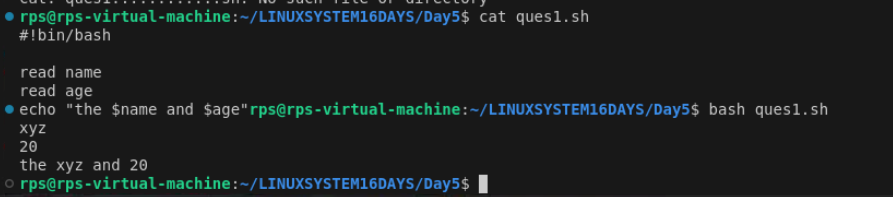
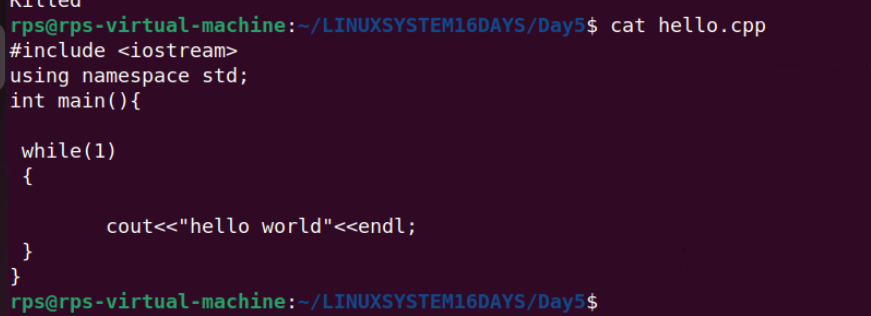
Day5 LSP 22/7/24

|  |  |  |  |
| --- | --- | --- | --- |
| **Ques Write a script that prompts the user for their name and age, then greets them with a personalized message.** | | |



How to kill process

1> create program



2> run it

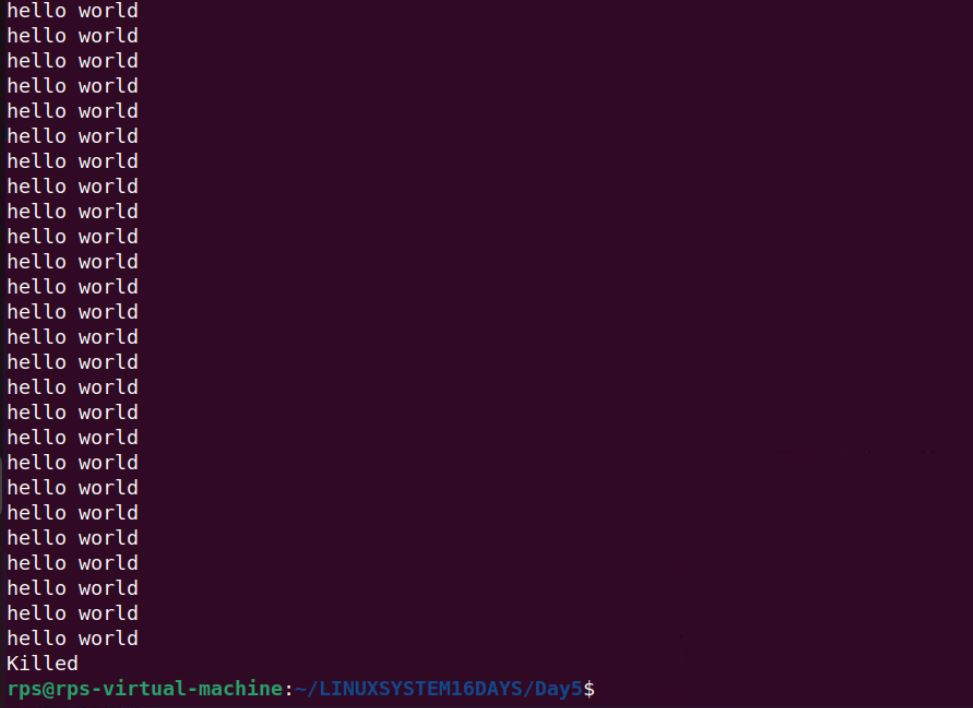
make hello

./hello

3>use htop command - it will show all process state

4> Now select the process you want to kill using the arrow key.

