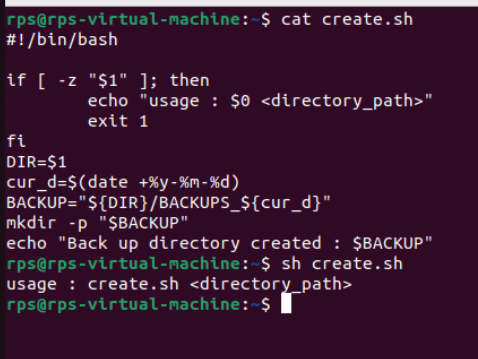
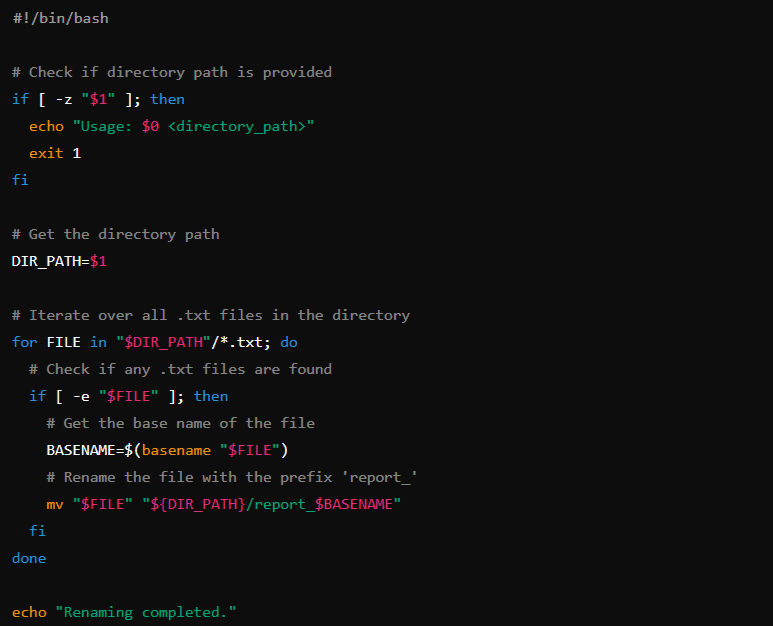
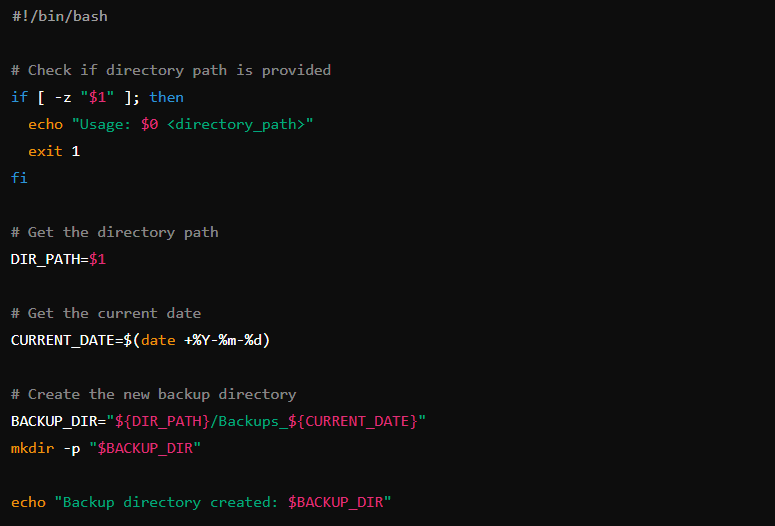
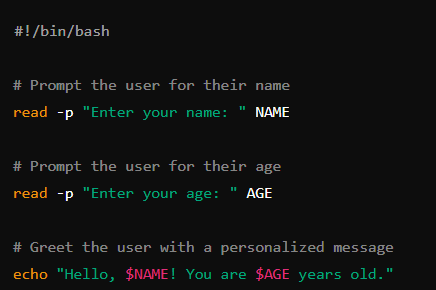
1.Write a script that takes a directory path as input and creates a new directory within it named "Backups\_$(date +%Y-%m-%d)". Create a script that renames all files in a directory with the extension ".txt" to have a prefix of "report\_".

