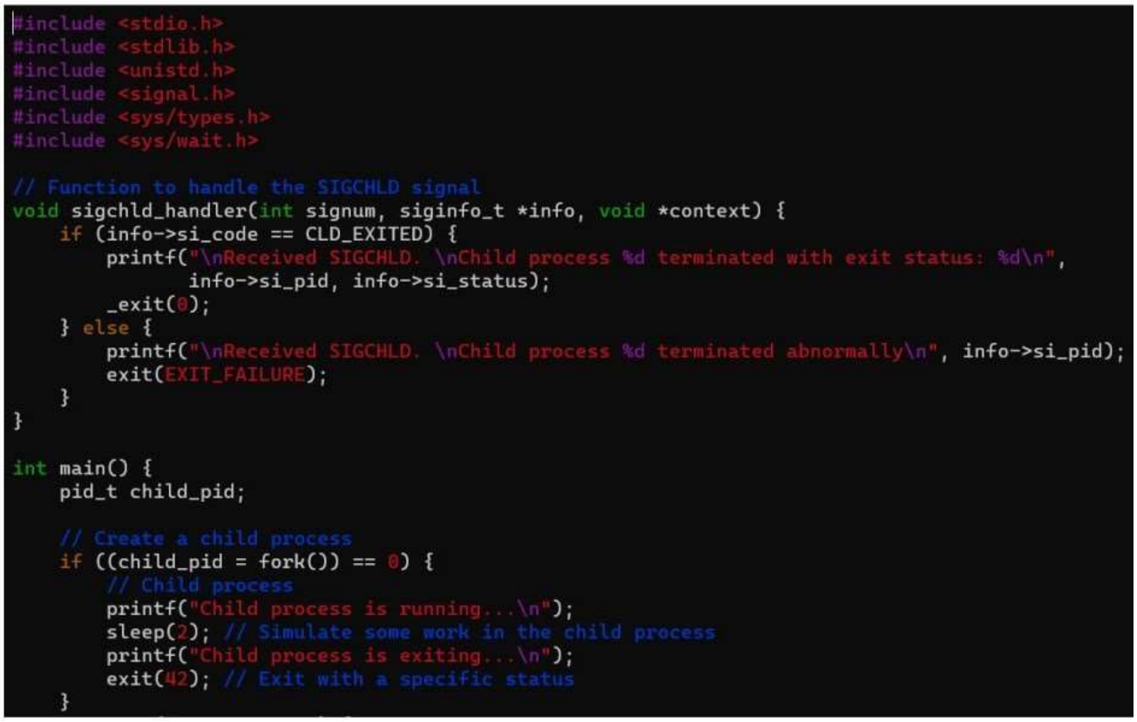
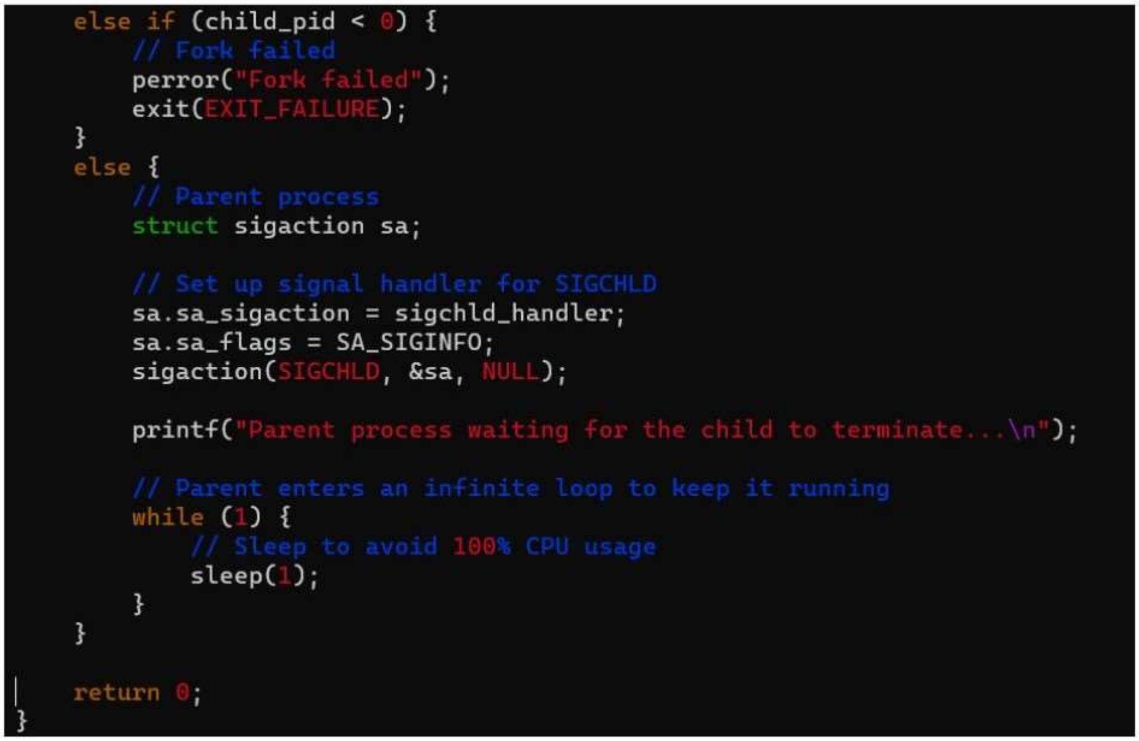
Abhishek Patel
20214053
CS5A

Assignment 8
System Call Lab - 2

1. Write a program a parent process receives a SIGCHLD signal after termination of child process. In this program create a child and a parent process. Once the child process terminates, the parent process will receive a SIGCHLD signal. This signal will invoke a handler which will print the EXIT STATUS of the child process.
Implement program # 3 of assignment for week # 7, and specify the action to be associated with the SIGCHLD signal using sigaction() function.

