# **Memory Layout:**

When the program runs, the processing is performed in two spaces called Kernel Spaceand User Spaceon the system. The two processing spaces implicitly interfere with each other and the processing of the program proceeds.

**Kernel Space**

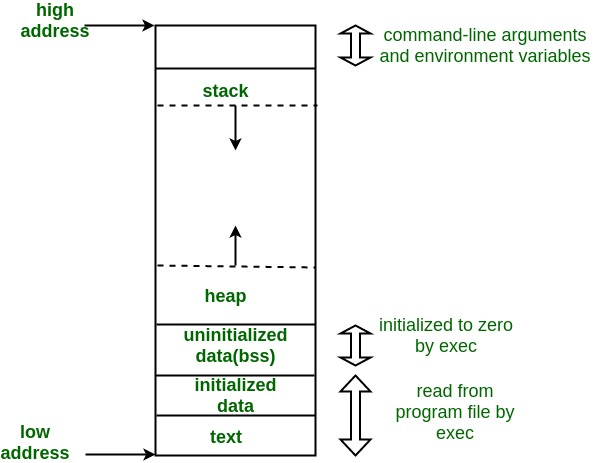
The kernel space can be accessed by user processes only through the use of system calls that are requests in a Unix-like operating system such as input/output (I/O) or process creation.

**User Space**

The user space is a computational resource allocated to a user, and it is a resource that the executing program can directly access. This space can be categorized into some segments.

When we executed any applications on the computer, the computer has to allocate memory for the program to run. The memory that is assigned to a program or application in a computer can be divided into five parts.

1. Text segment  
2. Initialized data segment  
3. Uninitialized data segment  
4. Stack  
5. Heap



1. Text Segment:

A text segment , also known as a code segment.

It contains executable instructions.

The text segment is often read-only, to prevent a program from accidentally modifying its instructions.

2. Initialized Data Segment:

It contains the global variables and static variables that are initialized by the programmer.

Ex:

#include <stdio.h>

**int** global = 10; /\* initialized global variable stored in DS\*/

**int** main(**void**)

{

**static** **int** i = 100; /\* Initialized static variable stored in DS\*/

**return** 0;

}

3. Uninitialized Data Segment:

The Uninitialized data segment called as “bss” (Block Started by Symbol**)** segment.

The data in this segment is initialized by the kernel to arithmetic 0 before the program starts executing.

Ex:

#include <stdio.h>

**int** global; /\* Uninitialized variable stored in bss\*/

**int** main(**void**)

{

**static** **int** i; /\* Uninitialized static variable stored in bss \*/

**return** 0;

}

4. Stack:

The stack is a segment of memory where data like your local variables and function calls get added and/or removed.

When you compile a program, the compiler enters through the main function and a stack frame is created on the stack.

The set of values pushed for one function call is termed a “stack frame”.

The automatic variables are stored, along with information that is saved each time a function is called.

The stack area traditionally adjoined the heap area and grew the opposite direction. When the stack pointer met the heap pointer, free memory was exhausted.

A “stack pointer” register tracks the top of the stack, it is adjusted each time a value is “pushed” onto the stack.

Mainly the following basic operations are performed in the stack:

* Push: Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition.
* Pop: Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.

## **Advantages of using Stack:**

* Stack automatically cleans up the object.
* A stack is used when a variable is not used outside that function.
* When a function is called the local variables are stored in a stack,

and it is automatically destroyed once returned.

* Not easily corrupted
* Variables cannot be resized.

