SINGLE EXAMINEE AFFIDAVIT
“I, the undersigned, promise that this assignment submission is my own work. I recognize that should this
not be the case; I will be subject to plagiarism penalties as outlined in the course syllabus.”
Student Name: Nathnael Atlaw
RED ID: 822655984
Date: 10/01/2023

1) In a multi-programming environment, the OS constantly switches processes in and out the CPU cores for concurrent executions. Considering the principles behind the caching mechanism, explain why running with more CPU cores could result in more effective caching (i.e., better cache hit rate).

- When a computer system is juggling a bunch of tasks at once in our multi-programming environment, having more CPU cores can manage each task more easily. Each of these cores has its little memory stash (cache) where it keeps the data it uses a lot, so it doesn’t have to keep reaching into the slower, main memory. Each core specifies which task is there so they do not have to fight over them. The operating system plays a key role here by trying to keep a process on the same core to reuse the data (that’s sticking to the cache) and avoid asking the main memory again. But, while having more cores sounds great, it’s not just about having more space to store and quickly grab data; it’s also about making sure all these cores are on the same page and not messing up each other’s data.

2) Suppose in programming assignment 2, instead of using one countvocabstrings thread for removing and processing lines from the queue sequentially, you now spawn several posix threads concurrently with each thread trying to process one line from the queue, i.e., removing a line from the queue and counting the contained vocab strings then exiting.
Assuming to run your program with one processor and a Linux environment, where a posix thread usually uses the kernel-level threading model (one to one multiplexing)

From the thread scheduling perspective, explain why this implementation could result in better throughput for processing the lines in the queue. Throughput means the number of lines processed within a specified timeframe.

- From a thread scheduling perspective, while the single-processor environment inherently limits true parallel execution, the concurrent threading model can optimize CPU utilization by ensuring that the processor is always engaged in meaningful work, reducing idle times associated with I/O waits, and managing the execution of threads in a manner that maximizes throughput. This can result in an increased number of lines

processed within a specified timeframe, even though the threads are not truly executing in parallel. This is important as it can reduce i/o waiting time, enhance responsiveness by allowing other threads to process when one thread gets stuck, and implement parallel processing.

3) Based on what was discussed in the class about the detailed steps in interrupt handling and exception handling, what are the similarities between handling an interrupt from disk controller and handling an illegal memory access run-time condition.

- Both cases first of all include a disruption to the workflow of the program. During the initial disruption, interruptions from disk controllers usually are asynchronous while illegal memory access exceptions are synchronous. Both of these issues are resolved by using a handler that contains the logic and tries to resolve their problems. Before the handler can be used, the operating system must first save important information like register values. Then the handler reads and writes data from the disk or terminates the illegal memory access until the issue is resolved. Once the issue is resolved, the handler disappears and control is returned to the program

4) In a Linux virtual machine running on top of a type I hypervisor, a user process runs the following C code. Describe the details of how the code execution is carried out and the run-time condition/s is/are handled (if there is any) by the VM user program, the VM kernel, the hypervisor, and the hardware working together.

```
int main() {
printf("Unix is great.");
int arr_pids[20];
int cid = fork();
arr_pids[20] = cid;
}
```

- This code is a simple c program that prints out Unix is great performs a fork system call and then tries to assign the child process ID to an array. The runtime conditions handled by the VM user program seek to execute the fork system call by duplicating the current process. The child process gets a PID (Process ID) of 0, and the parent process gets the PID of the child. The runtime conditions are handled by the kernel, which handles the system calls made by the printf and fork functions, managing the I/O and process creation, respectively. The VM kernel also handles memory allocated by the program and can see if illegal memory access is happening. The hypervisor manages the virtual CPU used by the VM, translating the VM's virtual CPU instructions to physical CPU instructions and managing CPU scheduling. It also manages the I/O operations requested by the VM, ensuring that they are directed to the appropriate physical I/O

devices. Finally, the hardware manages the runtime conditions by using RAM to control memory access and using the CPU to execute the tasks outlined in the program.

5) Several processes are loaded and in different process states. P8 (first in the queue) and P10 are waiting to be scheduled on the CPU. Processes P3 and P7 are executing. P4 and P34 (first in the queue) are waiting on cloud storage access. P21 (first in the queue) and P6 are waiting for web responses. Draw a process queueing diagram for these processes (note each type of I/O has its own waiting queue). Draw arrows of where exactly the processes would go (change of queues) if P7 tries to access cloud storage to write, and P6 completes the wait. Make sure to label each process with its process state.

-