MID TERM

Understand process scheduling and how long it would take to execute jobs sequentially vs in parallel.

Q.1. Multiple jobs can run in parallel and finish faster than if they had run sequentially. Suppose that two jobs, each needing 20 minutes of CPU time, start simultaneously. How long will the last one take to complete if they run sequentially? How long if they run in parallel? Assume 50% I/O wait.

A.1. If each job has 50% I/O wait, then it will take 40 minutes to complete in the absence of competition. If run sequentially, the second one will finish 80 minutes after the first one starts. With two jobs, the approximate CPU utilization is (1 - (0.5)^2) or 0.75. Thus each one gets 0.375 CPU utilization per minute of real time. To accumulate 20 minutes of CPU time, a job must run for (20/0.375) or about 53.33 minutes. Thus, a) Running sequentially the jobs finish after 80 minutes. b) Running in parallel they finish after 53.33 minutes. [For explanation refer https://cs.stackexchange.com/a/59486/72322]

Being moved from ready to running means the process has been **scheduled**; The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.

Different scheduling policies are compared using scheduling metrics. The first is the turnaround time. The turnaround time of a job is defined as the time at which the job completes minus the time at which the job arrived in the system. More formally, the turnaround time Tturnaround is:

$$T_{turnaround} = T_{completion} - T_{arrival}$$

Another metric is the response time which is the time from when the job arrives in a system to the first time it is scheduled.

$$T_{response} = T_{firstrun} - T_{arrival}$$

Also, the process scheduling policies can be of two types: Preemptive(willing to stop one process from running in order to run another) and Non-preemptive(system drives one process to completion before considering to run another).

The most basic algorithm a scheduler can implement is known as **First In, First Out** (**FIFO**) scheduling or sometimes **First Come, First Served** (**FCFS**). FIFO has a number of positive properties: it is clearly very simple and thus easy to implement. But, if all jobs do not run for the same time and the first job P1 is very long as compared to other jobs, FIFO gives a large turnaround time.

This can be solved by using Shortest Job First approach. This works good only until the jobs arrive at the same time. If the jobs arrive at any time and the first arriving job is longer than later arriving jobs then again we get poor performance.

The next approach and the first preemptive approach is the Shortest Time to Completion First(STCF). It preempts a running job if its time to completion is more than the newly arrived job. STCF might give a good turnaround time but gives a poor response time.
Theoretically good, SJF and SCTF are not practical because we don't know when a job completes.

To solve this problem came the Round Robin algorithm. It's basic idea is to run a job for a time slice (or quanta) and then switch to next job in the queue. Small quantas lead to better response time but large context switch overhead.

Understand virtual memory and virtual to physical mapping.

Q.2. You are given the following data about a virtual memory system:
(a)The TLB can hold 1024 entries and can be accessed in 1 clock cycle (1 nsec).
(b) A page table entry can be found in 100 clock cycles or 100 nsec.
(c) The average page replacement time is 6 msec.
If page references are handled by the TLB 99% of the time, and only 0.01% lead to a page fault, what is the effective address-translation time?

A.2. The chance of a hit is 0.99 for the TLB, 0.0099 for the page table, and 0.0001 for a page fault (i.e., only 1 in 10,000 references will cause a page fault). The effective address translation time in nsec is then: $0.99 * 1 + 0.099 * 100 + 0.0001 * 6 * 10^6 \approx 602$ clock cycles. Note that the effective address translation time is quite high because it is dominated by the page replacement time even when page faults only occur once in 10,000 references.

Virtual memory is a **memory** management capability of an **OS** that uses hardware and software to allow a computer to compensate for physical **memory** shortages by temporarily transferring data from random access **memory** (RAM) to disk storage.

Address Translation: Base and bound method - We'll need two hardware registers within each CPU: one is called the base register, and the other the bounds (sometimes called a

limit register). This base-and-bounds pair is going to allow us to place the address space anywhere we'd like in physical memory, and do so while ensuring that the process can only access its own address space. In this setup, each program is written and compiled as if it is loaded at address zero. Now, when any memory reference is generated by the process, it is translated by the processor in the following manner: physical address = virtual address + base.
Each memory reference generated by the process is a virtual address; the hardware in turn adds the contents of the base register to this address and the result is a physical address that can be issued to the memory system.

Now you might be asking: what happened to that bounds (limit) register? The bounds register is there to help with protection. Specifically, the processor will first check that the memory reference is within bounds to make sure it is legal. If a process generates a virtual address that is greater than the bounds, or one that is negative, the CPU will raise an exception, and the process will likely be terminated. The point of the bounds is thus to make sure that all addresses generated by the process are legal and within the "bounds" of the process.