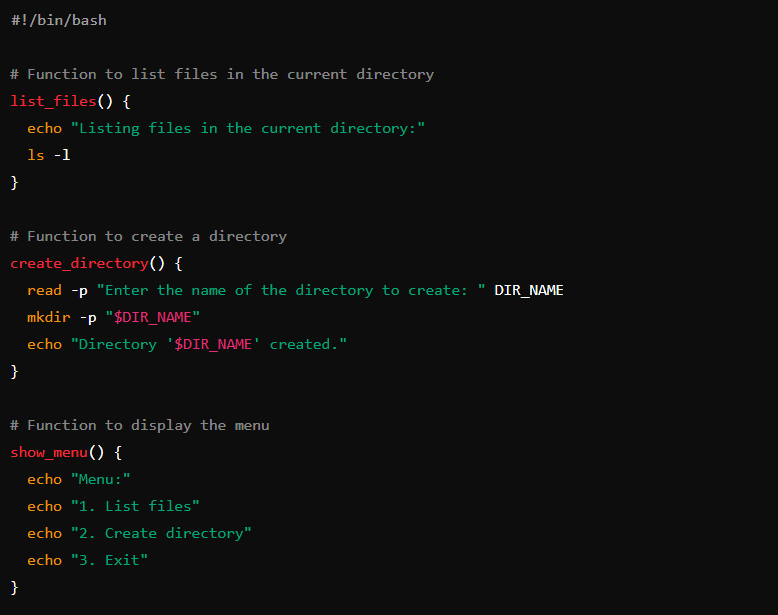


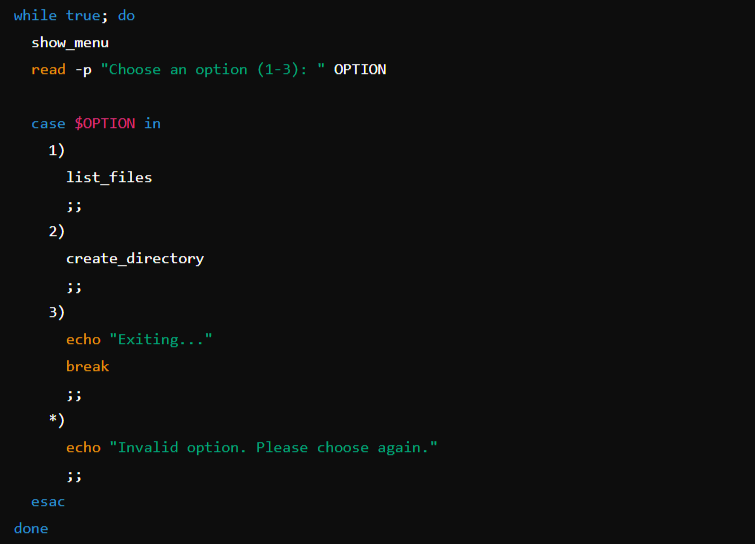




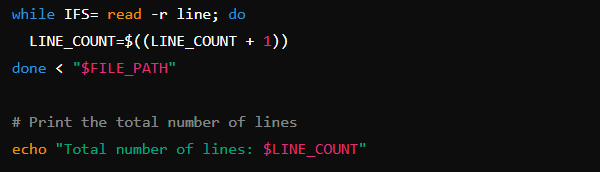
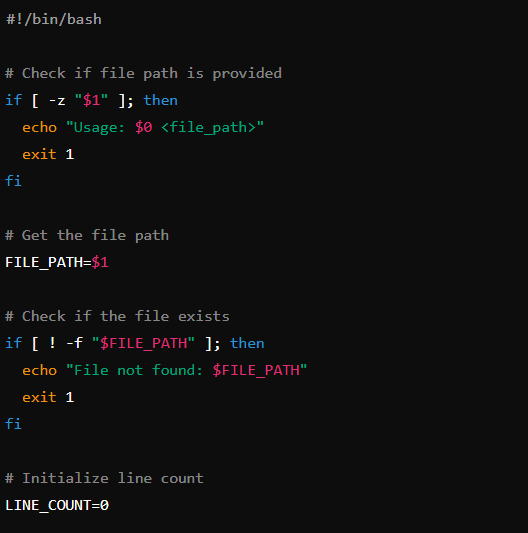
2. Write a script that prompts the user for their name and age, then greets them with a personalized message. Design a script that displays a menu with options like "List files," "Create directory," and "Exit." Allow the user to choose an option and perform the corresponding action.