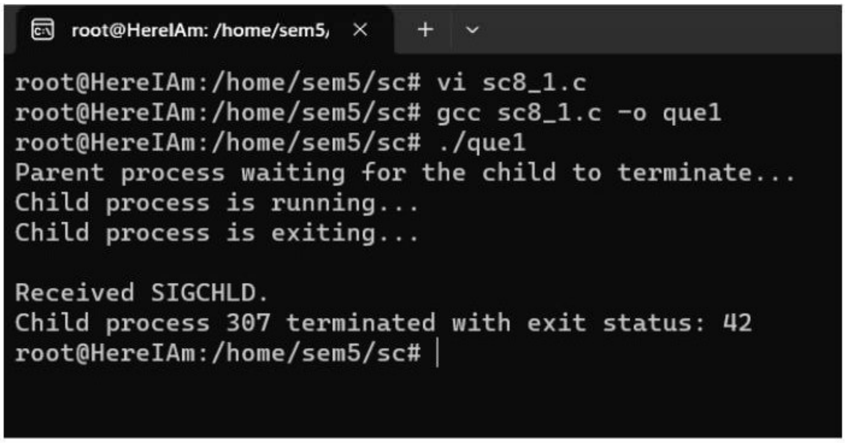


```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>

// Function to handle the SIGCHLD signal
void sigchld_handler(int signum, siginfo_t *info, void *context) {
    if (info->si_code == CLD_EXITED) {
        printf("\nReceived SIGCHLD. \nChild process %d terminated with exit status: %d\n",
               info->si_pid, info->si_status);
        _exit(0);
    } else {
        printf("\nReceived SIGCHLD. \nChild process %d terminated abnormally\n", info->si_pid);
        exit(EXIT_FAILURE);
    }
}

int main() {
    pid_t child_pid;

    // Create a child process
    if ((child_pid = fork()) == 0) {
        // Child process
        printf("Child process is running...\n");
        sleep(2); // Simulate some work in the child process
        printf("Child process is exiting...\n");
        exit(42); // Exit with a specific status
    }
```



```c
    else if (child_pid < 0) {
        // Fork failed
        perror("Fork failed");
        exit(EXIT_FAILURE);
    }
    else {
        // Parent process
        struct sigaction sa;

        // Set up signal handler for SIGCHLD
        sa.sa_sigaction = sigchld_handler;
        sa.sa_flags = SA_SIGINFO;
        sigaction(SIGCHLD, &sa, NULL);

        printf("Parent process waiting for the child to terminate...\n");

        // Parent enters an infinite loop to keep it running
        while (1) {
            // Sleep to avoid 100% CPU usage
            sleep(1);
        }
    }

    return 0;
}
```

Output:

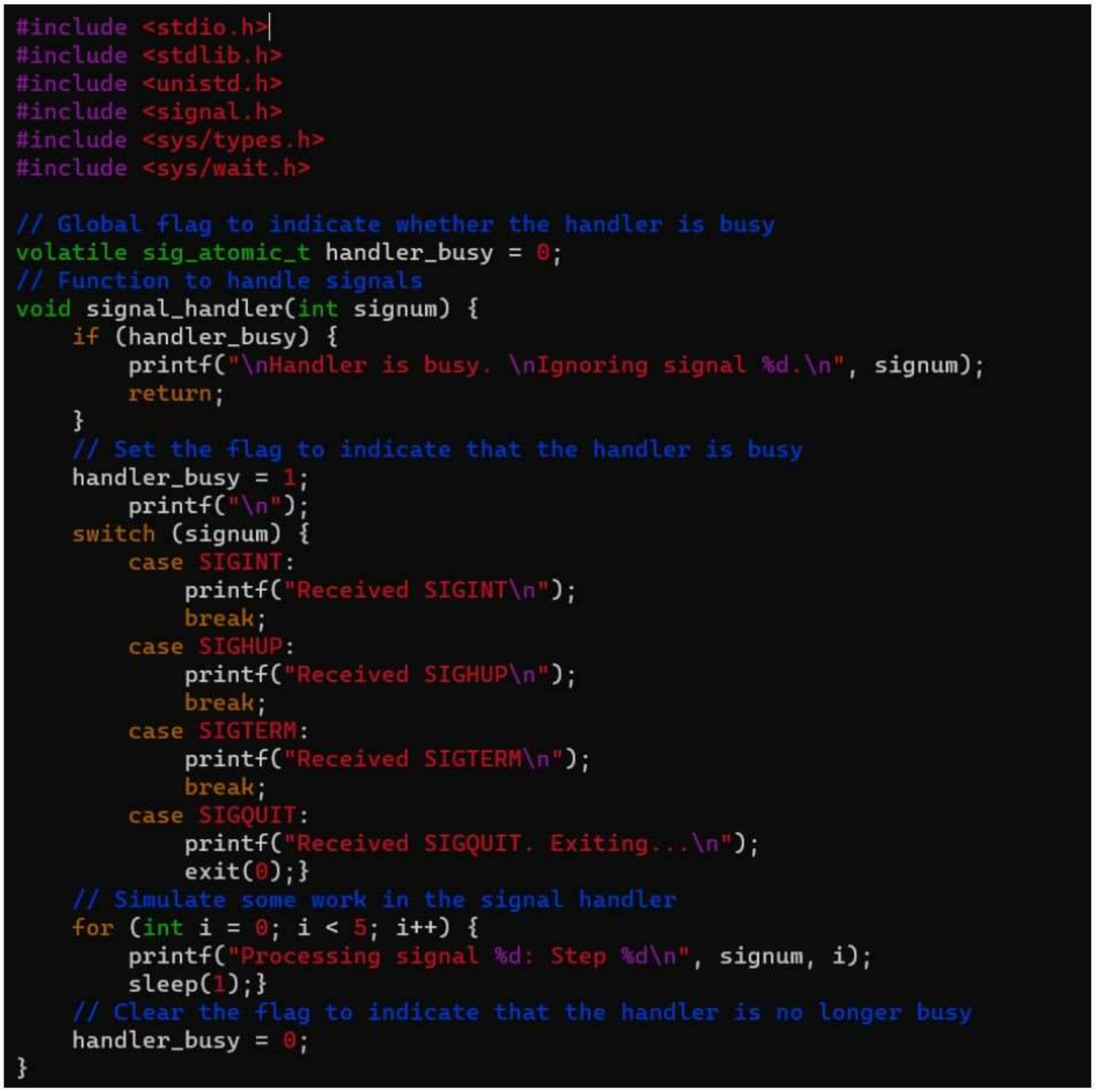


```
root@HereIAm:/home/sem5/sc# vi sc8_1.c
root@HereIAm:/home/sem5/sc# gcc sc8_1.c -o que1
root@HereIAm:/home/sem5/sc# ./que1
Parent process waiting for the child to terminate...
Child process is running...
Child process is exiting...

Received SIGCHLD.
Child process 307 terminated with exit status: 42
root@HereIAm:/home/sem5/sc# 
```

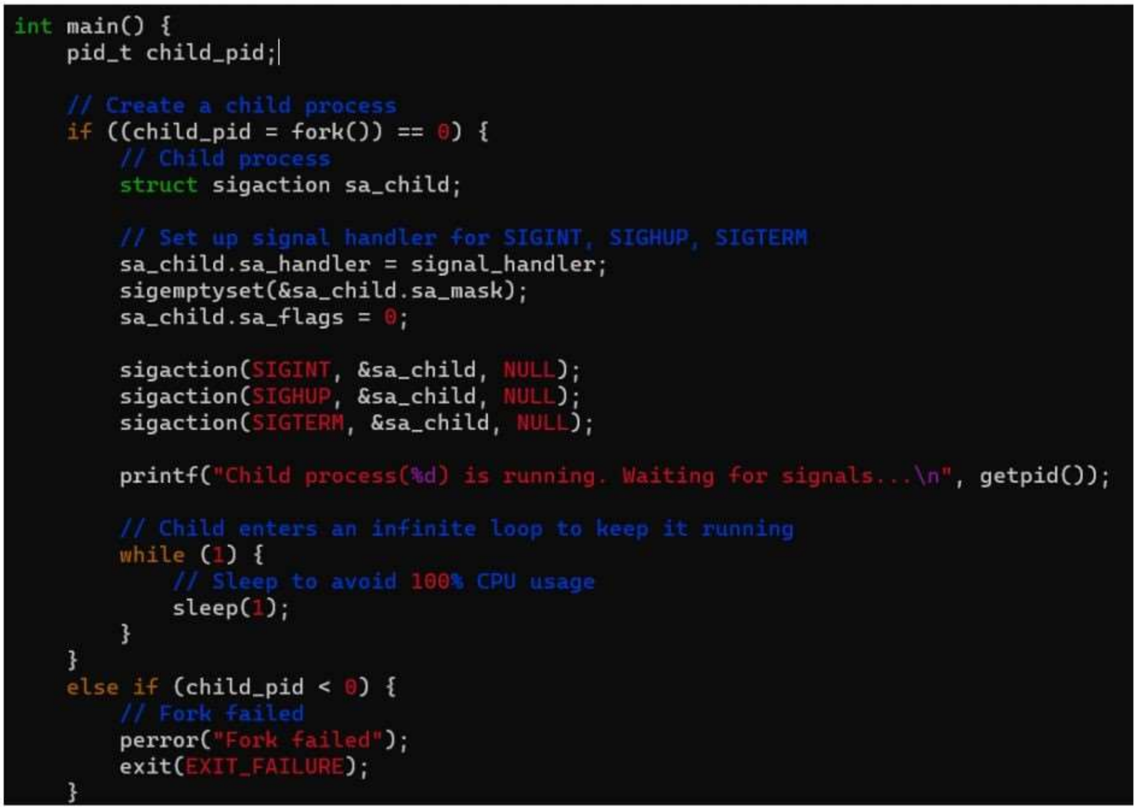
2. Write a program to handle different types of SIGNALS using the same handler function. In this program the process will receive different types of signals from system (some other process or by the user itself). The process needs to handle SIGINT, SIGHUP, SIGTERM and for every interrupt the same handler should inform the type of SIGNAL occurred. The process need to keep running until it encounters SIGQUIT.