**Stack overflow:**

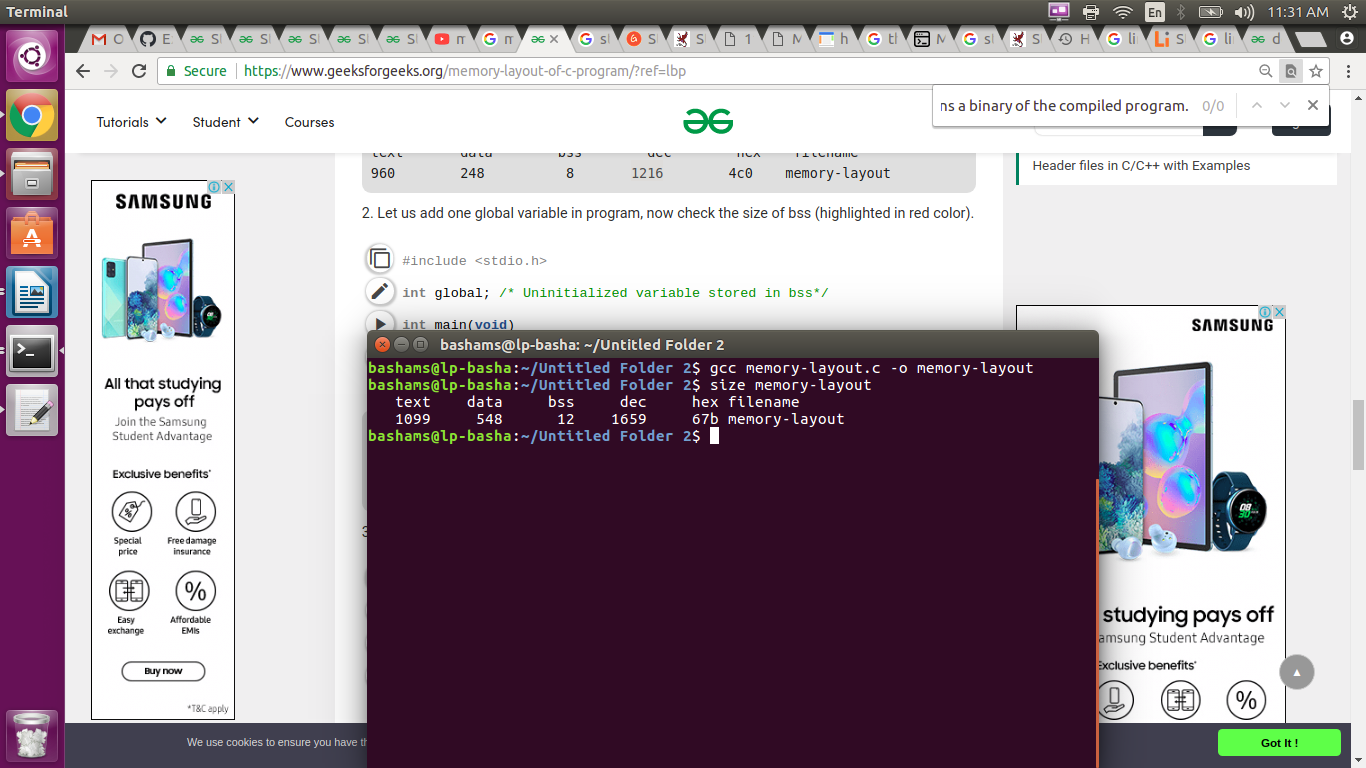
Stack consist of a limited amount of [address space](https://en.wikipedia.org/wiki/Address_space), often determined at the start of the program. The size of the call stack depends on many factors, including the programming language, machine architecture, multi-threading, and amount of available memory. When a program attempts to use more space than is available on the call stack (that is, when it attempts to access memory beyond the call stack's bounds, which is essentially a [buffer overflow](https://en.wikipedia.org/wiki/Buffer_overflow)), the stack is said to overflow, typically resulting in a program crash.

5. Heap:

Heap is the segment where dynamic memory allocation usually takes place.

The Heap area is managed by malloc, realloc, and free.

EX: When a memory-layout.c executed memory allocated for data segment, text segment.

