = 4

In binary, 18 is 0 0 0 1 0 0 1 0

The last two digits are transferred to the last two digits of the logical address

4 is found at index 3 of the third frame, which forms the 3rd and 4th digits of the logical address

2 is the index of the third frame, which is found at index 7 of the outer page table

The 7 is what makes up the remaining digits of the logical address, giving us an address of 0 1 1 1 1 1 1 0

Question 15

Not yet graded / 0 pts

How does a segmentation memory architecture differ from paging?

How can they both be integrated in a hybrid architecture?
Describe such a hybrid architecture with details on how logical addresses would be translated into their physical equivalents.

Your Answer:

The segmentation memory architecture makes a program a collection of segments, which is a logical unit such as a main program, procedure, function, method, object, local/global variable, stack, ect.

While paging puts the program into chunks of physical memory which can be found using a page and offset put into a page table for translating.

The CPU generates the logical address given to the segmentation unit, which produces linear addresses. The linear address is given to the paging unit which generates physical address in main memory. The paging units form the equivalent of the MMU.

Question 16

Not yet graded / 0 pts

Describe the buddy memory allocation scheme.

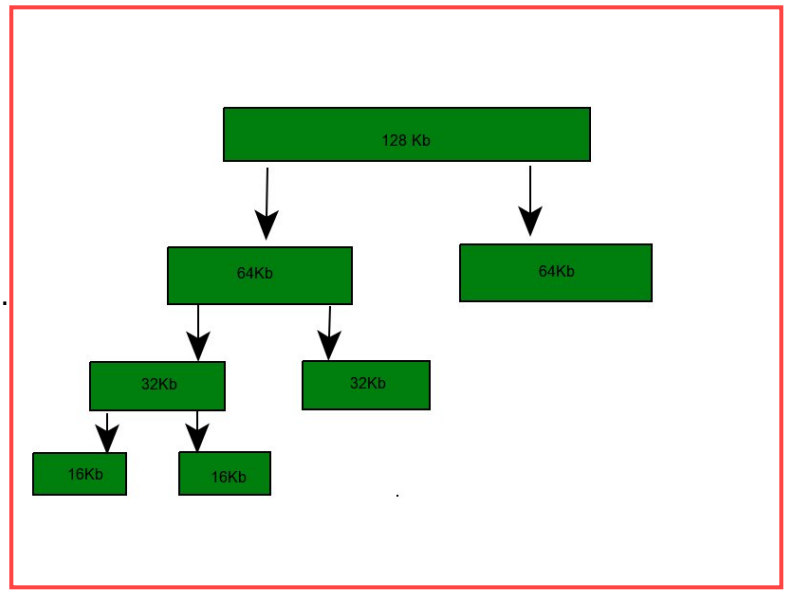
What are the benefits of using the buddy scheme for allocating memory for kernel needs?

Your Answer:

1. Buddy system –

Buddy allocation system is an algorithm in which a larger memory block is divided into small parts to satisfy the request. This algorithm is used to give best fit. The two smaller parts of block are of equal size and called as buddies. In the same manner one of the two buddies will further divide into smaller parts until the request is fulfilled. Benefit of this technique is that the two buddies can combine to form the block of larger size according to the memory request.

Example – If the request of 25Kb is made then block of size 32Kb is allocated.



1. Binary buddy system
2. Fibonacci buddy system
3. Weighted buddy system
4. Tertiary buddy system

Why buddy system?
If the partition size and procees size are different then poor match occurs and may use space inefficiently.
It is easy to implement and efficient then dynamic allocation.

Drawback –
The main drawback in buddy system is internal fragmentation as larger block of memory is acquired then required. For example if a 36 kb request is made then it can only be satisfied by 64 kb segment and reamining memory is wasted.

Question 17

Not yet graded / 0 pts

Describe the slab memory allocation scheme.

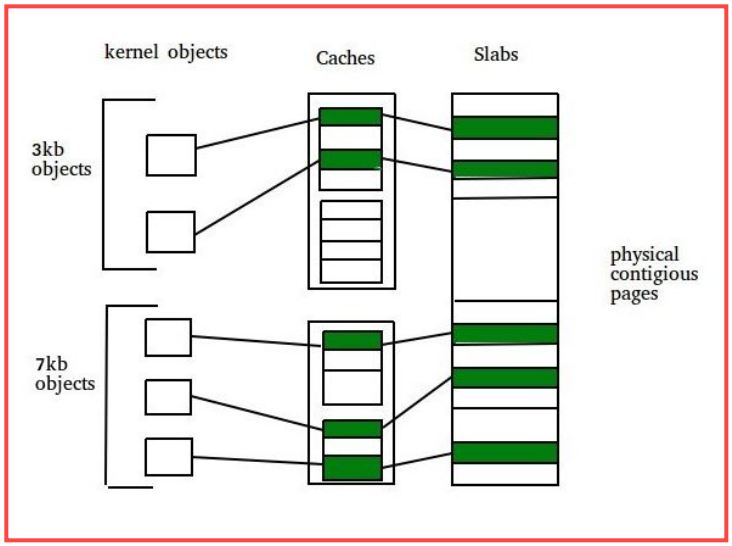
What are the benefits of using the slab scheme for allocating memory for kernel needs?

Your Answer:

2. Slab Allocation –

A second strategy for allocating kernel memory is known as slab allocation. It eliminates fragmentation caused by allocations and deallocations. This method is used to retain allocated memory that contains a data object of a certain type for reuse upon subsequent allocations of objects of the same type. In slab allocation memory chunks suitable to fit data objects of certain type or size are preallocated. Cache does not free the space immediately after use although it keeps track of data which are required frequently so that whenever request is made the data will reach very fast. Two terms required are:

- **Slab –** A slab is made up of one or more physically contiguous pages. The slab is the actual container of data associated with objects of the specific kind of the containing cache.
- **Cache –** Cache represents a small amount of very fast memory. A cache consists of one or more slabs. There is a single cache for each unique kernel data structure.



Each cache is populated with objects that are instantiations of the kernel data structure the cache represents. For example the cache representing semaphores stores instances of semaphore objects, the cache representing process descriptors stores instances of process descriptor objects.

Benefits of slab allocator –

- No memory is wasted due to fragmentation because each unique kernel data structure has an associated cache.
- Memory request can be satisfied quickly.

- The slab allocating scheme is particularly effective for managing when objects are frequently allocated or deallocated. The act of allocating and releasing memory can be a time consuming process. However, objects are created in advance and thus can be quickly allocated from the cache. When the kernel has finished with an object and releases it, it is marked as free and return to its cache, thus making it immediately available for subsequent request from the kernel.

Question 18

10 / 10 pts

I have submitted answers to all questions in this study set.

- (●) True
- ( ) False

Quiz Score: **10** out of 10

Study Set for Lecture 10: Virtual Memory

Due Nov 2 at 11:59pm **Points** 10 **Questions** 16
Available Oct 27 at 12pm - Dec 8 at 8pm about 1 month **Time Limit** None
Allowed Attempts Unlimited

Instructions

Review lecture notes from **lect10_Virtual_Memory.pdf**.

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

Take the Quiz Again

Attempt History

	Attempt	Time	Score
KEPT	Attempt 3	31 minutes	10 out of 10 *
LATEST	Attempt 3	31 minutes	10 out of 10 *
	Attempt 2	34 minutes	10 out of 10
	Attempt 1	less than 1 minute	0 out of 10 *

* Some questions not yet graded

ⓘ Correct answers are hidden.

Score for this attempt: **10** out of 10 *
Submitted Dec 7 at 8:59pm
This attempt took 31 minutes.

Question 1 **Not yet graded / 0 pts**

Explain what a virtual memory is including the role of demand paging realized by the lazy swapper in memory virtualization.

Furthermore, describe the benefits of memory virtualization.

Your Answer:

Virtual memory is separation of user logical memory from physical memory where only part of the program needs to be in memory for execution. This is known as the lazy swapper method where a page is swapped into memory only when the page is needed (demand paging). the benefits of memory virtualization are:□Flexible process management: The process doesn't need to wait for an opening in memory so only a portion can go in giving it a better response time, the total logical space for all processes can be larger than the available physical address space giving it more concurrent processes and users, and the address spaces can be shared by several processes such as dynamic libraries.□Less I/O: Fewer pages to swap in and out and improved performance

Advantages :

- More processes may be maintained in the main memory: Because we are going to load only some of the pages of any particular process, there is room for more processes. This leads to more efficient utilization of the processor because it is more likely that at least one of the more numerous processes will be in the ready state at any particular time.
- A process may be larger than all of main memory: One of the most fundamental restrictions in programming is lifted. A process larger than the main memory can be executed because of demand paging. The OS itself loads pages of a process in main memory as required.
- It allows greater multiprogramming levels by using less of the available (primary) memory for each process.

Causes of Thrashing :

1. **High degree of multiprogramming** : If the number of processes keeps on increasing in the memory than number of frames allocated to each process will be decreased. So, less number of frames will be available to each process. Due to this, page fault will occur more frequently and more CPU time will be wasted in just swapping in and out of pages and the utilization will keep on decreasing.

   For example:
   Let free frames = 400

Case 1: Number of process = 100
Then, each process will get 4 frames.

Case 2: Number of process = 400
Each process will get 1 frame.
Case 2 is a condition of thrashing, as the number of processes are increased,frames per process are decreased. Hence CPU time will be consumed in just swapping pages.

2. **Lacks of Frames**:If a process has less number of frames then less pages of that process will be able to reside in memory and hence more frequent swapping in and out will be required. This may lead to thrashing. Hence sufficient amount of frames must be allocated to each process in order to prevent thrashing.

Recovery of Thrashing :

- Do not allow the system to go into thrashing by instructing the long term scheduler not to bring the processes into memory after the threshold.
- If the system is already in thrashing then instruct the mid term schedular to suspend some of the processes so that we can recover the system from thrashing.

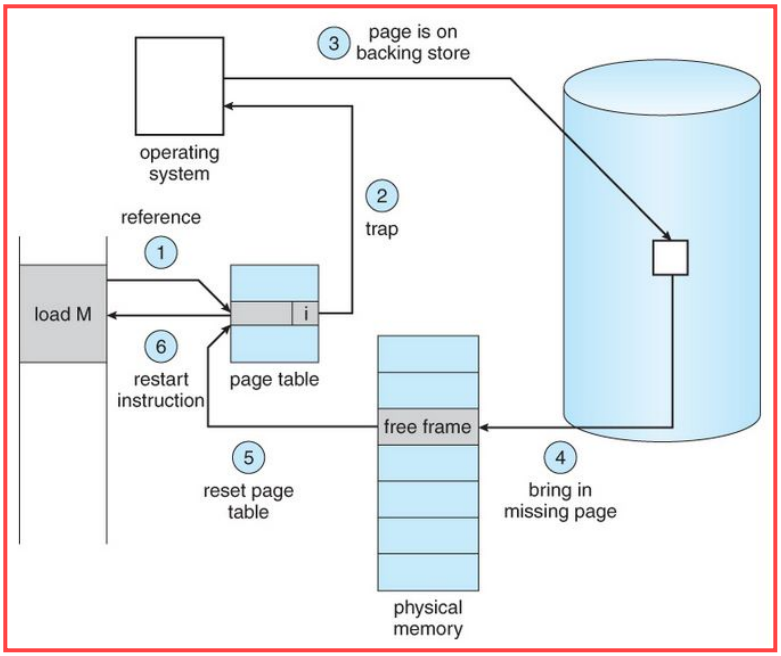
Question 2

Not yet graded / 0 pts

Explain what a page fault is, how it is detected, and describe step-by-step the process of page replacement.

Your Answer:

A page fault occurs when a program attempts to access data or code that is in its address space, but is not currently located in the system RAM. So when page fault occurs then following sequence of events happens :



- The computer hardware traps to the kernel and program counter (PC) is saved on the stack. Current instruction state information is saved in CPU registers.
- An assembly program is started to save the general registers and other volatile information to keep the OS from destroying it.
- Operating system finds that a page fault has occurred and tries to find out which virtual page is needed. Some times hardware register contains this required information. If not, the operating system must retrieve PC, fetch instruction and find out what it was doing when the fault occurred.
- Once virtual address caused page fault is known, system checks to see if address is valid and checks if there is no protection access problem.
- If the virtual address is valid, the system checks to see if a page frame is free. If no frames are free, the page replacement algorithm is run to remove a page.
- If frame selected is dirty, page is scheduled for transfer to disk, context switch takes place, fault process is suspended and another process is made to run until disk transfer is completed.
- As soon as page frame is clean, operating system looks up disk address where needed page is, schedules disk operation to bring it in.
- When disk interrupt indicates page has arrived, page tables are updated to reflect its position, and frame marked as being in normal state.

- Faulting instruction is backed up to state it had when it began and PC is reset. Faulting is scheduled, operating system returns to routine that called it.
- Assembly Routine reloads register and other state information, returns to user space to continue execution.

Question 3

Not yet graded / 0 pts

Discuss local frame allocation dilemma; i.e., options for distributing frames between processes.

Your Answer:

There are a few options for distributing frames between processes:

One is choosing the replacement scopes. Global replacement means the process selects a replacement frame from the set of all frames and one process may be given a frame taken away from another. Another is local replacement where each process selects only from it's own set of allocated frames.

Another option is how each does fixed allocation. Equal allocation means each process is given the same number of frames. On the other hand, proportional allocation allocates according to the size of the process.