Although the space between the stack and heap is not being used by the process, it is still taking up physical memory when we relocate the entire address space somewhere in physical memory; thus, the simple approach of using a base and bounds register pair to virtualize memory is wasteful.

Understand how the OS maintains free disk block list in the memory ?

Read **http://www.cs.unh.edu/~jlw/cs610/notes/free-space-mgmt.pdf**

We assume a basic interface such as that provided by malloc() and free(). Specifically, void *malloc(size t size) takes a single parameter, size, which is the number of bytes requested by the application; it hands back a pointer (of no particular type, or a void pointer in C lingo) to a region of that size (or greater). The complementary routine void free(void *ptr) takes a pointer and frees the corresponding chunk. Note the implication of the interface: the user, when freeing the space, does not inform the library of its size; thus, the library must be able to figure out how big a chunk of memory is when handed just a pointer to it The space that this library manages is known historically as the heap, and the generic data structure used to manage free space in the

heap is some kind of free list. This structure contains references to all of the free chunks of space in the managed region of memory.

Assume we have a 4096-byte chunk of memory to manage (i.e., the heap is 4KB). To manage this as a free list, we first have to initialize said list; initially, the list should have one entry, of size 4096 (minus the header size).

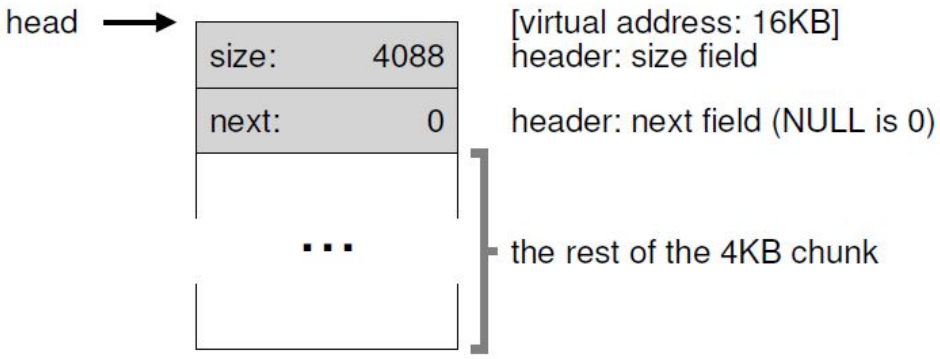

Figure 17.3: **A Heap With One Free Chunk**

After allocating the space to the heap, the status of the list is that it has a single entry, of size 4088(4096- header size). The head pointer contains the beginning address of this range; let's assume it is 16KB. Now, let's imagine that a chunk of memory is requested, say of size 100 bytes. To service this request, the library will first find a chunk that is large enough to accommodate the request; because there is only one free chunk (size: 4088), this chunk will be chosen. Then, the chunk will be split into two: one chunk big enough to service the request (and header, as described above), and the remaining free chunk. Assume the header to be of 8bytes.

