Part A -

All values in kB.

Event	VM-Alloc	VM-RSS
After Initialization, before memory mapping	12652	1176
Memory Mapping the file	23192	1236
Reading the first character (== 'k')	23192	1240
Character at offset 10000 bytes (== 'd')	23192	1244

On Memory Mapping the file, the VM allocated to the process increases dramatically, proportional to the file size of 10 MB. VM-RSS also increases a bit as some parts of the process' VM are also allocated physical frames.

On reading the first character, VM-Alloc doesn't change as expected as no additional (virtual) memory has been requested by the program. We see a 4kB increase in VM-RSS. This is because asking for the first byte of the file results in the first page of the file being allocated a physical frame, of size 4kB.

On reading the character at 10,000 B offset, VM-Alloc doesn't change for the same reason as above (no additional memory requested by the program). The 4kB increase is seen in VM-RSS this is because reading the last byte of the mapped file results in one more physical being allocated on RAM as 10,000 Bytes would not fit into a single physical frame.

NOTE - These values were obtained using the command 'pmap -x <pid> | tail -n 1'

Part-B

In this part, we open each file and read 10MB of its data. The file sizes are a little larger than 10MB. Hence, Total data read = 250 MB.

1. Avg throughput = 71 MBps.
   This is the the data access speed from the system RAM because the files have already been (virtually) memory-mapped. In this case, the bottleneck is the disk. This means that this is the maximum throughput possible.
2. Avg throughput = 62MBps.
   Again the bottleneck here is the disk. This is slightly lower than in 1) . This can be explained as - read() is a syscall and hence it results in lot's of context switches(once for every 512 bytes whereas mmap results in 1 page fault for 4K bytes) which can be avoided when using mmap. Also using reads results in a few extra data copying happening (between disk and kernel space buffer) before it reaches the user (program) space buffer. In the case of mmap it is directly copied from the disk to the RAM.
   The values are the same order of magnitude because ultimately, the disk itself is the bottleneck, with some additional overhead as mentioned above.
3. Using Memory Mapped files is better in this case. The benefit though is not very huge because we are just accessing the files just once each. Using Memory Mapping will give huge benefits if you need to access a particular part of a file multiple times because it forces the OS to keep the file contents in RAM and you are no longer dependent on the Disk Buffer Cache having the file for fast access.
4. Inspecting the size of the disk buffer cache in the above two experiments shows that there is increase in size when using the read call and not when using Memory Mapping. Hence this implies that when using Memory Mapping, buffer cache is not used and is bypassed and data directly copied to the user space mapped addresses (the mapped memory addresses are directly passed on to the user).
5. Avg throughput = 36.7 MBps.
   In this case, effectively all disk blocks have to be written back to the disk. Hence, here, the bottleneck resource is the disk and the average throughput is of the same order of magnitude as typical disk access rates. This is slower than 1 as disk writes are typically slower than disk reads.
6. Avg throughput = 28.95 MBps.
   The throughput value in this case is lower than part 1. Again the reason here is that using write() results in extra data copying happening to the kernel space buffers, finally from which the disk lifts the data to write. Also, This may be due to the fact that mmap writes just happen to the RAM initially then afterwards when the file is unmapped it is flushed back to the disk in whole. This results in sequential writes happening and hence faster speed. Using write() means parts of the file are written to the disk periodically (exactly times will be determined by OS) meaning lots of small disk accesses are required.
7. In the case of writes mmap gives an advantage over write(). Hence it is recommended to use this.

8. In this case, memory-mapped files do not give any performance benefits over regular files.
   If the files fit into the buffer cache, sequential writes do not translate to write requests to the disk (they can be done at a later stage, after the entire write operation gets finished, because the set of blocks to be written are all in memory, unlike cases 1-7). In this case the basic advantage of mmap i.e avoiding a extra copy operation to and fro the cache is nullified.

Note- Using Memory Mapping is only feasible only if file size is not too large so that it occupies most of the Virtual Memory of the process.