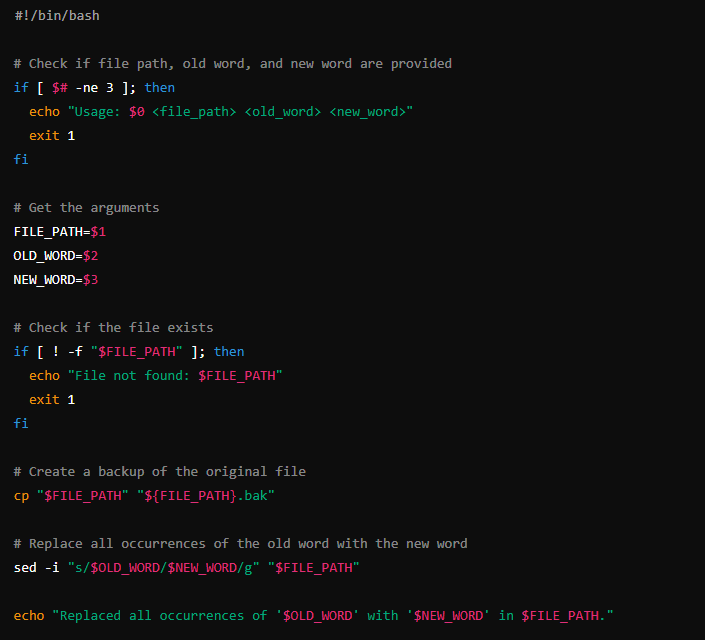




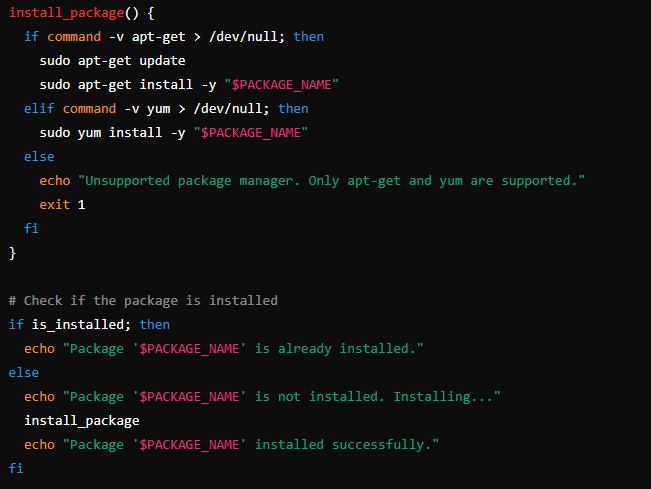
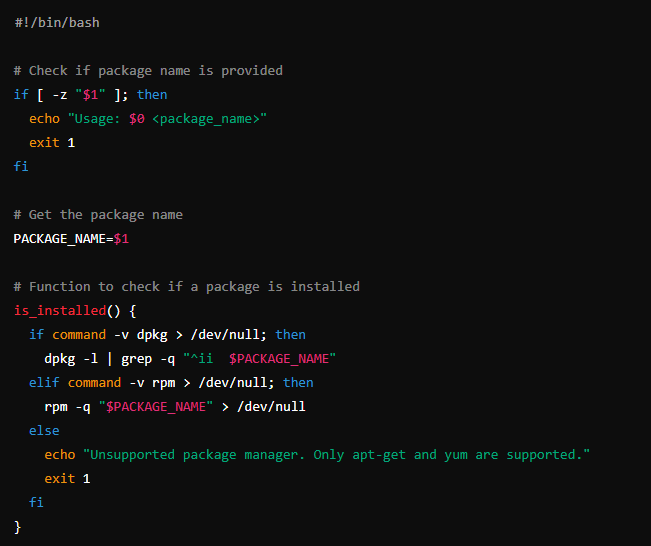