Extension of Program No. 1 Implement program no. 1 where the signals are generated by the parent process for child process. Once the child process receives signals generated by parent process. It needs to invoke the signal handler as per the signal generated.

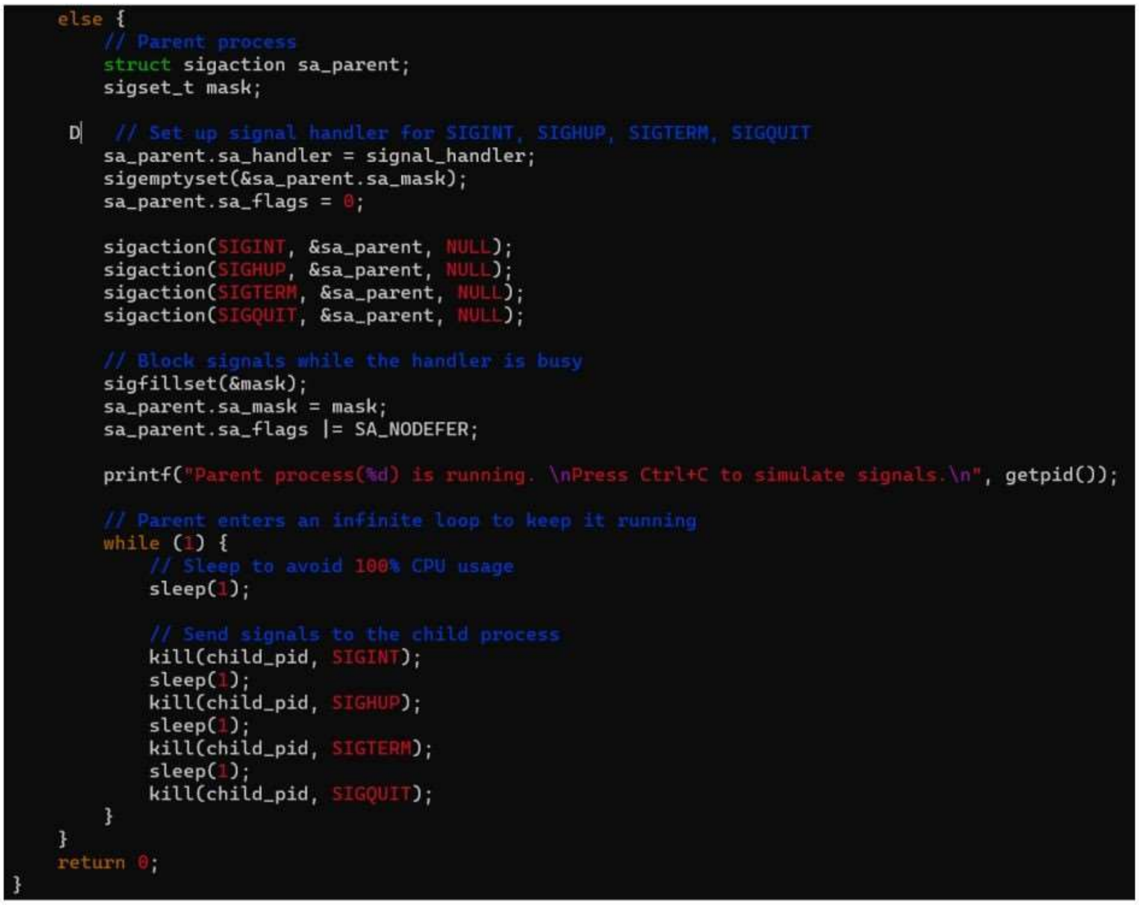
Implement program # 4 of assignment for week # 7, using sigaction() and modify the program to mask any other signal that appears, while the signal handler is busy processing a signal . Hint: Define the signal_handler in such a way, that it remains under execution for some time (loop may be used), when another signal gets generated.



```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>

// Global flag to indicate whether the handler is busy
volatile sig_atomic_t handler_busy = 0;
// Function to handle signals
void signal_handler(int signum) {
    if (handler_busy) {
        printf("\nHandler is busy. \nIgnoring signal %d.\n", signum);
        return;
    }
    // Set the flag to indicate that the handler is busy
    handler_busy = 1;
        printf("\n");
    switch (signum) {
        case SIGINT:
            printf("Received SIGINT\n");
            break;
        case SIGHUP:
            printf("Received SIGHUP\n");
            break;
        case SIGTERM:
            printf("Received SIGTERM\n");
            break;
        case SIGQUIT:
            printf("Received SIGQUIT. Exiting...\n");
            exit(0);}
    // Simulate some work in the signal handler
    for (int i = 0; i < 5; i++) {
        printf("Processing signal %d: Step %d\n", signum, i);
        sleep(1);}
    // Clear the flag to indicate that the handler is no longer busy
    handler_busy = 0;
}
```

