Lab5 – MPI(collective)

ex.1 Run the following MPI program on different servers

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

main(int argc, char* argv[])
{
    char processor_name[80];
    int pid, name_len;

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

    MPI_Get_processor_name(processor_name, &name_len);
    printf("%s, rank %d₩n", processor_name, pid);

    MPI_Finalize();
}
```

Use the following command to run the program on different servers(hpa, hpb, hpc).

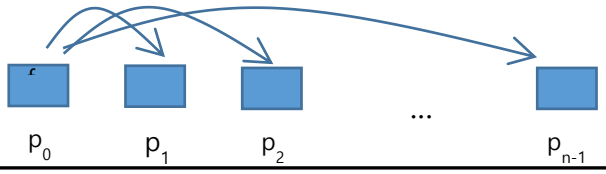
mpiexec –f mf –n 10 *prog*

mf: hpa:2
hpb:3
hpc:5

다양한 값 줘서 확인.

ex.2 Broadcasting(MPI_Bcast)

Complete the following MPI program to broadcast a data from p_0 to all other processes




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

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

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

    if (pid == 0) data = 100.0;
    [                                    ]

    printf("%f₩n", pid+data);
    MPI_Finalize();
}
```

just Bcast..
(원하는 숫자 try.)

ex.3 Reduction(MPI_Reduce)

Complete the following MPI program(rand.c) to find the minimum number (using MPI_Reduce() from random numbers generated in parallel.

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

#define N 12000

int next = 1;
int rand2(int a, int c)
{
    return (next = next * a + c);
}
void srand2(int seed)
{
    next = seed;
}
void random_p(int n, int a, int c, int arr[n])
{
    int i;

    for (i=1; i<n; i++)
        arr[i] = rand2(a, c);
}

int main(int argc, char *argv[])
{
    int X[N], *local_X, np, pid, local_N, i, min, temp;
    int a, c, A, C;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &np);
    MPI_Comm_rank(MPI_COMM_WORLD, &pid);
    local_N = N/np;
    srand2(1);
    a = 1103515245;
    c = 12345;
    // A and are for parallel random
numbers
    A = 1; C = 0;
    for (i=0; i<np; i++) {
        A = (a * A);
        C = (a * C + c);
    }

    local_X = malloc(sizeof(int)*local_N);

    // initial random numbers (sequential)
    srand2(1);
    for (i=0; i<pid+1; i++)
        local_X[0] = rand2(a, c);

    // parallel random number generator
    random_p(local_N, A, C, local_X);
                                &local_X[ ]
    min = INT_MAX;

    // FILL IN THIS BLANK



    if (pid == 0) printf("%d₩n", min);

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

총 12000개의 난수를 가지고 가장 최솟값 찾기.

Submit rand.c when you complete programming.