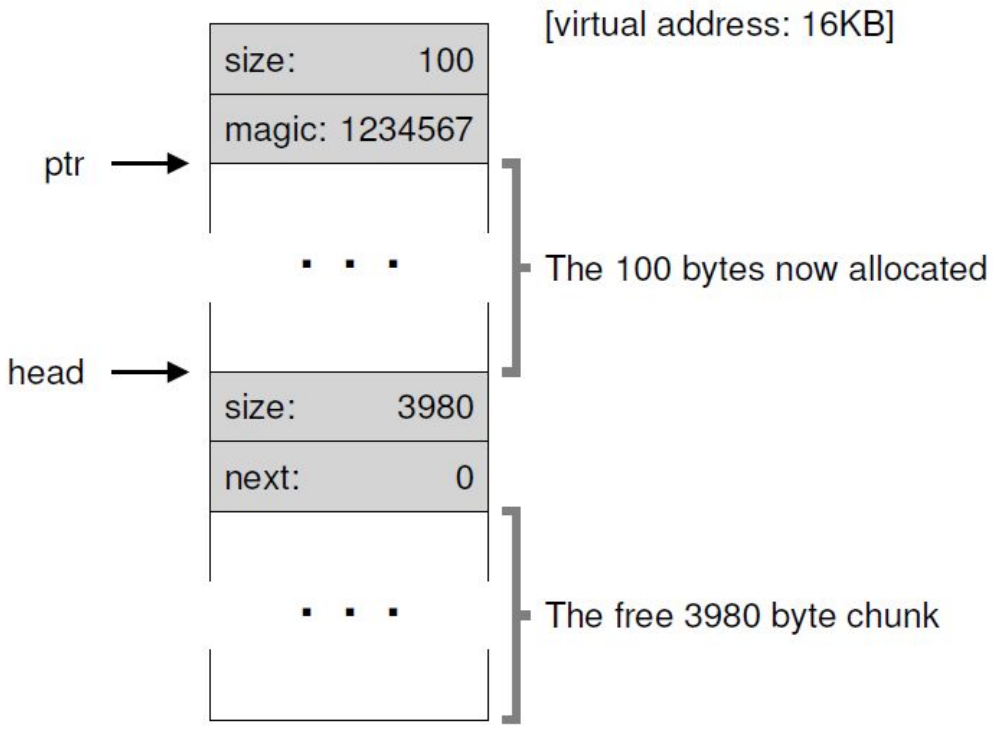

Figure 17.4: **A Heap: After One Allocation**

Thus, upon the request for 100 bytes, the library allocated 108 bytes out of the existing one free chunk, returns a pointer (marked ptr in the figure above) to it, stashes the header information immediately before the allocated space for later use upon free(), and shrinks the one free node in the list to 3980 bytes (4088 minus 108).

When the allocated memory chunk is freed, the nearby nodes in the free list and checked and if the memory locations are contiguous, they are coalesced or we get a highly fragmented memory.

Understand how OS can facilitate installation of a new device without any need of recompiling the OS ? [Device drivers, dynamically loaded modules]

Also read http://faculty.cs.niu.edu/~berezin/463/lec/08os/ufsnotes.html
http://www.angelfire.com/myband/binusoman/Unix.html#inode

UNIX does it as follows. There is a table indexed by device number, with each table entry being a C struct containing pointers to the functions for opening, closing, reading, and writing and a few other things from the device. To install a new device, a new entry has to be made in this table and the pointers filled in, often to the newly loaded device driver.

Linux operating systems contain a table, called the device table, which is indexed according to the number of devices. Each device is represented in the table by a device data structure; for each entry in the table, there is a pointer for all functions of the device. These pointers maintain the entry for read, write, open, close, etc, so that entries can be updated, added, and removed. When a new device is added and installed on the operating system, a new entry is made in the table. Then, pointers are filled in for the drivers of the new device.

Given disk rotation speed, number of sectors/cylinder, etc. How long does it take to read a sector ?

Q.5. A disk rotates at 7200 RPM. It has 500 sectors of 512 bytes around the outer cylinder. How long does it take to read a sector?

A.5. At 7200 RPM, there are 120 rotations per second, so 1 rotation takes about 8.33 msec. Dividing this by 500 we get a sector time of about 16.67 μ sec.

Q.6. If the data in disk is lost, how will you retrieve it ? [Superblock]

FINALS (Notes from Class)

Q.2. a)What techniques can help OS minimize TLB misses: CPU Affinity

To reduce TLB misses, sometimes the operating system can use its intuition to figure out which pages are likely to be used next and to preload entries for them in the TLB. For example, when a client process sends a message to a server process on the same machine, it is very likely that the server will have to run soon. Knowing this, while processing the trap to do the send, the system can also check to see where the server's code, data, and stack pages are and map them in before they get a chance to cause TLB faults.

The normal way to process a TLB miss, whether in hardware or in software, is to go to the page table and perform the indexing operations to locate the page referenced. The problem with doing this search in software is that the pages holding the page table may not be in the TLB, which will cause additional TLB faults during the processing. These faults can be reduced by maintaining a large (e.g., 4-KB) software cache of TLB entries in a fixed location whose page is always kept in the TLB. By first checking the software cache, the operating system can substantially reduce TLB misses.

When software TLB management is used, it is essential to understand the difference between two kinds of misses. A soft miss occurs when the page referenced is not in the TLB, but is in memory. All that is needed here is for the TLB to be updated. No disk I/O is needed. Typically a soft miss takes 10-20 machine instructions to handle and can be completed in a few nanoseconds. In contrast, a hard miss occurs when the page itself is not in memory (and of course, also not in the TLB). A disk access is required to bring in the page, which takes several milliseconds. A hard miss is easily a million times slower than a soft miss.

b)

Q.3. Can disabling interrupts handle concurrency correct ?

Ignore threads.

Q.5.