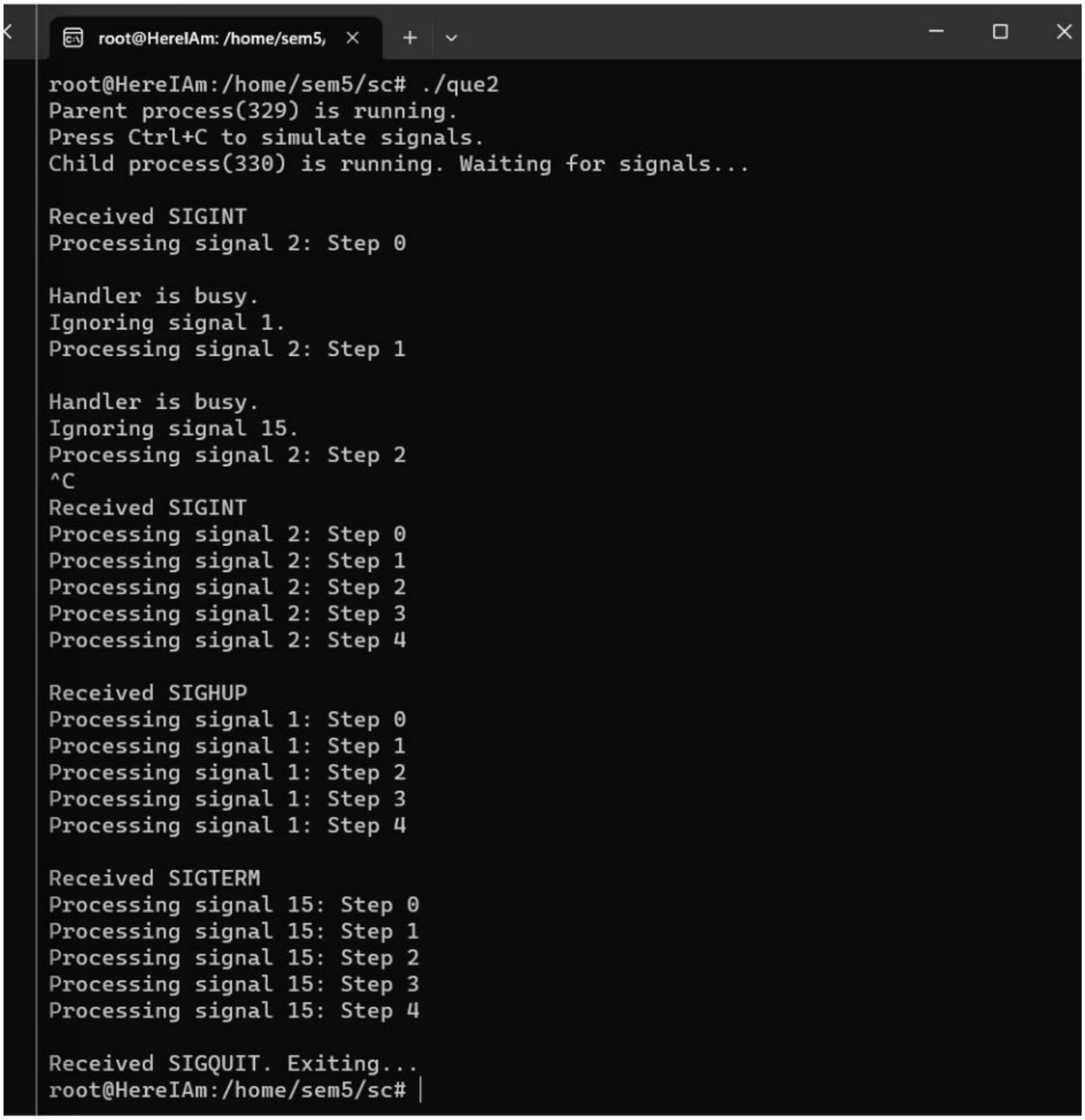


```c
int main() {
    pid_t child_pid;

    // Create a child process
    if ((child_pid = fork()) == 0) {
        // Child process
        struct sigaction sa_child;

        // Set up signal handler for SIGINT, SIGHUP, SIGTERM
        sa_child.sa_handler = signal_handler;
        sigemptyset(&sa_child.sa_mask);
        sa_child.sa_flags = 0;

        sigaction(SIGINT, &sa_child, NULL);
        sigaction(SIGHUP, &sa_child, NULL);
        sigaction(SIGTERM, &sa_child, NULL);

        printf("Child process(%d) is running. Waiting for signals...\n", getpid());

        // Child enters an infinite loop to keep it running
        while (1) {
            // Sleep to avoid 100% CPU usage
            sleep(1);
        }
    }
    else if (child_pid < 0) {
        // Fork failed
        perror("Fork failed");
        exit(EXIT_FAILURE);
    }
```