Finally, you could do dynamic allocation, also known as priority allocation. Using a proportional allocation scheme using priorities rather than size, if a process generates a page fault, select for replacement a frame from a process with a lower priority.

Question 4

Not yet graded / 0 pts

Compute the memory effective access time in a system with the following characteristics:

- page faults happen once every 2000 memory accesses on average,
- disk access time is 5 ms,
- probability that the dirty bit is set on the vitctim page is 0.1,
- memory access time is 100 nanoseconds,
- page fault overhead is 7 nanoseconds, and
- restart overhead is 3 nanoseconds.

Your Answer:

EAT = (1 – p) * memory access + p * [page fault trap overhead + swap page out + swap page in + restart overhead]

EAT = (1 – (1/2000)) * 100 + (1/2000) * [7 + 5000000 + 5000000 + 3]

EAT = 5099.955 nanoseconds = 5.1e-3 ms

ALSO reference

p = 1/2000 = 0.002

mem access = 100 ns

read/write = 5ms = 5000000 ns

dirty bit = 0.1

page fault/restart overhead = 10 ns

EAT = (1-0.0005) **100 + 0.0005** (10 + 5000000 + 5000000(.1) + 10)

= (.9995)*100 + 0.0005(5500020)

99.95 + 2750.01

2849.96 nanoseconds

Question 5

Not yet graded / 0 pts

Assume that an OS with 3 frames uses OPT replacement policy . For the following reference set

```
{4, 3, 6, 2, 1, 2, 5, 4, 3, 4, 2, 3, 6, 1, 2, 3}
```

1. Show with details all page replacements.
2. State clearly how many page faults have occurred.

Show frame content (which pages are loaded) for each page reference per line. Use "-" for empty frame; otherwise, use the page number of the page loaded into a particular frame. Precede the number corresponding to the page being referenced with "<". If there is a page fault, indicate which page was replaced by preceding the corresponding number with "*".

For example, the following line:

```
- 4 2<
```

indicates that the first frame is free, the second frame holds page number 4, and the page number 2 held in the third frame has just been referenced. The following line:

```
6* 4 2
```

indicates that page 6 was referenced and faulted; it indicates that page replacement occurred in the first frame.

For example, in FIFO, the following string:

```
1 1 5 2 4 2 0 2 5 1 0 5 0 2 3
```

will be processed as follows:

```
1* -  -
1< -  -
1  5* -
1  5  2*
4* 5  2
4  5  2<
```

```
4  0* 2
4  0  2<
4  0  5*
1* 0  5
1  0< 5
1  0  5<
1  0< 5
1  2* 5
1  2  3*
```

Your Answer:

{4, 3, 6, 2, 1, 2, 5, 4, 3, 4, 2, 3, 6, 1, 2, 3}

***4 - -**
4 *3 -
4 3 *6
4 3 *2
4 *1 2
4 1 >2
4 *5 2
>4 5 2
4 *3 2
>4 3 2
4 3 >2
4 >3 2
***6 3 2**
***1 3 2**
1 3 >2

1 >3 2

9 page faults

4: Memory is: 4 * * : Page Fault: (Number of Page Faults: 1)
3: Memory is: 4 3 * : Page Fault: (Number of Page Faults: 2)
6: Memory is: 4 3 6 : Page Fault: (Number of Page Faults: 3)
2: Memory is: 4 3 2 : Page Fault: (Number of Page Faults: 4)
1: Memory is: 4 1 2 : Page Fault: (Number of Page Faults: 5)
2: Memory is: 4 1 2 : Hit: (Number of Page Faults: 5)
5: Memory is: 4 5 2 : Page Fault: (Number of Page Faults: 6)
4: Memory is: 4 5 2 : Hit: (Number of Page Faults: 6)

3: Memory is: 4 3 2 : Page Fault: (Number of Page Faults: 7)
4: Memory is: 4 3 2 : Hit: (Number of Page Faults: 7)
2: Memory is: 4 3 2 : Hit: (Number of Page Faults: 7)
3: Memory is: 4 3 2 : Hit: (Number of Page Faults: 7)
6: Memory is: 6 3 2 : Page Fault: (Number of Page Faults: 8)
1: Memory is: 1 3 2 : Page Fault: (Number of Page Faults: 9)
2: Memory is: 1 3 2 : Hit: (Number of Page Faults: 9)
3: Memory is: 1 3 2 : Hit: (Number of Page Faults: 9)
Total Number of Page Faults: 9

Process finished with exit code 0

Question 6

Not yet graded / 0 pts

Assume that an OS with 3 frames uses FIFO replacement policy . For the following reference set

```
{4, 3, 6, 2, 1, 2, 5, 4, 3, 4, 2, 3, 6, 1, 2, 3}
```

1. Show with details all page replacements.
2. State clearly how many page faults have occurred.

Show frame content (which pages are loaded) for each page reference per line. Use "-" for empty frame; otherwise, use the page number of the page loaded into a particular frame. Precede the number corresponding to the page being referenced with "<". If there is a page fault, indicate which page was replaced by preceding the corresponding number with "*".

For example, the following line:

```
- 4 2<
```

indicates that the first frame is free, the second frame holds page number 4, and the page number 2 held in the third frame has just been referenced. The following line:

```
6* 4 2
```

indicates that page 6 was referenced and faulted; it indicates that page replacement occurred in the first frame.

For example, in FIFO, the following string:

```
1 1 5 2 4 2 0 2 5 1 0 5 0 2 3
```

will be processed as follows:

```
1* -  -
1< -  -
1  5* -
1  5  2*
4* 5  2
4  5  2<
4  0* 2
4  0  2<
4  0  5*
1* 0  5
1  0< 5
1  0  5<
1  0< 5
1  2* 5
1  2  3*
```

Your Answer:

{4, 3, 6, 2, 1, 2, 5, 4, 3, 4, 2, 3, 6, 1, 2, 3}

***4 - -**
4 *3 -
4 3 *6
***2 3 6**
2 *1 6
>2 1 6
2 1 *5
***4 1 5**
4 *3 5
>4 3 5
4 3 *2
4 >3 2
***6 3 2**
6 *1 2
6 1 >2
6 1 *3

12 page faults

4 -1 -1
4 3 -1
4 3 6
2 3 6
2 1 6
Hit & column > 0 :2 1 6
2 1 5
4 1 5
4 3 5
Hit & column > 0 :4 3 5
4 3 2
Hit & column > 1 :4 3 2
6 3 2
6 1 2
Hit & column > 2 :6 1 2
6 1 3
No. of page hit : 4

12 page faults

Process finished with exit code 0

Question 7

Not yet graded / 0 pts

Assume that an OS with 3 frames uses LRU replacement policy . For the following reference set

```
{4, 3, 6, 2, 1, 2, 5, 4, 3, 4, 2, 3, 6, 1, 2, 3}
```

1. Show with details all page replacements.
2. State clearly how many page faults have occurred.

Show frame content (which pages are loaded) for each page reference per line. Use "-" for empty frame; otherwise, use the page number of the page loaded into a particular frame. Precede the number corresponding to the page being referenced with "<". If there is a page fault, indicate which page was replaced by preceding the corresponding number with "*".

For example, the following line:

```
- 4 2<
```

indicates that the first frame is free, the second frame holds page number 4, and the page number 2 held in the third frame has just been referenced. The following line:

```
6* 4 2
```

indicates that page 6 was referenced and faulted; it indicates that page replacement occurred in the first frame.

For example, in FIFO, the following string:

```
1 1 5 2 4 2 0 2 5 1 0 5 0 2 3
```

will be processed as follows:

```
1* -  -
1< -  -
1  5* -
1  5  2*
4* 5  2
4  5  2<
4  0* 2
4  0  2<
4  0  5*
1* 0  5
1  0< 5
1  0  5<
1  0< 5
1  2* 5
1  2  3*
```

Your Answer:

{4, 3, 6, 2, 1, 2, 5, 4, 3, 4, 2, 3, 6, 1, 2, 3}

***4 - -**
4 *3 -
4 3 *6
***2 3 6**
2 *1 6
>2 1 6
2 1 *5

2 *4 5
*3 4 5
3 >4 5
3 4 *2
>3 4 2
3 *6 2
3 6 *1
*2 6 1

2 *3 1

13 page faults

Question 8

Not yet graded / 0 pts

Explain what Bellady's Anomaly is.

Illustrate the problem by computing page faults in an OS that uses memory with three frames and with four frames.

Use the following reference set:

```
{1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5}
```

Your Answer:

Belady's Anomaly is where more frames can cause more page faults where it should cause less page faults.

In computer storage (https://en.wikipedia.org/wiki/Computer_storage) , **Bélády's anomaly** is the phenomenon in which increasing the number of page frames results in an increase in the number of page faults (https://en.wikipedia.org/wiki/Page_fault) for certain memory access patterns. This phenomenon is commonly experienced when using the first-in first-

out (**FIFO (https://en.wikipedia.org/wiki/FIFO_(computing_and_electronics))**) **page replacement algorithm (https://en.wikipedia.org/wiki/Page_replacement_algorithm)**. In FIFO, the page fault may or may not increase as the page frames increase, but in Optimal and stack-based algorithms like LRU, as the page frames increase the page fault decreases. **László Béládi (https://en.wikipedia.org/wiki/L%C3%A1szl%C3%B3_B%C3%A9l%C3%A1dy)** demonstrated this in 1969.[1] **(https://en.wikipedia.org/wiki/B%C3%A9l%C3%A1dy%27s_anomaly#cite_note-1)**

FIFO

***1 - -**
1 *2 -
1 2 *3
***4 2 3**
4 *1 3
4 1 *2
***5 1 2**
5 >1 2
5 1 >2
5 *3 2
5 3 *4
>5 3 4

9 page faults

***1 - - -**
1 *2 - -
1 2 *3 -
1 2 3 *4
>1 2 3 4
1 >2 3 4
***5 2 3 4**
5 *1 3 4

5 1 *2 4
5 1 2 *3
***4 1 2 3**
4 *5 2 3

10 page faults

Question 9

Not yet graded / 0 pts

Assume the following page reference sequence:

```
{0, 3, 4, 3, 3, 0, 6, 5, 5, 6, 0, 2, 1, 2, 4, 3, 3, 4, 2, 1, 1, 1,
2, 3, 4, 5, 6, 4, 5, 6, 5, 5, 6, 6, 1}
```

Show all successive working sets using a window $\Delta = 4$. Use 4 reference bits that are set from the left for the pages that are referenced in the current window and then shifted right on every timeout. The "timeout" is simulated by counting page references; in this task assume a timeout every two page references.

Write C code that declares appropriate data structures and implements the algorithm using bit shifting to maintain the references bits. You do not need to use packing in this solution (i.e., you are allowed to utilize just 4 bits in a byte to hold the reference information.

Your Answer:

Step 1: Slide the window

Step 2: Do the right shift

Step 3: Look at everything within the window and mark the reference

Every 2 references is when the reference table gets updated

First working set is the first 2 in the window

Second working set takes into account all 4 in the window

When the window gets shifted, all 4 are taken into account

If the number of page references isn't a multiple of the window, then just use however many end up in the window

ALSO NOT SURE

-1 -1 -1

3 -1 -1

3 4 -1

3 4 -1

3 4 -1

3 4 -1

3 4 6

3 5 6

3 5 6

3 5 6

3 5 6

2 5 6

2 1 6

2 1 6

2 1 4

2 1 4

2 1 4

2 1 4

2 1 4

2 1 4

2 1 4

2 1 4

2 1 4

2 1 4

2 5 4

6 5 4

6 5 4

6 5 4

6 5 4

6 5 4

6 5 4

6 5 4

6 5 4

6 5 4

Total Page Faults = 12

Question 10

Not yet graded / 0 pts

Can a quasi-stack the top of which always keeps the last referenced page number and which provides access to every element (hence "quasi" in contrast to a true stack that can only push and pop at the top) be a foundation for implementing the LRU page replacement algorithm? How can you find the least recently used element?

Your Answer:

In operating systems that use paging for memory management, page replacement algorithm are needed to decide which page needed to be replaced when new page comes in. Whenever a new page is referred and

not present in memory, page fault occurs and Operating System replaces one of the existing pages with newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce number of page faults.

In **L**east **R**ecently **U**sed (LRU) algorithm is a Greedy algorithm where the page to be replaced is least recently used. The idea is based on locality of reference, the least recently used page is not likely

```
Let capacity be the number of pages that
memory can hold.  Let set be the current
set of pages in memory.

ALGORITHM

1- Start traversing the pages.
 i) If set holds less pages than capacity.
   a) Insert page into the set one by one until
      the size  of set reaches capacity or all
      page requests are processed.
   b) Simultaneously maintain the recent occurred
      index of each page in a map called indexes.
   c) Increment page fault
 ii) Else
   If current page is present in set, do nothing.
   Else
     a) Find the page in the set that was least
     recently used. We find it using index array.
     We basically need to replace the page with
     minimum index.
     b) Replace the found page with current page.
     c) Increment page faults.
     d) Update index of current page.

2. Return page faults.
```

Question 11

Not yet graded / 0 pts

Consider a double-linked list-based implementation of the LRU quasi-stack that was discussed in the lecture.

Explain step-by-step the algorithm to keep the last referenced page at the top of the stack. Note that the page being referenced may, but does not need to, be already on the stack. Your algorithm must handle both cases.

What is the complexity of updating the stack? That is, how many operations are required to add a new page number at the top, or to move an already existing page number from some location in the stack to the top?

Your Answer:

It would keep the top pointing to the correct victim page because it would run through the array to figure out which number would be used closest, so in that cause it will be able to figure out what to pop off the stack.