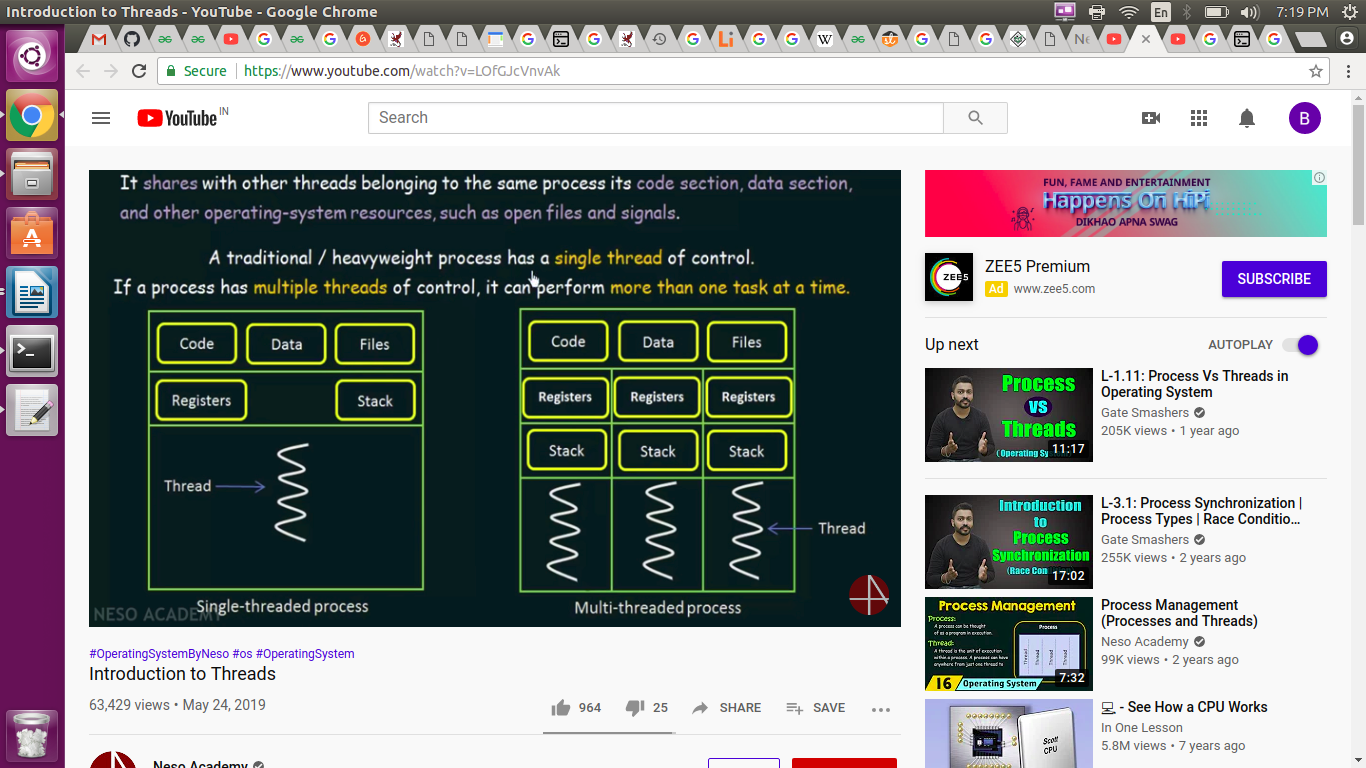


**Treads:**

Program under execution is known as process and thread is a basic unit of execution.

Each program may have number of processes associated with it and each process can have number of threads executing in it.

Threads shares with other threads belonging to the same process its code section, data section and other operating system resources. If a process has single thread it will perform single task, if a process has multiple threads of control, it can perform more than one task at a time.



**Single threaded proess:**

The block considered as process. The code, data, files, registers and stack belonging to the single thread process. This perform only one task at a time.

**multi threaded proess:**

It a single process contain multiple threads and each of thread has own stack and register. Code, data section and files belonging to these process are shared by these threads. Multi thread process much more efficient than single thread process it make computation faster and efficient.