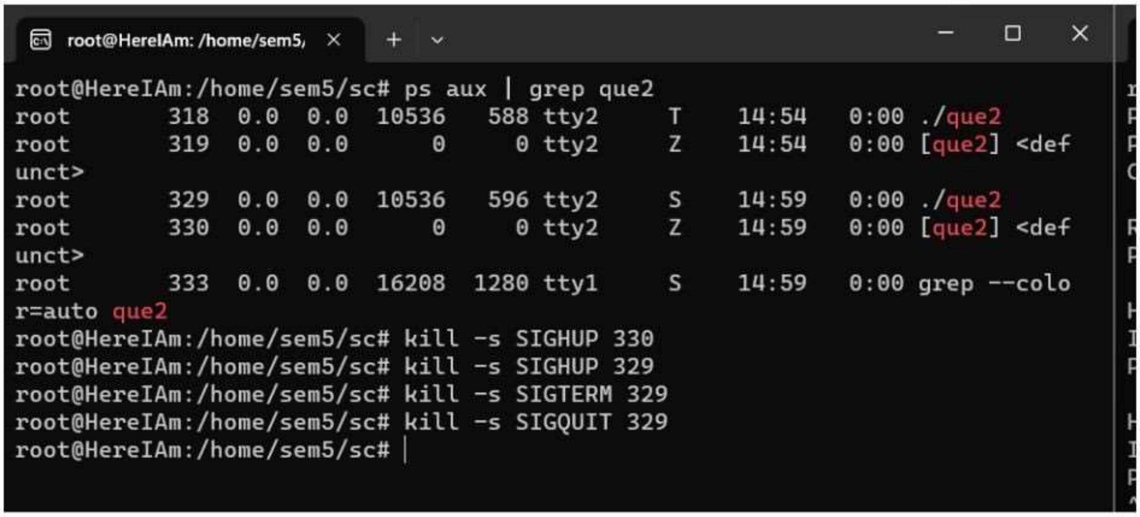


```c
    else {
        // Parent process
        struct sigaction sa_parent;
        sigset_t mask;

    D   // Set up signal handler for SIGINT, SIGHUP, SIGTERM, SIGQUIT
        sa_parent.sa_handler = signal_handler;
        sigemptyset(&sa_parent.sa_mask);
        sa_parent.sa_flags = 0;

        sigaction(SIGINT, &sa_parent, NULL);
        sigaction(SIGHUP, &sa_parent, NULL);
        sigaction(SIGTERM, &sa_parent, NULL);
        sigaction(SIGQUIT, &sa_parent, NULL);

        // Block signals while the handler is busy
        sigfillset(&mask);
        sa_parent.sa_mask = mask;
        sa_parent.sa_flags |= SA_NODEFER;

        printf("Parent process(%d) is running. \nPress Ctrl+C to simulate signals.\n", getpid());

        // Parent enters an infinite loop to keep it running
        while (1) {
            // Sleep to avoid 100% CPU usage
            sleep(1);

            // Send signals to the child process
            kill(child_pid, SIGINT);
            sleep(1);
            kill(child_pid, SIGHUP);
            sleep(1);
            kill(child_pid, SIGTERM);
            sleep(1);
            kill(child_pid, SIGQUIT);
        }
    }
    return 0;
}
```

Output:

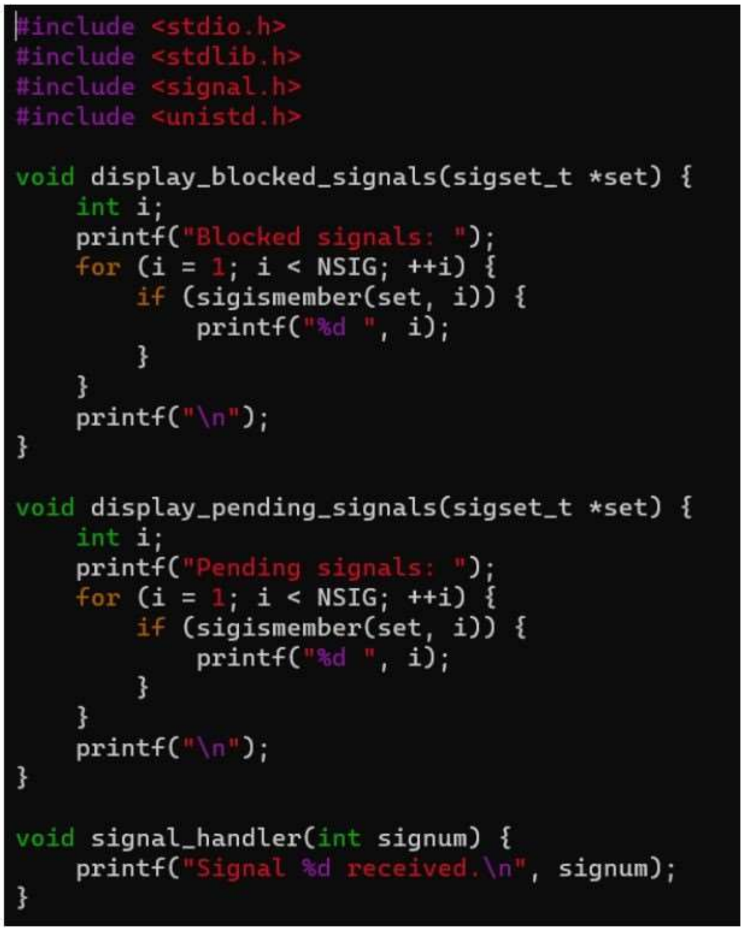


```
root@HereIAm:/home/sem5/sc# ./que2
Parent process(329) is running.
Press Ctrl+C to simulate signals.
Child process(330) is running. Waiting for signals...

Received SIGINT
Processing signal 2: Step 0

Handler is busy.
Ignoring signal 1.
Processing signal 2: Step 1

Handler is busy.
Ignoring signal 15.
Processing signal 2: Step 2
^C
Received SIGINT
Processing signal 2: Step 0
Processing signal 2: Step 1
Processing signal 2: Step 2
Processing signal 2: Step 3
Processing signal 2: Step 4

Received SIGHUP
Processing signal 1: Step 0
Processing signal 1: Step 1
Processing signal 1: Step 2
Processing signal 1: Step 3
Processing signal 1: Step 4

Received SIGTERM
Processing signal 15: Step 0
Processing signal 15: Step 1
Processing signal 15: Step 2
Processing signal 15: Step 3
Processing signal 15: Step 4

Received SIGQUIT. Exiting...
root@HereIAm:/home/sem5/sc# 
```

