Nachos 2 (Task Summary)

Task – 01 (Address Space : Virtual Memory Support)

Provide an interface **PageTable** to do the following tasks

- Keep track of free pages
- Allocate from free pages
- De-allocate pages

You also need to do the followings:

1. Edit constructor of AddrSpace to link virtual address/ virtual page number(vpn) and physical address/physical page number(ppn) properly
2. Write functions for reading from and writing to virtual memory.
   - First, convert virtual address to its physical address
   - Then read/write in physical address using Machine::ReadMem() and Machine::WriteMem()

Save the <processId, PageTable> pair inside kernel.

Task – 02 (System Call: Exec and Exit)

Exec

- Implement both system calls with standard arguments provided in **syscall.h**
- Carefully review the code inside **userprog/progtest.cc (**Startprocess() function to be precise)
- A new pageTable must be created when Exec is called. For now, you donot need to implement any **Page Replacement Policy**. If there are not enough number of free pages, throw an exception.
- If you find that there are not enough pages to run the new process and you have already allocated some pages for it, **de-allocate** them. Anyway, it is suggested to keep track of **Number of free pages** inside the **PageTable** data structure.
- Properly copy filename (you may need to use virtual memory read/write here)
- Check if the filename is too long to fit in free pages. Handle the case where a filename can be in non-contiguous pages.

Exit

- Implement both system calls with standard arguments provided in **syscall.h**
- Print status value (argument of Exit) for debugging purpose.
- You must **free the allocated pages** once a process is done executing.

Task – 03 (System Call: Read and Write)

- Read **console.cc** to see the implementation of a console. Also see **userprog/progtest.cc (consoleTest()** function) to see how a console object is used.

- Create a **console** object as global inside **Initialize()** in **threads/system.cc**
- Implement a thread safe, synchronized console interface (take help of **consoleTest**) to control and support read/write in console.
- Now, implement **Read** and **Write** system calls as in **syscall.h** (You don't have to worry about file read/write. Just support these system calls for console i/o)

Task – 04 (Bonus: User Exception Handling)

- Make the kernel bulletproof so that any exception in user program doesn't terminate the kernel or affect any other process. Go through **exception.cc** and try to understand how to do it. **As this is a bonus task, you won't get any detailed documentation** ☹ .

Advice regarding System Call Implementation

- Create a separate function for each system call inside **exception.cc .** Call this function from the exception handler.
- Increment **PC** properly. (You will get the instructions inside the slide provided)

Running User Programs

Those who are having hard time running nachos –x command, use the following after you are in **userprog** directory and **make** is complete

```
./nachos –x ../test/matmult
```

Here, **matmult** is a test program provided with nachos. Use other programs if you wish.