```
#define _CRT_SECURE_NO_WARNINGS
#define WIN32_LEAN_AND_MEAN

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

// for run use command: {mpiexec__path}/mpiexec -n 2 {mpi_df__path}/mpi_df.exe

// A*X + B
#define A 4
#define X 5
#define B 7

#define RESULT (A * X + B)

// (f0:A*X) (f1:B)
//     ||    ||
//     \/    \/
//       (f2:+)


void f0(int * argArr, int * resArr) {
	MPI_Request request;
	//printf("f0\r\n");

	argArr[0] = 4;
	argArr[1] = 5;
	resArr[0] = argArr[0] * argArr[1];
	//_sleep(1000);

	MPI_Isend(resArr, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &request);
	MPI_Request_free(&request);
}

void f1(int * argArr, int * resArr) {
	MPI_Request request;
	//printf("f1\r\n");

	resArr[0] = 7;
	//_sleep(1000);

	MPI_Isend(resArr, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &request);
	MPI_Request_free(&request);
}

void f2(int * argArr, int * resArr) {
	MPI_Status status;
	MPI_Request request;
	//printf("f2\r\n");

	MPI_Recv(argArr,     1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
	//printf("resv from f0 (value = %d)\r\n", argArr[0]);
	MPI_Recv(argArr + 1, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, &status);
	//printf("resv from f1 (value = %d)\r\n", argArr[1]);

	resArr[0] = argArr[0] + argArr[1];
	//_sleep(1000);

	MPI_Isend(resArr, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &request);
	MPI_Request_free(&request);
}
```

```
#define MAX_STAGE_COUNT 2
#define MAX_PE_COUNT 2

void(*fArr[MAX_STAGE_COUNT][MAX_PE_COUNT])(int * argArr, int * resArr) = {
	{ f0,   f1   },
	{ f2,   NULL }
};

void compute(int argc, char* argv[]){
	MPI_Status status;
	int procNum, procRank;//, recvRank;
	int argArr[MAX_PE_COUNT * 2], resArr[MAX_PE_COUNT * 2];

	MPI_Init(&argc, &argv);
	MPI_Comm_size(MPI_COMM_WORLD, &procNum);
	MPI_Comm_rank(MPI_COMM_WORLD, &procRank);

	for (int iIndex = 0; iIndex < MAX_STAGE_COUNT; ++iIndex) {
		switch (procRank){
		case 0:
			if (fArr[iIndex][0]) fArr[iIndex][0](argArr, resArr);
			break;
		case 1:
			if (fArr[iIndex][1]) fArr[iIndex][1](argArr, resArr);
			break;
		default:
			break;
		}
	}

	if (procRank == 0){
		MPI_Recv(resArr, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);

		printf("result of execution = %d \r\n", *resArr);
		printf("expected result     = %d \r\n", RESULT);
		printf("-------------------------------\r\n");
		if (*resArr == RESULT){
			printf("verify status: succes\r\n");
		}
		else{
			printf("verify status: not success\r\n");
		}
	}

	MPI_Finalize();

}

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

	return 0;
}
```