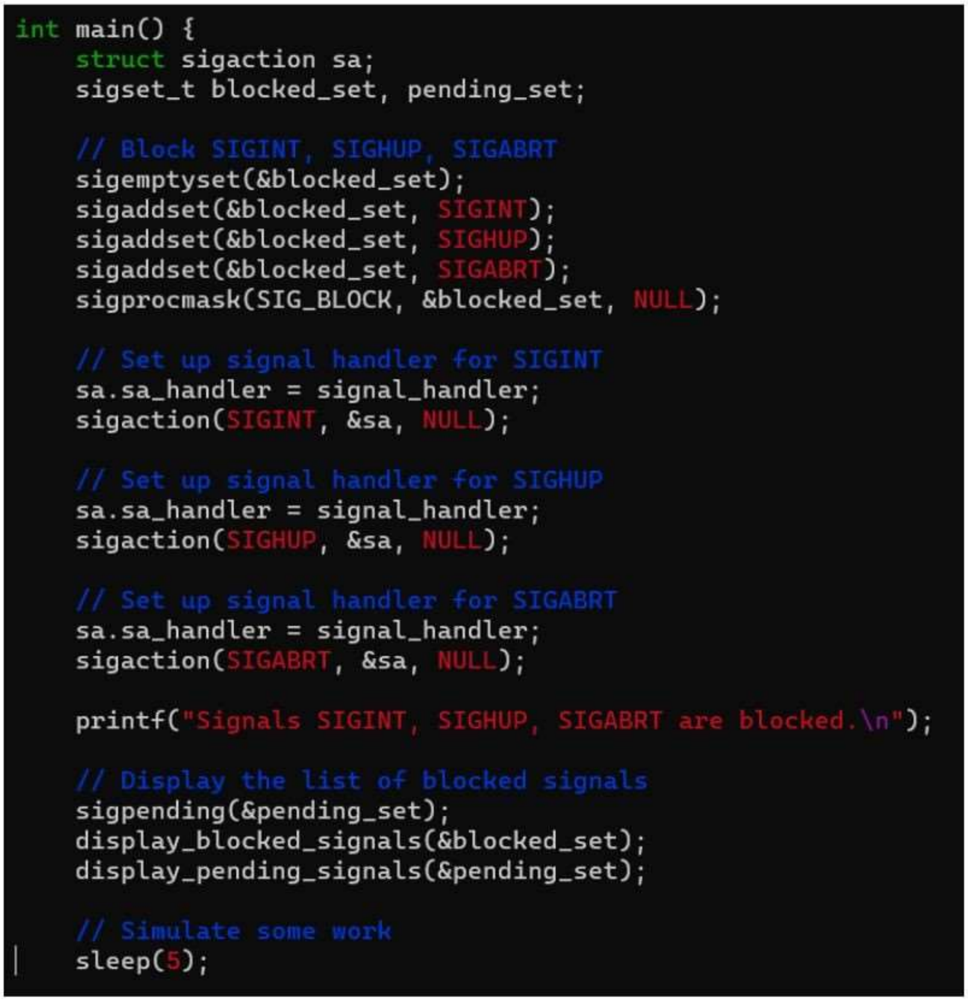


```
root@HereIAm:/home/sem5/sc# ps aux | grep que2
root       318  0.0  0.0  10536   588 tty2     T    14:54   0:00 ./que2
root       319  0.0  0.0      0     0 tty2     Z    14:54   0:00 [que2] <def
unct>
root       329  0.0  0.0  10536   596 tty2     S    14:59   0:00 ./que2
root       330  0.0  0.0      0     0 tty2     Z    14:59   0:00 [que2] <def
unct>
root       333  0.0  0.0  16208  1280 tty1     S    14:59   0:00 grep --colo
r=auto que2
root@HereIAm:/home/sem5/sc# kill -s SIGHUP 330
root@HereIAm:/home/sem5/sc# kill -s SIGHUP 329
root@HereIAm:/home/sem5/sc# kill -s SIGTERM 329
root@HereIAm:/home/sem5/sc# kill -s SIGQUIT 329
root@HereIAm:/home/sem5/sc# |
```

