**Parallel Mustaqil ish**

*Ilhomjonov Iqbolshoh*

1. Dasturlarni cmd orqali ishga tushirish usullari.

Cmd orqali dasturni ishga tushirish uchun dastur yo’lini qo’shtirnoq ichiga olib cmdga kiritamiz va Enter tugmasi bosiladi so’ng o’sha dastur ishga tushadi.

Misol uchun:

#include <iostream>

using namespace std;

int main()

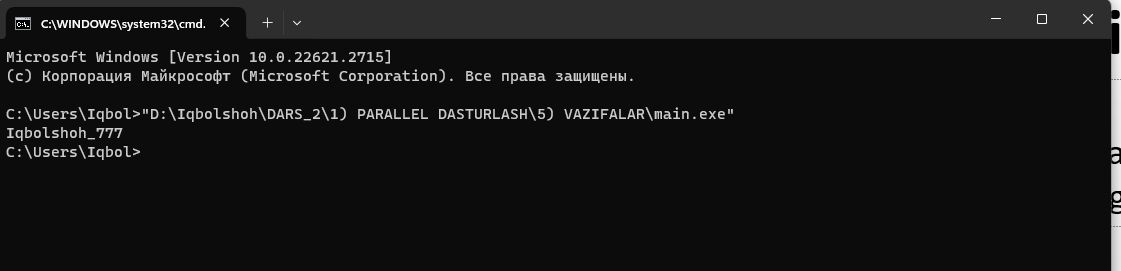
{

    cout << "Iqbolshoh\_777";

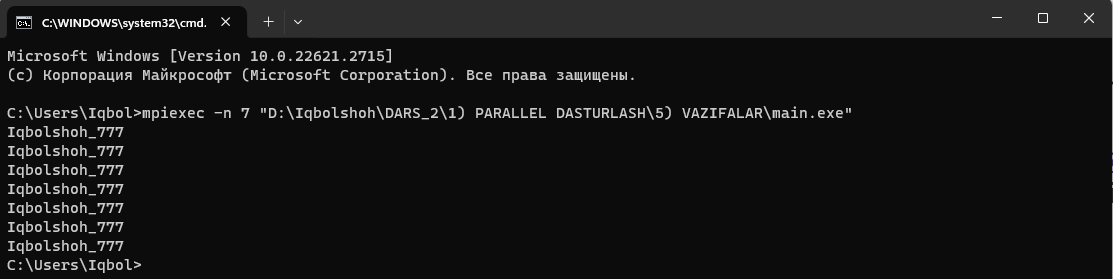
    return 0;

}

Bu dasturning manzilini cmd ga yozib ENTER ni bosamiz.



Dasturni ishga tushirishni yana bir yo’li bu bir necha jarayon bilan ishga tushirish. Dasturlarni bir nechta jarayonlar orqali ishga tushirish uchun bizga MPI texnologiyasi kerak bo’ladi. MPI texnologiyasini o’rnatamiz, so’ng dastur yo’li oldiga mpiexec –n 7 bunda 7 jarayonlar soni.



1. Cmd buyruqlari. MPI kutubxonasining yaratilish tarixi.

Cmd buyruqlari

* **dir** – File larni ko’rish.
* **mkdir** - Yangi papka (jild) yaratish mumkin.
* **ren** – File nomini o’zgartirish.
* **cd** – Filega kirish.
* **cd ..** - Filedan chiqish.
* **rmdir** – Fileni o’chirish.
* **copy** – Fileni nusxalash uchun ishlatiladi.
* **move** - Faylni boshqa joyga ko'chirish uchun ishlatiladi.

MPI kutubxonasining yaratilish tarixi.

MPI (Message Passing Interface), parallel dasturlash uchun ishlatiladigan kutubxona hisoblanadi. MPI kompyuterlar o'rtasida ma'lumot almashish va parallel amallarni bajarish uchun mo'ljallangan. MPI bir qancha tarmoqda yuritilgan kompyuterlar orasida koordinatsiyani ta'minlash uchun ham foydalaniladi.

