Shang-Ian Tan
a1818695

PDC Assignment 1 Report

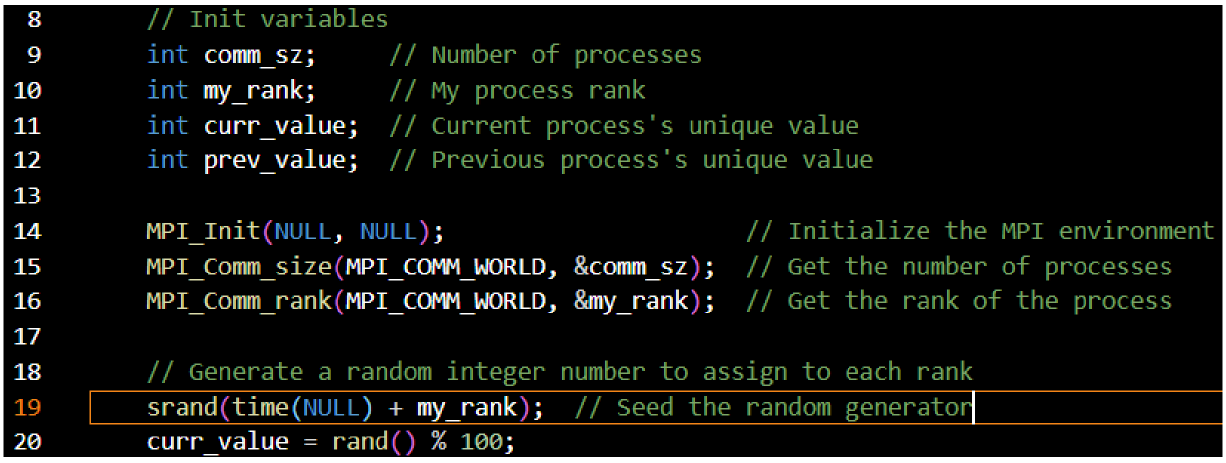
Introduction

The objective of the assignment is to write two MPI programs, within a SPMD framework, using message-passing operations to pass messages between separate processes. Both programs involve assigning a value to each process, out of the 5 called during program execution. The programs are designed to find numbers out of order and print information about them. I have referenced the program 3.1 in the textbook for a template.

Initialisation In Separate Processes

Each process initializes a random number and checks if these numbers are in order based on their ranks.

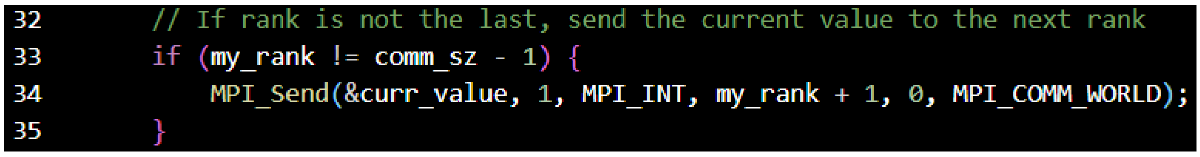
Upon initialization, each process sets up local variables including the number of processes (`comm_sz`), the rank of the current process (`my_rank`), the current process's unique value (`curr_value`), and the previous process's unique value (`prev_value`). MPI is then initialized for all processes using `MPI_Init()`. Random numbers are generated for each process using `rand() % 100`, ensuring uniqueness through seeding with `time(NULL) + my_rank`.



```c
    // Init variables
    int comm_sz;      // Number of processes
    int my_rank;      // My process rank
    int curr_value;   // Current process's unique value
    int prev_value;   // Previous process's unique value

    MPI_Init(NULL, NULL);                        // Initialize the MPI environment
    MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);   // Get the number of processes
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);   // Get the rank of the process

    // Generate a random integer number to assign to each rank
    srand(time(NULL) + my_rank);  // Seed the random generator
    curr_value = rand() % 100;
```

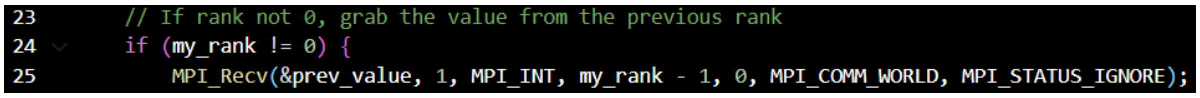
Program 1:

Program 1 communicates between processes using `MPI_Send()` & `MPI_Recv()`, where each process sends its current value (*curr_value*) to the next process



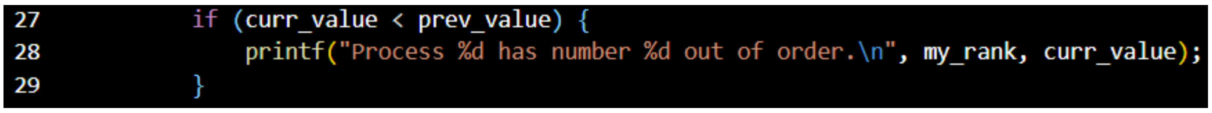
```c
    // If rank is not the last, send the current value to the next rank
    if (my_rank != comm_sz - 1) {
        MPI_Send(&curr_value, 1, MPI_INT, my_rank + 1, 0, MPI_COMM_WORLD);
    }
```

and receives the previous process's value (*prev_value*).



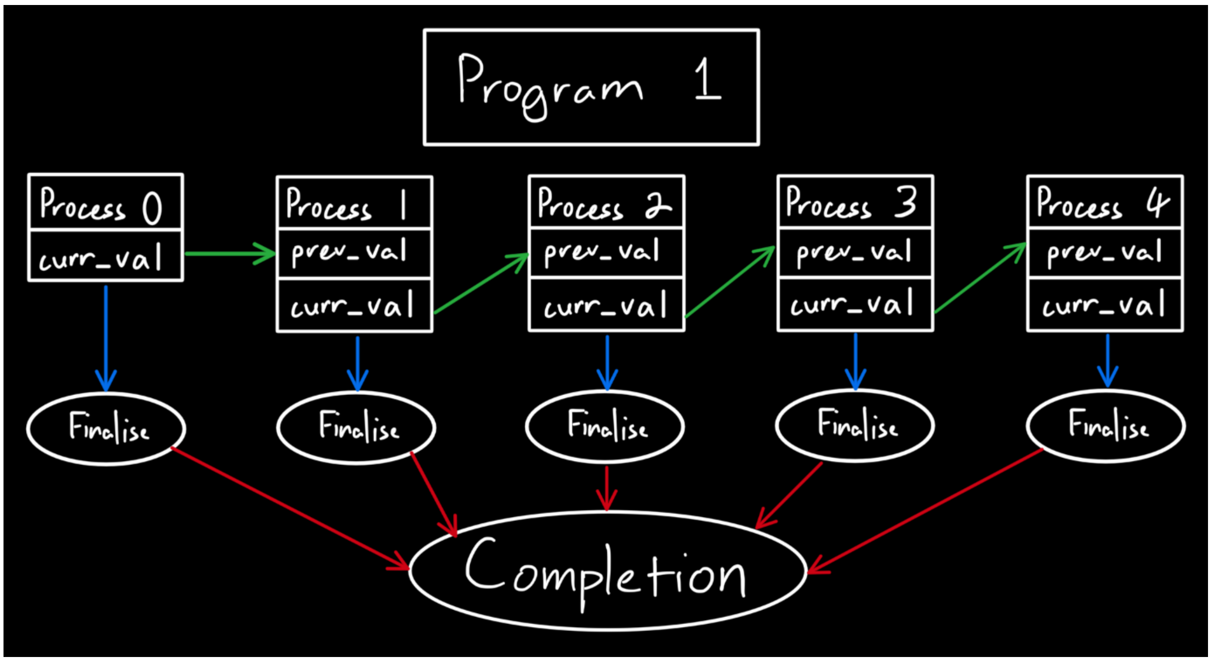
```c
    // If rank not 0, grab the value from the previous rank
    if (my_rank != 0) {
        MPI_Recv(&prev_value, 1, MPI_INT, my_rank - 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

If a current process discovers that its `` ` ``*curr_value*`` ` `` is less than the `` ` ``*prev_value*`` ` ``, it outputs that the current number is out order.