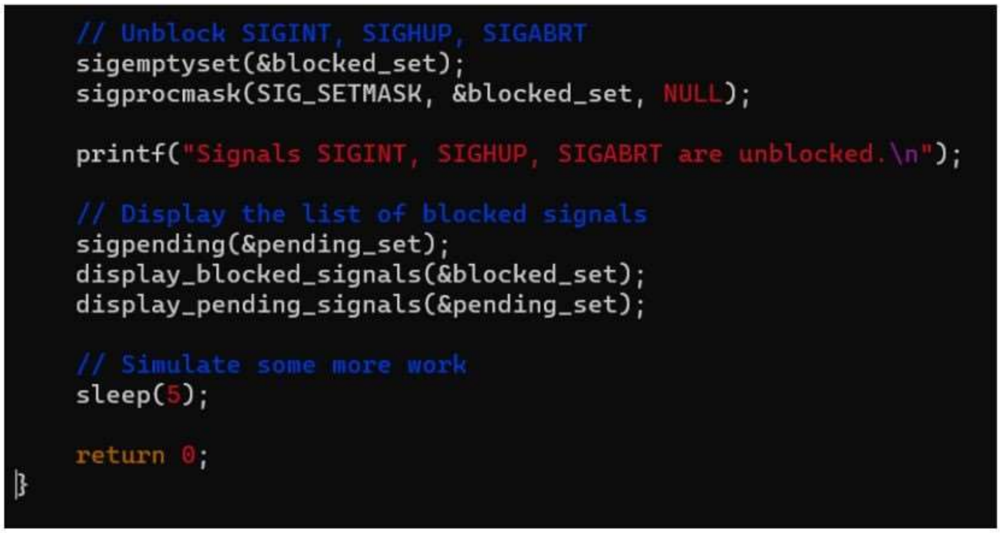
3. Implement a program using sigaction() and signal_set to block any of the signals SIGINT, SIGHUP and SIGABRT. The program should also display the list of blocked/pending signals



```c
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>

void display_blocked_signals(sigset_t *set) {
    int i;
    printf("Blocked signals: ");
    for (i = 1; i < NSIG; ++i) {
        if (sigismember(set, i)) {
            printf("%d ", i);
        }
    }
    printf("\n");
}

void display_pending_signals(sigset_t *set) {
    int i;
    printf("Pending signals: ");
    for (i = 1; i < NSIG; ++i) {
        if (sigismember(set, i)) {
            printf("%d ", i);
        }
    }
    printf("\n");
}

void signal_handler(int signum) {
    printf("Signal %d received.\n", signum);
}
```

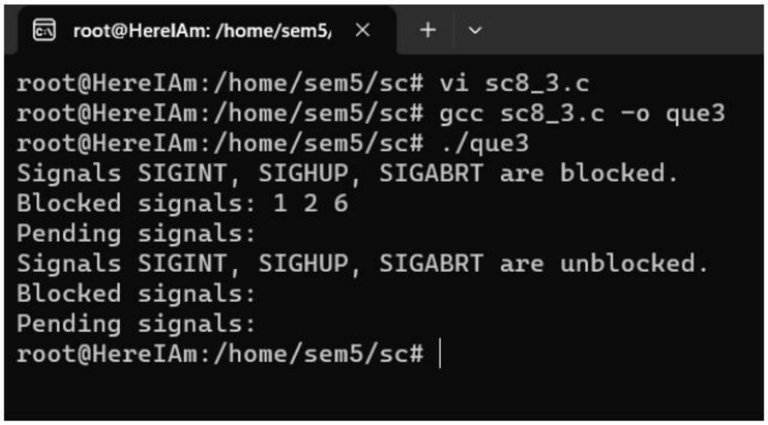


```c
int main() {
    struct sigaction sa;
    sigset_t blocked_set, pending_set;

    // Block SIGINT, SIGHUP, SIGABRT
    sigemptyset(&blocked_set);
    sigaddset(&blocked_set, SIGINT);
    sigaddset(&blocked_set, SIGHUP);
    sigaddset(&blocked_set, SIGABRT);
    sigprocmask(SIG_BLOCK, &blocked_set, NULL);

    // Set up signal handler for SIGINT
    sa.sa_handler = signal_handler;
    sigaction(SIGINT, &sa, NULL);

    // Set up signal handler for SIGHUP
    sa.sa_handler = signal_handler;
    sigaction(SIGHUP, &sa, NULL);

    // Set up signal handler for SIGABRT
    sa.sa_handler = signal_handler;
    sigaction(SIGABRT, &sa, NULL);

    printf("Signals SIGINT, SIGHUP, SIGABRT are blocked.\n");

    // Display the list of blocked signals
    sigpending(&pending_set);
    display_blocked_signals(&blocked_set);
    display_pending_signals(&pending_set);

    // Simulate some work
    sleep(5);
```



```c
    // Unblock SIGINT, SIGHUP, SIGABRT
    sigemptyset(&blocked_set);
    sigprocmask(SIG_SETMASK, &blocked_set, NULL);

    printf("Signals SIGINT, SIGHUP, SIGABRT are unblocked.\n");

    // Display the list of blocked signals
    sigpending(&pending_set);
    display_blocked_signals(&blocked_set);
    display_pending_signals(&pending_set);

    // Simulate some more work
    sleep(5);

    return 0;
}
```

Output:



```
root@HereIAm:/home/sem5/sc# vi sc8_3.c
root@HereIAm:/home/sem5/sc# gcc sc8_3.c -o que3
root@HereIAm:/home/sem5/sc# ./que3
Signals SIGINT, SIGHUP, SIGABRT are blocked.
Blocked signals: 1 2 6
Pending signals:
Signals SIGINT, SIGHUP, SIGABRT are unblocked.
Blocked signals:
Pending signals:
root@HereIAm:/home/sem5/sc# 
```