AOS Final Exam 2021

Name: Shriya Sarraf
Student Id: 00001606767
Email Id: ssarraf@scu.edu

Question 1- Multicore CPUs are beginning to appear in conventional desktop machines and laptop computers. Desktops and laptops with tens or hundreds of cores are not far off. One possible way to harness this power is to parallelize the application. Mention one application that is attractive to parallelize [3 pts]? Another approach is to parallelize the services offered by the OS (e.g., TCP processing), etc. Which approach is more promising between these two approaches [3 pts]?

Answer 1-

Parallelizing common desktop applications like word processors and web browsers is one method to tap into this capacity. Video gaming is the one desktop application that could benefit from large-scale parallelization as many components of the game require significant (parallel) computing. Although some desktop applications are multi-threaded wherein the threads are typically utilized to facilitate I/O programming and are thus not compute-intensive threads.

Parallelizing the Operating Systems' services appears to be a better option based on the two methodologies outlined above. Once the OS is parallelized and then it can be used by all programs. For example, Network cards now include on-board parallel processors (network processors) that speed up packet processing and enable higher-level network services at line speeds (e.g., encryption, intrusion detection).

Consider the powerful processors found in video cards, which are designed to offload video rendering from the main CPU and provide apps with higher-level graphics APIs (e.g., Open GL). These special-purpose cards could be replaced with single-chip multicore processors. And therefore, parallelizing the OS's services appears to be a better alternative.

- OS would provide synchronization of tasks with message passing and inter process communication.
- OS provides a uniform layer of interface for all its applications.
- OS has a great management of memory and files.

Although some desktop applications are multi-threaded, the threads are generally utilized to facilitate I/O programming and are thus not compute-intensive threads. A more promising approach is to parallelize operating system and library services. This is reflected in modern hardware and operating system architectures. The same fundamental approach can be used to parallelize other operating systems and common library services as the number of cores increases.

Also, parallelizing at the Operating System level is done once and used by any other application. If the parallelization is done at the application level, then each application needs to have their own implementation.
The services mentioned above thus makes Operating System a better option than desktop applications.

Question 2- Please answer the following:

(a) Affinity scheduling reduces cache misses. Does it also reduces TLB misses [1.5 pts]? What about page faults [1.5 pts]?

(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 [3 pts]?

Answer 2-

(a) Affinity scheduling is the process of placing the appropriate thread on the appropriate CPU. Yes, it reduces TLB missed because TLB misses are stored within each CPU and this could help to reduce them. It has no influence on *page faults* because a page in memory for one CPU is also in memory for all other CPUs.
TLBs are placed within the CPU and therefore there will be fewer TLB misses. Page errors are unaffected when a single CPU's memory is occupied with a page. In memory, all CPUs will have the same pages.

(b) The following issues can arise while migrating virtual machines:

- The migration period itself can be as long as the complete virtual machine, along with the guest 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:
  1. The registers of such devices may hold a state that is critical to the proper functioning of the system.
  2. Ex: A read or write operation to the disk that has been issued but not been completed.

- Network I/O is particularly difficult because other machines will continue to send packets to the hypervisor, unaware that the virtual machine has moved.
- Even if packets can be redirected to the new hypervisor, the virtual machine will be unresponsive during the migration period, which can be long because the entire virtual machine, including the guest operating system and all processes executing on it, must be moved to the new machine. As a result, packets can experience large delays or even packet loss if the device/hypervisor buffers overflow.

Question 3- Can disabling interrupts handle concurrency correctly [2 pts]? What are mutexes and condition variables and how they are used [3 pts]?

Answer 3-

No disabling interrupts can't handle concurrency currently. On a single-processor system, the simplest solution is to have each process disable all interrupts just after entering its critical region and re-enable them just before leaving it. With interrupts disabled, no clock interrupts can occur. The CPU is only switched from process to process as a result of clock or other interrupts and with interrupts turned off the CPU will not be switched to another process. Thus, once a process has disabled interrupts, it can examine and update the shared memory without fear that any other process will intervene.

This approach is generally unattractive because it is unwise to give user processes the power to turn off interrupts. And if one of them, did it and never turned them on again then that could be the end of the system. Furthermore, if the system is a multiprocessor i.e., with two or more CPUs then disabling interrupts affects only the CPU that executed the disable instruction. The other ones will continue running and can access the shared memory. Code that masks or unmasks interrupts will be executed slowly by modern CPUs(expensive context switch).

This approach does not work on multiprocessors. If multiple threads are running on different CPUs and each try to enter the same critical section so it does not matter whether interrupts are disabled as 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, 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.

Also, it is frequently convenient for the kernel itself to disable interrupts for a few instructions while it is updating variables or especially lists. If an interrupt occurs while the list of ready processes, for example, is in an inconsistent state, race conditions could occur. The conclusion is: disabling interrupts is often a useful technique within the operating system itself but is not appropriate as a general mutual exclusion mechanism for user processes.