Question 12

Not yet graded / 0 pts

Consider a fixed-size non-circular array-based implementation of the LRU quasi-stack that was discussed in the lecture.

Explain step-by-step the algorithm to keep the last referenced page at the top of the stack. Note that the page being referenced may, but does not need to, be already on the stack. Your algorithm must handle both cases.

Is it an efficient approach to implementing the LRU policy? What is the complexity of updating the stack? That is, how many operations are required to add a new page number at the top, or to move an already existing page number from some location in the stack to the top?

Your Answer:

When an LRU cache is full, we discard the *Least Recently* ***Used*** item.

If we're discarding items from the front of the queue, then, we have to make sure the item at the front is the one that hasn't been used for the longest time.

We ensure this by making sure that an item goes to the back of the queue whenever it *is* used. The item at the front is then the one that hasn't been moved to the back for the longest time.

To do this, we need to maintain the queue on every `put` OR `get` operation:

- When we `put` a new item in the cache, it becomes the *most* recently used item, so we put it at the back of the queue.
- When we `get` an item that is already in the cache, it becomes the *most* recently used item, so we *move* it from its current position to the back of the queue.

Moving items from the middle to the end is *not* a deque operation and is not supported by the `ArrayDeque` interface. It's also not supported efficiently by the underlying data structure that `ArrayDeque` uses. Doubly-linked lists are used because they *do* support this operation efficiently.

Question 13

Not yet graded / 0 pts

Consider a fixed-size circular array-based implementation of the LRU quasi-stack that was discussed in the lecture.

Explain step-by-step the algorithm to keep the last referenced page at the top of the stack. Note that the page being referenced may, but does not need to, be already in the stack. Your algorithm must handle both cases.

What is the complexity of updating the stack on each new page reference? That is, how many operations are required to add a new page number at the top, or to move an already existing page number from some location in the stack to the top?

HINT: Note that when the stack implemented as a circular array is full, the top is "circulated" by incrementing (modulo the size of the array) the index of the top and the index of the bottom, and then overwriting what used to be the stack bottom (which holds the number of currently least recently used page number) with the number of the newly referenced page.

Your Answer:

When an LRU cache is full, we discard the *Least Recently* ***Used*** item.

If we're discarding items from the front of the queue, then, we have to make sure the item at the front is the one that hasn't been used for the

longest time.

We ensure this by making sure that an item goes to the back of the queue whenever it *is* used. The item at the front is then the one that hasn't been moved to the back for the longest time.

To do this, we need to maintain the queue on every `put` OR `get` operation:

- When we `put` a new item in the cache, it becomes the *most* recently used item, so we put it at the back of the queue.
- When we `get` an item that is already in the cache, it becomes the *most* recently used item, so we *move* it from its current position to the back of the queue.

Moving items from the middle to the end is *not* a deque operation and is not supported by the `ArrayDeque` interface. It's also not supported efficiently by the underlying data structure that `ArrayDeque` uses. Doubly-linked lists are used because they *do* support this operation efficiently.

Question 14

Not yet graded / 0 pts

Explain how to implement efficient page replacement algorithms that approximate LRU. What is a "second chance" approach?

Your Answer:

LRU cannot be implemented without special hardware. Aging is a way to approximate LRU which is based on LRU and the workign set. We can approximate the age/usage. With each page associate a reference bit. Initially set to 0; Change to 1 upon reference.
Search: if 1, then change to 0 and continue if 0, replace the page and change to 1.
Second change approach needs a second reference bit to simulate a clock by shifting the first reference bit to the second one.

Question 15

Not yet graded / 0 pts

Let's consider the following array:

```
char data[4][4];
```

on a trivial machine with the memory consisting of a single 4-byte frame.

1) How many pages does `data` encompass?

Consider two alternate ways to traverse `data`:

```
// A
for (i = 0; i < 4; i++)
   for (j = 0; j < 4; j++)
      data[i][j] = 0;
```

```
// B
for (j = 0; j < 4; j++)
   for (i = 0; i < 4; i++)
      data[i][j] = 0;
```

2) Assuming that the row major array order is used on this machine (i.e., arrays are stored by rows — like in C), calculate how many page faults will occur executing `A` and how many will occur executing `B`.

Your Answer:

ssuming the generality that the character size is of 1 Byte, the given character array data[4][4] contains 16 elements each of size 1 Byte.

So the total space required to store the array = 4*4 Bytes = 16 Bytes.

Also, given that the frame size is of 4 Byte which implies the page size is also of 4 Byte according to standard memory management technique.

1) Number of pages that data encompasses = 16 Bytes/ 4 Bytes = 4 pages.

2) Given the array is stored in row major order. And given only one frame exists in the memory of the trivial machine given.

So, at a time it implies the memory can store four elements of the data array in a frame which is loaded with a page.

Execution of A is similar to row-wise access of the array because the variable i in data[i][j] is fixed as it is outer loop intially and then the inner loop variable j is varying from 0 to 3. So the data is accesses in a fashion data[0][0], data[0][1], data[0][2], data[0][3], data[1][0],..... so on. It is similar to the way the elements get stored in a page.

There will be four pages in the Virtual address space as shown below

Page 1----> data[0][0], data[0][1], data[0][2], data[0][3]

Page 2----> data[1][0], data[1][1], data[1][2], data[1][3]

Page 3----> data[2][0], data[2][1], data[2][2], data[2][3]

Page 4----> data[3][0], data[3][1], data[3][2], data[3][3]

So, while executing the A code, assuming initally there is no page in the only single frame present in the physical address space, the first page fault occurs while accessing data[0][0]. So, as the first page(Page 1) is loaded into the frame, accessing data[0][1], data[0][2], data[0][3] there won't be any page faults because the Page 1 itself contains all these three elements. Again while accesing data[1][0] there will be page fault and similar case for the rest of the elements in the page as told previously. And for every first element of the row, there will be a page fault.

**********So, totally while executing code A, there will be 4 page faults.

Execution of B is similar to column-wise access of the array because the variable j in data[i][j] is fixed as it is outer loop intially and then the inner loop variable i is varying from 0 to 3. So the data is accesses in a fashion data[0][0], data[1][0], data[2][0], data[3][0], data[0][1], data[1][1]..... so on. It is similar to the way the elements get stored in a page.

So, while executing the B code, assuming initally there is no page in the only single frame present in the physical address space, the first page fault occurs while accessing data[0][0]. So, as the Page 1 is loaded into the frame, accessing data[1][0] next to data[0][0] as per the code will again cause a page fault since the Page 1 which is already in the frame do not contain data[1][0] which is accessed next to data[0][0]. So, Page 2 is loaded on to the frame replacing Page 1. Similar is the case with data[2][0] and data[3][0]. After that data[0][1] is accessed when Page 4 is present in the frame. So, again page fault occurs in this case. Similarly, as told previously again the page faults occurs for each and every access in this case.

**********So, totally while executing code B, there will be 16 page faults.

Question 16

10 / 10 pts

I have submitted answers to all questions in this study set.

- (•) True
- ( ) False

Quiz Score: **10** out of 10

Study Set for Lecture 11: File System Interface

Due Nov 9 at 11:59pm **Points** 10 **Questions** 13
Available Nov 3 at 12pm - Dec 8 at 8pm about 1 month **Time Limit** None
Allowed Attempts Unlimited

Instructions

Review lecture notes from **lect11_File_System_Interface.pdf**. Please download the accompanying code from **lect11code.zip**.

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

Take the Quiz Again

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	1,373 minutes	10 out of 10

Correct answers are hidden.

Score for this attempt: **10** out of 10
Submitted Nov 6 at 11:26am
This attempt took 1,373 minutes.

Question 1 — 0 / 0 pts

What kind of data are necessary for maintaining open files in operating systems. Where is the data kept?

Your Answer:

Information about files is kept in the directory structure; maintained by the OS on the disk. In order toopen a file, the OS needs to know:

□file position: pointer to last read/write position.

□file-open count: number of times a file has been opened.

□disk location of the file: cached in memory, so it doesn't need to be read from file all the time.

□access rights: per-process access mode information

Question 2

0 / 0 pts

Describe the ways for an operating system to tell the type of a file?

Your Answer:

An OS can use an extension, internal meta-data, a magic number stored at the beginning of the file, or areference to creating the application to determine the type of file.

Question 3

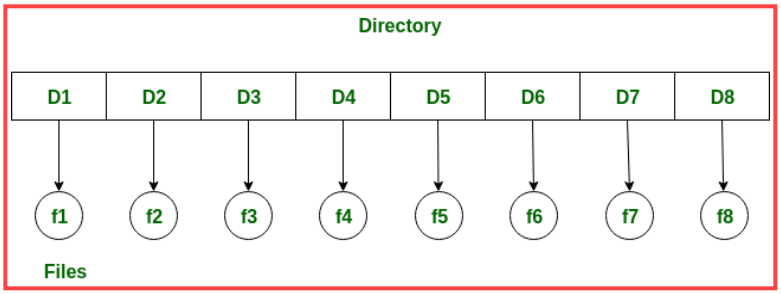
0 / 0 pts

Describe with details all types of directory structures that were discussed in the lecture.

Your Answer:

Single-Level Directory

- A single directory for all users



- Many problems with:

- searching

-- plenty of files

- naming

-- no duplicates

● grouping

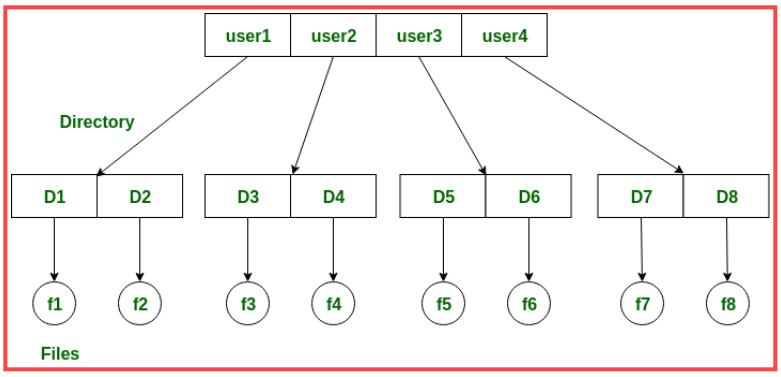
-- no support other as by name

● security

-- shared space

Two-Level Directory Structure

● Separate directory for each user



● Use a path name rather than just a file name

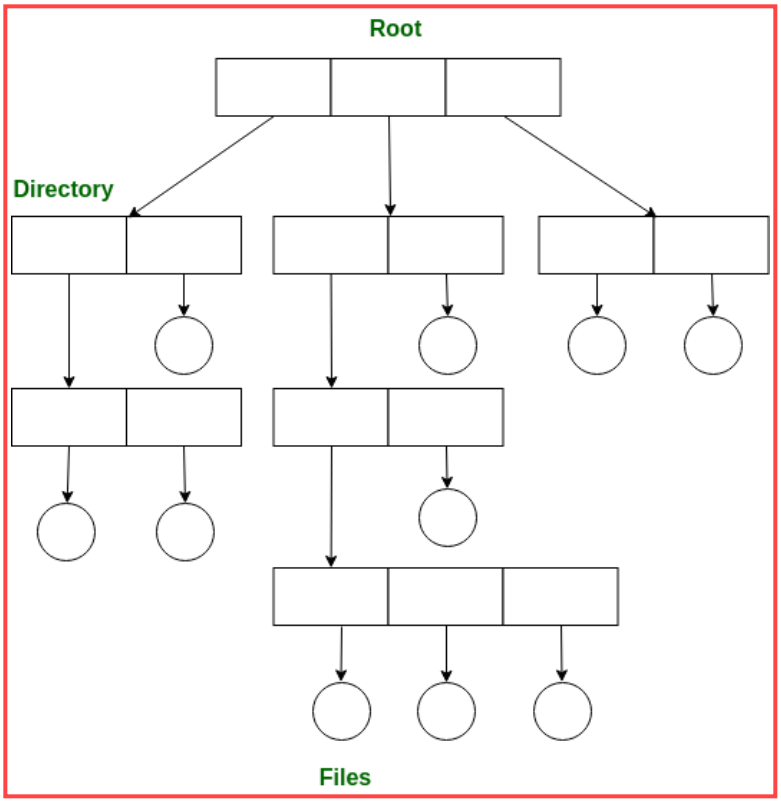
● Can have the same file name for different user

● Efficient searching per user

● ...but, still not enough flexibility

- e.g., no grouping capability

Tree-structured directory –
Once we have seen a two-level directory as a tree of height 2, the natural generalization is to extend the directory structure to a tree of arbitrary height.
This generalization allows the user to create there own subdirectories and to organize on their files accordingly.



A tree structure is the most common directory structure. The tree has a root directory, and every file in the system have a unique path.

Advantages:

- Very generalize, since full path name can be given.
- Very scalable, the probability of name collision is less.
- Searching becomes very easy, we can use both absolute path as well as relative.