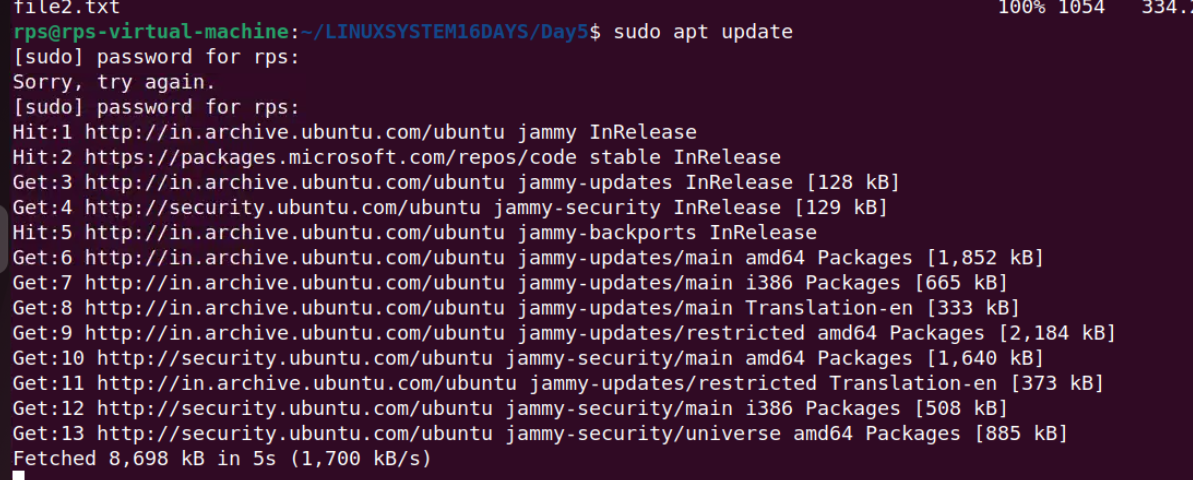
5> then press k and select sigkill



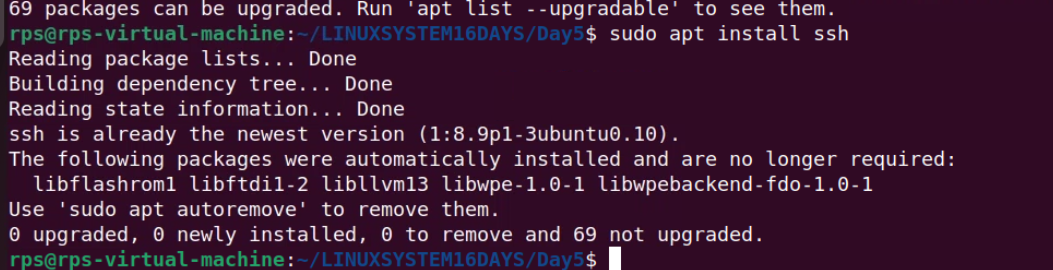
Kill - **The kill command in Unix-like systems is used to send signals to processes, typically to terminate them**

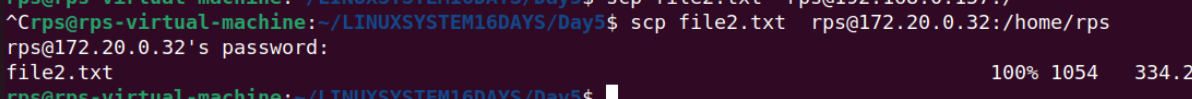
scp -The scp (Secure Copy Protocol) command is used to securely transfer files **and directories between two locations over a network using SSH (Secure Shell).**

