README

All the programs used in this report are attached alongside in the same folder. To reproduce the same below steps should be followed:

Execute “build” - This will compile all required C programs into the current directory. The command is:

```
./build
```

- To execute Part 1

  ```
  ./run <filename> [-r|-w] <block_size> <block_count>
  ```

- To execute Part 2

  ```
  ./run2 <filename> <block_size>
  ```

- To execute Part 6

  ```
  ./fast <file_to_read>
  ```

> NOTE: The results may vary based on the system specifications.

Part 1:

- The program takes in a filename, flags indicating read or write, the block size and the block count to be read.
- Block size is a value in bytes while block count is an integer value. Read requires only block size. If a block count is mentioned, it will stop executions once blocksize * blockcount bytes are read.
- Write requires both block size and block count to be provided.

Part 2:

The program takes in a filename and a block size in bytes as input. Returns the block count to be read for a given block size, such that the total read time is atleast 7 seconds. Outputs the block count. To execute this script, use:

```
python3 part2.py
```

The script for Part 2 requires a text file called ***blocksize.txt***. This contains the various block sizes which the read program should execute and return block count for. We have provided a sample file that contains block sizes from 1 byte to 134217728 bytes.

Part 2 Extra Credit:

This compares the read speeds of our program with dd. We have a script for this which must be run as superuser. This is because we clear the cache on every run so that we can compare the cached and non cached speeds with dd. To execute this script, use:

```
sudo python3 part2EC.py
```

Part 3 and 4

Since Part 3 and 4 are comparisons of performances with non cached and cached reads, our program takes in the block size and block counts to read a given file. This returns the non cached performance (for part 4) and an average of cached performance over 4 executions (for part 3). The results are printed in MiB/s for every blocksize - blockcount pair. To execute this python script, use:

```
sudo python3 part3,4.py
```

The python script requires a CSV containing blocksize,blockcount data. We have provided a sample file named ***part3,4.txt*** that was compiled as part of our experiment.

Part 5

Part 5 was comparing the system call speeds for very small block size. Our build produces ***part5***, an executable that can be run to get the system call performance comparing lseek(), read() and pread() for block size of 1, 2, 4 and 8 bytes.

Part 6

Part 6 is the fastest read times for a given file. We have used mmap() - a system call that created memory mapping for a file directly in system memory. This executes atleast as fast as dd, if not faster. To execute this, we just use:

```
./fast <filename>
```