Disadvantages:

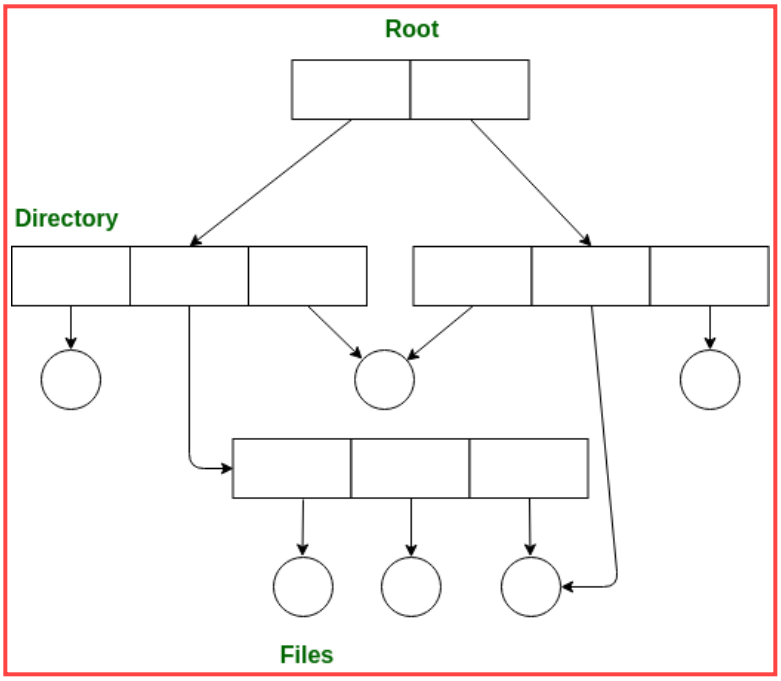
- Every file does not fit into the hierarchical model, files may be saved into multiple directories.
- We can not share files.
- It is inefficient, because accessing a file may go under multiple directories.

Acyclic graph directory –
An acyclic graph is a graph with no cycle and allows to share subdirectories and files. The same file or subdirectories may be in two different directories. It is a natural generalization of the tree-structured directory.

It is used in the situation like when two programmers are working on a joint project and they need to access files. The associated files are stored

in a subdirectory, separating them from other projects and files of other programmers, since they are working on a joint project so they want the subdirectories to be into their own directories. The common subdirectories should be shared. So here we use Acyclic directories.

It is the point to note that shared file is not the same as copy file . If any programmer makes some changes in the subdirectory it will reflect in both subdirectories.



Advantages:

- We can share files.
- Searching is easy due to different-different paths.

Disadvantages:

- We share the files via linking, in case of deleting it may create the problem,
- If the link is softlink then after deleting the file we left with a dangling pointer.
- In case of hardlink, to delete a file we have to delete all the reference associated with it.

Question 4

0 / 0 pts

Allowing a general graph to represent a directory structure has a number of pitfalls. Describe them along with potential solutions.

Your Answer:

he pitfall with allowing a general graph to represent a directory structure is the introduction of cycles, which can lead to infinite traversal of a directory. If a directory contains a link to itself or even another directory which leads to a cycle, then infinite traversal is a possibility. There is a way to guarantee no cycles though, if you make it so that files are the only things allowed to be linked, then that eliminates the possibility of cycles. Every time a new link is added use a cycle detection algorithm to determine whether it is ok. Unfortunately, this doesn't fix the problem with dangling pointers, which can be fixed using a reference count like acyclic graphs

Question 5

0 / 0 pts

Thoroughly explain a difference between Unix soft links and hard links to files.

Your Answer:

What is Soft Link And Hard Link In Linux?

A **symbolic** or **soft link** is an actual link to the original file, whereas a **hard link** is a mirror copy of the original file. If you delete the original file, the soft link has no value, because it points to a non-existent file. But in the case of hard link, it is entirely opposite. Even if you delete the original file, the hard link will still has the data of the original file. Because hard link acts as a mirror copy of the original file.

In a nutshell, a soft link

- can cross the file system,
- allows you to link between directories,

- has different inode number and file permissions than original file,
- permissions will not be updated,
- has only the path of the original file, not the contents.

A hard Link

- can't cross the file system boundaries (i.e. A hardlink can only work on the same filesystem),
- can't link directories,
- has the same inode number and permissions of original file,
- permissions will be updated if we change the permissions of source file,
- has the actual contents of original file, so that you still can view the contents, even if the original file moved or removed.
-

Question 6

0 / 0 pts

Following the code distributed with the lecture notes and discussed during the lecture, write a program that recursively traverses the directory hierarchy of a file system and finds the largest file in the whole file system.

Will your program work if a general graph is used for the directory structure? How could the problem be addressed to solve the potential cycle problem?

Your Answer:

Once you obtain information about the files, and getting information from the current folder loop. The best way to do it would be a recursive call.

You would put the recursive call if you have an entry that is a directory. You know you are in a directory because DT_DIR: iis called in the loop, then you would call recursively the same function. This makes it guaranteed that I go through the whole hierarchy.

General Graph Directory

● We allow for any graph to represent a directory

● Problem with possibility of introducing cycles

● How can we guarantee no cycles?

- allow only links to files, and not directories

-- files are terminal nodes in the graph (still problem with dangling pointers)

- every time a new link is added use a cycle detection algorithm to determine whether it is OK

-- expensive!

To fix the cycle problem, we either need to remember the path, and check if the path is repeating itself. Or an algorithm that detects cycles in the whole file systems. Neither of them would be good solutions.... Its usually pushed on the user not to have cycle problems.

Question 7

0 / 0 pts

1. Show Unix commands that change access modes to protect the following files:
   - file named "A" (the owner can do anything, your group can only read, nobody else can see)
   - file named "B" (everybody can execute, but only the owner can write)
   - file named "C" (everybody has total access to this non-executable)
2. Show a Unix command that changes the access mode to protect directory named "D", so only you can open it.

Your Answer:

A: chmod u+rwx A

chmod g+r A

chmod o-rwx A

B: chmod a+x B

chmod u+w B

C: chmod a+rw B

2.

chmod u+rw D

chmod g-rwx D

chmod o-rwx D

Question 8

0 / 0 pts

Describe how to mount and unmount file systems on Unix. How to list the mounted file systems.

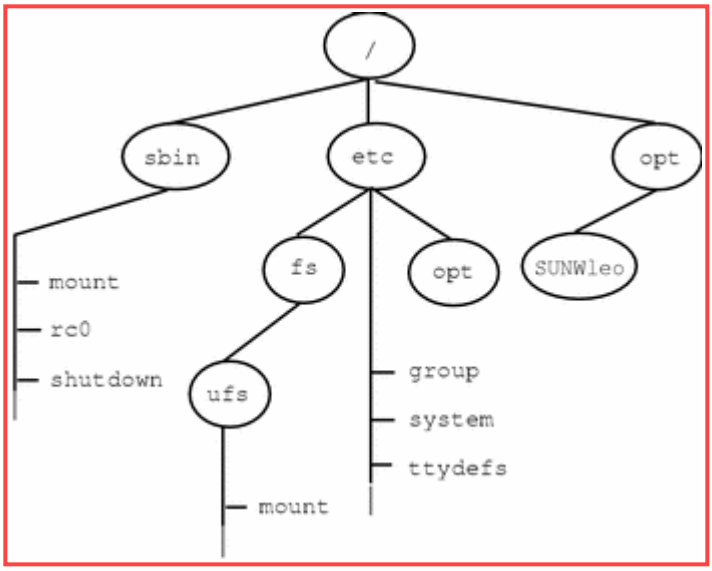
Hint: explore mount, umount, and df Unix commands.

Your Answer:

Before you can access the files on a file system, you need to mount the file system. Mounting a file system attaches that file system to a directory (**mount point**) and makes it available to the system. The root (/) file system is always mounted. Any other file system can be connected or disconnected from the root (/) file system.

When you mount a file system, any files or directories in the underlying mount point directory are unavailable as long as the file system is mounted. These files are not permanently affected by the mounting process, and they become available again when the file system is unmounted. However, mount directories are typically empty, because you usually do not want to obscure existing files.

For example, the figure below shows a local file system, starting with a root (`/`) file system and subdirectories `sbin`, `etc`, and `opt`.



Question 9

0 / 0 pts

What is the difference between the following commands? Which one always requires super user access?

```
> chmod
> chgrp
> chown
```

Your Answer:

chmod changes the permissions of a file directory, chgrp changes group ownership, and chown changes files owner and group. chowm always needs super user access since you might not own the file

Question 10

0 / 0 pts

A user executes the following commands in bash in an empty directory:

```
> umask 77
> touch test_file_A
> umask 27
> touch test_file_B
> umask 2
> touch test_file_C
```

What will be the system response to the following command:

```
> ls -l
```

Explain how did you derive the answer by describing the results of each command.

Your Answer:

unmask 77- determines default permissions for creating new files
touch test_file_A - creates test_file_A
unmask 27- determines default permissions for creating new files.
touch test_file_B - creates test_file_B
unmask 2- determines default permissions for creating new files
touh test_file_C - creates test_file_C

ls -l
this will display permissions and status

Question 11

0 / 0 pts

Explain the meaning and ranslate the following access modes to their letter counterparts:

```
777
600
400
644
666
```

Your Answer:

777= rwx rwx rwx
600= rw- --- ---
400= -w- --- ---
644= rw- -w- -w-
666= rw- rw- rw-

-binary

Question 12

0 / 0 pts

Explain the meaning and translate the following access modes to their octal counterparts:

```
rwx------
rwxr--r--
r--r--r--
rw-rw-rw-
rwxrwxrwx
rw--w--w-
```

Your Answer:

1. The owner has access to read,write, and execute, while the group and world has none
2. owner has access to read write and execute, group has read and world has read
3. everyone has read
4. everyone has read and write
5. everyone has access to read write and execute

6. owner has access to read and write. group and world has access to write

Unanswered

Question 13

10 / 10 pts

I have submitted answers to all questions in this study set.

- True
- False

Quiz Score: **10** out of 10

Study Set for Lecture 12: File System Implementation

Due Nov 16 at 11:59pm **Points** 10 **Questions** 12
Available Nov 10 at 12pm - Dec 8 at 8pm 28 days **Time Limit** None
Allowed Attempts Unlimited

Instructions

Review lecture notes from **lect12_File_System_Implementation.pdf**.

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

Take the Quiz Again

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	69 minutes	10 out of 10

Correct answers are hidden.

Score for this attempt: **10** out of 10
Submitted Nov 16 at 11:11am
This attempt took 69 minutes.

Question 1 — 0 / 0 pts

Describe with details typical memory-resident structures that support file system after it has been mounted.

What is a File Control Block (FCB)? What is its purpose?

Using the memory organization and data structures that you have just described, provide pseudocode for opening a file, and for reading from an

opened file.

Your Answer:

The two main memory-resident structures that support the file system after mounting are the directory structure and the open file tables. The directory is used to indicate where file contents are on the disk. The open file tables hold file descriptors of opened files, can be either system-wide open or only show files open per process.

A **File Control Block** (**FCB**) is a file system structure in which the state of an open **file** **(https://en.wikipedia.org/wiki/Computer_file)** is maintained. A FCB is managed by the operating system, but it resides in the memory of the program that uses the file, not in operating system memory. This allows a process to have as many files open at one time as it wants to, provided it can spare enough memory for an FCB per file. A file control block is an on-disk file system structure that contains details about and individual file or directory.

if (file is in the directory and has a file control block)

if (the file is already opened)

return file handle

else

create entry in the local/global file table for the file using the directory entry and the info in the file control block

return handle for the new entry

Question 2

0 / 0 pts

Describe the algorithms to

1. add
2. remove

a file or folder to a file system directory based on hashing functions.

Discuss its advantages and disadvantages.

Your Answer:

Hash Table

● direct access through a hash function
applied to the name
● decreases directory search time
● must deal with the collisions
● efficient solutions do exist though
● fixed size

1. **Hash Table –**
   The hash table takes a value computed from the file name and returns a pointer to the file. It decreases the directory search time. The insertion and deletion process of files is easy. The major difficulty is hash tables are its generally fixed size and hash tables are dependent on hash function on that size.

How can one store a large number of files while maintaining a high level of performance during access? One solution is file name hashing.

File name hashing in the simplest terms can be defined as, creating a known and reproducible path, based on the name of the file.

Lets do a test. The following java code will create a 2-level directory hash for the String “cat.gif”.

```
public static void main(String[] args) {

    String fileName = "cat.gif";
    int hash = fileName.hashCode();    int mask = 255;
    int firstDir = hash & mask;
    int secondDir = (hash >> 8) & mask;    String path = new StringBui
lder(File.separator)
            .append(String.format("%03d", firstDir))
            .append(File.separator)
            .append(String.format("%03d", secondDir))
            .append(File.separator)
            .append(fileName)
            .toString();    System.out.println(path);
}
```

The code generates the following output:

```
/172/029/cat.gif
```

If we ever need to find “cat.gif”, we can easily reproduce this path using the same algorithm.

Question 3

0 / 0 pts

Describe the algorithms to

1. add
2. remove

a file or folder to a file system using a linear directory.

Discuss its advantages and disadvantages.

Your Answer:

1. **Linear List –**
   It maintains a linear list of filenames with pointers to the data blocks.It is time-consuming also. To create a new file, we must first search the directory to be sure that no existing file has the same name then we add a file at end of the directory. To delete a file, we search the directory for the named file and release the space. To reuse the directory entry either we can mark the entry as unused or we can attach it to a list of free directories.