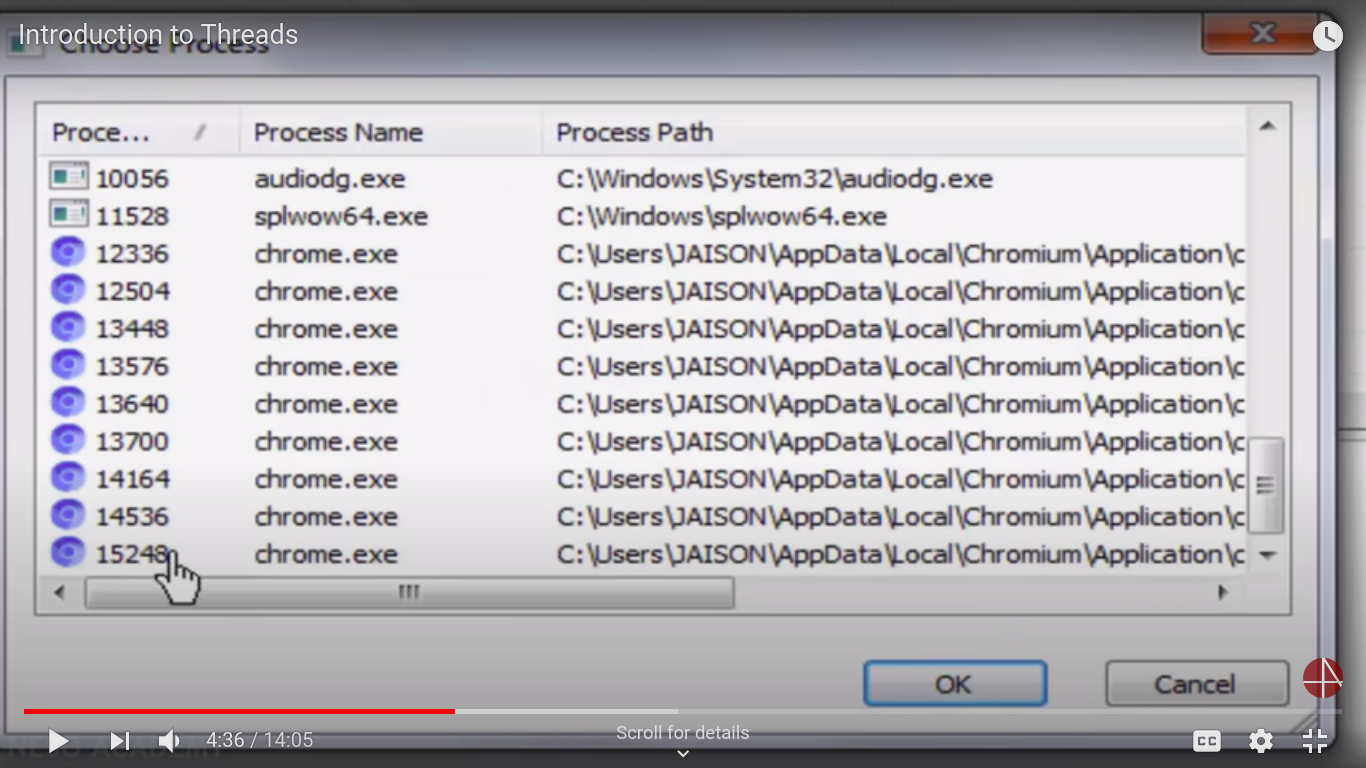
In a multi-threaded process, all of the process threads share the same memory and open files. Within the shared memory, each thread gets its own stack. Each thread has its own instruction pointer and registers. An operating system had to keep track of processes, and stored its per-process information in a data structure called a process control block (PCB).

