Walchand College of Engineering, Sangli
Department of Computer Science and Engineering

Class: Final Year (Computer Science and Engineering)

Year: 2025-26 **Semester:** 1

Course: High Performance Computing Lab

Practical No. 6

Exam Seat No: 22510064 – Parshwa Herwade

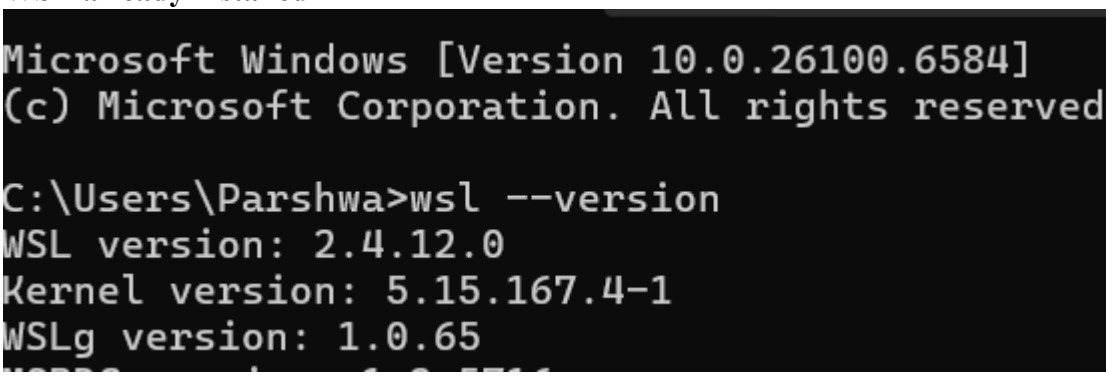
Github Link: [Sem-7-Assign/HPC lab at main · parshwa913/Sem-7-Assign · GitHub]

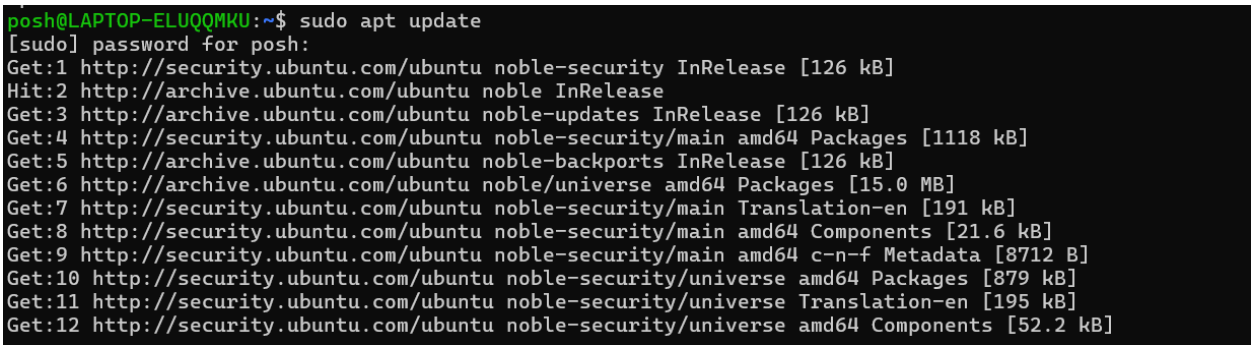
Title of practical:

Installation of MPI & Implementation of basic functions of MPI

ANSWER

WSL already installed

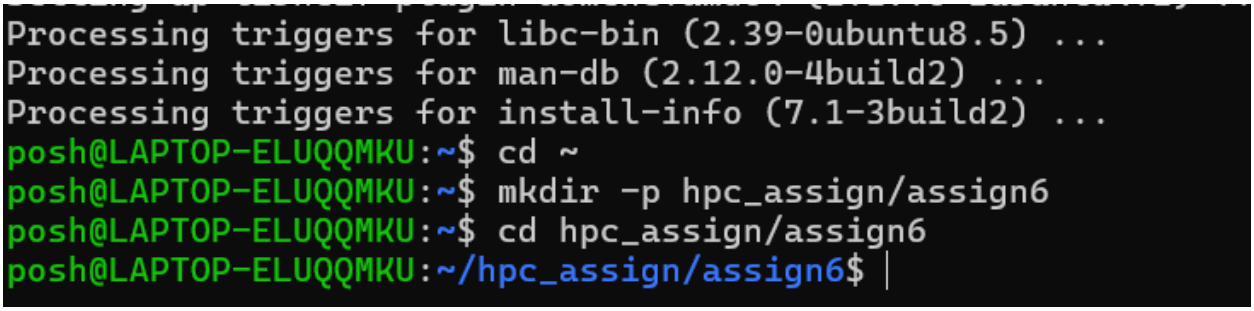


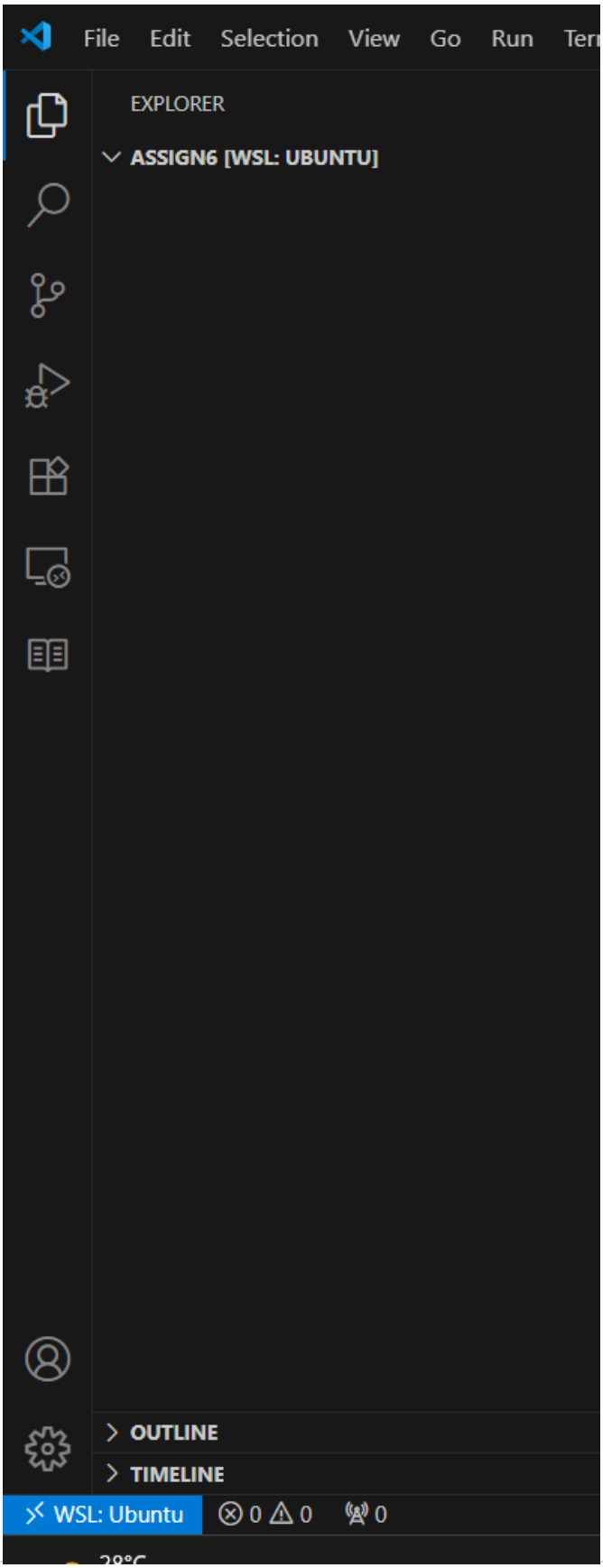


```
Get:50 http://archive.ubuntu.com/ubuntu noble-backports/universe Translation-en [17.4 kB]
Get:51 http://archive.ubuntu.com/ubuntu noble-backports/universe amd64 Components [19.2 kB]
Get:52 http://archive.ubuntu.com/ubuntu noble-backports/universe amd64 c-n-f Metadata [1304 B]
Get:53 http://archive.ubuntu.com/ubuntu noble-backports/restricted amd64 Components [216 B]
Get:54 http://archive.ubuntu.com/ubuntu noble-backports/restricted amd64 c-n-f Metadata [116 B]
Get:55 http://archive.ubuntu.com/ubuntu noble-backports/multiverse amd64 Components [212 B]
Get:56 http://archive.ubuntu.com/ubuntu noble-backports/multiverse amd64 c-n-f Metadata [116 B]
Fetched 37.1 MB in 1min 43s (360 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
160 packages can be upgraded. Run 'apt list --upgradable' to see them.
posh@LAPTOP-ELUQQMKU:~$
```

```
160 packages can be upgraded. Run 'apt list --upgradable' to see them.
posh@LAPTOP-ELUQQMKU:~$ sudo apt install -y build-essential openmpi-bin libopenmpi-dev
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  autoconf automake autotools-dev bzip2 cpp cpp-13 cpp-13-x86-64-linux-gnu cpp-x86-64-linux-gnu dpkg-dev fakeroot g++
  g++-13 g++-13-x86-64-linux-gnu g++-x86-64-linux-gnu gcc gcc-13 gcc-13-base gcc-13-x86-64-linux-gnu
  gcc-x86-64-linux-gnu gfortran gfortran-13 gfortran-13-x86-64-linux-gnu gfortran-x86-64-linux-gnu ibverbs-providers
  javascript-common libalgorithm-diff-perl libalgorithm-diff-xs-perl libalgorithm-merge-perl libamd-comgr2
  libamdhip64-5 libaom3 libasan8 libatomic1 libc-bin libc-dev-bin libc-devtools libc6 libc6-dev libcaf-openmpi-3t64
  libcc1-0 libcoarrays-dev libcoarrays-openmpi-dev libcrypt-dev libde265-0 libdpkg-perl libevent-2.1-7t64 libevent-dev
  libevent-extra-2.1-7t64 libevent-openssl-2.1-7t64 libevent-pthreads-2.1-7t64 libfabric1 libfakeroot
  libfile-fcntllock-perl libgcc-13-dev libgd3 libgfortran-13-dev libgfortran5 libgomp1 libheif-plugin-aomdec
  libheif-plugin-aomenc libheif-plugin-libde265 libheif1 libhsa-runtime64-1 libhsakmt1 libhwasan0 libhwloc-dev
  libhwloc-plugins libhwloc15 libibverbs-dev libibverbs1 libisl23 libitm1 libjs-jquery libjs-jquery-ui libllvm17t64
  liblsan0 libltdl-dev libltdl7 libmpc3 libmunge2 libnl-3-200 libnl-3-dev libnl-route-3-200 libnl-route-3-dev
  libnuma-dev libnuma1 libopenmpi3t64 libpmix-dev libpmix2t64 libpsm-infinipath1 libpsm2-2 libquadmath0 librdmacm1t64
  libstdc++-13-dev libtool libtsan2 libubsan1 libucx0 libxnvctrl0 libxpm4 linux-libc-dev locales lto-disabled-list m4
```

```
tting up libcoarrays-openmpi-dev:amd64 (2.10.2+ds-2.1build2) ...
date-alternatives: using /usr/lib/x86_64-linux-gnu/open-coarrays/openmpi/bin/caf to provide /usr/bin/caf.openmpi (
enmpi) in auto mode
date-alternatives: using /usr/bin/caf.openmpi to provide /usr/bin/caf (caf) in auto mode
tting up libheif-plugin-aomdec:amd64 (1.17.6-1ubuntu4.1) ...
tting up libheif1:amd64 (1.17.6-1ubuntu4.1) ...
tting up libgd3:amd64 (2.3.3-9ubuntu5) ...
tting up libc-devtools (2.39-0ubuntu8.5) ...
tting up libheif-plugin-libde265:amd64 (1.17.6-1ubuntu4.1) ...
tting up libheif-plugin-aomenc:amd64 (1.17.6-1ubuntu4.1) ...
ocessing triggers for libc-bin (2.39-0ubuntu8.5) ...
ocessing triggers for man-db (2.12.0-4build2) ...
ocessing triggers for install-info (7.1-3build2) ...
sh@LAPTOP-ELUQQMKU:~$
```


Processing triggers for libc-bin (2.39-0ubuntu8.5) ...
Processing triggers for man-db (2.12.0-4build2) ...
Processing triggers for install-info (7.1-3build2) ...
posh@LAPTOP-ELUQQMKU:~$ cd ~
posh@LAPTOP-ELUQQMKU:~$ mkdir -p hpc_assign/assign6
posh@LAPTOP-ELUQQMKU:~$ cd hpc_assign/assign6
posh@LAPTOP-ELUQQMKU:~/hpc_assign/assign6$


File
Edit
Selection
View
Go
Run
EXPLORER
ASSIGN6 [WSL: UBUNTU]
OUTLINE
TIMELINE
WSL: Ubuntu
0
0
0

Problem Statement 1:

Implement a simple hello world program by setting number of processes equal to 10

```c
#include <mpi.h>
#include <stdio.h>

int main(int argc, char *argv[]) {
    MPI_Init(&argc, &argv);
    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    printf("Hello from rank %d out of %d processes\n", rank, size);

    MPI_Finalize();
    return 0;
}
```

Screenshots:

```
posh@LAPTOP-ELUQQMKU:~/hpc_assign/assign6$ mpicc -o hello hello.c
posh@LAPTOP-ELUQQMKU:~/hpc_assign/assign6$ mpirun -np 10 ./hello
--------------------------------------------------------------------------
There are not enough slots available in the system to satisfy the 10
slots that were requested by the application:

  ./hello

Either request fewer slots for your application, or make more slots
available for use.

A "slot" is the Open MPI term for an allocatable unit where we can
launch a process.  The number of slots available are defined by the
environment in which Open MPI processes are run:

  1. Hostfile, via "slots=N" clauses (N defaults to number of
     processor cores if not provided)
  2. The --host command line parameter, via a ":N" suffix on the
     hostname (N defaults to 1 if not provided)
  3. Resource manager (e.g., SLURM, PBS/Torque, LSF, etc.)
  4. If none of a hostfile, the --host command line parameter, or an
     RM is present, Open MPI defaults to the number of processor cores

In all the above cases, if you want Open MPI to default to the number
of hardware threads instead of the number of processor cores, use the
--use-hwthread-cpus option.

Alternatively, you can use the --oversubscribe option to ignore the
number of available slots when deciding the number of processes to
launch.
--------------------------------------------------------------------------
posh@LAPTOP-ELUQQMKU:~/hpc_assign/assign6$ mpirun --oversubscribe -np 10 ./hello
Hello from rank 2 out of 10 processes
Hello from rank 3 out of 10 processes
Hello from rank 5 out of 10 processes
Hello from rank 8 out of 10 processes
Hello from rank 7 out of 10 processes
Hello from rank 9 out of 10 processes
Hello from rank 0 out of 10 processes
Hello from rank 4 out of 10 processes
Hello from rank 1 out of 10 processes
Hello from rank 6 out of 10 processes
posh@LAPTOP-ELUQQMKU:~/hpc_assign/assign6$
```

Information 1:

Theory/Info:

Every MPI program begins with MPI_Init and ends with MPI_Finalize.

MPI_Comm_rank gives the rank of the process (unique ID).

MPI_Comm_size gives the total number of processes.

Purpose: To understand process identification in MPI and parallel execution.

Demonstration: Each process prints its rank and the total number of processes.

Key Concepts: Process initialization, basic communication, parallel output.

Conclusion:

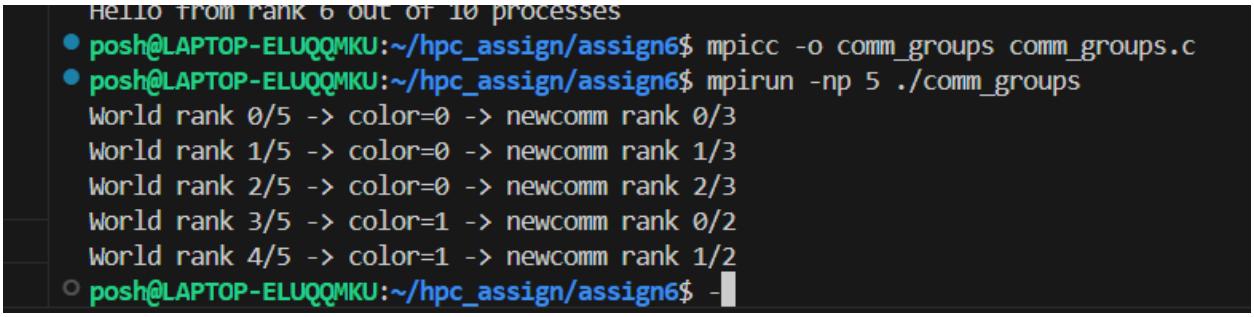
Verified that multiple processes are launched and each process has a unique rank.

Problem Statement 2:

Implement a program to display rank and communicator group of five processes

```c
#include <mpi.h>
#include <stdio.h>

int main(int argc, char *argv[]) {
    MPI_Init(&argc, &argv);

    int world_rank, world_size;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    int color = (world_rank < (world_size+1)/2) ? 0 : 1;
    MPI_Comm newcomm;
    MPI_Comm_split(MPI_COMM_WORLD, color, world_rank, &newcomm);

    int new_rank, new_size;
    MPI_Comm_rank(newcomm, &new_rank);
    MPI_Comm_size(newcomm, &new_size);

    printf("World rank %d/%d -> color=%d -> newcomm rank %d/%d\n",
           world_rank, world_size, color, new_rank, new_size);

    MPI_Comm_free(&newcomm);
    MPI_Finalize();
    return 0;
}
```

Screenshots:



Topic: Communicator Splitting and Subgroups
Theory/Info:

- MPI allows splitting the global communicator (MPI_COMM_WORLD) into sub-communicators using MPI_Comm_split.
- Processes can be divided into groups based on rank, color, or other criteria.
- Purpose: Learn group communication and logical partitioning of processes.
  Demonstration: Processes are divided into two groups; each process prints its original rank and its new rank in the subgroup.
  Key Concepts: Process grouping, sub-communicators, parallel task partitioning.
  Conclusion:
- Demonstrates how MPI sub-communicators can divide work among process groups.
- Helps design programs with group-specific communication patterns.

Q3: Implement a MPI program to give an example of Deadlock.

Program and screenshots

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

#define BIGSIZE 10000000

int main(int argc, char* argv[]) {
    int rank, size;
    int *buffer = NULL;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    if (size != 2) {
        if (rank == 0) {
            printf("Run this program with exactly 2 processes!\n");
        }
        MPI_Finalize();
        return 0;
    }

    buffer = (int*)malloc(BIGSIZE * sizeof(int));

    if (rank == 0) {
        MPI_Send(buffer, BIGSIZE, MPI_INT, 1, 0, MPI_COMM_WORLD);
        MPI_Recv(buffer, BIGSIZE, MPI_INT, 1, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
        printf("Process 0 finished exchange\n");
    } else if (rank == 1) {
        MPI_Send(buffer, BIGSIZE, MPI_INT, 0, 0, MPI_COMM_WORLD);
        MPI_Recv(buffer, BIGSIZE, MPI_INT, 0, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
```

```
        printf("Process 1 finished exchange\n");
    }

    free(buffer);
    MPI_Finalize();
    return 0;
}
```

```
Process 0 received 100
posh@LAPTOP-ELUQQMKU:~/hpc_assign/assign6$ mpicc deadlock.c -o deadlock
posh@LAPTOP-ELUQQMKU:~/hpc_assign/assign6$ mpirun -np 2 ./deadlock
^Cposh@LAPTOP-ELUQQMKU:~/hpc_assign/assign6$ mpicc deadlock.c -o deadlock
```

Process didn't complete
Forced to stop the code

DEADLOCK

FIXED CODE:

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

#define BIGSIZE 10000000   // 10 million ints ~ 40 MB

int main(int argc, char* argv[]) {
    int rank, size;
    int *sendbuf = NULL;
    int *recvbuf = NULL;
```

```
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    if (size != 2) {
        if (rank == 0) {
            printf("Run this program with exactly 2 processes!\n");
        }
        MPI_Finalize();
        return 0;
    }

    sendbuf = (int*)malloc(BIGSIZE * sizeof(int));
    recvbuf = (int*)malloc(BIGSIZE * sizeof(int));

    for (int i = 0; i < BIGSIZE; i++) {
        sendbuf[i] = rank;   // fill with process rank
    }

    // Safe exchange using MPI_Sendrecv
    MPI_Sendrecv(sendbuf, BIGSIZE, MPI_INT, 1 - rank, 0,
                 recvbuf, BIGSIZE, MPI_INT, 1 - rank, 0,
                 MPI_COMM_WORLD, MPI_STATUS_IGNORE);

    printf("Process %d successfully exchanged data with process %d\n",
           rank, 1 - rank);

    free(sendbuf);
    free(recvbuf);

    MPI_Finalize();
    return 0;
}
```

```
posh@LAPTOP-ELUQQMKU:~/hpc_assign/assign6$  mpicc -o deadlock_fixed_sendrecv deadlock_fixed_sendrecv.c
posh@LAPTOP-ELUQQMKU:~/hpc_assign/assign6$ mpirun -np 2 ./deadlock_fixed_sendrecv
Process 0 successfully exchanged data with process 1
Process 1 successfully exchanged data with process 0
```

Theory/Info:

- **A deadlock occurs when two or more processes wait indefinitely for each other to send/receive messages.**
- **Using blocking sends (MPI_Send), if all processes try to send simultaneously without a matching receive, the program hangs.**
- **Purpose: Learn the importance of ordering communication operations to avoid deadlocks.**
  Demonstration: Processes attempting to send large messages to each other simultaneously will hang.
  Key Concepts: Deadlock conditions, blocking communication, synchronization issues.
  Conclusion:
- **Demonstrates the consequences of improper communication ordering.**
- **Highlights the need for careful design in parallel programs to avoid deadlocks.**

Topic: MPI_Sendrecv
Theory/Info:

- **MPI_Sendrecv allows a process to simultaneously send and receive a message safely.**
- **This prevents deadlocks because each process posts a send and receive in one atomic call.**
- **Purpose: Learn safe data exchange patterns in MPI to avoid deadlocks.**
  Demonstration: Two processes exchange messages successfully and print confirmation.
  Key Concepts: Synchronous communication, deadlock prevention, atomic send-receive.
  Conclusion:
- **Confirms that deadlocks can be prevented using proper MPI functions.**
- **Demonstrates reliable communication even with simultaneous data exchange.**

Q4. Implement blocking MPI send & receive to demonstrate Nearest neighbor exchange of data in a ring topology.

Program and screenshots

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char *argv[]) {
    MPI_Init(&argc, &argv);
    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    int my_value;
    if (rank == 0) {
        printf("Enter an integer value (rank 0): ");
        fflush(stdout);
        scanf("%d", &my_value);
    }
    MPI_Bcast(&my_value, 1, MPI_INT, 0, MPI_COMM_WORLD);

    int left = (rank - 1 + size) % size;
    int right = (rank + 1) % size;
    int recv_value = -1;

    if (rank % 2 == 0) {
        MPI_Send(&my_value, 1, MPI_INT, right, 0, MPI_COMM_WORLD);
        MPI_Recv(&recv_value, 1, MPI_INT, left, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
    } else {
        MPI_Recv(&recv_value, 1, MPI_INT, left, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
        MPI_Send(&my_value, 1, MPI_INT, right, 0, MPI_COMM_WORLD);
    }

    printf("Rank %d sent %d to %d, received %d from %d\n",
```

```
            rank, my_value, right, recv_value, left);

    MPI_Finalize();
    return 0;
}
```

```
Process 1 successfully exchanged data with process 0
posh@LAPTOP-ELUQQMKU:~/hpc_assign/assign6$ mpicc -o ring_exchange ring_exchange.c
posh@LAPTOP-ELUQQMKU:~/hpc_assign/assign6$ mpirun -np 4 ./ring_exchange
Enter an integer value (rank 0): 42
Rank 0 sent 42 to 1, received 42 from 3
Rank 1 sent 42 to 2, received 42 from 0
Rank 2 sent 42 to 3, received 42 from 1
Rank 3 sent 42 to 0, received 42 from 2
```

Theory/Info:

- **In a ring topology, each process communicates only with its neighbors (left and right).**
- **MPI allows designing custom topologies for structured communication.**
- **Purpose: Understand neighbor communication patterns, blocking sends, and data propagation in a ring.**
  Demonstration: Each process sends its data to the next process and receives from the previous one; output shows correct exchange.
  Key Concepts: Process topologies, nearest neighbor communication, synchronization in a structured network.
  Conclusion:
- **Validates communication in a ring structure is correctly implemented.**
- **Useful for algorithms like token passing, pipeline computations, or distributed workflows.**

Q5. Write a MPI program to find the sum of all the elements of an array A of size

n. Elements of an array can be divided into two equals groups. The first [n/2]

elements are added by the first process, P0, and last [n/2] elements the by second process, P1. The two sums then are added to get the final result.

Program and screenshots

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[]) {
    int rank, size;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    if (size != 2) {
        if (rank == 0) printf("Run this program with exactly 2
processes!\n");
        MPI_Finalize();
        return 0;
    }

    int n;
    int *arr = NULL;

    if (rank == 0) {
        printf("Enter total number of elements: ");
        fflush(stdout);
        scanf("%d", &n);
        arr = (int*)malloc(n * sizeof(int));
        printf("Enter %d integers: ", n);
        for (int i = 0; i < n; i++) scanf("%d", &arr[i]);
    }

    MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
    int half = n / 2;
    int remaining = n - half;
```

```
    int local_sum = 0;

    if (rank == 0) {
        for (int i = 0; i < half; i++) local_sum += arr[i];
        MPI_Send(arr + half, remaining, MPI_INT, 1, 0,
MPI_COMM_WORLD);
        int sum1;
        MPI_Recv(&sum1, 1, MPI_INT, 1, 1, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
        printf("Final sum = %d (P0=%d + P1=%d)\n", local_sum + sum1,
local_sum, sum1);
        free(arr);
    } else if (rank == 1) {
        int *subarr = (int*)malloc(remaining * sizeof(int));
        MPI_Recv(subarr, remaining, MPI_INT, 0, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
        for (int i = 0; i < remaining; i++) local_sum += subarr[i];
        MPI_Send(&local_sum, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
        free(subarr);
    }

    MPI_Finalize();
    return 0;
}
```

```
posh@LAPTOP-ELUQQMKU:~/hpc_assign/assign6$ mpicc -o sum_two_processes sum_two_processes.c
posh@LAPTOP-ELUQQMKU:~/hpc_assign/assign6$ mpirun -np 2 ./sum_two_processes
Enter total number of elements: 7
1 2 3 4 5 6 7
Enter 7 integers: Final sum = 28 (P0=6 + P1=22)
posh@LAPTOP-ELUQQMKU:~/hpc_assign/assign6$
```

Theory/Info:

- Large arrays can be split across multiple processes for parallel computation.
- Each process computes a partial sum and sends it to a master process to compute the final sum.

- Purpose: Learn data partitioning, communication of partial results, and parallel reduction.
  Demonstration:
  - Process 0 sums the first half of the array.
  - Process 1 sums the second half.
  - Partial sums are combined at process 0 to get the total.
    Key Concepts: Data parallelism, reduction operations, inter-process communication.
    Conclusion:
- Shows the effectiveness of parallel computation in reducing workload.
- Confirms correct summation of array elements across multiple processes.