**sudo apt update**



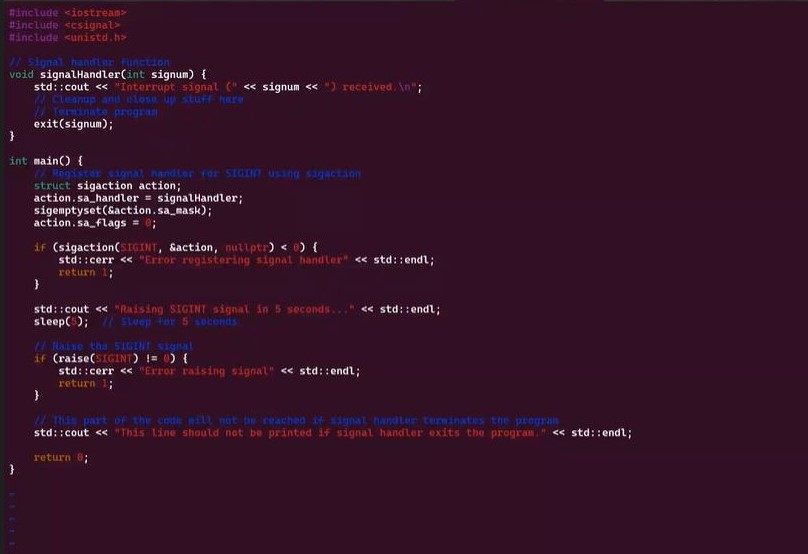
**Sudo apt install ssh**

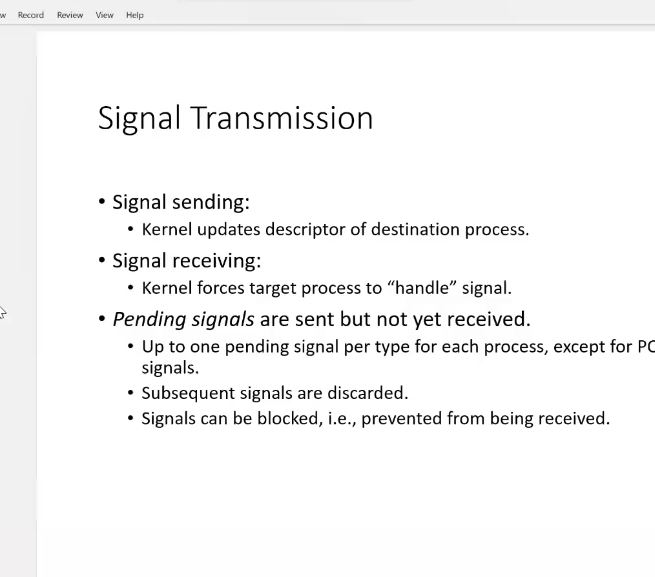


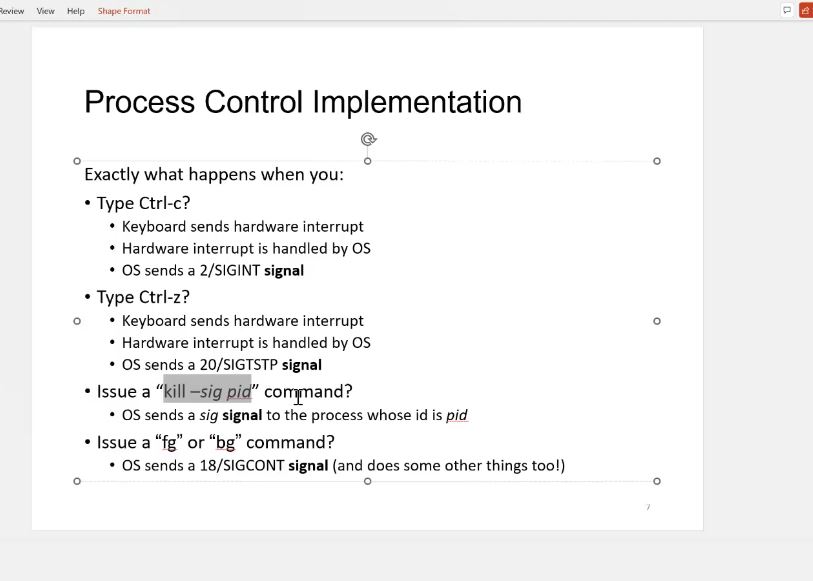


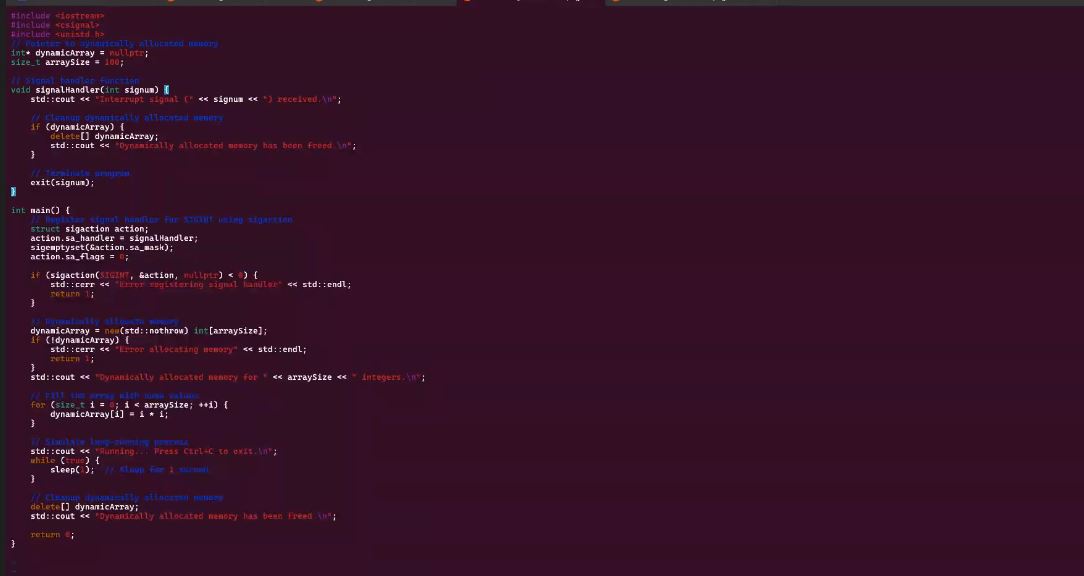












**Basic Handling vs. Advanced Control: Implement signal handling using both signal and sigaction (in separate program runs). Observe the behavior. Which API allows for more control over the signal handler? Explain the key difference in a comment within your code**

**Control Over Flags**: sigaction allows you to specify various flags that control the behavior of the signal handler. For example, SA\_RESTART can be used to automatically restart certain system calls if they are interrupted by a signal.

**Signal Masking**: sigaction allows you to specify a signal mask, which determines which signals are blocked during the execution of the signal handler.

**Handler Function Type**: The sigaction structure can hold both a simple signal handler (sa\_handler) and a more complex handler that takes additional parameters (sa\_sigaction).

**signal**: Simple to use but less control. Suitable for basic signal handling tasks.

**sigaction**: Provides more control and flexibility, allowing you to fine-tune the behavior of your signal handlers. It is generally recommended for more robust and complex signal handling needs.