```c
        if (curr_value < prev_value) {
            printf("Process %d has number %d out of order.\n", my_rank, curr_value);
        }
```

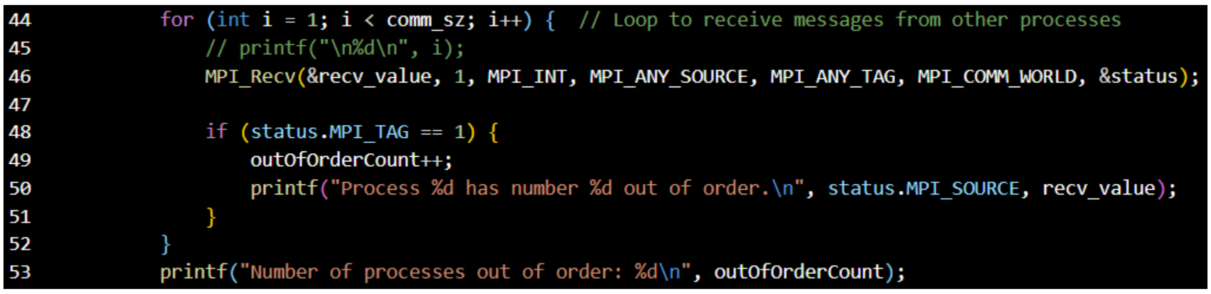
Interaction Diagram (Program 1):



Program 2:

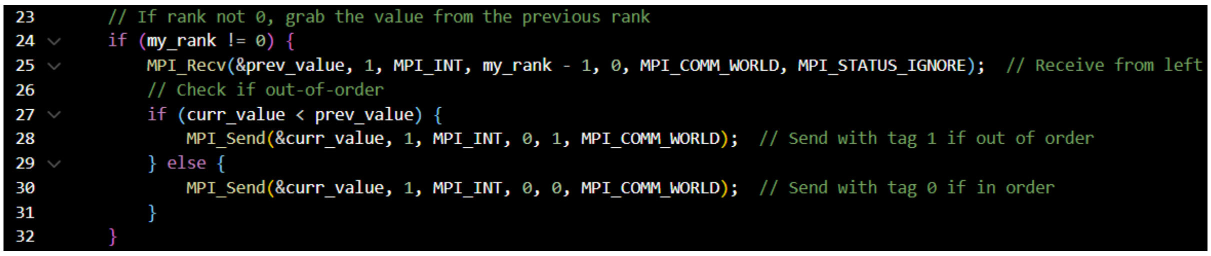
Program 2 uses the similar methodology for communication between processes as program 1. The difference between the two programs lies in the roles of the master (rank 0) and every other process.

The master's role is to receive and compute output based on the messages sent from subsequent processes, incrementing a counter for every out-of-order process. It does this by checking whether the message received had the tag value 1, and if so, that process had an out-of-order value. The counter is incremented and the rank number and value is output. After checking all processes, the final count of out-of-order processes is displayed.



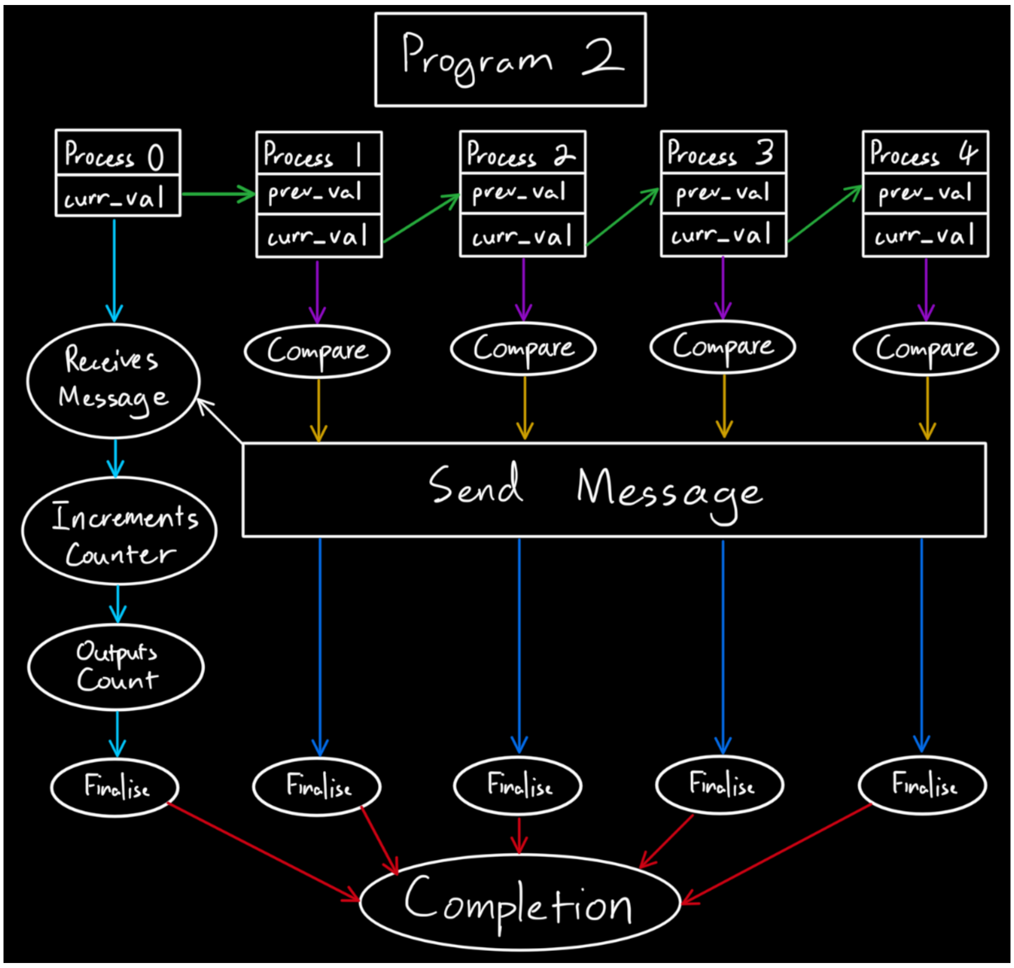
```c
        for (int i = 1; i < comm_sz; i++) {  // Loop to receive messages from other processes
            // printf("\n%d\n", i);
            MPI_Recv(&recv_value, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &status);