MPI, 1992 yilida MPI Forum tomonidan ishlab chiqilgan va bu forumda ko'plab soha mutaxassislarining ishtirok etgan. Ushbu forum MPI standartini yaratish uchun qo'llanilgan va MPI-ni ishlab chiqish, yangilanishlar va rivojlanishlar bo'yicha qarorlarni olish uchun yillar davomida faoliyat yuritgan.

MPI standarti, C, C++, Fortran dasturlash tillari uchun buyruq va funksiyalarni o'z ichiga oladi. MPI yordamida, distributiv tizimlarda joylashgan kompyuterlar orasida ma'lumot almashish va parallel amallarni bajarish mumkin bo'ladi. MPI tarmoqlar orasidagi axborot almashuvini, ma'lumotlarni almashish va boshqa amallarni bajarish uchun xizmat qiladi. Shuningdek, MPI, massivlar, strukturalar va yorliqlar kabi ma'lumot turlarini yuborish va qabul qilish uchun mo'ljallangan funksiyalarga ega.

MPI, foydalanuvchilariga turli xil ma'lumotlar almashuvini va amallarni bajarishni o'rganish uchun qulay foydalanuvchilar interfeysini taklif etadi. Ushbu standart, sifatli va mukammal parallel hisoblashni ta'minlash maqsadida yaratilgan, shuningdek, MPI yordamida yaratilgan dasturlar distributiv hisoblash tizimlari uchun yaxshi ishlashini ta'minlaydi.

1. MPI\_status strukturasi. MPI umumiy protseduralari.

MPI\_Status struktura MPI (Message Passing Interface) kutubxonasida o'zgaruvchilar va ma'lumotlar almashuvini tekshirish uchun ishlatiladi. Bu struktura, MPI operatsiyalaridan so'ng olib kelgan ma'lumotlarni saqlash uchun qo'llaniladi. MPI\_Status strukturasining asosiy ma'lumotlariga quyidagilar kiradi:

* MPI\_SOURCE: Bu maydon, xabar yuborgan yoki qabul qilgan kompyuter (process) raqamini (rank) o'z ichiga oladi.
* MPI\_TAG: Xabar yuborishda yoki olishda ishlatilgan teg.
* MPI\_ERROR: Xabar o'qish yoki yozish operatsiyasi natijasida yuzaga kelgan xatolikni ifodalovchi maydon. Agar operatsiya muvaffaqiyatli bo'lgan bo'lsa, bu qiymat MPI\_SUCCESS bo'ladi.
* Quyidagi misol MPI\_Status strukturasi ishlatishni ko'rsatadi. Bu misolda MPI MPI\_Recv funksiyasi orqali xabarni olish va MPI\_Get\_count orqali xabar uzunligini aniqlash ko'rsatilgan:

#include <iostream>

#include <mpi.h>

using namespace std;

int main(int argc, char \*\*argv)

{

    MPI\_Init(&argc, &argv);

    int size, rank, count;

    MPI\_Status status;

    MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

    int a = 0;

    if (rank == 0)

    {

        a = 777;

        MPI\_Send(&a, 1, MPI\_INT, 1, 11, MPI\_COMM\_WORLD);

        cout << "0 - jarayondan ma'lumot yubordi:" << a << endl;

    }

    if (rank == 1)

    {

        MPI\_Recv(&a, 1, MPI\_INT, 0, 11, MPI\_COMM\_WORLD, &status);

        cout << "1 - jarayon ma'lumotni oldi:" << a << endl;

        MPI\_Get\_count(&status, MPI\_INT, &count);

        cout << count << " ta element qabul qilindi." << endl;