Modes for lock: Exclusive and Shared.
a) 6 proc-esses

Q.6. b)YARN:

FINALS

1. Multicore CPUs are beginning to appear in conventional desktop machines and laptop computers. Desktops with tens or hundreds of cores are not far off. One possible way to harness this power is to parallelize standard desktop applications such as the word processor or the web browser. Another possible way to harness the power is to parallelize the services offered by the operating system -- e.g., TCP processing -- and commonly-used library services -- e.g., secure http library functions). Which approach appears the most promising? Why?

Among the above two approaches, parallelize the services offered by the OS is more promising. The reason is, each CPU in the multicore contains its own operating system. Therefore, a master OS schedules the jobs or tasks to remaining OSs.

- OS provides synchronization of tasks with message passing and Inter Process Communication (IPC).
- OS provides better management of memory and files.
- OS provides a uniform layer of interface for all its applications.

The above mentioned services offered by the OS makes it is appeared to be most promising then the standard desktop applications.

2. a) Affinity scheduling reduces cache misses. Does it also reduce TLB misses? What about page faults

Ans : -

Affinity Scheduling

A process is any program which is under execution. When any process is large, it is divided into threads. So, to execute the right thread on right CPU affinity scheduling is done.

Translation Look aside Buffer (TLB) misses will also reduce. This is because; TLB are placed inside the CPU. So, TLB will also executed by right CPU.

The page fault will not be affected by affinity scheduling. This is because; when a page is placed in the memory of single CPU, then, all the different CPUs have the same page in their memory also.

2. b) Migrating virtual machines may be easier than migrating processes, but migration can still be difficult. What problems can arise when migrating a virtual machine

Ans :-

Migration:

• **Migration** is the process of moving a virtual machine from one server to another server while it is still in the production environment. It should involve minimal disruption to any existing sessions.

• Hypervisors is a piece of computer firmware and software that create an illusion of many virtual machines.

• A **hypervisor** is also called a virtual machine monitor (VMM).

• Hypervisors remove the virtual machine (VM's) dependency on physical hardware. This eases out the process of migration a bit.

• Apart from migrating entire virtual machine, the care should be taken in migrating

o VM configuration file.

o Virtual disk file

o Memory pages used by the virtual machine

o Memory bitmap

o Transfer of storage and network connectivity

o The IP addresses.

Migrating virtual machines might be easier than migrating processes, but yet it poses some difficulty while **implementation**.

While migrating virtual machine's the following issues can arise:

• The migration period can itself be long as the complete virtual machine, along with the guest operating system (OS) and all processes executing on it must be moved to the new machine.

• **Physical I/O** devices present problems as they do not migrate with the virtual machine.

• Consider the following:

o The registers of such devices may hold a state that is critical to the proper functioning of the system.

o For example, a read or write operations to disk that have been issued, but have not been completed yet.

• **Network I/O** is can also be an issue as other machines will continue to send packets to the hypervisor, blissfully unaware that the virtual machine has migrated.

• Consider the following:

o Even if packets are redirected to a new hypervisor, the virtual machine will be unresponsive during the long migration period.

o As a result, packets can experience long delays or even lost if the hypervisor buffers end up overflowing.

Therefore, the steps to be followed when issues arise while migrating a virtual machine's are as follows:

• Check migration settings.

• Reconnect the host servers.

• Check for compatible server hardware and device dependencies.

• Check network connectivity between the servers.

• Check computing resources on the destination server.

3. Can disabling interrupts handle concurrency correct ? What are mutex and condition variables and do we need both ?

Ans : Assume we are running on such a single-processor system. By turning off interrupts (using some kind of special hardware instruction) before entering a critical section, we ensure that the code inside the critical section will not be interrupted, and thus will execute as if it were

atomic. When we are finished, we re-enable interrupts (again, via a hardware instruction) and thus the program proceeds as usual. The main positive of this approach is its simplicity. The negatives, unfortunately, are many. First, this approach requires us to allow any calling thread to perform a privileged operation (turning interrupts on and off), and thus trust that this facility is not abused.

Second, the approach does not work on multiprocessors. If multiple threads are running on different CPUs, and each try to enter the same critical section, it does not matter whether interrupts are disabled; threads will be able to run on other processors, and thus could enter the critical section. As multiprocessors are now commonplace, our general solution will have to do better than this. Third, and probably least important, this approach can be inefficient. Compared to normal instruction execution, code that masks or unmasks interrupts tends to be executed slowly by modern CPUs. For these reasons, turning off interrupts is only used in limited contexts as a mutual-exclusion primitive.

