Assignment1

- Changes in Nachos
  - Changes in Address Space
    - Nachos was a single process system , so it used the whole of its memory for the address space of a single process .
    - Every address space consist of pagetable which is made up of pages .
    - The first page of the address space starts with virtual address 0 ,but it has some mapping to the physical address on the main memory.
    - In the original nachos both virtual address and the physical address were same (equals 0) as it had to deal with only one process .
    - The class Bitmap from Bitmap.h was used to create a bitmap for the pages in the physical memory and to find the empty physical address while creating a new address space.
    - Initially after creating the new address space , the whole physical memory was wiped out to 0 . So we had to change it so that it only assigns 0 to its own address space bits.
    - We had to translate the virtual address to the corresponding physical address while copying any file to the address space.
    - To handle the specific need of the Fork system call in which we had to copy the current address space to a newly created address space we used **constructor overloading** to create a new address space constructor deep copies the address space from the address space pointer passed to it.
    - We had to move the pagetable creation step from the original constructor to the Load function . So it creates new address space when Load function is called and assigns required amount of pages for it .
    - The Address space destructor has to take care to free up the memory and mark the corresponding pages in the pagetable bitmap as dirty so that the space can be reused.
    - We also created a class element called id which assigns a unique id to each process based on the starting page of its address space.
  - System calls
    - **Fork**
      - Fork creates a new kernel level thread
      - Copies the current address space to the address space of the new thread .
      - Also keep the registers to be used by the child process .
      - Puts child on the ready queue and starts the child process.
      - It returns 0 in the child process and the *PID* of the child in the parent process.
    - There are two implementations of the exec system call described below
      - **Exec2**
      - To be used with Fork system call only .
      - It replaces the current address space with the program to be loaded .

    - Resets all the registers and thus moves the program counter to the beginning .
    - The control returns to the parent process after the child process finishes.
    - **Exec**
    - An amalgamation of the above Exec2 and Fork calls .
    - Instead of calling Fork and then Exec2 , We can directly call Exec with the executable of the new process and it directly loads and starts the new process.
    - The control returns to the parent process after the child process finishes .
  - **Exit**
    - Exit system call is called automatically whenever a program exits .
    - Handles the cleanup of the system .
    - It stops the thread if it is not the main thread .
    - Since the last thread can not stop itself , so we can not just finish the last *main* thread .
    - So if the pid of the thread calling main is 0 (0 means it is the main thread) it calls nachos Halt instead of finishing the thread.