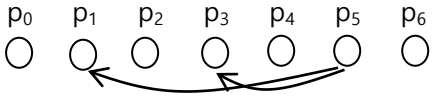
## Lab11 – Communicator and Grid

### ex1. Create new communicator group from old communicator.

Complete the following MPI program(odd.c) to create a new communicator with odd numbered processes and to broadcast the data from the biggest numbered process. The next figure show an example of running 7 processes.




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

int main(int argc, char* argv[])
{
    int pid, np, tag = 0, i;
    MPI_Status status;

    MPI_Group group_world, group_odd;
    MPI_Comm comm_odd;
    int *newrank;
    int data;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &pid);
    MPI_Comm_size(MPI_COMM_WORLD, &np);

    data = 10 + pid;
```

```
    newrank = (int*)(malloc(np/2*sizeof(int)));
    for (i=0; i<np/2; i++)
        newrank[i] = ...;

    MPI_Comm_group(..., &group_world);
    MPI_Group_incl(..., &group_odd);
    MPI_Comm_create(..., &comm_odd);

    if (...) MPI_Bcast(..., comm_odd);

    printf("%d %d\n", pid, data);

    free(newrank);

    MPI_Finalize();
}
```

### ex2. Split a communicator

Complete the following MPI program(odd_split.c) to split a communicator into even numbered processes and odd numbered processes and to broadcast from the biggest numbered process. The execution result is the same as the above exercise (ex1).

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

int main(int argc, char* argv[])
{
    int pid, np, flag, tag = 0, i;
    MPI_Status status;

    MPI_Comm comm_odd;
    int data;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &pid);
    MPI_Comm_size(MPI_COMM_WORLD, &np);
```

```
    data = 10 + pid;

    flag = ...;
    MPI_Comm_split(..., &comm_odd);

    if (...) MPI_Bcast(..., comm_odd);

    printf("%d %d\n", pid, data);

    MPI_Finalize();
}
```

**ex3. Grid topology(1)**

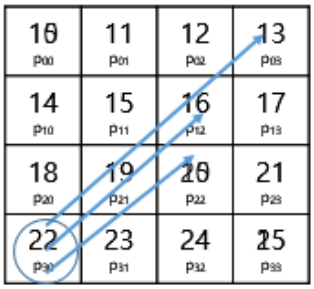
Complete the following MPI program(grid.c) to shift the row or the column direction provided by command-line arguments.

Run the program "mpiexec -n #proc *prog* direction shift" and find the out results. Use #proc = a number of square of 2, direction = 0 or 1, shift = 1 or -1.

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

int main(int argc, char* argv[])
{
    int np, pid, inp, jnp, i, j, k, pid_from, *buf;
;
    int dim_sizes[2], wrap_around[2];
    int direct, shift, source, dest, reorder, tag=0;
    MPI_Comm grid_comm;
    MPI_Status status;

    if (argc != 3) {
        printf("usage %s direct shift\n", argv[0]);
        exit(1);
    }
    direct = atoi(argv[1]); shift = atoi(argv[2]);

    MPI_Init(&argc, &argv);

    MPI_Comm_rank(MPI_COMM_WORLD, &pid);
    MPI_Comm_size(MPI_COMM_WORLD, &np);

    inp = sqrt(np); jnp = np/inp;
    dim_sizes[0] = inp;
    dim_sizes[1] = jnp;
    wrap_around[0] = wrap_around[1] = 1;
    MPI_Cart_create(..., &grid_comm);
    MPI_Cart_shift(grid_comm, direct, shift, &source, &dest);

    MPI_Sendrecv(&pid, ..., &pid_from, ...);

    // print the shifted matrix
    if (pid == 0 ) buf = (int *)malloc(sizeof(int)*np);
    MPI_Gather(&pid_from, 1, MPI_INT, buf, 1, MPI_INT, 0,
MPI_COMM_WORLD);
    if (pid == 0 ) {
        k = 0;
        for (i = 0; i < inp; i++) {
            for (j = 0; j < jnp; j++)
                printf("%2d ", buf[k++]);
            printf("₩n");
        }
        free(buf);
    }

    MPI_Finalize();
}
```

**ex4. Grid topology(2)**

Complete the following MPI program(diag.c) for the leftmost bottom process to broadcast data. Follow figure illustrates a case of using 16 processes.




Run the program "mpiexec -n #proc *prog*" where #proc is a number of square of 2.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "mpi.h"

main(int argc, char* argv[])
{
    int np, pid, inp, jnp, i, j, k, *buf;
    int dim_sizes[2], wrap_around[2];
    int direct, shift, source, dest, reorder, tag=0;
    MPI_Comm grid_comm;
    MPI_Status status;

    int coordinates[2]; /* i and j location in the grid */

    MPI_Group diag_world;
    MPI_Group diag_group;
    MPI_Comm diag_comm;
    int *diag_rank;
    int data;

    MPI_Init(&argc, &argv);

    MPI_Comm_rank(MPI_COMM_WORLD, &pid);
    MPI_Comm_size(MPI_COMM_WORLD, &np);

    inp = sqrt(np); jnp = np/inp;
    dim_sizes[0] = inp;
    dim_sizes[1] = jnp;
    wrap_around[0] = wrap_around[1] = 1;

    MPI_Cart_create(..., &grid_comm);
    MPI_Cart_coords(..., coordinates);
```

```
    // new communicator and broadcast
    diag_rank = (int*)malloc(inp*sizeof(int));
    for (i = 0; i < jnp; i++)
        diag_rank[i] = ...;

    MPI_Comm_group(..., &diag_world);
    MPI_Group_incl(..., &diag_group);
    MPI_Comm_create(..., &diag_comm);

    data = pid+10;
    if (...)
        MPI_Bcast(&data, 1, MPI_INT, jnp-1, diag_comm);

    // print the shifted matrix
    if (pid == 0 ) buf = (int *)malloc(sizeof(int)*np);
    MPI_Gather(&data, 1, MPI_INT, buf, 1, MPI_INT, 0,
MPI_COMM_WORLD);
    if (pid == 0 ) {
        k = 0;
        for (i = 0; i < inp; i++) {
            for (j = 0; j < jnp; j++)
                printf("%2d ", buf[k++]);
            printf("₩n");
        }
        free(buf);
    }
}
```

Submit 4 programs - odd.c, odd_split.c, grid,c and diag.c – individually when you are done.