**Objective: Modify your program to demonstrate graceful termination upon receiving a specific signal (e.g., SIGINT). Within the signal handler, perform any necessary cleanup tasks (e.g., closing files, releasing resources) before exiting the program gracefully.**

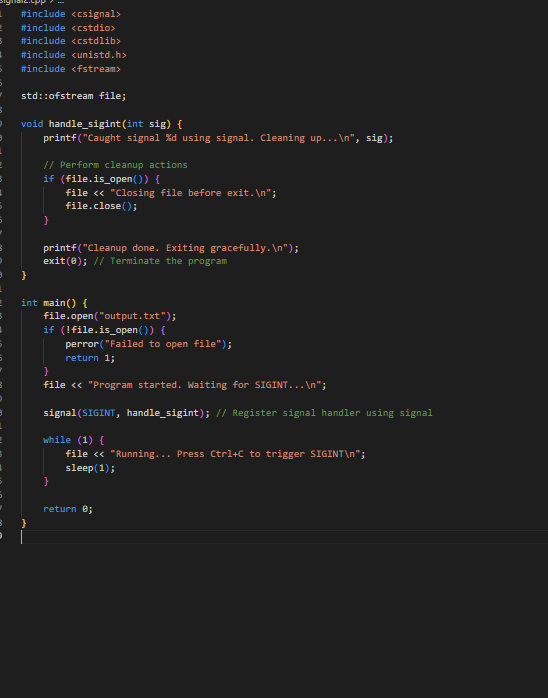
**Implementation:**

**In your signal handler function, include code to perform cleanup actions. This might involve closing open files, releasing memory, or writing data to disk.**

**Use exit(0) or similar methods to terminate the program after cleanup is complete.**

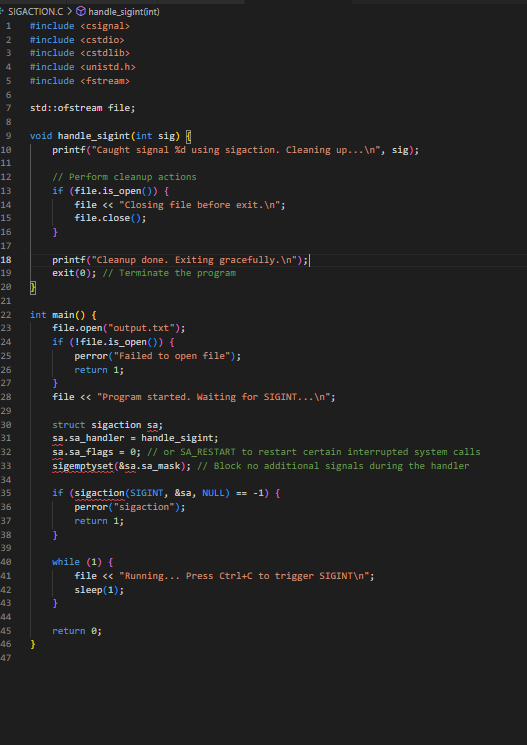
**To demonstrate graceful termination upon receiving a signal like SIGINT, we will modify the signal handler to perform cleanup actions before exiting the program. The program will use both the signal and sigaction APIs in separate runs to show the handling.**

