**Assignment**

**Question 1: Add user in Sudoers File?**

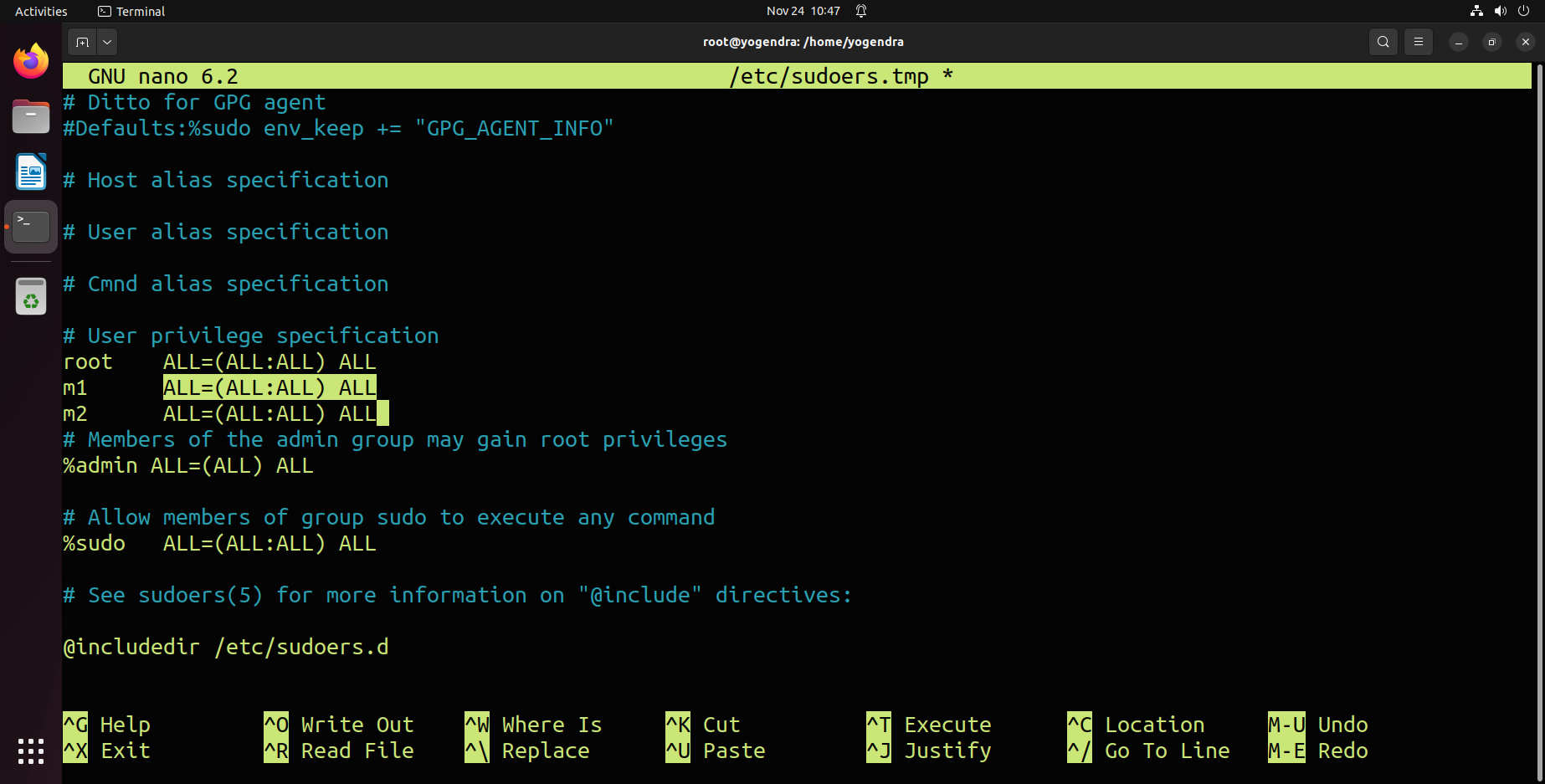
**Answer:** Open visudo file and add below line under **User privilege specification** and save the file.

# User privilege specification

root ALL=(ALL:ALL) ALL

m1 ALL=(ALL:ALL) ALL

m2 ALL=(ALL:ALL) ALL



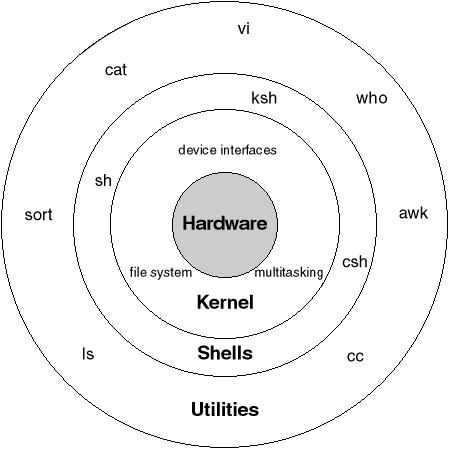
**Question 2:** What process loads when the terminal opens?

