1. **(5 pts) What are the three main purposes of an operating system? Explain how the old mainframe computers were different from the computers we have today.**

The tree main purposes of operating systems are:

* Software that manages the computer hardware
* Common services for programs and apps
* Middleman between the computer and user

The old mainframe computers lacked operating systems and were much larger in size compared to modern computers. As seen in the videos in class, previous computers were easily 3 to 5 times bigger than a today’s small laptop. Another characteristic was the punch cards system they had, where in order to run programs in their mainframe you had to punch a specific type of card that will instruct the computer what to do – and these were only able to run one software at a time.

**2) (5 pts) Explain the differences between a program, executable, process, and task.**

- Program: A set of instructions for a computer to execute  
- Executable: files that are ready to be loaded into memory and executed  
- Process: Program loaded into the memory and executing. A process is a unit of work in a system.  
- Taks: A thread activity/process. Generally includes I/O operations.

3) **(5 pts) Explain the purpose of system calls within an operating system? Find two POSIX system calls (using an Internet search) and explain the purpose of the system call, the parameters it accepts, and the return values, if any.**

Additional system calls: <https://docs.oracle.com/cd/E19048-01/chorus4/806-3328/index.html>

A system provides an interface to the services made available by the OS.

1. open(2POSIX) - Accepts three arguments: const char \*path, int oflag, and mode\_t mode. This function initiates a file descriptor for the specified file referenced by path. The oflag parameter determines the file access mode (either for reading, writing, or both) and may be combined with additional flags to enable further actions (such as file creation if it does not exist and appending during write operations). The mode argument defines the file's permission settings. On successful execution, open(...) yields a file descriptor; if it fails, it returns -1 and assigns an error code to errno based on the failure reason.
2. close(2POSIX) - Requires a single argument: int fildes. The close(...) function is used to terminate a file descriptor that was previously opened via open(...), duplicated with dup(...), created for a socket(...), or established through shm\_open(...) operations. Should close(...) complete successfully, it outputs 0. If it encounters an error, it outputs -1 and changes errno to EBADF (indicating an invalid file descriptor).

4) **(10 pts) Research an operating system of your choice, that is not Windows, Linux, or Mac OS. Answer the following:**

**(a) What is the name of the OS?**

Chromium OS.

**(b) What company or group of people is responsible for maintaining it?**

Google.

**(c) When was its first release?**

In 2009 by Google.

**(d) How much does it cost?**

It is free.

**(e) What type of hardware does it run on?**

It has run on the Kogan Agora Chromium laptop and it got also got ported to Rasberry Pi 2/3.

**(f) What features set it apart from other operating systems?**  
It has the open-source version of ChromeOS and is designed to use a browser engine; thus the reason it is broadly use for web applications rather than software applications.

5) **(5 pts) Using the program below, explain what the output will be at LINE A.**

At Line A, the output will display "PARENT: value = 32." Despite being a global variable, executing fork() causes the child process to inherit a copy of the parent's variables into its separate memory space. Consequently, modifications to the value in the child process remain isolated and do not impact the parent's value. Therefore, at Line A, the printf function will output the initial value of 32. It's critical to understand that this scenario unfolds only after the child process concludes, as wait(NULL) compels the parent process to pause until the child has finished its execution.

6) **(5 pts) Assume that the OS implements Many-to-Many multithreading model. What is the minimum number of kernel threads required to achieve better concurrency than in the Many-to-One model and why?**

To surpass the concurrency offered by the many-to-one model, the many-to-many model necessitates a minimum of two kernel threads. This setup permits the operating system to distribute user threads across the kernel threads. Consequently, it enables concurrent execution of both kernel threads, mitigating the impact of a process that blocks one thread by allowing the other to continue operating.

7) **(10 pts) Explain in detail whether the following problems exhibit data parallelism or task parallelism:**

**(a) Sum of integers from 1 to n.**  
Data parallelism – same operation of addition

**(b) Find the dot product of two different matrices.**  
Task parallelism – needs addition and multiplication for each element of the matrix

**(c) Search of a text file for a string s.**  
Data parallelism – text is divided into chunks and same search operation is run for all of them

**(d) Convert the first character of each string in an array to uppercase.**  
Data parallelism – same operation of converting the first char of each string to uppercase

**(e) Find the product of the integers from 5 to 500 and the average of the integers from 5 to 500.**  
Task parallelism – both are running different operations of multiplication, sum, and then division

8) **(5 pts) What are the differences between a monolithic kernel and a microkernel? Explain.**

Windows, Unix, and Linux employ monolithic kernels, which consist of a unified static binary operating within a singular address space, where all elements are closely integrated. This architecture makes the system less flexible in terms of modifications but enhances efficiency as a trade-off. Conversely, microkernels, utilized by MacOS and iOS, strip away all non-critical components, primarily facilitating interaction between client programs and services operating in user space. Consequently, services on systems with microkernels typically function as user-level applications.