Question 4

0 / 0 pts

Describe the algorithms to

1. add
2. remove

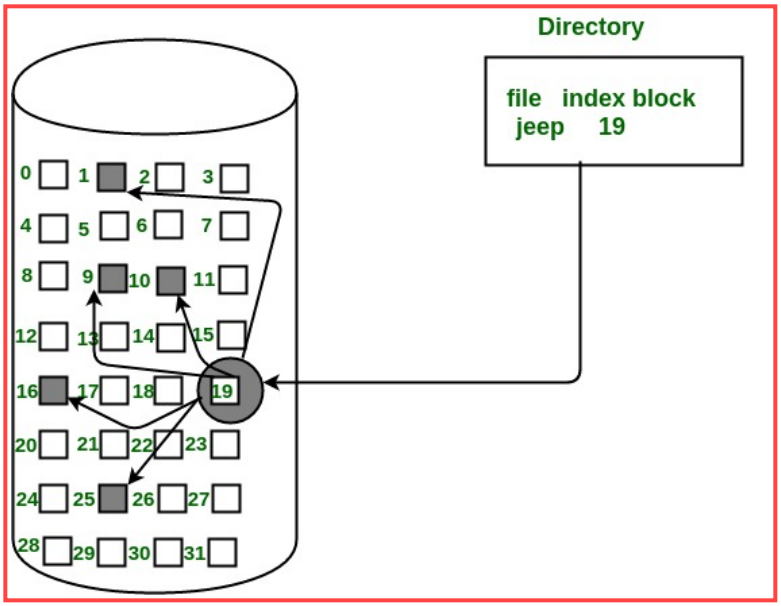
a file or folder to a file system using indexed allocation scheme for allocating storage space.

Discuss its advantages and disadvantages.

Your Answer:

3. Indexed Allocation

In this scheme, a special block known as the **Index block** contains the pointers to all the blocks occupied by a file. Each file has its own index block. The ith entry in the index block contains the disk address of the ith file block. The directory entry contains the address of the index block as shown in the image:



Advantages:

- This supports direct access to the blocks occupied by the file and therefore provides fast access to the file blocks.
- It overcomes the problem of external fragmentation.

Disadvantages:

- The pointer overhead for indexed allocation is greater than linked allocation.
- For very small files, say files that expand only 2-3 blocks, the indexed allocation would keep one entire block (index block) for the pointers which is inefficient in terms of memory utilization. However, in linked allocation we lose the space of only 1 pointer per block.

Question 5

0 / 0 pts

Describe with details how to use linked lists for managing free space in a file system.

What are the advantages and disadvantages of this approach?

Your Answer:

Linked list-based free space management involves starting from the head of the list and traversing through the list starting from the head. Because of this traversals are very expensive, but fortunately traversing is not needed frequently, since it's a list of free blocks so the first block will usually be sufficient. No space is wasted since there won't be any disconnections as long as the list is properly managed and there are no gaps. But unfortunately it's hard to get contiguous space in a file system, and contiguous space speeds up reading and writing.

Question 6

0 / 0 pts

Describe with details how to use a bit vector for managing free space in a file system.

What are the advantages and disadvantages of this approach?
(i.e., compare to other methods of managing free storage)

Your Answer:

Bitmap or Bit vector –
A Bitmap or Bit Vector is series or collection of bits where each bit corresponds to a disk block. The bit can take two values: 0 and 1: *0 indicates that the block is allocated* and 1 indicates a free block.
The given instance of disk blocks on the disk in *Figure 1* (where green blocks are allocated) can be represented by a bitmap of 16 bits as: **0000111000000110**.

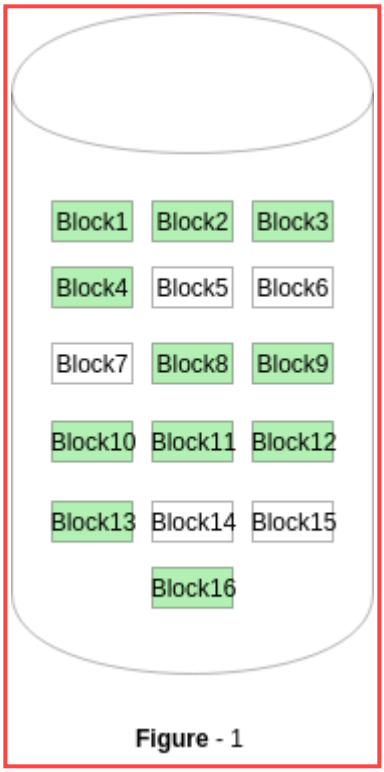


Figure - 1

When the **operating system** **(https://en.wikipedia.org/wiki/Operating_system)** (OS) needs to write a file, it will scan the bitmap until it finds enough free locations to fit the file. If a 12 KiB file were stored on the example drive, three zero bits would be found, changed to ones, and the data would be written across the three sectors represented by those bits. If the file were subsequently truncated down to 8 KiB, the final sector's bit would be set back to zero, indicating that it is again available for use.

Advantages

- Simple: Each bit directly corresponds to a sector
- Fast random access allocation check: Checking if a sector is free is as simple as checking the corresponding bit
- Fast deletion: Data need not be overwritten on delete; flipping the corresponding bit is sufficient
- Fixed cost: Both an advantage and disadvantage. Other techniques to store free space information have a variable amount of overhead depending on the number and size of the free space extents. Bitmaps can never do as well as other techniques in their respective ideal circumstances, but don't suffer pathological cases either. Since the bitmap never grows, shrinks or moves, fewer lookups are required to find the desired information
- Low storage overhead as a percentage of the drive size: Even with relatively small sector sizes, the storage space required for the bitmap

is small. A 2 TiB (https://en.wikipedia.org/wiki/Tebibyte) drive could be fully represented with a mere 64 MiB (https://en.wikipedia.org/wiki/Mebibyte) bitmap.

Disadvantages

- Wasteful on larger disks: The simplistic design starts wasting large amounts of space (in an absolute sense) for extremely large volumes[1] (https://en.wikipedia.org/wiki/Free_space_bitmap#cite_note-bonwick-1)
- Poor scalability: While the size remains negligible as a percentage of the disk size, finding free space becomes slower as the disk fills. If the bitmap is larger than available memory (https://en.wikipedia.org/wiki/Random-access_memory), performance drops precipitously on all operations[1] (https://en.wikipedia.org/wiki/Free_space_bitmap#cite_note-bonwick-1)
- Fragmentation (https://en.wikipedia.org/wiki/File_system_fragmentation): If free sectors are taken as they are found, drives with frequent file creation and deletion will rapidly become fragmented. If the search attempts to find contiguous blocks, finding free space becomes much slower for even moderately full disks.

NOTE - Other way

Linked List –
In this approach, the free disk blocks are linked together i.e. a free block contains a pointer to the next free block. The block number of the very first disk block is stored at a separate location on disk and is also cached in memory.

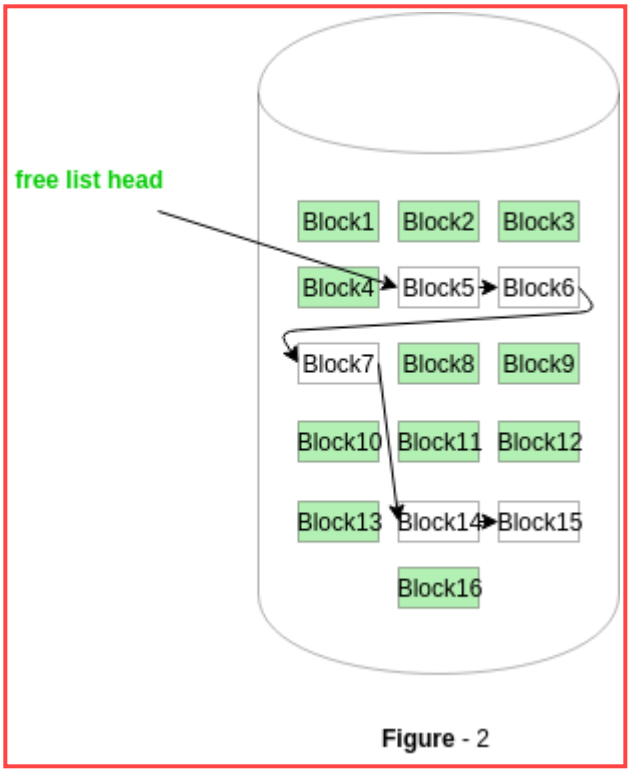


Figure - 2

In *Figure-2*, the free space list head points to Block 5 which points to Block 6, the next free block and so on. The last free block would contain a null pointer indicating the end of free list.
A drawback of this method is the I/O required for free space list traversal.

Question 7

0 / 0 pts

If you have a disk with 1,048,576 blocks, how many bytes do you need to allocate space for the bit vector in a file system using a bit vector-based management of free space assuming that a single bit in a byte indicates the status (free or taken) of one disk block?

Assuming that bit `1` is used to indicate a free block, and bit `0` to indicate a taken block, write a C function that implements the algorithm to find, and to claim, the first (from the beginning of the logical address space) free block. The function should:

- use bit-wise and bit shifting operations to run fast,
- flip the corresponding bit to indicate that the block is taken (not free anymore), and
- return the LBA (logical block address) of the free block to use.

Your Answer:

1 byte = 8 bits

and 1 bit representing the status (free or taken) of 1 block in disk.

So,

Total number of bits required = number of blocks = 1048576

Converting into bytes = 1048576/8 = 131072 bytes

C function to find the block number of first free block:-

```
// Assuming size of "int" data type is 4 bytes.
// So value of n = 32768.
int find(int bitmap[], int n)
{
   for(int i = 0; i < n; i++)
   {
      // If all the bits of the integer presented at i'th index are '1' means,
      // All blocks are taken represented by bits of this integer.
      // Otherwise some blocks are free.
      if(bitmap[i] < 4294967295)
      {
         // Assuming that system is big endian.
         int j = 31;
         while(j >= 0 && ((1 << j) && bitmap[i]) != 0)
         {
            j--;
         }
         return (i*32 + 31 - j);

         /* If system is little endian, remove the above code and uncomment
below code segment.
         int j = 0;
         while(j < 32 && ((1 << j) && bitmap[i]) != 0)
         {
            j++;
         }
         return (i*32 + j);
         */
      }
```

```
  }
}
```

Question 8

0 / 0 pts

Describe what Virtual File System is. What's its purpose? How is it used?

Your Answer:

A virtual file system (VFS) or virtual filesystem switch is an abstraction layer on top of a more concrete file system. The purpose of a VFS is to allow client applications to access different types of concrete file systems in a uniform way. A VFS can, for example, be used to access local and network storage devices transparently without the client application noticing the difference. It can be used to bridge the differences in Windows, classic Mac OS/macOS and Unix filesystems, so that applications can access files on local file systems of those types without having to know what type of file system they are accessing.

A VFS specifies an interface (or a "contract") between the kernel and a concrete file system. Therefore, it is easy to add support for new file system types to the kernel simply by fulfilling the contract. The terms of the contract might change incompatibly from release to release, which would require that concrete file system support be recompiled, and possibly modified before recompilation, to allow it to work with a new release of the operating system; or the supplier of the operating system might make only backward-compatible changes to the contract, so that concrete file system support built for a given release of the operating system would work with future versions of the operating system.

Question 9

0 / 0 pts

Describe with details how a log-structured file system works.

Your Answer:

Log structured file systems write information to media into no blocks without rewriting. The file structure is a circular list buffer where new or modified information is added always at the head and when a new version is written, then the old data is marked as obsolete. A "garbage collector is run to make room at the tail. If there are no blocks with obsolete information then the disk is full. Even if the file system crashes then the system is guaranteed to remain consistent since the old data is still there and will only be made obsolete if the new data is successfully written. It is possible to use logging in parts of a file system.

SUMMARY of why we use log-structured file systems:

Summary: The File Systems That We Have Loved
• The original Unix file system had a simple
design, but was slow
- Data layout did not provide spatial locality for
directories and files with temporal locality
- Only provided 4% of the sequential disk
bandwidth
• FFS leveraged knowledge of disk geometry
to improve performance
- To reduce seeks:
-Related files and directories are stored in the
same cylinder group
-Block size increased from 512 bytes to 4KB
- To minimize rotational latency, FFS used “skip
sectors”

• However, FFS had slow failure recovery due
to the horrifying multi-pass nature of fsck

• Journaling file systems use write-ahead logging
to make crash recovery faster
- ext3 performs redo logging of physical blocks
- NTFS performs redo+undo logging of operations
- The journal converts random writes into sequential
ones, but the file system must eventually issue
random writes to perform checkpoints

• LFS turns the entire file system into a log
- Assumes that the buffer cache will handle most reads, so making writes fast is the most important thing
- The log turns all writes (random or sequential, large or small) into large sequential writes
- Fast recovery
- Requires garbage collection, which (depending on the workload) may eliminate benefits from making all writes sequential

Question 10

0 / 0 pts

Describe with details how journaling file system provides data protection from system crashes.

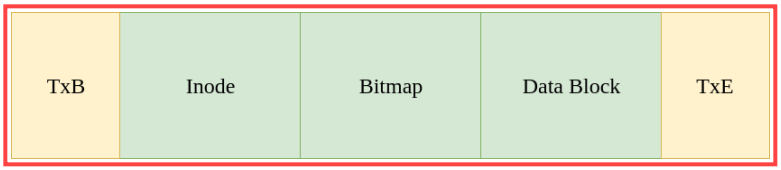
Your Answer:

ournaling, or **write-ahead logging** is a sophisticated solution to the problem of **file system inconsistency (https://www.geeksforgeeks.org/file-system-inconsistency/)** in operating systems. Inspired by database management systems, this method first writes down a summary of the actions to be performed into a “log” before actually writing them to the disk. Hence the name, “write-ahead logging”. In the case of a crash, the OS can simply check this log and pick up from where it left off. This saves multiple disk scans to fix inconsistency, as is the case with FSCK.