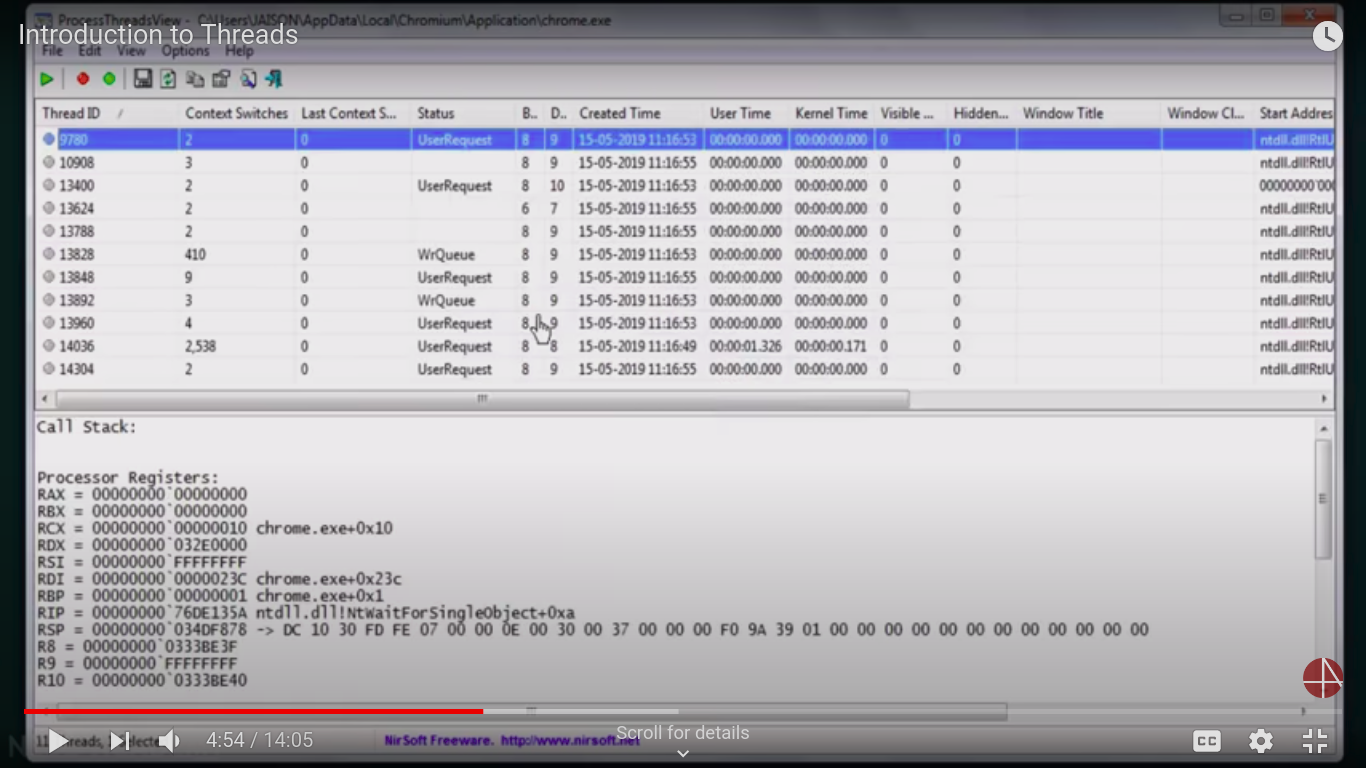
The items that the operating system must store that are unique to each thread are:

* Thread ID
* Saved registers, stack pointer, instruction pointer
* Stack (local variables, temporary variables, return addresses)
* Signal mask
* Priority (scheduling information)

The items that are shared among threads within a process are:

* Text segment (instructions)
* Data segment (static and global data)
* BSS segment (uninitialized data)
* Open file descriptors
* Signals
* Current working directory
* User and group IDs