A mutex is a shared variable that can be in one of two states: unlocked or locked. Consequently, only 1 bit is required to represent it, but in practice an integer often is used, with 0 meaning unlocked and all other values meaning locked.
Pthreads provides a number of functions that can be used to synchronize threads. The basic mechanism uses a mutex variable, which can be locked or unlocked, to guard each critical region. In addition to mutexes, Pthreads offers a second synchronization mechanism: condition variables. Mutexes are good for allowing or blocking access to a critical region. Condition variables allow threads to block due to some condition not being met. Almost always the two methods are used together

4. Authentication mechanisms are divided into three categories: Something the user knows, something the user has, and something the user is. Imagine an authentication system that uses a combination of these three categories. For example, it first asks the user to enter a login and password, then insert a plastic card (with magnetic strip) and enter a PIN, and finally provide fingerprints. Can you think of two drawbacks of this design?

Ans : -

In terms of authentication, using all three would provide highly reliable results. But on the other hand authentication systems should not only focus entirely on the security aspect but also make the authenticating system easy to use for the user as a whole.

One drawback that i think with this system is that you would be using different varieties of elements to verify a person when even one of them can do the perfect job for you. Say for example you just plan to implement user authentication based only on fingerprints, then it would be sufficient for you to validify the user based only on that as neither can someone possess that same set of prints on his hands nor is it easy to obtain someone else's fingerprints. Coming to magnetic strips, ATM cards have over a couple of decades been working with this technology, authenticating users/ account holders worldwide, so if implemented, they are self sufficient to carry out authentication. A login id and password is the only aspect that might not be self sufficient as we have seen in the past as well as present that login id and passwords can be vulnerable to attacks by hackers.

Secondly, logistics and cost related issues can be a sort of drawback, because implementing fingerprint scanning would require special devices and the physical presence of the user as well. Also scanning magnetic strips would also require the use of special devices. Besides that the more complex an authentication process is, the more less users would it attract. A simple hassle free implementation can not only be successful but also be secure in terms of verifying the user.

5. a) 4a. What does the following Linux shell pipeline do?

grep nd xyz | wc -l

Grep command returns the lines where a pattern exists in a file or std input. Here, finds matches for “nd” in the file named xyz and returns only those lines. This output is supplied to wc -1 via | operator. wc -1 gives the number of lines in file.

x

5. b) A user at a terminal types the following commands:

a | b | c &

d | e | f &

How many processes are running?

{a,b,c,d,e,f} - 6 processes

| redirects the output of the first process to the next process. Trailing & implies the process is run in the background. So, the processes that run are {a,b,c,d,e,f} - 6 processes.

5. c) Does it make sense to take away a process’ memory when it enters zombie state? Why

or why not?

Zombie process

When the resources are not shared among different processes, then, the condition occurred is known as zombie state.

In the shared memory multiprocessor system process want to enter the critical section, and it becomes a problem if more than one process wants to enter into the critical section. To avoid a situation for multiple processes into critical section, the code segment shown is utilized, as a solution.

When any process is in zombie state, which means it is not executed any more. So, to free the memory is a good idea.

5. d) A computer has a pipeline with four stages. Each stage takes the same time to do its work, namely, 1 nsec. How many instructions per second can this machine execute

Ans :-

Pipelining

It is known that $1\ \text{nsec} = 10^{-9}\ \text{sec}$

Or there are $10^9\ \text{nsecs}$ in one second. Since it is a four stage pipeline, when first instruction is in the final stage, the second instruction is in the third stage, the third instruction is in the second stage and the fourth instruction is the first stage.

Thus the first instruction is completely executed in the fourth stage. After which, the instructions are executed one after another because of the pipeline.

Thus, the first three stages of the pipeline for the first instruction take 3nsec to execute the instruction in the fourth stage.

Therefore, the total number of instructions which are executed in one second is:

$10^9 - 3 = 999999997$

Thus, $\boxed{999999997}$ instructions are executed in one second.

Visit problems on page 851 in Modern Operating Systems- Tannenbaum. (http://stst.elia.pub.ro/news/SO/Modern%20Operating%20System%20-%20Tanenbaum.pdf)

6. a) Is Hadoop better for interactive or batch processes ?

A. Batch processes

Batch processing is the execution of a series of jobs in a program on a computer without manual intervention (non-interactive). Strictly speaking, it is a processing mode: the execution of a series of programs each on a set or "batch" of inputs, rather than a *single* input. Hadoop MapReduce is the best framework for processing data in batches. Hadoop is based on batch processing of big data. This means that the data is stored over a period of time and is then processed using Hadoop.

6. b) YARN

https://searchdatamanagement.techtarget.com/definition/Apache-Hadoop-YARN-Yet-Another-Resource-Negotiator

https://www.youtube.com/watch?v=ZFbkNY6Xn94