Good examples of systems that implement data journaling include Linux ext3 and ext4 file systems, and Windows NTFS.

Data Journaling:

A log is stored in a simple data structure called the journal. The figure below shows its structure, which comprises of three components.



1. **TxB (Transaction Begin Block):**
   This contains the transaction ID, or the TID.
2. **Inode, Bitmap and Data Blocks (Metadata):**
   These three blocks contain a copy of the contents of the blocks to be updated in the disk.
3. **TxE (Transaction End Block)**
   This simply marks the end of the transaction identified by the TID.

As soon as an update is requested, it is written onto the log, and thereafter onto the file system. Once all these writes are successful, we can say that we have reached the **checkpoint** and the update is complete.

What if a crash occurs during journaling ?

One could argue that journaling, itself, is not atomic. Therefore, how does the system handle an un-checkpointed write ? To overcome this scenario, journaling happens in two steps: simultaneous writes to TxB and the following three blocks, and then write of the TxE. The process can be summarized as follows.

1. **Journal Write:**
   Write TxB, inode, bitmap and data block contents to the journal (log).
2. **Journal Commit:**
   Write TxE to the journal (log).
3. **Checkpoint:**
   Write the contents of the inode, bitmap and data block onto the disk.

A crash may occur at different points during the process of journaling. If a crash occurs at step 1, i.e. before the TxE, we can simply skip this transaction altogether and the file system stays consistent.

If a crash occurs at step 2, it means that although the transaction has been logged, it hasn't been written onto the disk completely. We cannot be sure which of the three blocks (inode, bitmap and data block) were

actually updated and which ones suffered a crash. In this case, the system scans the log for recent transactions, and performs the last transaction again. This does lead to redundant disk writes, but ensures consistency. This process is called **redo logging**.

Question 11

0 / 0 pts

Describe with details how copy-on-write file system provides data protection from crashes.

Copy on Write.docx
(https://cilearn.csuci.edu/files/2295535/download)

Question 12

10 / 10 pts

I have submitted answers to all questions in this study set.

- (•) True
- ( ) False

Quiz Score: **10** out of 10

Study Set for Lecture 13: Mass Storage Structure

Due Dec 7 at 11:59pm **Points** 10 **Questions** 7
Available Nov 24 at 12pm - Dec 8 at 8pm 14 days **Time Limit** None
Allowed Attempts Unlimited

Instructions

Review lecture notes from **lect13_Mass_Storage_Structure.pdf**.

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

Take the Quiz Again

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	10,229 minutes	10 out of 10 *

* Some questions not yet graded

Correct answers are hidden.

Score for this attempt: **10** out of 10 *
Submitted Dec 6 at 11:18am
This attempt took 10,229 minutes.

Question 1

Not yet graded / 0 pts

Explain what NAS and SAN are.
Describe the difference between the two.

Your Answer:

Both network-attached storage (NAS) and storage area network (SAN) were developed to solve the problem of making stored data available to a

lot of users at once. Each of them provides dedicated storage for a group of users, but they couldn't be more different in their approach to achieving their mission.

A NAS is a single storage device that serves files over Ethernet and is relatively inexpensive and easy to set up, while a SAN is a tightly coupled network of multiple devices that work with block-based data and is more expensive and complex to set up and manage. From a user perspective, the biggest difference between NAS and SAN is that NAS devices look like volumes on a file server and use protocols like NFS and SMB/CIFS, while SAN-connected disks appear to the user as local drives.

Benefits of NAS

A NAS is frequently the next step up for a home office or small business that is using DAS (direct attached storage). The move up to NAS results from the desire to share files locally and remotely, having files available 24/7, data redundancy, the ability to replace and upgrade hard drives in the system, and and the availability of other services such as automatic backup.

Summary of NAS Benefits

- Relatively inexpensive
- 24/7 and remote data availability
- Good expandability
- Redundant storage architecture
- Automatic backups to other devices and cloud
- Flexibility

Limitations of NAS

The weaknesses of a NAS are related to scale and performance. As more users need access, the server might not be able to keep up and could require the addition of more server horsepower. The other weakness is related to the nature of Ethernet itself. By design, Ethernet transfers data from one place to another via packets, dividing the source into a number of segments and sending them along to their destination. Any of those packets could be delayed, or sent out of order, and might not be available to the user until all of the packets arrive and are put back in order.

Any latency (slow or retried connections) is usually not noticed by users for small files, but can be a major problem in demanding environments such as video production, where files are extremely large and latency of more than a few milliseconds can disrupt production steps such as rendering.

Benefits of SAN

Because it's considerably more complex and expensive than NAS, SAN is typically used by large corporations and requires administration by an IT staff. For some applications, such as video editing, it's especially desirable due to its high speed and low latency. Video editing requires fair and prioritized bandwidth usage across the network, which is an advantage of SAN.

A primary strength of a SAN is that all of the file access negotiation happens over Ethernet while the files are served via extremely high speed Fibre Channel, which translates to very snappy performance on the client workstations, even for very large files. For this reason SAN is widely used today in collaborative video editing environments.

Summary of SAN Benefits

- Extremely fast data access
- Dedicated network for storage relieves stress on LAN
- Highly expandable
- OS level (block level) access to files
- High quality-of-service for demanding applications such as video editing

Limitations of SAN

The challenge of SAN can be summed up in its cost and administration requirements — having to dedicate and maintain both a separate Ethernet network for metadata file requests and implement a Fibre Channel network can be a considerable investment. That being said, SANs are really the only way to provide very fast data access for a large number of users that also can scale to supporting hundreds of users at the same time.

What's the Diff: NAS vs SAN

NAS	SAN
Typically used in homes and small to medium sized businesses.	Typically used in professional and enterprise environments.
Less expensive	More expensive
Easier to manage	Requires more administration
Data accessed as if it were a network-attached drive (files)	Servers access data as if it were a local hard drive (blocks)
Speed dependent on local TCP/IP usually Ethernet network, typically 100 megabits to one gigabit per second. Generally slower throughput and higher latency due to slower file system layer.	High speed using Fibre Channel, 2 gigabits to 128 gigabits per second. Some SANs use iSCSI as a less expensive but slower alternative to Fibre Channel.
I/O protocols: NFS, SMB/CIFS, HTTP	SCSI, iSCSI, FCoE
Lower-end not highly scalable; high-end NAS scale to petabytes using clusters or scale-out nodes	Network architecture enables admins to scale both performance and capacity as needed
Does not work with virtualization	Works with virtualization
Requires no architectural changes	Requires architectural changes
Entry level systems often have a single point of failure, e.g. power supply	Fault tolerant network with redundant functionality
Susceptible to network bottlenecks	Not affected by network traffic bottlenecks. Simultaneous access to cache, benefiting applications such as video editing.
File backups and snapshots economical and schedulable.	Block backups and mirrors require more storage.

Question 2

Not yet graded / 0 pts

Implement two functions that translate a logical block address (LBA) into a cylinder-head-sector (CHS) address, and reverse, translating a CHS address into its LBA equivalent.

```
typedef struct chs {
   int cylinder;
   int head;
   int sector;
} CHS_TYPE;

typedef long LBA_TYPE;

CHS_TYPE lba2chs(LBA_TYPE lba);

LBA_TYPE chs2lba(CHS_TYPE chs);
```

Using the algorithms, compute the location of the logical block 162656 in a hard drive with 200 cylinders, 128 sectors, and 5 two-sided platters.

Two-sided platters are served by two heads; one for each side.

Assume that a logical block is the same size as a sector and that there are no bad sectors on this perfect disk.

Your Answer:

STRUCT lba2chs(long addr)

{

STRUCT struct;

struct.cyl = addr/sectors per cylinder;

struct.head = (addr % sectors per cylinder) / sectors per track;

struct.sect = (addr % sectors per cylinder) % sectors per track;

return struct;

}

long chs2lba (STRUCT struct)

{

long logic = 0;

logic += struct.sect;

logic += struct.head * sectors per track;

logic += struct.cyl * sectors per cylinder

return logic;

}

The cylinder value is the number of tracks on one side of each platter.

200 cylinder means there 200 tracks on one side of the platter.

The number of sectors is 128 per track.

So total sector on each platter is 128*200= 25600 sectors per platter

we have to find the location of 162656 logical block (sector)

each platter can store 25600*2=51200 sectors or logical block

1st platter=51200

2nd platter =51200

3rd platter =51200

so total upto 3 platter including both side we get
51200+51200+51200=153600

we have to find the location of 162656 so it will be on the 4th platter.

Question 3

Not yet graded / 0 pts

Explain why some aspects of IO scheduling are addressed by the operating system rather than the disk controller. Give concrete examples.

What aspects of optimization would be more appropriate in the disk controller?

Your Answer:

The operating system knows about contraints that the disc controller does not, such as that demand paging takes priority over application I/O, or that writing is more important than reading if the cache is almost full.

The disk controllers can perform optimization on request ordering, coalescing, and sector numbering schemes

Question 4

Not yet graded / 0 pts

Describe:

1. advantages and disadvantages of solid state drives (SSDs) in comparison with disk hard drives,
2. what wear leveling is,
3. why is wear leveling used in the current SSDs, and
4. why wear leveling is a problem for ensuring information security.

Your Answer:

SSDs have faster starting, reading, and writing times with DRAM cache, has no noise that comes from a spinning disk, has decent mechanical reliability depending on the level type of the cells, file fragmentation is not

a problem since SSDs use direct access to any location, uses less power usually, and since problems usually happen with writes, writes can be repeated in good cells. Unfortunately they are still very expensive, have relatively low capacity, and have limited write cycles.

Wear leveling is circularly rewriting to every cell on the disk so that there is uniformal usage of all cells instead of having hotspots of rewriting. **Wear leveling** (also written as **wear levelling**) is a technique[1] (https://en.wikipedia.org/wiki/Wear_leveling#cite_note-Fundamental_Patent-1) for prolonging the **service life** (https://en.wikipedia.org/wiki/Service_life) of some kinds of erasable **computer storage** (https://en.wikipedia.org/wiki/Computer_storage) media, such as **flash memory** (https://en.wikipedia.org/wiki/Flash_memory), which is used in **solid-state drives** (https://en.wikipedia.org/wiki/Solid-state_drive) (SSDs) and **USB flash drives** (https://en.wikipedia.org/wiki/USB_flash_drive), and **phase-change memory** (https://en.wikipedia.org/wiki/Phase-change_memory). There are several wear leveling mechanisms that provide varying levels of longevity enhancement in such memory systems.

To reduce wear on current SSDs, wear leveling is used to reduce the number of hotspots.

Wear leveling causes problems in information security since you are rewriting to different cells, the old information stays in the old cell.

Question 5

Not yet graded / 0 pts

Assume that there are 256 cylinders in a disk and the current disk controller request queue is as follows:

```
67 223 45 183 233 12 238 54 29 218 156 48 22 191 78 73 224 251 2 15
184 101
```

Show the order in which the requests will be carried out for each of the following scheduling methods:

1. FCFS

2. SSTF (shortest-seek-time-first)
3. LOOK (elevator algorithm)
4. C-LOOK (circular unidirectional elevator algorithm)

NOTE: In `SCAN`, `C-SCAN`, `LOOK` and `C-LOOK`, assume that the first number is the current head position and that the head starts to move in the direction of increasing cylinder numbers.

Your Answer:

First Come First Serve (FCFS)

FCFS is the simplest **disk scheduling algorithm (https://www.geeksforgeeks.org/disk-scheduling-algorithms/)** . As the name suggests, this algorithm entertains requests in the order they arrive in the disk queue. The algorithm looks very fair and there is no starvation (all requests are serviced sequentially) but generally, it does not provide the fastest service.

Algorithm:

1. Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. ‘head’ is the position of disk head.
2. Let us one by one take the tracks in default order and calculate the absolute distance of the track from the head.
3. Increment the total seek count with this distance.
4. Currently serviced track position now becomes the new head position.
5. Go to step 2 until all tracks in request array have not been serviced.

Total number of seek operations = 2542
Seek Sequence is
223
45
183
233
12
238
54
29
218
156
48
22

191
78
73
224
251
2
15
184
101

Shortest Seek Time First (SSTF) –
Basic idea is the tracks which are closer to current disk head position should be serviced first in order to *minimise the seek operations*.

Advantages of Shortest Seek Time First (SSTF) –

1. Better performance than FCFS scheduling algorithm.
2. It provides better throughput.
3. This algorithm is used in Batch Processing system where throughput is more important.
4. It has less average response and waiting time.

Disadvantages of Shortest Seek Time First (SSTF) –

1. Starvation is possible for some requests as it favours easy to reach request and ignores the far away processes.
2. Their is lack of predictability because of high variance of response time.
3. Switching direction slows things down.

Total number of seek operations = 382
Seek Sequence is
67
73
78
101
54
48
45
29
22
15

12
2
156
183
184
191
218
223
224
233
238
251

LOOK Disk Scheduling Algorithm:
LOOK is the advanced version of **SCAN (elevator) disk scheduling algorithm (https://www.geeksforgeeks.org/scan-elevator-disk-scheduling-algorithms/)** which gives slightly better seek time than any other algorithm in the hierarchy *(FCFS->SRTF->SCAN->C-SCAN->LOOK)*. The LOOK algorithm services request similarly as SCAN algorithm meanwhile it also “looks” ahead as if there are more tracks that are needed to be serviced in the same direction. If there are no pending requests in the moving direction the head reverses the direction and start servicing requests in the opposite direction.
The main reason behind the better performance of LOOK algorithm in comparison to SCAN is because in this algorithm the head is not allowed to move till the end of the disk.

