1. A. 8 CPUs(logical CPUs) , 4 cores -- 2 threads per core(Hyper threading)

```
codemaxx@Trojan-Horse:~$ lscpu
Architecture:            x86_64
CPU op-mode(s):          32-bit, 64-bit
Byte Order:              Little Endian
CPU(s):                  8
On-line CPU(s) list:     0-7
Thread(s) per core:      2
Core(s) per socket:      4
Socket(s):               1
```

B. All CPUs have 2.80 GHz frequency

```
codemaxx@Trojan-Horse:~$ cat /proc/cpuinfo | grep "model name"
model name      : Intel(R) Core(TM) i7 CPU         930  @ 2.80GHz
model name      : Intel(R) Core(TM) i7 CPU         930  @ 2.80GHz
model name      : Intel(R) Core(TM) i7 CPU         930  @ 2.80GHz
model name      : Intel(R) Core(TM) i7 CPU         930  @ 2.80GHz
model name      : Intel(R) Core(TM) i7 CPU         930  @ 2.80GHz
model name      : Intel(R) Core(TM) i7 CPU         930  @ 2.80GHz
model name      : Intel(R) Core(TM) i7 CPU         930  @ 2.80GHz
model name      : Intel(R) Core(TM) i7 CPU         930  @ 2.80GHz
```

C. Total memory is 8162888 kB

```
codemaxx@Trojan-Horse:~$ cat /proc/meminfo
MemTotal:        8162888 kB
```

D. Free - 4273744 kB
Available - 6469364 kB

MemFree is the amount of physical memory not used by the system while MemAvailable is an estimate of how much memory is available for starting new applications, without swapping.

Citation:
https://superuser.com/questions/980820/what-is-the-difference-between-memfree-and-memavailable-in-proc-meminfo

```
codemaxx@Trojan-Horse:~$ cat /proc/meminfo
MemTotal:        8162888 kB
MemFree:         4273744 kB
MemAvailable:    6469364 kB
```

E. **454763**

```
codemaxx@Trojan-Horse:~$ cat /proc/stat | ag processes
processes 454763
```

F. **599104237**

```
codemaxx@Trojan-Horse:~$ cat /proc/stat | grep ctxt
ctxt 599104237
```

2. The resident size of a process (as shown in top or ps) represents the amount of non-swapped memory the kernel has already allocated to the process. VmSize is the total memory of a program including its resident size, swap size, code, data, shared libraries etc.

Memory1: VmSize: 8140 kb VmRSS: 648kb
Memory2: VmSize: 12048kb VmRSS: 740 kb
Memory3: VmSize:8144 kb VmRSS: 3572 kb
Memory4: VmSize:12044 kb VmRSS: 6924kb

VM size increases by almost 4000kB for memory_2 as compare to memory_1 which the space required for 1000000 integers. VMRSS is almost same for both memory_1 and memory_2 since the space has not been initialised yet.

VmSize of memory_3 is similar to that of memory_1 since the size of the array is same. VmRSS increases by more than 2000kb which is the space required for the 500000 integers initialised.

VmRSS for memory_4 is double that of memory_3 since we are allocating double the number of integers. VMsize increases by 4000kB due to the space for 1000000 integers.

```
codemaxx@Trojan-Horse:~/lab1/memory$ ./memory_1


Program : 'memory_1'
_____________________


PID : 11051
Size of int : 4
```

```
codemaxx@Trojan-Horse:/proc/11051$ cat status | ag Vm
VmPeak:     8140 kB
VmSize:     8140 kB
VmLck:         0 kB
VmPin:         0 kB
VmHWM:       648 kB
VmRSS:       648 kB
VmData:      188 kB
VmStk:      3916 kB
VmExe:         4 kB
VmLib:      1952 kB
VmPTE:        36 kB
VmPMD:        12 kB
VmSwap:        0 kB
```

```
codemaxx@Trojan-Horse:~/lab1/memory$ ./memory_2


Program : 'memory_2'
_____________________


PID : 11353
Size of int : 4
```

```
codemaxx@Trojan-Horse:/proc/11353$ cat status | ag Vm
VmPeak:    12048 kB
VmSize:    12048 kB
VmLck:         0 kB
VmPin:         0 kB
VmHWM:       740 kB
VmRSS:       740 kB
```

```
codemaxx@Trojan-Horse:~/lab1/memory$ ./memory_3


Program : 'memory_3'
____________________


PID : 11375
Size of int : 4
```

```
codemaxx@Trojan-Horse:/proc/11375$ cat status | ag Vm
VmPeak:     8144 kB
VmSize:     8144 kB
VmLck:         0 kB
VmPin:         0 kB
VmHWM:      3572 kB
VmRSS:      3572 kB
```

```
codemaxx@Trojan-Horse:~/lab1/memory$ ./memory_4


Program : 'memory_4'
____________________


PID : 11382
Size of int : 4
```

```
codemaxx@Trojan-Horse:/proc/11382$ cat status | ag Vm
VmPeak:    12044 kB
VmSize:    12044 kB
VmLck:         0 kB
VmPin:         0 kB
VmHWM:      6924 kB
VmRSS:      6924 kB
```