The possibility of achieving mutual exclusion by disabling interrupts even within the kernel is becoming less every day due to the increasing number of multicore chips even in low-end PCs. Two cores are already common, four are present in many machines and 8, 16, or 32 are possible very soon. In a multicore/multiprocessor system- disabling the interrupts of one CPU does not prevent other CPUs from interfering with operations the first CPU is performing. Consequently, more sophisticated schemes are needed.

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. 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.

Mutex is good for locking i.e., enforce serialization and condition variable is good for synchronization. They are typically used together because if you tried to use mutex for both locking and synchronization then you will get deadlock.

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.

Question 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 login name and password, then insert a plastic card (has magnetic strip) and enter a PIN, and finally provide fingerprints. Can you think of two drawbacks of this design [6 pts]?

Answer 4-

The combination of different authentication mechanisms will provide stronger authentication. However, there are two drawbacks which are as follows-

First, the expense of developing this system is substantial. Three separate authentication systems are used in the system which adds to the expense. Fingerprint scanning would necessitate the usage of special gadgets and the user's physical presence is also required. Then, scanning magnetic strips would necessitate the use of specialized equipment as well.

Second, this authentication mechanism puts extra burden on the user. The user has to remember his login/password, carry his plastic card and remember its PIN and has to go through the process of fingerprint matching. The key issue is that all this will increase the time it takes to authenticate a user thereby resulting in increased user dissatisfaction. Also, you would be using different varieties of elements to verify a person when even one of them can do the perfect

job for you. For example, you just plan to implement user authentication based only on fingerprints, then it would be sufficient for you to validate 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 are 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.

Question 5-

5 (a) What does the following Linux shell pipeline do?
grep nd xyz | wc -l

Answer- Grep command returns the lines where a pattern exists in a file or std input. Here, find matches for "nd" in the file named xyz and returns only those lines. This output is supplied to wc -1 via | operator. Output should be # of lines from the file that contain the pattern "nd".

5 (b) A user at a terminal types the following commands:
a | b| c &
d | e| f &
How many processes are running?

Answer- {a,b,c,d,e,f} So, 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?

Answer- Yes, it makes sense to take away a process' memory when it enters zombie state for the following reasons-

- Zombie process holds the PID. Rapidly growing zombies can exhaust the available PIDs and no new processes will be launched.
- Zombies hold tiny memory and huge number of zombies can make a difference. Large amount can consume considerable amount of memory contributing to high memory utilization of machine.

5 (d) Explain under what situation a process may request a shared lock or an exclusive lock. What problem may a process requesting an exclusive lock suffer from [1.5 pts]?

Answer- Shared locks used to maintain read integrity. A process will request a shared lock if it wants to read some bytes whereas it will request an exclusive lock if it wants to update some bytes. A process requesting an exclusive lock maybe blocked indefinitely if there is a sequence of processes requesting shared locks. That is, if readers always go before writers, then writers could suffer from starvation.

- When a process is reading the lock, it should not get updated.
- Many processes can use the same shared lock on the same object simultaneously without any issues. This is because all of them are reading it.
- If an object is already under a shared lock and a request for another shared lock arrives then the shared lock is permitted. Exclusive locks used to protect write integrity.
- When a process is modifying the object, it should neither be read from nor written to.
- Only one process can access an exclusive lock at a time.
- If an object is already under exclusive lock, any request for locking will fail.

Question 6- Why Hadoop is considered a batch processing platform and not an interactive platform [3 pts]? What is YARN, why needed, and how it can be used [3 pts]?

Answer 6-

Hadoop is considered a batch processing platform. Batch processing is the execution of a series of jobs in a program on a computer without manual intervention. 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. Hadoop solution to availability includes Shuffling which causes significant delay (rpt) but it does not impact throughput. As a result, Hadoop is platform for batch processing with good throughput but not very good rpt.

YARN stands for *Yet Another Resource Negotiator*. Its goal is to manage cluster resources and support multiple application frameworks/paradigms concurrently whereas Hadoop was built to support M/R Framework only. This is significant as it enables Hadoop to support multiple application framework rather than being limited to batch processing only.

Apache Hadoop YARN is the resource management and job scheduling technology in the open source Hadoop distributed processing framework. YARN is responsible for allocating system resources to the various applications running in a Hadoop cluster and scheduling tasks to be executed on different cluster nodes.

YARN can dynamically allocate resources to applications as needed. This feature is designed to improve resource utilization and application performance compared with MapReduce's more static allocation approach.

In a cluster architecture, Apache Hadoop YARN is between HDFS and the processing engines being used to run applications. It combines a central resource manager with containers, application coordinators and node-level agents that monitor processing operations in individual cluster nodes. Hence, this defines YARN and its utilization.