**2. In the above program,  
● add signal handling for SIGINT (etc) and prevent termination.  
● Convert the above threads to individual functions and note down the time taken and the flow of execution.**

#include <stdio.h>

#include <pthread.h>

#include <unistd.h>

#include <signal.h>

#include <time.h>

volatile sig\_atomic\_t stop = 0;

void sig\_handler(int sig) {

printf("\nCtrl+C pressed. Program will not terminate.\n");

stop = 1;

}

int is\_prime(int num) {

if (num < 2) return 0;

for (int i = 2; i \* i <= num; i++) {

if (num % i == 0) return 0;

}

return 1;

}

void\* calc\_prime\_sum(void\* arg) {

int n = \*((int\*)arg);

int count = 0, num = 2, sum = 0;

printf("Thread A started...\n");

time\_t start = time(NULL);

while (count < n && !stop) {

if (is\_prime(num)) {

sum += num;

count++;

}

num++;

}

time\_t end = time(NULL);

printf("Sum of first %d prime numbers = %d\n", n, sum);

printf("Thread A ended. Time taken: %ld seconds\n", end - start);

return NULL;

}

void\* thread1\_func(void\* arg) {

printf("Thread B started...\n");

time\_t start = time(NULL);

while (!stop && time(NULL) - start < 100) {

printf("Thread 1 running\n");

sleep(2);

}

printf("Thread B ended.\n");

return NULL;

}

void\* thread2\_func(void\* arg) {

printf("Thread C started...\n");

time\_t start = time(NULL);

while (!stop && time(NULL) - start < 100) {

printf("Thread 2 running\n");

sleep(3);

}

printf("Thread C ended.\n");

return NULL;

}

int main() {

pthread\_t t1, t2, t3;

int n;

signal(SIGINT, sig\_handler);

printf("Enter N: ");

scanf("%d", &n);

pthread\_create(&t1, NULL, calc\_prime\_sum, &n);

pthread\_create(&t2, NULL, thread1\_func, NULL);

pthread\_create(&t3, NULL, thread2\_func, NULL);

pthread\_join(t1, NULL);

pthread\_join(t2, NULL);

pthread\_join(t3, NULL);

printf("All threads finished.\n");

return 0;

}

