Lab 4 – Collective Communication and Error Handling in MPI

Solved.

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

int main(int argc, char *argv[]) {
    int rank, size, fact = 1, factsum, i;

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

    for (i = 1; i <= rank + 1; i++)
        fact = fact * i;

    MPI_Reduce(&fact, &factsum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

    MPI_Barrier(MPI_COMM_WORLD);

    if (rank == 0)
        printf("Sum of all the factorial=%d\n", factsum);

    MPI_Finalize();
    return 0;
}
```

Output.

```
6CSEC1@debian:~/Documents/230905152_PPL/L4$ mpicc -o abc solved.c
6CSEC1@debian:~/Documents/230905152_PPL/L4$ mpirun -np 5 ./abc
Sum of all the factorial=153
6CSEC1@debian:~/Documents/230905152_PPL/L4$
```

Q1.

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

int main(int argc, char *argv[]) {
    int rank, size, i, err;
    long long fact = 1;
    long long sum_of_facts = 0;
```

```
    char err_msg[MPI_MAX_ERROR_STRING];
    int msg_len;

    err = MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    MPI_Comm_set_errhandler(MPI_COMM_WORLD, MPI_ERRORS_RETURN);

    err = MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if (err != MPI_SUCCESS) {
        MPI_Error_string(err, err_msg, &msg_len);
        printf("Error in MPI_Comm_rank: %s\n", err_msg);
        MPI_Abort(MPI_COMM_WORLD, err);
    }

    err = MPI_Comm_size(MPI_COMM_WORLD, &size);
    if (err != MPI_SUCCESS) {
        MPI_Error_string(err, err_msg, &msg_len);
        printf("Error in MPI_Comm_size: %s\n", err_msg);
        MPI_Abort(MPI_COMM_WORLD, err);
    }

    for (i = 1; i <= rank + 1; i++) {
        fact *= i;
    }

    err = MPI_Scan(&fact, &sum_of_facts, 1, MPI_LONG_LONG, MPI_SUM,
MPI_COMM_WORLD);

    if (err != MPI_SUCCESS) {
        MPI_Error_string(err, err_msg, &msg_len);
        printf("Error in MPI_Scan: %s\n", err_msg);
    }

    if (rank == size - 1) {
        printf("Sum of factorials 1! to %d! is: %lld\n", size, sum_of_facts);
    }
    MPI_Finalize();
    return 0;
}
```

Output.

Q2.

```
bevu@6CSEC2:~/Desktop/230905152_PPL/L4$ mpicc -o abc q1.c
bevu@6CSEC2:~/Desktop/230905152_PPL/L4$ mpirun -np 4 ./abc
Invalid MIT-MAGIC-COOKIE-1 key
Invalid MIT-MAGIC-COOKIE-1 key
Sum of factorials 1! to 4! is: 33
bevu@6CSEC2:~/Desktop/230905152_PPL/L4$ mpirun -np 400 ./abc
[proxy:0@6CSEC2] HYDU_create_process (lib/utils/launch.c:24): pipe error (Too many open
 files)
[proxy:0@6CSEC2] launch_procs (proxy/pmip_cb.c:1000): create process returned error
[proxy:0@6CSEC2] handle_launch_procs (proxy/pmip_cb.c:585): launch_procs returned erro
[proxy:0@6CSEC2] HYD_pmcd_pmip_control_cmd_cb (proxy/pmip_cb.c:495): launch_procs retu
ned error
[proxy:0@6CSEC2] HYDT_dmxu_poll_wait_for_event (lib/tools/demux/demux_poll.c:76): call
ack returned error status
[proxy:0@6CSEC2] main (proxy/pmip.c:122): demux engine error waiting for event
[mpiexec@6CSEC2] control_cb (mpiexec/pmiserv_cb.c:280): assert (!closed) failed
[mpiexec@6CSEC2] HYDT_dmxu_poll_wait_for_event (lib/tools/demux/demux_poll.c:76): call
ack returned error status
[mpiexec@6CSEC2] HYD_pmci_wait_for_completion (mpiexec/pmiserv_pmci.c:173): error wait
ng for event
[mpiexec@6CSEC2] main (mpiexec/mpiexec.c:260): process manager error waiting for compl
tion
bevu@6CSEC2:~/Desktop/230905152_PPL/L4$
```

