

### Question 10 (2024eb03003):

Please find screenshot of C program below and attaching **signal\_handling.c** code to GitHub repository:

```
som@linux-vm: ~/Desktop
GNU nano 7.2 signal_handling.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>

volatile sig_atomic_t signal_received = 0;
pid_t sigterm_child = 0, sigint_child = 0;

// Signal handler for SIGTERM
void sigterm_handler(int sig) {
    signal_received = SIGTERM;
    printf("\n[Parent PID %d] Caught SIGTERM from child PID %d\n", getpid(), sigterm_child);
}

// Signal handler for SIGINT
void sigint_handler(int sig) {
    signal_received = SIGINT;
    printf("\n[Parent PID %d] Caught SIGINT from child PID %d\n", getpid(), sigint_child);
}

int main() {
    pid_t pid1, pid2;

    // Register signal handlers
    signal(SIGTERM, sigterm_handler);
    signal(SIGINT, sigint_handler);

    printf("Parent PID %d starting - waiting indefinitely for signals...\n", getpid());

    // Create first child (SIGTERM after 5s)
    pid1 = fork();
    if (pid1 == 0) {
        // Child 1: SIGTERM sender
        sigterm_child = getpid();
        sleep(5);
        printf("[Child PID %d] Sending SIGTERM to parent PID %d\n", getpid(), getppid());
        kill(getppid(), SIGTERM);
        exit(0);
    }

    // Create second child (SIGINT after 10s)
    pid2 = fork();
    if (pid2 == 0) {
        // Child 2: SIGINT sender
        sigint_child = getpid();
        sleep(10);
    }
}
```

```
printf("[Child PID %d] Sending SIGINT to parent PID %d\n", getpid(), getppid());
kill(getppid(), SIGINT);
exit(0);
}

// Parent runs indefinitely, handling signals
while (1) {
    pause(); // Wait for signals

    if (signal_received == SIGTERM) {
        printf("[Parent] Handling SIGTERM: cleaning up child %d\n", pid2);
        kill(pid2, SIGTERM); // Terminate remaining child
        break;
    }
    else if (signal_received == SIGINT) {
        printf("[Parent] Handling SIGINT: cleaning up child %d\n", pid1);
        kill(pid1, SIGTERM); // Terminate remaining child
        break;
    }
}

// Cleanup: wait for children
int status;
waitpid(pid1, &status, 0);
waitpid(pid2, &status, 0);

printf("[Parent PID %d] Exiting gracefully after signal handling\n", getpid());
return 0;
}

^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo
^X Exit      ^R Read File  ^_ Replace    ^U Paste      ^J Justify    ^_ Go To Line M-E Redo
```

## Testing the signal\_handling.c code

### Test Case:

Compile the c program by running below command

```
gcc signal_handling.c -o signal_handling
```

```
./signal_handling
```

```
som@linux-vm: ~/Desktop
som@linux-vm:~/Desktop$ gcc signal_handling.c -o signal_handling
som@linux-vm:~/Desktop$ ./signal_handling
Parent PID 8090 starting - waiting indefinitely for signals...
[Child PID 8091] Sending SIGTERM to parent PID 8090

[Parent PID 8090] Caught SIGTERM from child PID 8091
[Parent] Handling SIGTERM - cleaning up SIGINT child 8092

[Parent PID 8092] Caught SIGTERM from child PID 8091
[Child PID 8092] Sending SIGINT to parent PID 8090

[Parent PID 8090] Caught SIGINT from child PID 8092
[Parent PID 8090] Exited gracefully after signal handling
som@linux-vm:~/Desktop$
```

Validation Complete: The C program successfully:

The program demonstrates:

- Indefinite parent execution (pause())
- Timed child signal transmission (sleep() + kill())
- Correct PID tracking in handlers (fixed from 0 → actual PIDs)
- Distinct SIGTERM/SIGINT handling
- Graceful termination with child cleanup (waitpid())