Algorithm:

1. Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. ‘head’ is the position of disk head.
2. The initial direction in which head is moving is given and it services in the same direction.
3. The head services all the requests one by one in the direction head is moving.
4. The head continues to move in the same direction until all the request in this direction are not finished.
5. While moving in this direction calculate the absolute distance of the track from the head.

6. Increment the total seek count with this distance.
7. Currently serviced track position now becomes the new head position.
8. Go to step 5 until we reach at last request in this direction.
9. If we reach where no requests are needed to be serviced in this direction reverse the direction and go to step 3 until all tracks in request array have not been serviced.

Initial position of head: 67
Total number of seek operations = 433
Seek Sequence is
73
78
101
156
183
184
191
218
223
224
233
238
251
54
48
45
29
22
15
12
2

C-LOOK (Circular LOOK) Disk Scheduling Algorithm:
C-LOOK is an enhanced version of both **SCAN** as well as **LOOK** disk scheduling algorithms. This algorithm also uses the idea of wrapping the tracks as a circular cylinder as C-SCAN algorithm but the seek time is better than C-SCAN algorithm. We know that C-SCAN is used to avoid starvation and services all the requests more uniformly, the same goes for

C-LOOK.
In this algorithm, the head services requests only in one direction(either left or right) until all the requests in this direction are not serviced and then jumps back to the farthest request on the other direction and service the remaining requests which gives a better uniform servicing as well as avoids wasting seek time for going till the end of the disk.

Algorithm-

1. Let Request array represents an array storing indexes of the tracks that have been requested in ascending order of their time of arrival and **head** is the position of the disk head.
2. The initial direction in which the head is moving is given and it services in the same direction.
3. The head services all the requests one by one in the direction it is moving.
4. The head continues to move in the same direction until all the requests in this direction have been serviced.
5. While moving in this direction, calculate the absolute distance of the tracks from the head.
6. Increment the total seek count with this distance.
7. Currently serviced track position now becomes the new head position.
8. Go to step 5 until we reach the last request in this direction.
9. If we reach the last request in the current direction then reverse the direction and move the head in this direction until we reach the last request that is needed to be serviced in this direction without servicing the intermediate requests.
10. Reverse the direction and go to step 3 until all the requests have not been serviced.

Initial position of head: 67
Total number of seek operations = 485
Seek Sequence is
73
78
101
156
183
184
191

218
223
224
233
238
251
2
12
15
22
29
45
48
54

SCAN (Elevator) algorithm

In SCAN disk scheduling algorithm, head starts from one end of the disk and moves towards the other end, servicing requests in between one by one and reach the other end. Then the direction of the head is reversed and the process continues as head continuously scan back and forth to access the disk. So, this algorithm works as an elevator and hence also known as the elevator algorithm. As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.

Total number of seek operations = 318
Seek Sequence is
54
48
45
29
22
15
12
2
0
73
78
101
156
183

184
191
218
223
224
233
238
251

What is C-SCAN (Circular Elevator) Disk Scheduling Algorithm?
Circular SCAN (C-SCAN) scheduling algorithm is a modified version of SCAN disk scheduling algorithm that deals with the inefficiency of SCAN algorithm by servicing the requests more uniformly. Like SCAN (Elevator Algorithm) C-SCAN moves the head from one end servicing all the requests to the other end. However, as soon as the head reaches the other end, it immediately returns to the beginning of the disk without servicing any requests on the return trip (see chart below) and starts servicing again once reaches the beginning. This is also known as the "Circular Elevator Algorithm" as it essentially treats the cylinders as a circular list that wraps around from the final cylinder to the first one.

Algorithm:

1. Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. 'head' is the position of disk head.
2. The head services only in the right direction from 0 to size of the disk.
3. While moving in the left direction do not service any of the tracks.
4. When we reach at the beginning(left end) reverse the direction.
5. While moving in right direction it services all tracks one by one.
6. While moving in right direction calculate the absolute distance of the track from the head.
7. Increment the total seek count with this distance.
8. Currently serviced track position now becomes the new head position.
9. Go to step 6 until we reach at right end of the disk.
10. If we reach at the right end of the disk reverse the direction and go to step 3 until all tracks in request array have not been serviced.

Initial position of head: 67
Total number of seek operations = 238
Seek Sequence is
73
78
101
156
183
184
191
199
218
223
224
233
238
251
0
2
12
15
22
29
45
48
54

Question 6

Not yet graded / 0 pts

Design a data structure (or data structures) necessary to hold I/O requests for disk scheduler that can use FIFO (first-in-first-out; a.k.a.

NOOP), SSTF (shortest-seek-time-first), LOOK (elevator), and C-LOOK (circular elevator), and then write a program that implements the scheduler.

The program is given the input that specifies:

- the name of an algorithm to use, followed by
- the current head position given as a cylinder number, and finally followed by
- a sequence of I/O requests also given as corresponding cylinder numbers.

For example:

```
FIFO 67 223 45 183 233 12 238 54 29 218 156 48 22 191 78 73 224 251
2 15 184 101
```

states the program should use `FIFO` algorithm, the current head position is at cylinder `67`, and the remaining numbers are cylinder numbers associated with the I/O requests.

The output should be the sequence of requests ordered according to the specified algorithm.

For example, the sample input should generate the following output:

```
67 223 45 183 233 12 238 54 29 218 156 48 22 191 78 73 224 251 2 15
184 101
```

Your Answer:

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;


// function to print request for LOOK
void LOOK(int head, vector<int> request)
{
// vector for cylinders left and right of head
vector<int> left, right;
for(int i=0; i<request.size(); i++)
{
```

```
if(request[i] < head)
left.push_back(request[i]);
if(request[i] > head)
right.push_back(request[i]);
}

// sort the 2 vectors
sort(left.begin(), left.end());
sort(right.begin(), right.end());

// start looking towards right => print right
for(int i=0; i<right.size(); i++)
cout << " " << right[i];
// look towards left => print left but in reverse order
for(int i=left.size()-1; i>=0; i--)
cout << " " << left[i];
}



// function to print request for SSTF
void SSTF(int head, vector<int> request)
{
int n = request.size();
// array to account for visited tasks
bool visited[n];
for(int i=0; i<n; i++)
visited[i] = false;

// loop for n times, find min for each run
int min = INT_MAX, idx = -1;
for(int j=1; j<=n; j++)
{
for(int i=0; i<n; i++)
{
// compare if the difference to reach the cylinder is minimum
if(!visited[i] && abs(request[i] - head) < min)
{
min = abs(head - request[i]);
idx = i;
}
}
// print min
```

```
cout << " " << request[idx];
// update head and visited
head = request[idx];
visited[idx] = true;
// clear min and idx
min = INT_MAX; idx = -1;
}
}


// function to print request for CLOOK
void CLOOK(int head, vector<int> request)
{
// vector for cylinders left and right of head
vector<int> left, right;
for(int i=0; i<request.size(); i++)
{
if(request[i] < head)
left.push_back(request[i]);
if(request[i] > head)
right.push_back(request[i]);
}

// sort the 2 vectors
sort(left.begin(), left.end());
sort(right.begin(), right.end());

// start looking towards right => print right
for(int i=0; i<right.size(); i++)
cout << " " << right[i];
// look towards left => print left (start from minimum => same order)
for(int i=0; i<left.size(); i++)
cout << " " << left[i];
}


//MAIN:

int main()
{
// take input
string type;
```

```
int head, tmp;
vector<int> request;
cin >> type;
cin >> head;
while(cin >> tmp)
{
request.push_back(tmp);
}

// print head cylinder
cout << head;

// call function according to request type
if(type == "FIFO")
{
// simply print the request in same order
for(int i=0; i<request.size(); i++)
{
cout << " " << request[i];
}
}
else if(type == "SSTF")
{
SSTF(head, request);
}
else if(type == "LOOK")
{
LOOK(head, request);
}
else if(type == "C-LOOK")
{
CLOOK(head, request);
}
cout << endl;
return 0;
}
```

Question 7

10 / 10 pts

I have submitted answers to all questions in this study set.

- (•) True
- ( ) False

Quiz Score: **10** out of 10

Study Set for Lecture 14: I/O Systems

Due Dec 7 at 11:59pm **Points** 10 **Questions** 9
Available Nov 24 at 12pm - Dec 8 at 8pm 14 days **Time Limit** None
Allowed Attempts Unlimited

Instructions

Review lecture notes from **lect14_I_O_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 Quiz Again

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	9,479 minutes	10 out of 10 *

* Some questions not yet graded

ⓘ Correct answers are hidden.

Score for this attempt: **10** out of 10 *
Submitted Dec 6 at 9:55am
This attempt took 9,479 minutes.

Question 1 **Not yet graded / 0 pts**

Using bit fields in a C struct design a type for variables that can be used to encode the following controls for an imaginary serial port:

- a choice between full-duplex and half-duplex communication,
- enabling or disabling parity checking with selection for no parity, odd parity, or even parity,

- selection of the word of data length that can range from 5 to 8 bits, and
- selection of one of the 8 speeds supported by the port.

For each field, provide comments that describe the value choices for each of the options.

When the struct is ready, declare a variable of its type, and then using the hex format set the fields to configure the port for communicating over a full-duplex, odd-parity, 6-bit-word, and the highest speed available channel.

Your Answer:

typedef struct serial_r{

//Two selections
unsigned duplex: 1;
//Three selections
unsigned parity: 2;
//5-8 bits -> 4 bits 2^2
unsigned word_len: 2;
//Selection of one of the 8 speeds -> 8 bits 2^3
unsigned speed: 3;

} SERIAL_R

SERIAL_R register= {0x0, 0x1, 0x3, 0x5};

Bit Field Declaration

The declaration of a bit-field has the following form inside a structure −

```
struct {
   type [member_name] : width ;
};
```

The following table describes the variable elements of a bit field −

Sr.No.	Element & Description
1	**type**

	An integer type that determines how a bit-field's value is interpreted. The type may be int, signed int, or unsigned int.
2	**member_name** The name of the bit-field.
3	**width** The number of bits in the bit-field. The width must be less than or equal to the bit width of the specified type.

The variables defined with a predefined width are called **bit fields**. A bit field can hold more than a single bit; for example, if you need a variable to store a value from 0 to 7, then you can define a bit field with a width of 3 bits as follows −

```
struct {
   unsigned int age : 3;
} Age;
```

The above structure definition instructs the C compiler that the age variable is going to use only 3 bits to store the value. If you try to use more than 3 bits, then it will not allow you to do so. Let us try the following example −

Live Demo **(http://tpcg.io/wVJ3IM)**

```
#include <stdio.h>
#include <string.h>

struct {
   unsigned int age : 3;
} Age;

int main( ) {

   Age.age = 4;
   printf( "Sizeof( Age ) : %d\n", sizeof(Age) );
   printf( "Age.age : %d\n", Age.age );

   Age.age = 7;
   printf( "Age.age : %d\n", Age.age );

   Age.age = 8;
   printf( "Age.age : %d\n", Age.age );
```

```
   return 0;
}
```

When the above code is compiled it will compile with a warning and when executed, it produces the following result −

```
Sizeof( Age ) : 4
Age.age : 4
Age.age : 7
Age.age : 0
```

Question 2

Not yet graded / 0 pts

Describe what a device port is and how it is used to communicate with a device.

Describe a sample layout of the port.

Your Answer:

A device port is a 1 to 1 communication channel, common ones are the parallel port and the serial port. A lot of ports interact with a larger bus which can connect to multiple sources in either direction.

When referring to a physical **device** (https://www.computerhope.com/jargon/d/device.htm), a **hardware port** or **peripheral port** is a hole or connection found on the front or back of a computer. Ports allow computers to **access** (https://www.computerhope.com/jargon/a/access.htm) **external** (https://www.computerhope.com/jargon/e/external.htm) devices such as **printers** (https://www.computerhope.com/jargon/p/printer.htm). Below is a short listing of the different computer ports you may find on a computer.

Question 3

Not yet graded / 0 pts

Describe what are block devices and character devices. How do they differ?

What's the difference in how the OS needs to handle each of them?

Your Answer:

Block devices are devices that transfer data in blocks rather than in individual data bits, commands include read, write, and seek. Gets raw I/O or file system access and memory-mapped file access is possible.

Character devices get input character by character, like keyboards and serial ports. Commands include get and put and libraries layered on top allow line editing.

- A Character ('c') Device is one with which the Driver communicates by sending and receiving single characters (bytes, octets).
- A Block ('b') Device is one with which the Driver communicates by sending entire blocks of data.
- Examples for Character Devices: serial ports, parallel ports, sounds cards.
- Examples for Block Devices: hard disks, USB cameras, Disk-On-Key.

Question 4

Not yet graded / 0 pts

What is polling?

How does a polling-based I/O differ from interrupt-based I/O?

Your Answer:

Polling is a way of communicating with a device, where you constantly ask the device what its state is. Can cause busy-wait cycle when waiting for I/O device, but can use a timer to remedy this.

Polling is pervasive throughout CS whenever one program, process, or similar is waiting on another. Basically process A somehow asks process B "do you have anything for me?" repeatedly until it gets something.

This can be anywhere from a very low level (e.g. polling the status of a physical wire looking for a change in voltage) to a very high level (e.g. one server sending a web request to another to see if the price on an item has changed).

Polling is relatively simple to implement and is often (though not always) the right tool for the job. In UNIX-style socket programming, you'll often see poll() compared with select(), which is interrupt-driven (more of a "just wake me up when you have something; I'll sleep for now and let the system do other things" approach).

Interrupt based runs on a CPU interrupt triggered by the I/O device. The interrupts can be masked so they are ignored or delayed. Doesn't cause busy-waiting as long as the I/O eventually goes through

Question 5

Not yet graded / 0 pts

Describe in general terms what buffering is?

Why is buffering needed in dealing with I/O from and to devices?

How is buffering different from caching?

How is buffering different from spooling?

Your Answer:

Buffering is storing data in memory while transferring between devices.

Buffering is needed to cope with the device speed mismatch and the transfer size mismatch and to maintain the collection of coherent chunks.

Caching is used for performance sake since it's main operation is quickly copying data, while the buffer is just for coping with device mismatch.

Spooling is for holding the output of a device if a device can serve only one request at a time, just so it doesn't get overwhelmed with requests.

EXTRA:

Buffers are often used in conjunction with **I/O** (https://en.wikipedia.org/wiki/I/O) to **hardware** (https://en.wikipedia.org/wiki/Computer_hardware), such as **disk drives** (https://en.wikipedia.org/wiki/Disk_drives), sending or receiving data to or from a **network** (https://en.wikipedia.org/wiki/Computer_network), or playing sound on a speaker. A line to a **rollercoaster** (https://en.wikipedia.org/wiki/Rollercoaster) in an amusement park shares many similarities. People who ride the coaster come in at an unknown and often variable pace, but the roller coaster will be able to load people in bursts (as a coaster arrives and is loaded). The **queue area** (https://en.wikipedia.org/wiki/Queue_area) acts as a buffer—a temporary space where those wishing to ride wait until the ride is available. Buffers are usually used in a **FIFO** (https://en.wikipedia.org/wiki/FIFO_(computing_and_electronics)) (first in, first out) method, outputting data in the order it arrived.

Buffers can increase application performance by allowing **synchronous** (https://en.wikipedia.org/wiki/Synchronous) operations such as file reads or writes to complete quickly instead of blocking while waiting for hardware interrupts to access a physical disk subsystem; instead, an operating system can immediately return a successful result from an API call, allowing an application to continue processing while the kernel completes the disk operation in the background. Further benefits can be achieved if the application is reading or writing small blocks of data that do not correspond to the block size of the disk subsystem, allowing a buffer to be used to aggregate many smaller read or write operations into

block sizes that are more efficient for the disk subsystem, or in the case of a read, sometimes to completely avoid having to physically access a disk.

Question 6

Not yet graded / 0 pts

Describe what a device driver is and how it helps the operating system to operate with multitude of devices out there.

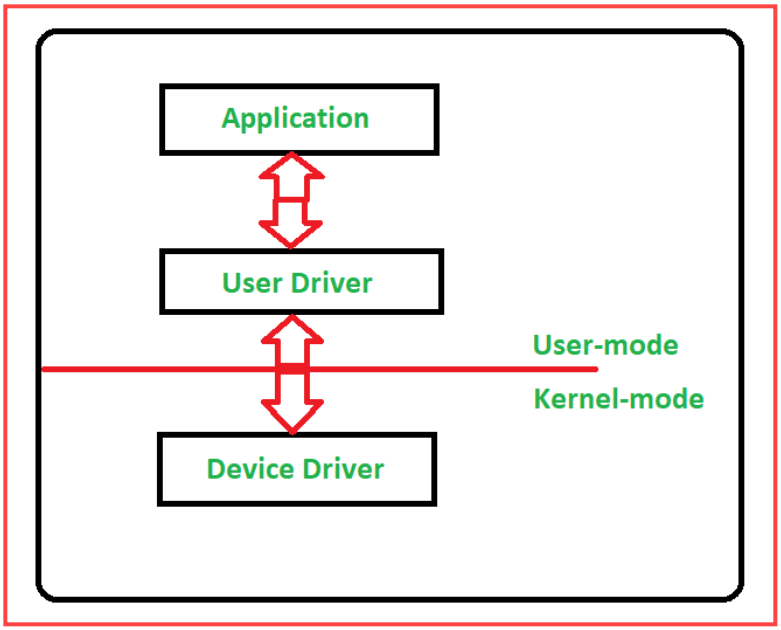
Your Answer:

Device drivers are classes that hide differences among I/O contollers from the kernel. Each class serves similar devices.

Device Driver in computing refers to a special kind of software program or a specific type of software application which controls a specific hardware device that enables different hardware devices for communication with the computer's Operating System **(https://www.geeksforgeeks.org/introduction-of-operating-system-set-1/)** A device driver communicates with the computer hardwares by computer subsystem or computer bus connected to the hardware.

Device Drivers are very essential for a computer system to work properly because without device driver the particular hardware fails to work accordingly means it fails in doing a particular function/action for which it has been created.

In a very common way most term it as only a **Driver** also when someone says **Hardware Driver** that also refers to this **Device Driver.**



Types of Device Driver:
For almost every device associated with the computer system there exist Device Driver for the particular hardware.But it can be broadly classified into two types i.e.,

1. **Kernel-mode Device Driver –**
   This Kernel-mode device driver includes some generic hardwares which loads with operating System as part the OS these are BIOS, motherboard, processor and some other hardwares which are part of kernel software. These includes the minimum system requirement device drivers for each operating system.
2. **User-mode Device Driver –**
   Other than the devices which are brought by kernel for working of the system the user also bring some devices for use during the using of a system that devices needs device drivers to functions those drivers falls under User mode device driver. For example user needs any plug and play action that comes under this.

Question 7

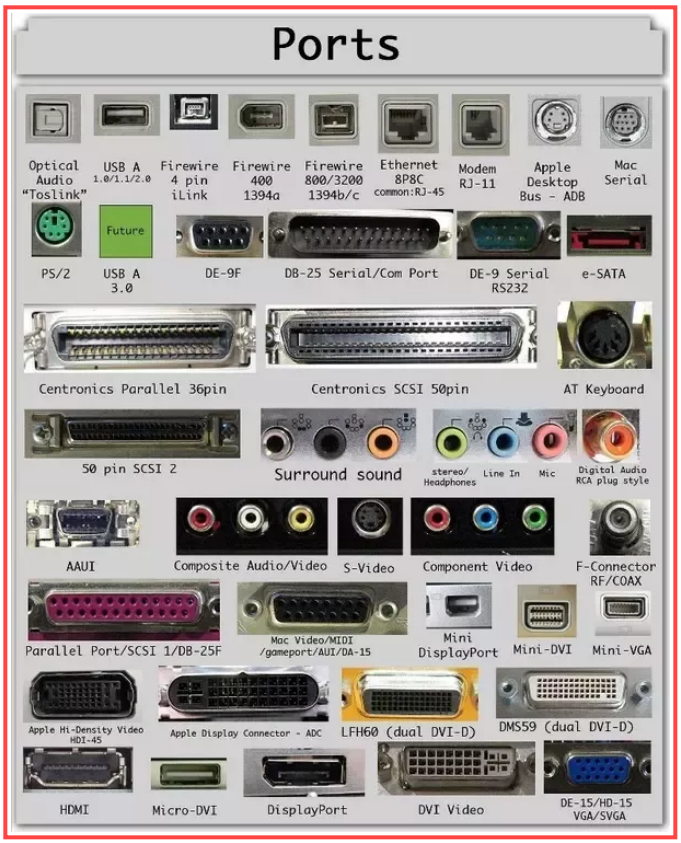
Not yet graded / 0 pts

Compare port-based communication with bus-based communication.

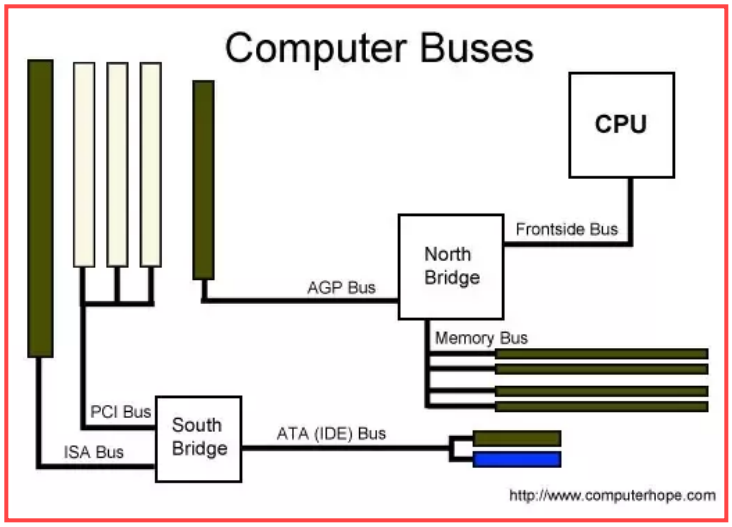
Your Answer:

in port based input output the input output peripheral is connected to the general port of the processor while in bus best input output the input output peripheral is connected via address bus

- A port is commonly defined as a connection point (or interface) between external hardware, like a mouse, keyboard or harddrive.
  - An example is the **serial port (https://en.wikipedia.org/wiki/Serial_port)** .



- A bus is a communication system that transfers data between components inside a computer.
  - An example is the bus that connects your **CPU (https://en.wikipedia.org/wiki/Central_processing_unit)** to your internal memory. It's inside of the motherboard & needs to be very fast.

- 

So to summarise, think of an interface as basically anything that connects up two different parts, this includes a bus or port. A protocol is basically a definition of something, a set of rules everyone agreed upon so communication can be smooth. A bus is a specific kind of interface and it uses an implementation of some protocol to manage communication going through itself, so the CPU knows what to say at what time to access memory and the memory knows how to reply with data when it is requested. A port is basically a kind of external bus, right now way slower than an average bus, since speed is no bottleneck here. A port mostly connects stuff from the outside of the pc and a bus connects things inside.

Question 8

Not yet graded / 0 pts

Describe how CPU communicates with devices using Memory-Mapped I/O (MMIO).

Describe how CPU communicates with devices using Port-Mapped I/O (PMIO).

Explain the key difference between MMIO and PMIO.

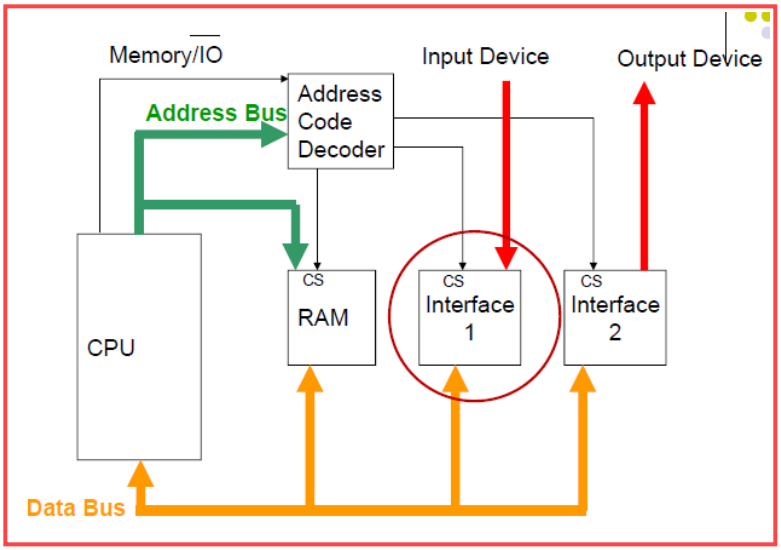
Your Answer:

Memory-mapped I/O and port-mapped I/O are two complementary methods for I/O.

Memory-Mapped I/O

In memory-mapped systems, the I/O device is accessed like it is a part of the memory. `Load` and `Store` commands are executed for reading from and writing to I/O devices, just like they are used for the memory (port-mapped has special commands for I/O). This means I/O devices use the same address bus as memory, meaning that CPU can refer to memory *or* the I/O device based on the value of the address. This approach requires isolation in the address space: that is, addresses reserved for I/O should not be available to physical memory.

Below is an image of a *simple, basic computer system*. The case is much more complicated in contemporary systems.



Port-Mapped I/O

According to **Wikipedia** **(http://en.wikipedia.org/wiki/Port-mapped_IO)**

> Port-mapped I/O often uses a special class of CPU instructions specifically for performing I/O. This is found on Intel microprocessors, with the IN and OUT instructions. These instructions can read and write one to four bytes (outb, outw, outl) to an I/O device. I/O devices have a separate address space from general memory, either accomplished by an extra "I/O" pin on the CPU's physical interface, or an entire bus dedicated to I/O. Because the address space for I/O is isolated from that for main memory, this is sometimes referred to as isolated I/O.

As for the advantages and disadvantages: since the peripheral devices are slower than the memory, sharing data and address buses may slow the memory access. On the other hand, by the I/O simplicity memory-mapped systems provide, CPU requires less internal logic and this helps for faster, cheaper, less power consuming CPUs to be implemented. The logic is similar to that of RISC systems: reduce the complexity, get a more dedicated and a robust system which comes quite handy for embedded systems, for example.

On the contrary (again from Wiki):

> Port-mapped I/O instructions are often very limited, often providing only for simple load and store operations between CPU registers and I/O ports, so that, for example, to add a constant to a port-mapped device register would require three instructions: read the port to a CPU register, add the constant to the CPU register, and write the result back to the port.

Question 9

10 / 10 pts

I have submitted answers to all questions in this study set.

- (●) True
- ( ) False

Quiz Score: **10** out of 10