REPORT
CS20BTECH11028

Implementation:

Part 1:

- Added a system call pgtPrint() similar to the one done in Assignment 2.
- In the sysproc.c file, added function sys_pgtPrint(), which iterates over all page tables in the page directory which we can get using myproc()->pgdir and over all page numbers in the pagetables checks whether the pages are valid and can be accessed by user using bitwise ands with PTE_P and PTE_U flags, and ouputs the valid page table entries.

Part 2:

- In the exec.c file previously the total memory was being allocated. Now we have to allocate only read-only data, so we allocate only ph.vaddr + ph.filesz, and this won't allocate the memory for uninitialized global variables etc, during the process creation.
- So, when the process is being executed, whenever the variables with unallocated memory are accessed, it results in a pagefault. Hence we have to handle the page fault. In XV6 pagefault corresponds to trap number 14. So, in the trap.c file, we have to handle the trap by keeping an extra case statement in the existing switch case statement corresponding to page fault.
- In the page fault handler, first we have to check whether the process is accessing the memory which is in the process's address space. The address accessed due to which page fault occured will be stored in CR2 register, which can be accessed using rcr2().
- If the address is not in process's address space we kill the process, else we have to do the following
  - Get a free frame using kalloc()
  - Initialize all the entries to 0 in the frame using memset()
  - Now, map the virtual address corresponding to the page fault (stored in CR2) to the newly created frame. (similar to that done in mappages() function).

Note:

- The walkpgdir() function is written in the trap.c by copying it from vm.c

Observations and Analysis:

Part 1

- Without any array declarations, number of pages allocated are 2.
- When declared a global array int arr[10000], total 12 pages are being allocated. Even if the gloabal array is not initialized with values, the memory is being allocated for them. Hence we get a total of 12 pages. With increasing size of the array, the number of pages being allocated is increased.
- When we execute the program multiple times, each time the virtual address is being same, but the physical address is different. The difference in physical address each time might be due to the following reason : As each time a

process is being start, required number of frames are allocated using kalloc(), which returns the frame from the list of free frames. After process completion the memory is released and the frames are added to free frames list. Now again on calling the process, again kalloc() will be called and from the list of frames, it assigns a frame. So, they may not be the same all the time. (some frame might be allocated for other process)

- Also, the number of valid page table entries remain the same. This is because, the same amount of memory will be required for the process each time it is run, hence number of valid entries doesn't change.
- If i declare a local array arr[10000] and initialize only few entries, then it shows 2 valid page entries (same as that without the array), but if we initialize the array completely then it is resulting in a pagefault error, which might be due to the stack size limit is crossed.

Part 2

- In demand paging, even if we declare a global variable without initializing it, there won't be any extra pages allocated for it. Once we start initializing the array, a page fault occurs, as the memory being accessed is not evaluated yet. In the code mydemandPage.c, we print the valid page table entries for every 1000 iterations, each time we see an extra page allocated.
- Number of page faults increases with increasing size of the array (only if the array is being initialized, else there wouldn't be any change due to array size),
  3 for N = 3000
  5 for N = 5000
  10 for N = 10000
  because as more memory needs to be allocated, more pages will be needed and hence more page faults.

Comparison with and without demand paging:

- If we run a process, which has a global array arr[10000] declared in it but it is not yet initialized, without demand paging we get 12 pages, but with demand paging we get only 2 valid page table entries, as the memory for the uninitialized global variables is allocated on demand which is not the case with implementing without demand paging

Learning from the assignment:

- Got clarity on how two level paging is useful in reducing the size for storing the page tables and on how the corresponding physical page number can be accessed.
- Understood how the page directory entries and page tables entries can be checked using flag bits.
- Understood about ELF file and various information in the program headers, memsz, filesz etc and differences between them
- Got an idea at somewhat low-level on how a process is allocated memory, mapping virtual to physical address, etc

- Understood how traps are handled generally.
- Got some more clarity on how demand paging helps in reducing the number of frames used in case of sparsely filled arrays etc.