            if (status.MPI_TAG == 1) {
                outOfOrderCount++;
                printf("Process %d has number %d out of order.\n", status.MPI_SOURCE, recv_value);
            }
        }
        printf("Number of processes out of order: %d\n", outOfOrderCount);
```

Processes other than the master are responsible for comparing their *curr_value* and the prev_value and sending a message to the master with an appropriate tag.

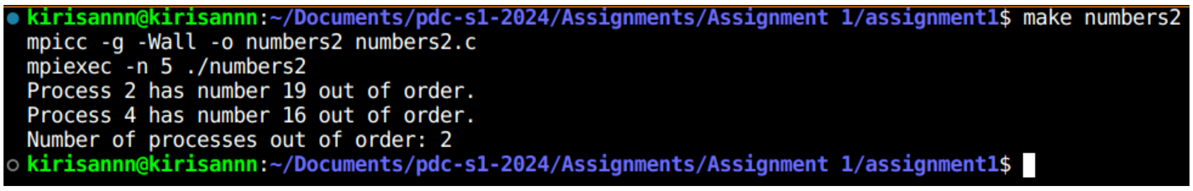


```c
    // If rank not 0, grab the value from the previous rank
    if (my_rank != 0) {
        MPI_Recv(&prev_value, 1, MPI_INT, my_rank - 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);  // Receive from left
        // Check if out-of-order
        if (curr_value < prev_value) {
            MPI_Send(&curr_value, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);  // Send with tag 1 if out of order
        } else {
            MPI_Send(&curr_value, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);  // Send with tag 0 if in order
        }
    }
```

Interaction Diagram (Program 2):



Copy of the Ordered Output



```
● kirisannn@kirisannn:~/Documents/pdc-s1-2024/Assignments/Assignment 1/assignment1$ make numbers2
  mpicc -g -Wall -o numbers2 numbers2.c
  mpiexec -n 5 ./numbers2
  Process 2 has number 19 out of order.
  Process 4 has number 16 out of order.
  Number of processes out of order: 2
○ kirisannn@kirisannn:~/Documents/pdc-s1-2024/Assignments/Assignment 1/assignment1$ 
```