

### //Assignment 3: Matrix Multiplication using POSIX Threads

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>

typedef struct _thread_data_t { //structure for matrix
    int *A;
    int m,n;
} thread_data_t;

void *multiplication(void *arg) { //multiplication of one row with column of other matrix

    int *data=(int*)arg;
    printf("%d ",(int)pthread_self());

    int *mul=(int*)malloc(sizeof(int));
    for(int i=1;i<=data[0];i++) {
        *mul=*mul+(data[i]*data[i+data[0]]);
    }
    pthread_exit(mul);
}

void *accept(void *arg) { //accepts elements of matrix

    printf("====In Accept====\n\n");

    thread_data_t *data=(thread_data_t *)arg;
    printf("PID      : %d\n",getpid());
    printf("Thread ID: %d\n\n",(int)(int)pthread_self());

    printf("Size of Matrix\n");
    printf("Number of rows  : ");
    scanf("%d",&data->m);
    printf("Number of columns: ");
    scanf("%d",&data->n);

    data->A=(int*)malloc(sizeof(int)*data->m*data->n);

    printf("\nEnter elements of Matrix:\n\n");
    for(int i=0;i<data->m;i++) {
        for(int j=0;j<data->n;j++) {
            scanf("%d",&data->A+i*data->n+j);
        }
    }
    printf("\n");
    pthread_exit(NULL);
}

void *defaults(void *arg) { //setting default value to a matrix

    printf("====In Default====\n\n");

    thread_data_t *data=(thread_data_t *)arg;
    printf("PID      : %d\n",getpid());
    printf("Thread ID: %d\n\n",(int)pthread_self());

    data->A=(int*)malloc(sizeof(int)*data->m*data->n);
    for(int i=0;i<data->m;i++) {
```

```

    for(int j=0;j<data->n;j++) {
        *(data->A+i*data->n+j)=0;
    }
}
printf("Successfully Allotted Default Value!\n\n");
pthread_exit(NULL);
}

```

```

void *display(void *arg) { //displaying the result

```

```

    printf("====In Display====\n\n");

```

```

    thread_data_t *data=(thread_data_t *)arg;
    printf("PID      : %d\n",getpid());
    printf("Thread ID: %d\n\n",(int)pthread_self());

```

```

    printf("Matrix A:\n\n");
    for(int i=0;i<data[0].m;i++) {
        for(int j=0;j<data[0].n;j++) {
            printf("%d ",*(data[0].A+(i*data[0].n+j)));
        }
        printf("\n");
    }

```

```

    printf("\nMatrix B:\n\n");
    for(int i=0;i<data[1].m;i++) {
        for(int j=0;j<data[1].n;j++) {
            printf("%d ",*(data[1].A+(i*data[1].n+j)));
        }
        printf("\n");
    }

```

```

    printf("\nMatrix A x B:\n\n");
    for(int i=0;i<data[2].m;i++) {
        for(int j=0;j<data[2].n;j++) {
            printf("%d ",*(data[2].A+i*data[2].n+j));
        }
        printf("\n");
    }
    pthread_exit(NULL);
}

```

```

int main(int argc, char **argv) {

```

```

    printf("\n====In Main====\n\n");
    pthread_t acceptThread[3],displayThread,*mulThread;
    thread_data_t data[3];
    int i,count=0;

```

```

    //printing pid
    printf("PID: %d\n\n",getpid());

```

```

    for(i=0;i<2;i++) {

```

```

        //creating 2 threads for accepting
        pthread_create(&acceptThread[i],NULL,accept,&data[i]);
        pthread_join(acceptThread[i],NULL);
    }

```

```

    //creating one thread for accepting default values

```

```

data[i].m=data[0].m;
data[i].n=data[1].n;
pthread_create(&acceptThread[i],NULL,defaults,&data[i]);
pthread_join(acceptThread[i],NULL);

//creating threads of size m1 x n2 for each i,j element of resultant matrix
mulThread=(pthread_t*)malloc(sizeof(pthread_t)*(data[0].m+data[1].n));
printf("====In Multiplication====\n\n");
printf("PID      : %d\n",getpid());
printf("Thread IDs: ");

if(data[0].n==data[1].m) {

    for(i=0;i<data[0].m;i++) {
        for(int j=0;j<data[1].n;j++) {

            //for storing row and column elements
            int l=1,*Data=(int*)malloc(sizeof(int)*((2*data[0].n)+1));
            void *value;

            Data[0]=data[0].n;
            for(int k=0;k<data[0].n;k++) {
                Data[l++]=*(data[0].A+i*data[0].n+k);
            }

            for(int k=0;k<data[1].m;k++) {
                Data[l++]=*(data[1].A+k*data[1].n+j);
            }

            //thread for multiplication
            pthread_create(&mulThread[count],NULL,multiplication,(void*)Data);
            pthread_join(mulThread[count],&value); //storing return value from multiplication in resultant
matrix
            *(data[2].A+i*data[2].n+j)=*(int*)value;
            count++;
        }
    }
}

//thread for display
printf("\n\n");
pthread_create(&displayThread,NULL,display,&data);
pthread_join(displayThread,NULL);

printf("\n====Back In Main====\n\n");
printf("PID: %d\n\n",getpid());

return 0;
}

```

## Output:

```

Someshwars-MacBook-Pro:Assignments someshwargaikwad$ gcc Assignment3.c -o
Assignment3
Someshwars-MacBook-Pro:Assignments someshwargaikwad$ ./Assignment3

```

```

====In Main====

```

```

PID: 4569

```

=====In Accept=====

PID : 4569  
Thread ID: 144322560

Size of Matrix  
Number of rows : 3  
Number of columns: 3

Enter elements of Matrix:

1 2 3  
4 5 6  
7 8 9

=====In Accept=====

PID : 4569  
Thread ID: 144322560

Size of Matrix  
Number of rows : 3  
Number of columns: 3

Enter elements of Matrix:

1 1 1  
1 1 1  
1 1 1

=====In Default=====

PID : 4569  
Thread ID: 144322560

Successfully Allotted Default Value!

=====In Multiplication=====

PID : 4569  
Thread IDs: 144322560 144322560 144322560 144322560 144322560 144322560 144322560  
144322560 144322560

=====In Display=====

PID : 4569  
Thread ID: 144322560

Matrix A:

1 2 3  
4 5 6  
7 8 9

Matrix B:

1 1 1  
1 1 1  
1 1 1

Matrix A x B:

6 6 6  
15 15 15  
24 24 24

=====Back In Main=====

PID: 4569

Someshwars-MacBook-Pro:Assignments someshwargaikwad\$