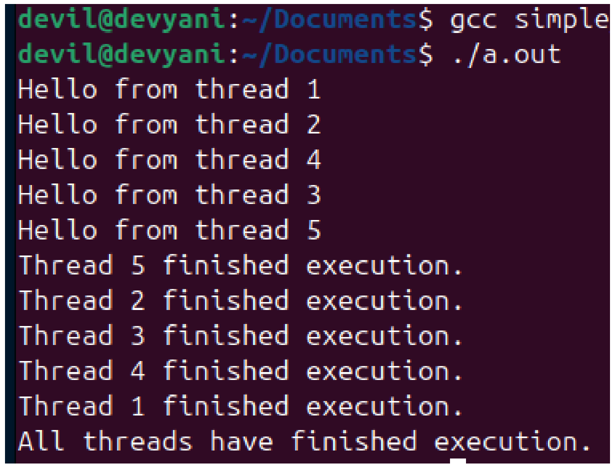
1. SIMPLE PTHREAD

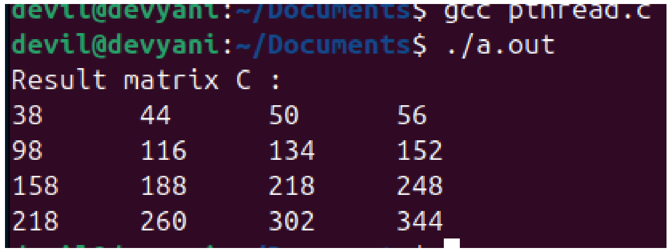
```
#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

#include<unistd.h>

void *threadFunction(void* arg) {

      int threadID = *((int*)arg);

      printf("Hello from thread %d\n",threadID);

      sleep(1); //simulate some work

      printf("Thread %d finished
execution.\n",threadID);

      return NULL;

}

int main() {

      pthread_t threads[5]; //Array to hold
thread identifers

      int threadIDs[5]; //Array to hold thread IDs

      //Create two threads

      for(int i = 0; i < 5 ; i++) {

        threadIDs[i] = i + 1;

        //Assigning IDs to threads

if(pthread_create(&threads[i],NULL,threadFun
ction,&threadIDs[i]) != 0)  {

          perror("Failed to create thread");

          return 1;

        } }

      //wait for both threads to finish

      for(int i = 0 ; i < 5 ; i++) {

          pthread_join(threads[i],NULL);

      }

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

      return 0;

}
```



```
devil@devyani:~/Documents$ gcc simple
devil@devyani:~/Documents$ ./a.out
Hello from thread 1
Hello from thread 2
Hello from thread 4
Hello from thread 3
Hello from thread 5
Thread 5 finished execution.
Thread 2 finished execution.
Thread 3 finished execution.
Thread 4 finished execution.
Thread 1 finished execution.
All threads have finished execution.
```

2. Simple Pthread Without Pointer

```
#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

#define N 4 // Size of the matrices (N * N)

//Structure to pass multiple arguments to
thread function

typedef struct {

      int row; // Row index for the first matrix

      int(*A)[N]; // First matrix

      int(*B)[N]; // Second matrix

      int(*C)[N]; // Result matrix

} ThreadData;

//Function for each thread to compute a
specific row of the result matrix

void* multiply_row(void* arg){

      ThreadData* data = (ThreadData*)arg;

      int row = data->row;

      for(int j = 0; j < N; j++)

      {

          data->C[row][j] = 0;
      for(int k = 0;k < N;k++)

      {

          data->C[row][j] += data->A[row][k]*data-
>B[k][j];

      }

      }

        return NULL;

}

int main()

{

      int A[N][N] = {

                    {1,2,3,4},

                    {5,6,7,8},

                    {9,10,11,12},

                    {13,14,15,16}

      };

      int B[N][N] = {

                    {1,2,3,4},

                    {5,6,7,8},

                    {9,10,11,12},

      };

      int C[N][N]; //Resultant matrix

      pthread_t threads[N];

      ThreadData threadData[N];

      //Create threads for each row of the result
matrix

      for(int i = 0;i < N;i++)

      {

          threadData[i].row = i;

          threadData[i].A = A;

          threadData[i].B = B;

          threadData[i].C = C;
if(pthread_create(&threads[i],NULL,multiply_r
ow,(void*)&threadData[i]) != 0)

          {

              perror("Failed to create thread");

              return 1;

          }

      }

      //Join threads

      for(int i = 0;i < N;i++)

      {

        pthread_join(threads[i],NULL);

      }

      //Print the result

      printf("Result matrix C : \n");

      for(int i = 0;i < N;i++)

      {

        for(int j = 0;j < N;j++)

        {

            printf("%d\t",C[i][j]);

        }

        printf("\n");

      }

      return 0;

}
```



```
devil@devyani:~/Documents$ gcc pthread.c
devil@devyani:~/Documents$ ./a.out
Result matrix C :
38      44      50      56
98      116     134     152
158     188     218     248
218     260     302     344
```