**Answer:**

**Question 3:**  What process runs when you enter ls?

**Answer: Link:** [**https://levelup.gitconnected.com/what-happens-where-you-type-ls-l-in-a-linux-shell-98ad95430cbf**](https://levelup.gitconnected.com/what-happens-where-you-type-ls-l-in-a-linux-shell-98ad95430cbf)

The shell is a program that takes your commands from the keyboard and gives them to the operating system to perform.The shell reads your input after you press Enter. It determines the command you want executed by looking at the first word of your input. A word is an unbroken set of characters. Spaces and tabs separate words. Shell script is a list of commands, which are listed in the order of execution.



So…What’s REALLY Happening?

First of all, before you type anything, the shell prints a prompt to you which normally ends with a $ sign. The prompt or command line is where you’ll type your command.

**1.** The shell prompt is displayed

**2.** The shell reads the command from standard input that was entered by the user

The getline function reads the entered line as one string, from the standard input and stores it in a buffer.

**The getline() Function**

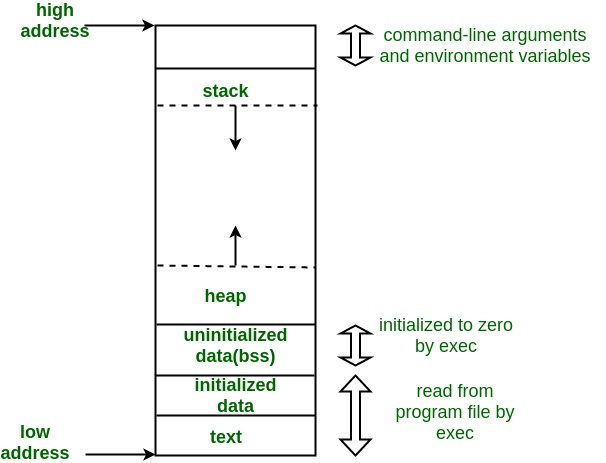
The latest and most trendy function for reading a string of text is getline(). It’s a new C library function, having appeared around 2010 or so.

**Question 4: Understand the following:**

* code segment
* Data segment
* Stack
* heap

**Answer:** A typical memory representation of a program consists of the following sections.

1. Text segment (i.e. instructions)
2. Initialised data segment
3. Uninitialized data segment (bss)
4. Heap
5. Stack



**Figure:** A typical memory layout of a running process.

**1. Text Segment:** A text segment, also known as a code segment or simply as text, is one of the sections of a program in an object file or in memory, which contains executable instructions.As a memory region, a text segment may be placed below the heap or stack in order to prevent heaps and stack overflows from overwriting it.

**2. Initialized Data Segment:** Initialized data segment, usually called simply the Data Segment. A data segment is a portion of the virtual address space of a program, which contains the global variables and static variables that are initialised by the programmer.

**3. Uninitialized Data Segment:** Uninitialized data segment often called the “bss” segment, named after an ancient assembler operator that stood for “block started by symbol.” Data in this segment is initialised by the kernel to arithmetic 0 before the program starts executing uninitialized data starts at the end of the data segment and contains all global variables and static variables that are initialised to zero or do not have explicit initialization in source code.

For instance, a variable declared static int i; would be contained in the BSS segment. For instance, a global variable declared int j; would be contained in the BSS segment.

**4. Stack:** The stack area traditionally adjoined the heap area and grew in the opposite direction; when the stack pointer met the heap pointer, free memory was exhausted. (With modern large address spaces and virtual memory techniques they may be placed almost anywhere, but they still typically grow in opposite directions.)

The stack area contains the program stack, a LIFO structure, typically located in the higher parts of memory. On the standard PC x86 computer architecture, it grows toward address zero; on some other architectures, it grows in the opposite direction. A “stack pointer” register tracks the top of the stack; it is adjusted each time a value is “pushed” onto the stack.

**5. Heap:** Heap is the segment where dynamic memory allocation usually takes place.The heap area begins at the end of the BSS segment and grows to larger addresses from there. The Heap area is managed by malloc, realloc, and free, which may use the brk and sbrk system calls to adjust its size (note that the use of brk/sbrk and a single “heap area” is not required to fulfil the contract of malloc/realloc/free; they may also be implemented using mmap to reserve potentially non-contiguous regions of virtual memory into the process’ virtual address space). The Heap area is shared by all shared libraries and dynamically loaded modules in a process.