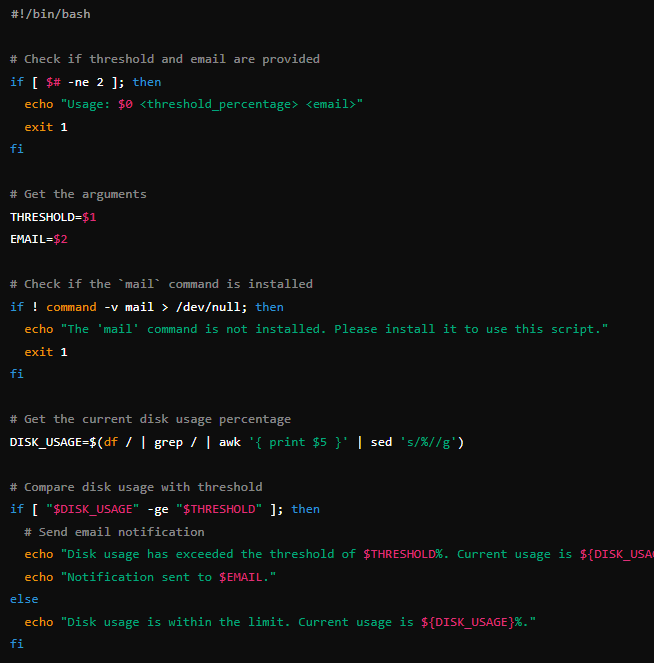
3. Write a script that reads the contents of a file line by line, counts the number of lines, and prints the total.Create a script that takes a text file as input and replaces all occurrences of a specific word with another word.



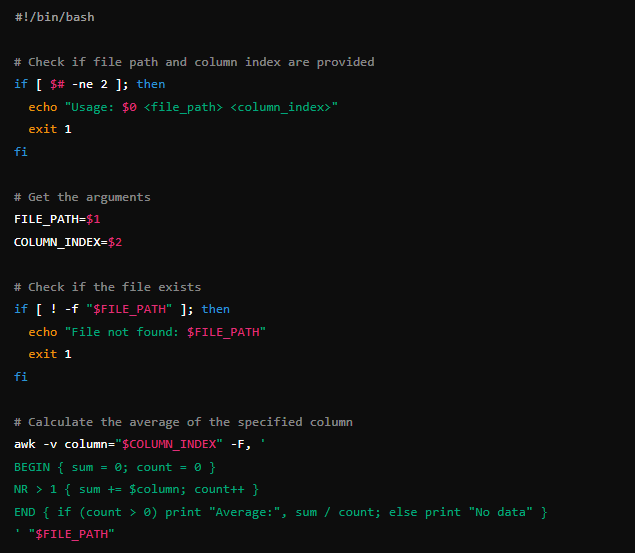


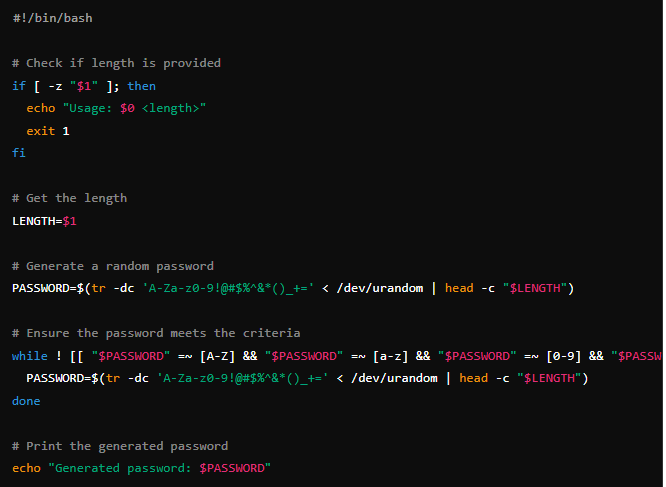
4.Write a script that checks if a specific package is installed and, if not, installs it using the appropriate package manager (e.g., apt-get, yum).Create a script that monitors disk usage and sends an email notification if it exceeds a certain threshold.