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

int main(int argc, char *argv[]) {
    int rank, size, mat[3][3], key, local = 0, total;

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

    if (size != 3) MPI_Abort(MPI_COMM_WORLD, 1);

    if (rank == 0) {
        fprintf(stdout, "Enter a 3*3 matrix: \n");
        fflush(stdout);
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                scanf("%d", &mat[i][j]);
            }
        }
        fprintf(stdout, "Enter element to count: \n");
        fflush(stdout);
        scanf("%d", &key);
    }
```

```
    MPI_Bcast(mat, 9, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Bcast(&key, 1, MPI_INT, 0, MPI_COMM_WORLD);

    for (int j = 0; j < 3; j++)
        if (mat[rank][j] == key)
            local++;

    MPI_Reduce(&local, &total, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

    if (rank == 0)
        printf("The number %d appears a total of %d times in the given 3x3
matrix.\n", key, total);

    MPI_Finalize();
    return 0;
}
```

Output.

```
6CSEC1@debian:~/Documents/230905152_PPL/L4$ mpicc -o abc q2.c
6CSEC1@debian:~/Documents/230905152_PPL/L4$ mpirun -np 3 ./abc
Enter a 3*3 matrix:
1 2 3
3 2 1
2 4 5
Enter element to count:
2
The number 2 appears a total of 3 times in the given 3x3 matrix.
6CSEC1@debian:~/Documents/230905152_PPL/L4$
```

Q3.

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

int main(int argc, char *argv[]) {
    int rank, size;
    int mat[4][4], result[4][4];

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

    if (size != 4)
```

```
        MPI_Abort(MPI_COMM_WORLD, 1);

    if (rank == 0) {
        fprintf(stdout, "Enter a 4*4 matrix: \n");
        fflush(stdout);
        for (int i = 0; i < 4; i++)
            for (int j = 0; j < 4; j++)
                scanf("%d", &mat[i][j]);
    }

    MPI_Bcast(mat, 16, MPI_INT, 0, MPI_COMM_WORLD);

    MPI_Scan(mat[rank], result[rank], 4, MPI_INT, MPI_SUM, MPI_COMM_WORLD);

    MPI_Gather(result[rank], 4, MPI_INT, result, 4, MPI_INT, 0, MPI_COMM_WORLD);

    if (rank == 0) {
        fprintf(stdout, "Resultant Matrix: \n");
        fflush(stdout);
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++)
                printf("%d ", result[i][j]);
            printf("\n");
        }
    }

    MPI_Finalize();
    return 0;
}
```

Output.

```
6CSEC1@debian:~/Documents/230905152_PPL/L4$ mpicc -o abc q3.c
6CSEC1@debian:~/Documents/230905152_PPL/L4$ mpirun -np 4 ./abc
Enter a 4*4 matrix:
1 1 1 1
2 2 2 2
3 3 3 3
4 4 4 4
Resultant Matrix:
1 1 1 1
3 3 3 3
6 6 6 6
10 10 10 10
6CSEC1@debian:~/Documents/230905152_PPL/L4$
```

Q4.

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

int main(int argc, char *argv[]) {
    int rank, size, len;
    char word[100], result[1000] = "";
    char my_char;

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

    if (rank == 0) {
        printf("Enter Word: \n");
        scanf("%s", word);
        len = strlen(word);
        if (len != size) MPI_Abort(MPI_COMM_WORLD, 1);
    }

    MPI_Scatter(word, 1, MPI_CHAR, &my_char, 1, MPI_CHAR, 0, MPI_COMM_WORLD);

    char temp[100] = "";
    for (int i = 0; i <= rank; i++) {
        temp[i] = my_char;
    }
    temp[rank + 1] = '\0';
```

```
    MPI_Gather(temp, 100, MPI_CHAR, result, 100, MPI_CHAR, 0, MPI_COMM_WORLD);

    if (rank == 0) {
        for (int i = 0; i < size; i++)
            printf("%s", result + i * 100);
        printf("\n");
    }

    MPI_Finalize();
    return 0;
}
```

Output.

```
6CSEC1@debian:~/Documents/230905152_PPL/L4$ mpicc -o abc q4.c
6CSEC1@debian:~/Documents/230905152_PPL/L4$ mpirun -np 4 ./abc
Enter Word:
abcd
abbcccdddd
6CSEC1@debian:~/Documents/230905152_PPL/L4$ mpirun -np 4 ./abc
Enter Word:
1234
1223334444
```

A1.

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

#define N 5

int main(int argc, char *argv[]) {
    int rank, size;
    int A[N][N], B[N][N];
    int colMin[N], colMax[N];
    int localRow[N];

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

    if (size != N) {
        if (rank == 0) printf("Run with exactly 5 processes\n");
```

```
        MPI_Finalize();
        return 0;
    }

    if (rank == 0) {
        int temp[N][N] = {
            {1, 2, 3, 4, 5},
            {5, 4, 3, 2, 1},
            {10, 3, 13, 14, 15},
            {11, 22, 11, 33, 44},
            {1, 12, 5, 4, 6}
        };

        fprintf(stdout, "Matrix A:\n");
        fflush(stdout);
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++)
                printf("%3d ", temp[i][j]);
            printf("\n");
        }

        for (int i = 0; i < N; i++)
            for (int j = 0; j < N; j++)
                A[i][j] = temp[i][j];

        for (int j = 0; j < N; j++) {
            colMin[j] = A[0][j];
            colMax[j] = A[0][j];
            for (int i = 1; i < N; i++) {
                if (A[i][j] < colMin[j]) colMin[j] = A[i][j];
                if (A[i][j] > colMax[j]) colMax[j] = A[i][j];
            }
        }
    }

    MPI_Bcast(A, N * N, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Bcast(colMin, N, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Bcast(colMax, N, MPI_INT, 0, MPI_COMM_WORLD);

    for (int j = 0; j < N; j++) {
        if (rank == j)
            localRow[j] = 0;
        else if (rank > j)
```

```
            localRow[j] = colMax[rank];
        else
            localRow[j] = colMin[rank];
    }

    MPI_Gather(localRow, N, MPI_INT, B, N, MPI_INT, 0, MPI_COMM_WORLD);

    if (rank == 0) {
        printf("Matrix B:\n");
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++)
                printf("%3d ", B[i][j]);
            printf("\n");
        }
    }

    MPI_Finalize();
    return 0;
}
```

Output.

```
6CSEC1@debian:~/Documents/230905152_PPL/L4$ mpicc -o abc a1.c
6CSEC1@debian:~/Documents/230905152_PPL/L4$ mpirun -np 5 ./abc
Matrix A:
  1   2   3   4   5
  5   4   3   2   1
 10   3  13  14  15
 11  22  11  33  44
  1  12   5   4   6
Matrix B:
  0   1   1   1   1
 22   0   2   2   2
 13  13   0   3   3
 33  33  33   0   2
 44  44  44  44   0
```

A2.

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

double f(double x) {
    return 4.0 / (1.0 + x * x);
```

```c
}

int main(int argc, char *argv[]) {
    int rank, size;
    long long n = 1000000;
    double h, local_sum = 0.0, pi = 0.0;

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

    if (rank == 0) {
        printf("The number of intervals(rectangles) are: %d\n", size);
    }

    h = 1.0 / n;

    for (long long i = rank; i < n; i += size) {
        double x = h * (i + 0.5);
        local_sum += f(x);
    }

    MPI_Reduce(&local_sum, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

    if (rank == 0) {
        pi *= h;
        fprintf(stdout, "Estimated Pi = %.10f\n", pi);
        fflush(stdout);
    }

    MPI_Finalize();
    return 0;
}
```

Output.

```
6CSEC1@debian:~/Documents/230905152_PPL/L4$ mpicc -o abc a2.c
6CSEC1@debian:~/Documents/230905152_PPL/L4$ mpirun -np 5 ./abc
The number of intervals(rectangles) are: 5
Estimated Pi = 3.1415926536
6CSEC1@debian:~/Documents/230905152_PPL/L4$
```