**USING SIGNAL**



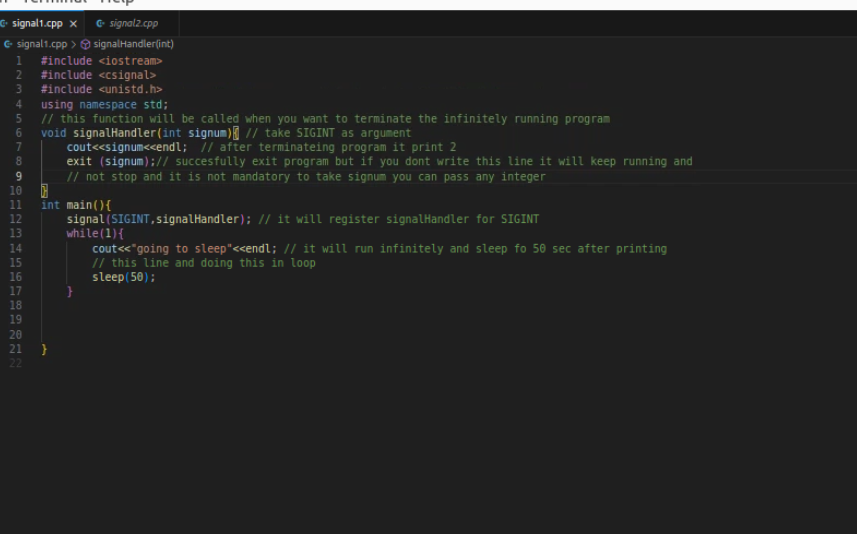
### Using sigaction

**Here is the modified program using the sigaction function:**

****

**#Use of Sigint to terminate the running program**

**For ex- If your program is running infinitely and you want to terminate it so you can use sigint or Ctrl +c to terminate it.**

****

**Signal: It sends a message to the operating system.**

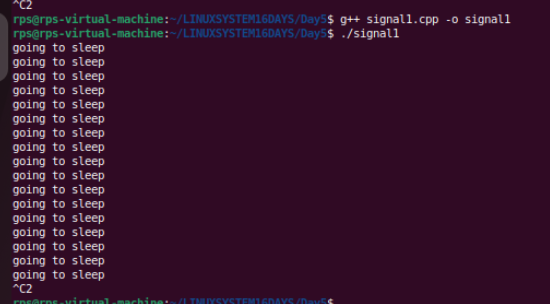
**Signal Handler: It is a function which runs when we get a signal.**

**SIGINT: Special type of function send when when we press Ctrl +C**

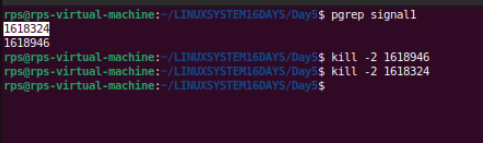
**Signal Handler Registration: To tell the program to run which function when we got a signal.**

**First way to handle infinite loop program**

**-> enter ctrl+c it will call signal handler function it will perform SIGINT and terminate program**

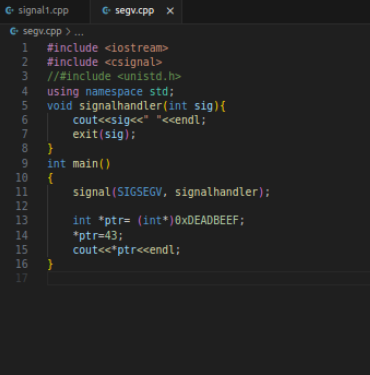
****

**Another way to terminate program using SIGINT**

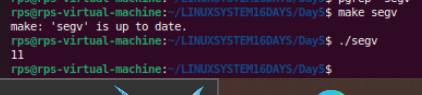
****

* **Pgrep programName —-- to get pid of that program**
* **Kill - signal name or signal no PID**

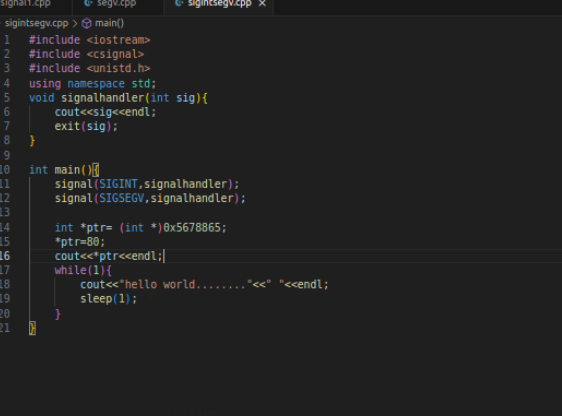
**# Use of SIGSEGV**

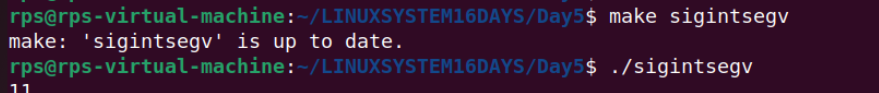
****

**#to show segmentation fault using SIGSEGV**

****

**#SIGINT and SIGSEGV**

****

****