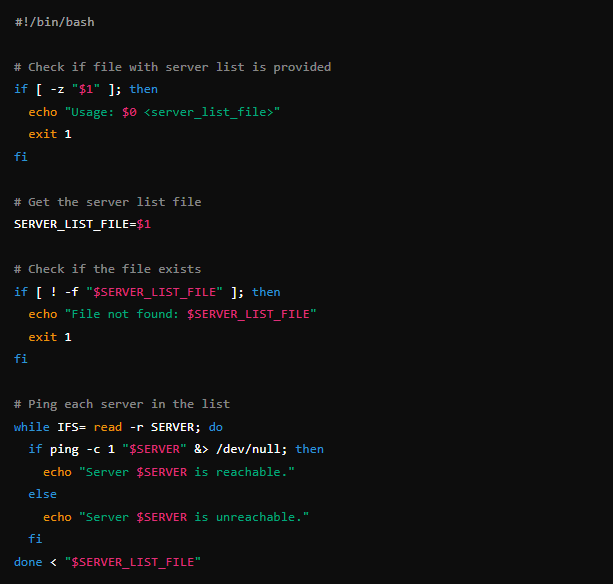


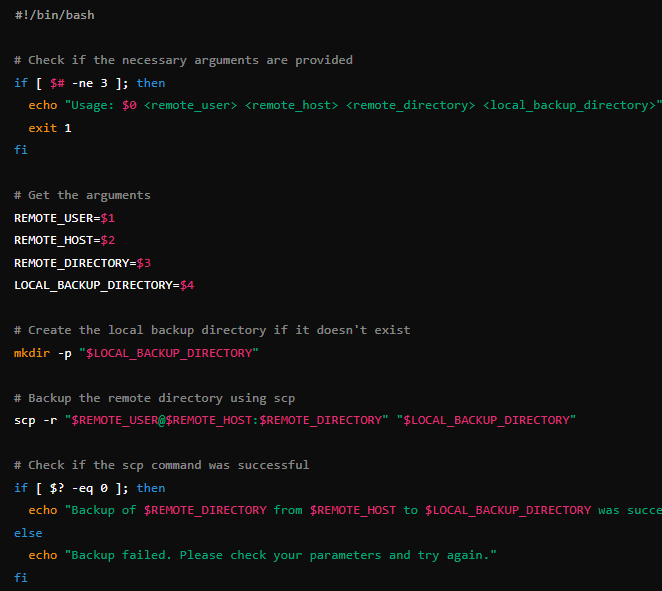
5. Write a script that reads a CSV file, calculates the average of a specific column, and prints the result. Create a script that generates a random password of a specified length, meeting certain criteria like uppercase, lowercase, numbers, and symbols.



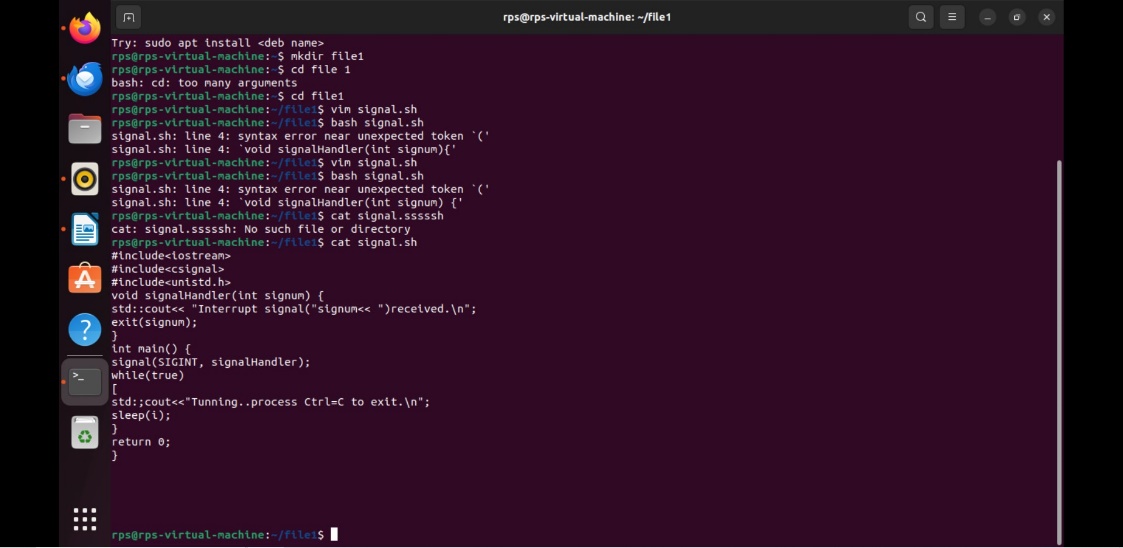


6. Write a script that pings a list of servers and reports if any are unreachable. Create a script that backs up a remote directory on another server to a local directory using tools like scp.

