Advanced C Assignment 2

Sri Gnana Saravan.N

VIT Chennai

1.
```
#include <stdio.h>
#include <unistd.h>
#include <time.h>
#include <stdlib.h>
#include <pthread.h>
#include <stdbool.h>

int N;

int isprime(int num)
{
   if (num <= 1) return 0;
   for (int i = 2; i*i <= num; i++) {
      if (num % i == 0) return 0;
   }
   return 1;
}

void* sum_of_prime(void* arg)
{
   int count=0,num =2,sum=0;
   while(count<N)
   {
```

```
        if(isprime(num))
        {
            sum += num;
            count++;
        }
        num++;
    }
    printf("The sum of first N numbers:%d\n",sum);
     pthread_exit(NULL);
}

void* thread_1_job(void* arg)
{
    time_t start = time(NULL);
    while (time(NULL) - start < 10) {
        printf("Thread 1 running\n");
        sleep(2);
    }
    pthread_exit(NULL);

}

void* thread_2_job(void* arg)
{
    time_t start = time(NULL);
     while (time(NULL) - start < 10) {
        printf("Thread 2 running\n");
        sleep(3);
```

```
    }
    pthread_exit(NULL);

}

int main()
{
    printf("Enter the Input:\n");
    scanf("%d",&N);
    pthread_t thread_1,thread_2,thread_3;
    pthread_create(&thread_1,NULL,sum_of_prime,NULL);
    pthread_create(&thread_2,NULL,thread_1_job,NULL);
    pthread_create(&thread_3,NULL,thread_2_job,NULL);

    pthread_join(thread_1,NULL);
    pthread_join(thread_2,NULL);
    pthread_join(thread_3,NULL);

    return 0;
}
```

Output:

```
Enter the Input:
5
The sum of first N numbers:28
Thread 1 running
Thread 2 running
Thread 1 running
Thread 2 running
Thread 1 running
```

2.
```
#include <stdio.h>
#include <unistd.h>
#include <time.h>
#include <stdlib.h>
#include <pthread.h>
#include <stdbool.h>
#include <signal.h>

void seq_sum_of_prime();
void seq_thread_1_job();
void seq_thread_2_job();

int N;



void handle_signal_interrupt()
{
   printf("Detected Signal termination and avoiding it\n");
}

int isprime(int num)
{
   if (num <= 1) return 0;
   for (int i = 2; i*i <= num; i++) {
      if (num % i == 0) return 0;
   }
   return 1;
}
```

```
void* sum_of_prime(void* arg)
{
   int count=0,num =2,sum=0;
   while(count<N)
   {
      if(isprime(num))
      {
         sum += num;
         count++;
      }
      num++;
   }
   printf("The sum of first N numbers:%d\n",sum);
    pthread_exit(NULL);
}

void* thread_1_job(void* arg)
{
   time_t start = time(NULL);
   while (time(NULL) - start < 10) {
      printf("Thread 1 running\n");
      sleep(2);
   }
   pthread_exit(NULL);

}
```

```
void* thread_2_job(void* arg)
{
   time_t start = time(NULL);
    while (time(NULL) - start < 10) {
      printf("Thread 2 running\n");
      sleep(3);
   }
   pthread_exit(NULL);

}

//Sequential Function callls


void seq_sum_of_prime()
{
   int count=0,num =2,sum=0;
   while(count<N)
   {
      if(isprime(num))
      {
         sum += num;
         count++;
      }
      num++;
   }
   printf("Sequential call:The sum of first N numbers:%d\n",sum);
}
```

```
void seq_thread_1_job()
{
   time_t start = time(NULL);
   while (time(NULL) - start < 10) {
      printf("Sequential:Thread 1 running\n");
      sleep(2);
   }

}

void seq_thread_2_job()
{
   time_t start = time(NULL);
    while (time(NULL) - start < 10) {
      printf("Sequential:Thread 2 running\n");
      sleep(3);
   }
}



int main()
{
   signal(SIGINT,handle_signal_interrupt);
   printf("Enter the Input:\n");
   scanf("%d",&N);
   printf("Running in the thread mode\n");
   struct timespec t1,t2;
```

```
clock_gettime(CLOCK_MONOTONIC,&t1);
pthread_t thread_1,thread_2,thread_3;
pthread_create(&thread_1,NULL,sum_of_prime,NULL);
pthread_create(&thread_2,NULL,thread_1_job,NULL);
pthread_create(&thread_3,NULL,thread_2_job,NULL);

pthread_join(thread_1,NULL);
pthread_join(thread_2,NULL);
pthread_join(thread_3,NULL);
clock_gettime(CLOCK_MONOTONIC,&t2);
//Computing the time it takes to run the threads
double time_taken_by_threads = (t2.tv_sec-t1.tv_sec) + (t2.tv_nsec-t1.tv_nsec)/(1e9);
printf("Time taken by threads:%.2f\n",time_taken_by_threads);

//Sequential calling
clock_gettime(CLOCK_MONOTONIC,&t1);
seq_sum_of_prime();
seq_thread_1_job();
seq_thread_2_job();
clock_gettime(CLOCK_MONOTONIC,&t2);
double time_taken_by_seq = (t2.tv_sec-t1.tv_sec) + (t2.tv_nsec-t1.tv_nsec)/(1e9);
printf("Time taken by Sequential function calls:%.2f\n",time_taken_by_seq);

return 0;
```

}

```
Enter the Input:
5
Running in the thread mode
The sum of first N numbers:28
Thread 1 running
Thread 2 running
^CDetected Signal termination and avoiding it
Thread 1 running
Thread 2 running
Thread 1 running
Thread 2 running
Thread 1 running
Thread 1 running
Thread 2 running
Time taken by threads:12.00
Sequential call:The sum of first N numbers:28
Sequential:Thread 1 running
Sequential:Thread 1 running
^CDetected Signal termination and avoiding it
Sequential:Thread 1 running
Sequential:Thread 1 running
Sequential:Thread 1 running
Sequential:Thread 1 running
Sequential:Thread 2 running
Sequential:Thread 2 running
Sequential:Thread 2 running
Sequential:Thread 2 running
Time taken by Sequential function calls:22.92
```

3.

Fork():

fork() is a system call in UNIX/Linux used to create a new process by duplicating the calling process. The new process created is called the *child process*, and the original is the *parent process*. After a successful fork(), both processes will execute the next instructions independently.

Signal Handing:

Signals are software interrupts delivered to a process to notify it of events like the keyboard interrupt.

SIGINT (Ctrl+C) ----> It is the signal used to terminate the process.

Kernel Crashes:

Kernel crashes are critical system-level failures that halt the OS due to severe issues like:

- Dereferencing null or invalid pointers
- Deadlocks
- Stack overflows
- Hardware incompatibilities

We can use dmesg command tp know about the kernel crashes.

Time Complexity:

Time complexity describes the amount of time the program takes to run.It is denoted by Big O notation.

Mutex (Mutual Exclusion): A blocking mechanism; if the lock is not available, the thread sleeps until it can acquire it. Useful in user space or kernel space when contention is low.

Spinlock: A non-blocking mechanism where the thread keeps checking (spinning) until the lock becomes available. Useful in kernel space when holding lock for a very short time.