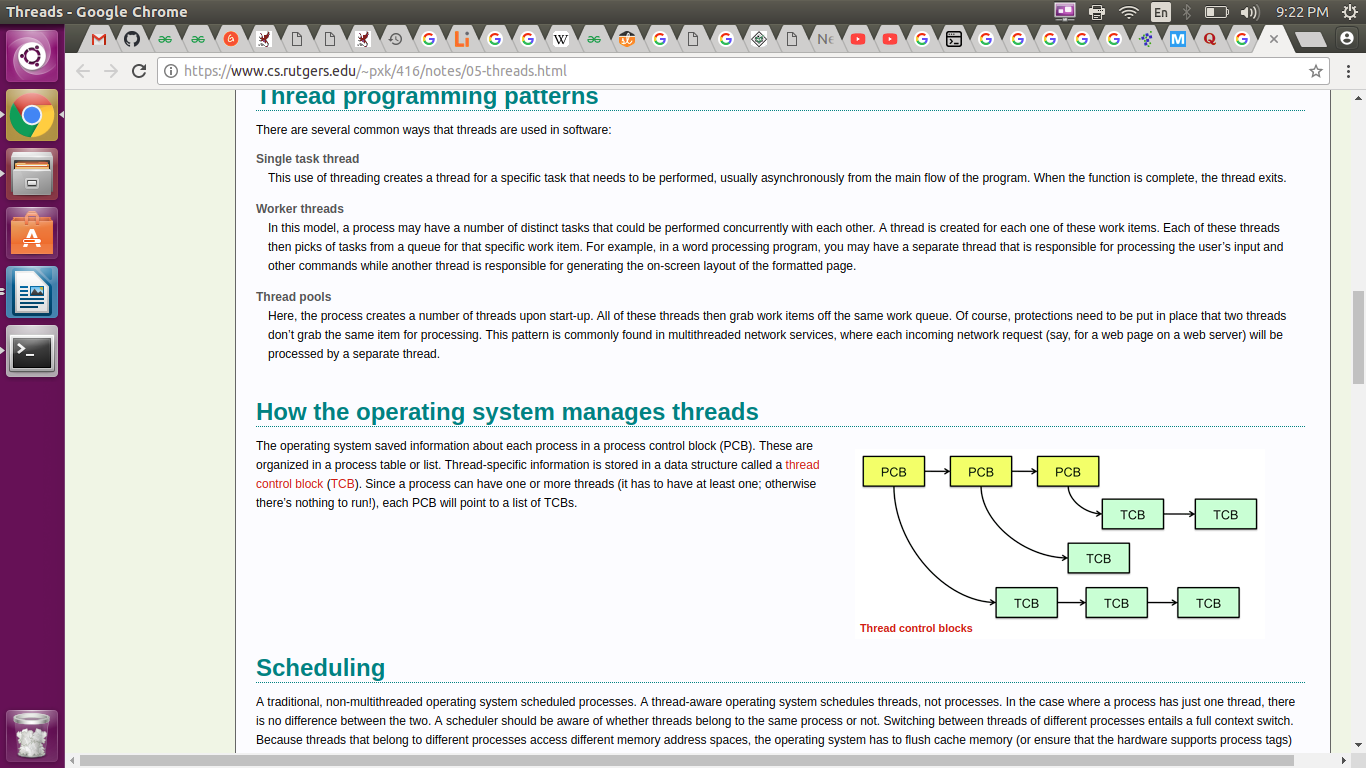
****

****

Ex: One thread using for display window and another thread download from internet.

# **How the operating system manages threads:**

The operating system saved information about each process in a process control block (PCB). These are organized in a process table or list. Thread-specific information is stored in a data structure called a thread control block(TCB), each PCB will point to a list of TCBs.



**Benefits of multi thread :**

**Resource Sharing:**

All the threads of a process share its resources such as memory, data, files etc. A single application can have different threads within the same address space using resource sharing.

**Responsiveness:**

Program responsiveness allows a program to run even if part of it is blocked using multithreading. This can also be done if the process is performing a lengthy operation. For example - A web browser with multithreading can use one thread for user contact and another for image loading at the same time.

**Utilization of Multiprocessor Architecture:**

In a multiprocessor architecture, each thread can run on a different processor in parallel using multithreading. This increases concurrency of the system. This is in direct contrast to a single processor system, where only one process or thread can run on a processor at a time.

**Economy:**

It is more economical to use threads as they share the process resources. Comparatively, it is more expensive and time-consuming to create processes as they require more memory and resources. The overhead for process creation and management is much higher than thread creation and management.