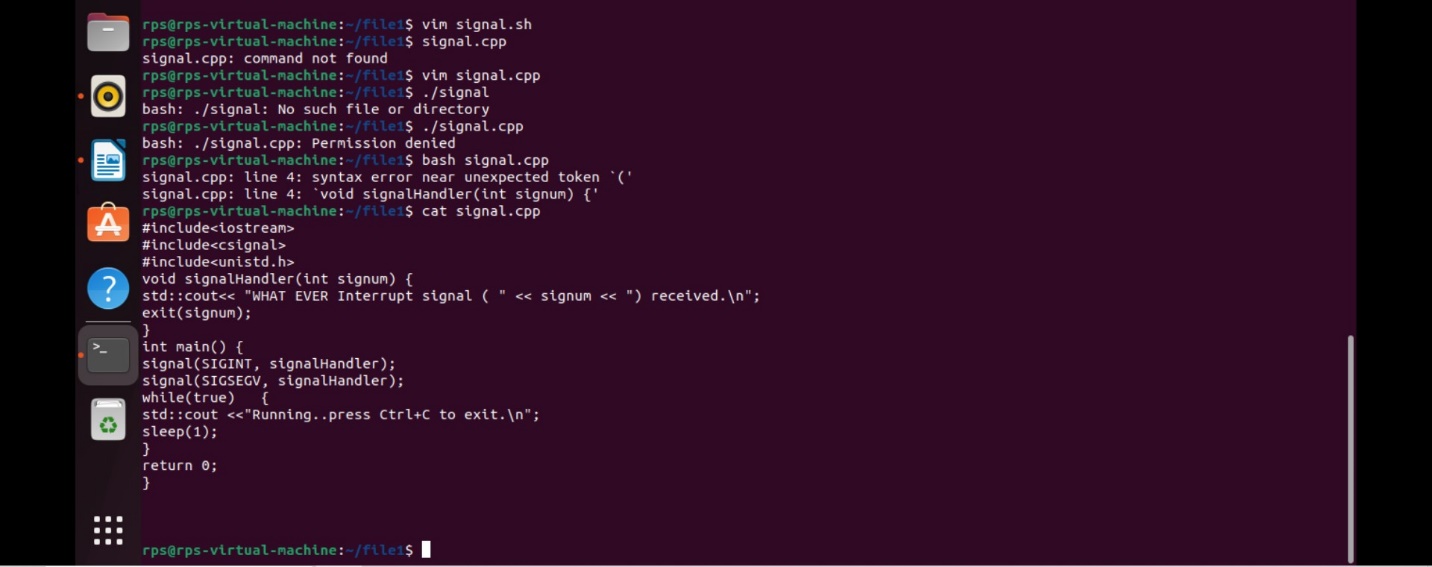




**Signal:**









**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.**

#include <iostream>

#include <signal.h>

#include <unistd.h>

void handle\_signal(int sig) {

std::cout << "Caught signal " << sig << std::endl;

}

int main() {

if (signal(SIGINT, handle\_signal) == SIG\_ERR) {

perror("signal");

return 1;

}

while (true) {

std::cout << "Running... Press Ctrl+C to send SIGINT" << std::endl;

sleep(1);

}

return 0;

}

#include <iostream>

#include <signal.h>

#include <unistd.h>

void handle\_signal(int sig) {

std::cout << "Caught signal " << sig << std::endl;

}

int main() {

struct sigaction sa;

sa.sa\_handler = handle\_signal;

sa.sa\_flags = 0;

sigemptyset(&sa.sa\_mask);

if (sigaction(SIGINT, &sa, NULL) == -1) {

perror("sigaction");

return 1;

}

while (true) {

std::cout << "Running... Press Ctrl+C to send SIGINT" << std::endl;

sleep(1);

}

return 0;

}

Example 1:

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.

#include <iostream>

#include <signal.h>

#include <unistd.h>

void handle\_signal(int sig) {

std::cout << "Caught signal " << sig << std::endl;

}

int main() {

if (signal(SIGINT, handle\_signal) == SIG\_ERR) {

perror("signal");

return 1;

}

while (true) {

std::cout << "Running... Press Ctrl+C to send SIGINT" << std::endl;

sleep(1);

}

return 0;

}

Example 2:

Graceful Termination with Signal Handling

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.

#include <iostream>

#include <signal.h>

#include <unistd.h>

void handle\_signal(int sig) {

std::cout << "Caught signal " << sig << std::endl;

}

int main() {

struct sigaction sa;

sa.sa\_handler = handle\_signal;

sa.sa\_flags = 0;

sigemptyset(&sa.sa\_mask);

if (sigaction(SIGINT, &sa, NULL) == -1) {

perror("sigaction");

return 1;

}

while (true) {

std::cout << "Running... Press Ctrl+C to send SIGINT" << std::endl;

sleep(1);

}

return 0;

}

Example 3:

