CO21BTECH11002
Aayush Kumar

Part 1:

Changes have been made to ‘exec.c’, ‘trap.c’, ‘defs.h’, ‘vm.c’ & ‘MAKEFILE’. Also, a new file ‘mydemandPage.c’ has been added.
The changes in exec.c does not allocate memory to dynamic variables when calling allocuvm but sz is still the same as before. When a page fault occurs, the fault is handled in trap.c. trap.c allocates memory to the program and maps virtual address to the physical address.

Observations & Reasons:

1. On increasing the size of the global array, the number of page faults increases. This is because only some parts of the process are stored in virtual memory. When a process requests a page of memory that is not currently in physical memory, it results in a page fault. If the size of the process is increased, it means that there are more pages that need to be stored in virtual memory. This leads to an increase in the number of page faults as the operating system has to constantly retrieve pages.

Part 2:

Changes have been made to ‘defs.h’, ‘proc.h’, ‘trap.c’, ‘proc.c’, and ‘vm.c’. Also, a new file ‘myCOW.c’ has been added.
In proc.h, we add an extra variable ‘W’ which stores whether the process has write access or not. In proc.c, in fork() function, the copyuvm() is changed to copyuvm_cow() which is a custom made function. copyuvm_cow() (defined in vm.c) prints the current state of the page directory of the process. Then it maps the parent’s physical frame read-only in both the parent and the child.
When a process tries to get write access a page fault occurs which is handled by the handle_pgflt() function. handle_pgflt() allocates a new memory page for the process and copies the data from the parent page and marks that the given process now has write access.

Observations & Reasons:

1. On increasing the size of the global array, the number of page faults increases.
2. When a process is forked, its write bit is 0 but when it gains access to write, its write bit becomes 1.