3. Number of subprocesses = 3 (Roll no. 150050031)

```
codemaxx@Trojan-Horse:/proc/11597/fd$ ps -efj | ag subprocesses
codemaxx 11892 10991 11892 10991  0 21:35 pts/27   00:00:00 ./subprocesses 150050031
codemaxx 11893 11892 11892 10991  0 21:35 pts/27   00:00:00 ./subprocesses 150050031
codemaxx 11894 11892 11892 10991  0 21:35 pts/27   00:00:00 ./subprocesses 150050031
codemaxx 11895 11892 11892 10991  0 21:35 pts/27   00:00:00 ./subprocesses 150050031
```

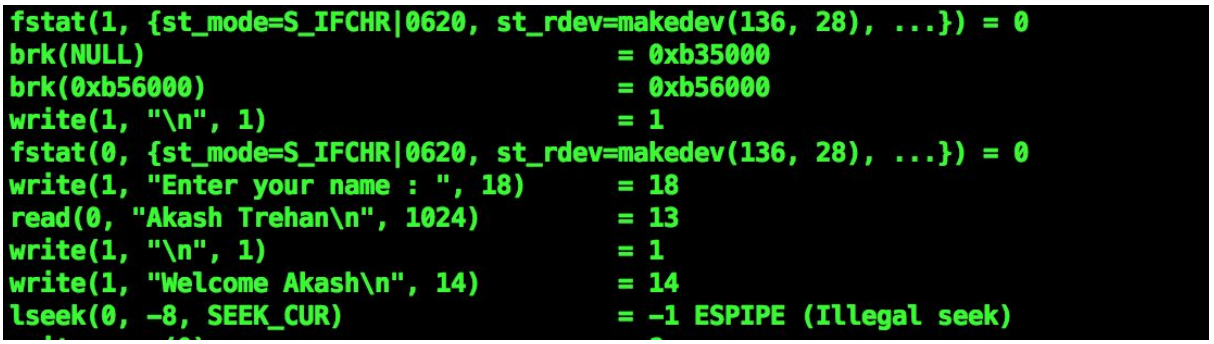
The last 3 processes have a PPID of 11892 which is the PID of the first process, the one we started.

4. 12 different system call functions for empty.c. It shows the system calls used for securing memory for the files, loading the binary, loading the associated libraries etc.

Common part: The loading of the binary.

```
execve("./hello", ["./hello"], [/* 24 vars */]) = 0
brk(NULL)                               = 0xb35000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f67aed30000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=164277, ...}) = 0
mmap(NULL, 164277, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f67aed07000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\t\2\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1868984, ...}) = 0
mmap(NULL, 3971488, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f67ae743000
mprotect(0x7f67ae903000, 2097152, PROT_NONE) = 0
mmap(0x7f67aeb03000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1c0000) = 0x7f67aeb03000
mmap(0x7f67aeb09000, 14752, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f67aeb09000
close(3)                                = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f67aed06000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f67aed05000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f67aed04000
arch_prctl(ARCH_SET_FS, 0x7f67aed05700) = 0
mprotect(0x7f67aeb03000, 16384, PROT_READ) = 0
mprotect(0x600000, 4096, PROT_READ)     = 0
mprotect(0x7f67aed32000, 4096, PROT_READ) = 0
munmap(0x7f67aed07000, 164277)          = 0
```

Different part: Reading from stdin and writing to stdout



B. execve - Takes the filename, arguments and environment variables as input. Returns -1 on error. It starts the execution of a program

brk - returns 0 on success, -1 on error. Used to change the data segment size. Sets the end of the data segment to the address mentioned in the argument.

Access - Arguments are filename and a permission. Returns 0 if the user has that permission on that file else returns -1

Mmap- Maps files into memory. Returns the pointer to the location where it's mapped. The various arguments specify the property of the mapped addresses - protection, flags and the file descriptor for the file and the offset from where to start

Munmap - deletes the mapping for the specified address space. Returns 0 on success, -1 on failure.

Open - Opens/creates a file. Takes input as the filepath and flags for read/write etc. and returns the file descriptor

Read - Input is file descriptor of the file to be read, the buffer to read it in and the number of bytes to be read. Returns the number of bytes read. Reads from files.

Write - Input is file descriptor of the file to be written, the buffer contaning the bytes to write and the number of bytes to be written. Returns the number of bytes written. Writes to files.

Close - takes file descriptor as input. Closes the file. Returns 0 on success, -1 on failure.

Arch_prctl - Sets architecture specific thread state.

Fstat - takes in a file descriptor and buffer as input. Gets the status of the file(info about the file) and puts it in the buffer.

Lseek - repositions the file offset of a file. Takes in the file descriptor, and offset and a directive which tells it what to do with the offset. Returns the final offset after the seek.

Exit_group - exits all threads in the calling process's thread group
Mprotect - Sets the protection for a given region of the memory. Takes the starting address, size of the memory region and the protection to set as input.

5. Found the pid of `openfiles` using `ps aux | grep openfiles`, then used the pid with `lsof -p PID` to find the files opened.

```
codemaxx@Trojan-Horse:~/lab1$ ps aux | grep openfiles
codemaxx 11597  0.0  0.0   4356   632 pts/27   S+   21:17   0:00 ./openfiles
```

The files opened are with fd 0-5 (last 6 in the screenshot)
0,1,2 are the standard file descriptors for stdin, stdout, stderr which are opened.
/tmp/welocme to OS, /tmp/CS333, /tmp/CS347 are the opened files

```
COMMAND     PID     USER   FD   TYPE DEVICE SIZE/OFF     NODE NAME
openfiles 11597 codemaxx  cwd    DIR   0,42     4096 25167666 /home/codemaxx/lab1/files
openfiles 11597 codemaxx  rtd    DIR    8,3     4096        2 /
openfiles 11597 codemaxx  txt    REG   0,42     8760 25167674 /home/codemaxx/lab1/files/openfiles
openfiles 11597 codemaxx  mem    REG    8,3  1868984  7341135 /lib/x86_64-linux-gnu/libc-2.23.so
openfiles 11597 codemaxx  mem    REG    8,3   162632  7341113 /lib/x86_64-linux-gnu/ld-2.23.so
openfiles 11597 codemaxx    0u   CHR 136,27      0t0       30 /dev/pts/27
openfiles 11597 codemaxx    1u   CHR 136,27      0t0       30 /dev/pts/27
openfiles 11597 codemaxx    2u   CHR 136,27      0t0       30 /dev/pts/27
openfiles 11597 codemaxx    3w   REG    8,3        0 11927554 /tmp/welocme to OS
openfiles 11597 codemaxx    4w   REG    8,3        0 11927592 /tmp/CS333
openfiles 11597 codemaxx    5w   REG    8,3        0 11927593 /tmp/CS347
```

Another way is to see the /proc/pid/fd directory

```
|codemaxx@Trojan-Horse:/proc/11597/fd$ ls -l
|total 0
|lrwx------ 1 codemaxx codemaxx 64 Jul 21 21:18 0 -> /dev/pts/27
|lrwx------ 1 codemaxx codemaxx 64 Jul 21 21:18 1 -> /dev/pts/27
|lrwx------ 1 codemaxx codemaxx 64 Jul 21 21:17 2 -> /dev/pts/27
|l-wx------ 1 codemaxx codemaxx 64 Jul 21 21:18 3 -> /tmp/welocme to OS
|l-wx------ 1 codemaxx codemaxx 64 Jul 21 21:18 4 -> /tmp/CS333
|l-wx------ 1 codemaxx codemaxx 64 Jul 21 21:18 5 -> /tmp/CS347
```

6. `lsblk -f` (contains filesystem + mountpoint for block devices)

```
codemaxx@Trojan-Horse:~/lab1/files$ lsblk -f
NAME           FSTYPE LABEL UUID                                   MOUNTPOINT
sr0
sda
├─sda2         ext4         d8c176a7-574a-457a-8b4e-31d347f9d3ee /home
├─sda3         ext4         c7e9a320-893f-4f5c-916e-d47595c89354 /
└─sda1         swap         b6d15d0f-48df-408f-aec5-e0b97492265e
  └─cryptswap1 swap         f9da1e43-879e-4561-bce3-d1a84840135e [SWAP]
```

7.

```
Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
sda               0.00   115.00  699.00  143.00 60008.00  1088.00   145.12     2.27    2.70    2.12    5.54   1.13  95.20
dm-0              0.00     0.00    0.00  272.00     0.00  1088.00     8.00     2.56    9.40    0.00    9.40   0.29   8.00

07/21/2017 11:20:54 PM
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.00    0.00    6.78    6.41    0.00   86.81

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
sda               0.00   178.00  763.00  183.00 65152.00  1456.00   140.82     1.93    2.04    1.88    2.73   0.99  93.60
dm-0              0.00     0.00    0.00  364.00     0.00  1456.00     8.00     1.40    3.84    0.00    3.84   0.13   4.80

07/21/2017 11:20:55 PM
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.25    0.00    6.26    6.88    0.00   86.61

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
sda               0.00   136.00  722.00  135.00 61936.00  1112.00   147.14     2.97    3.47    2.05   11.05   1.10  94.40
dm-0              0.00     0.00    0.00  278.00     0.00  1112.00     8.00     3.47   12.47    0.00   12.47   0.32   8.80
```

While ./disk1 is running the disk utilization is close to 95% .

The above is the output from the command `iostat -xtc 1`

```
Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
sda               0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
dm-0              0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00

07/21/2017 11:23:12 PM
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           5.38    0.00    7.12    0.12    0.00   87.38

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
sda               0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
dm-0              0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00

07/21/2017 11:23:13 PM
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           3.76    0.00    4.39    0.00    0.00   91.85

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
sda               0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
```

While running ./disk2, the idleness of the cpu is not 100% so we know a process is running, but the dis utilization is ~0% (can't be seen on the 1 second scale). Once the file is in the cache, it need not read from the disk.

This difference is because ./disk1 is reading all the different files while ./disk2 is reading only 1 file again and again. This 1 file is in the cache now so fast read. While 10000 files can't be put in cache so have to be read from the disk again and again.