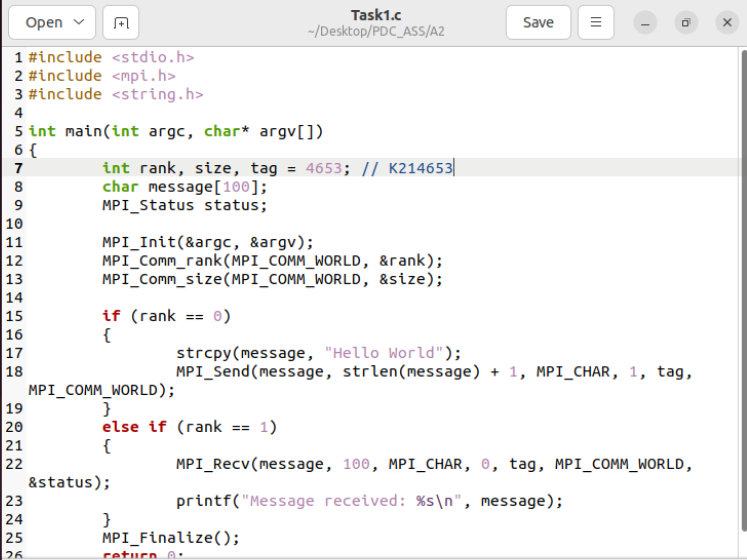
PDC A2

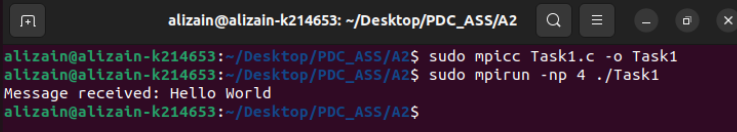
Ali Zain 21K4653

Task #1:

Code:



Output:



Task #2:

Code:

#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

#include <time.h>

#define N 100

int main(int argc, char\* argv[]) {

int rank, size;

time\_t t;

int num[N] = { 0 };

int\* arr = NULL, \* arr2 = NULL;

int sum = 0, total\_sum = 0;

MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

srand((unsigned)time(&t));

if (rank == 0) {

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

num[i] = rand() % 100;

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

}

}

arr = (int\*)malloc(N / size \* sizeof(int));

MPI\_Scatter(num, N / size, MPI\_INT, arr, N / size, MPI\_INT, 0, MPI\_COMM\_WORLD);

for (int i = 0; i < N / size; i++) {

sum += arr[i];

}

if (rank == 0) {

arr2 = (int\*)malloc(size \* sizeof(int));

}

MPI\_Gather(&sum, 1, MPI\_INT, arr2, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

if (rank == 0) {

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

total\_sum += arr2[i];

}

printf("\nTotal sum of 100 elements = %d\n", total\_sum);

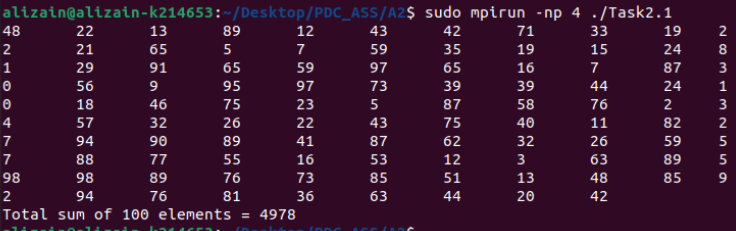
}

MPI\_Finalize();

return 0;

}

Output:



Task #3:

Code:

#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

int main(int argc, char\* argv[])

{

int rank, size, i, n, \* a, \* b, \* c, \* d;

MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

if (rank == 0)

{

printf("Enter the number of elements\n");

scanf("%d", &n);

}

MPI\_Bcast(&n, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

a = (int\*)malloc(n \* sizeof(int));

b = (int\*)malloc(n \* sizeof(int));

c = (int\*)malloc(n \* sizeof(int));

d = (int\*)malloc(n \* sizeof(int));

if (rank == 0)

{

printf("Enter the elements\n");

for (i = 0; i < n; i++)

scanf("%d", &a[i]);

}

MPI\_Scatter(a, 1, MPI\_INT, b, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

for (i = 0; i < n; i++)

c[i] = b[i] \* b[i];

MPI\_Gather(c, 1, MPI\_INT, d, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

if (rank == 0)

{

printf("The squared elements are\n");

for (i = 0; i < n; i++)

printf("%d ", d[i]);

printf("\n");

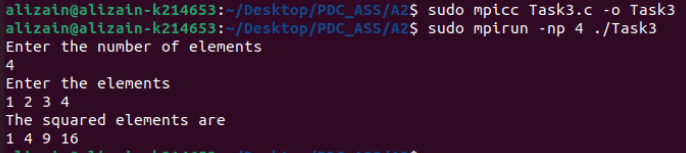
}

MPI\_Finalize();

return 0;

}

Output:



Task #4:

Code:

#include <stdio.h>

#include <mpi.h>

#include <stdlib.h>

#include <time.h>

int main(int argc, char\* argv[])

{

int rank, size, tag = 4653;

double start, end;

MPI\_Status status;

MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

start = MPI\_Wtime();

if (rank == 0)

{

int a = 10;

MPI\_Send(&a, 1, MPI\_INT, 1, tag, MPI\_COMM\_WORLD);

}

else if (rank == 1)

{

int b;

MPI\_Recv(&b, 1, MPI\_INT, 0, tag, MPI\_COMM\_WORLD, &status);

printf("Received %d from process 0\n", b);

}

end = MPI\_Wtime();

printf("Time taken by process %d is %f\n", rank, end - start);

MPI\_Finalize();

return 0;

}

Output:

A computer code with blue text

Description automatically generated

Task #5:

Code:

#include <stdio.h>

#include <mpi.h>

#include <stdlib.h>

#include <time.h>

int main(int argc, char\* argv[])

{

int rank, size, i, n = 10;

int\* arr = (int\*)malloc(n \* sizeof(int));

double start, end;

MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size); //size = 2

if (rank == 0)

{

printf("Enter %d elements: ", n);

for (i = 0; i < n; i++)

scanf("%d", &arr[i]);

start = MPI\_Wtime();

MPI\_Send(arr, n, MPI\_INT, 1, 0, MPI\_COMM\_WORLD);

end = MPI\_Wtime();

printf("Time taken by processor %d is %f\n", rank, end - start);

}

else if (rank == 1)

{

MPI\_Recv(arr, n, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

printf("Received array is: ");

for (i = 0; i < n; i++)

printf("%d ", arr[i]);

printf("\n");

}

MPI\_Finalize();

return 0;

}

Output:

A computer screen with text

Description automatically generated

Task #6:

Code:

#include<stdio.h>

#include<mpi.h>

int main(int argc, char\* argv[])

{

int rank, size, fact = 1, i, j, n = 10, fact1 = 1;

MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

if (rank == 0)

{

for (i = 1; i <= n; i++)

{

fact = fact \* i;

printf("Factorial of %d is %d\n", i, fact);

}

}

MPI\_Finalize();

return 0;

}

Output: because of 2 processor 2 outputs each from different processor

A screenshot of a computer

Description automatically generated