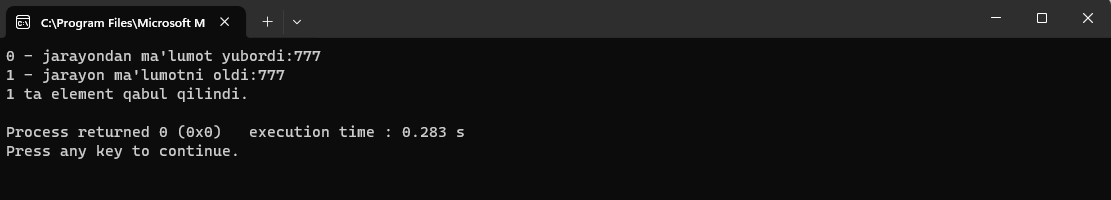
    }

    MPI\_Finalize();

    return 0;

}

// NATIJA:



1. MPI da xabar almashish turlari va ular orasidagi farqlar.

**MPI (Message Passing Interface)** dagi xabar almashish turlari va ularning farqlari quyidagicha:

**Point-to-Point Communication (nuqtadan-nuqtaga aloqalar)** - har bir jarayon orasida o'zaro ma'lumot almashish uchun ishlatiladigan xususiy aloqalarni ifodalaydi. Bu usul orqali bir kompyuterdan boshqa bir kompyuterga yoki kompyuterlar orasida ma'lumot almashish mumkin.

**Collective Communication (Jamoa Aloqalar) -** kompyuter tarmoqlarida joylashgan kompyuterlar orasida o'zaro aloqalarni boshqarish uchun ishlatiladigan usullarni ifodalaydi. Bu usullar, kompyuterlar orasidagi ma'lumot almashuvini yaxshi boshqarish uchun ishlatiladi va bir qancha kompyuterlar orasidagi parallel hisoblash amallarini boshqarishda juda samarali bo'ladi.

**Sinxron Communication (Blakirofkali Aloqalar) -** Eng oddiy xabar almashish usuli. Bir kompyuterdan boshqa bir kompyuterga xabar yuborish uchun MPI\_Send ishlatiladi, va MPI\_Recv orqali xabar olinadi. MPI\_Send va MPI\_Recv funksiyalari doim bloklaydigan funksiyalar hisoblanadi. Bunda xabar qabul qilinmasdan qolishi mumkin. Chunki bunda xabar qabul qilishini kuttirib bo’lmaydi bu esa dastur tugab qolishiga sabab bo’ladi va xabar qabul qilnmasligi mumkin.

MPI\_Send(&data, 1, MPI\_INT, destination\_rank, tag, MPI\_COMM\_WORLD);

MPI\_Recv(&received\_data, 1, MPI\_INT, source\_rank, tag, MPI\_COMM\_WORLD, &status);

**Asinxron Communication (Blakirofkasiz Aloqalar) -** Asinxron xabar almashish operatsiyalari boshqa operatsiyalarni bajarishni to'xtatmaydi. Ushbu usul MPI\_Isend va MPI\_Irecv funksiyalari orqali bajariladi. Keyingi qadamni boshqarish uchun MPI\_Wait yoki MPI\_Waitall funksiyalari ishlatiladi.

MPI\_Isend(&data, 1, MPI\_INT, destination\_rank, tag, MPI\_COMM\_WORLD, &request);

MPI\_Irecv(&received\_data, 1, MPI\_INT, source\_rank, tag, MPI\_COMM\_WORLD,&request);

MPI\_Wait(&request, &status); // yoki MPI\_Waitall

**MPI kutubxonasi tomonidan taqdim etilgan jamoaviy aloqalar**

**MPI\_Bcast -** Barcha kompyuterlarga bir xil ma'lumotni yuborish uchun ishlatiladi. Yani, bitta kompyuter ma'lumotni barcha kompyuterlarga uzatadi. MPI\_Bcast funksiyasi shunday ishlaydi: bitta kompyuter ma'lumotni yuboradi, qolgan kompyuterlar esa uni qabul qilarlar.

MPI\_Bcast(&data, 1, MPI\_INT, root\_rank, MPI\_COMM\_WORLD);

**MPI\_Scatter -** Bir massivni barcha kompyuterlarga bo'lib taqsimlash uchun ishlatiladi. Massiv asosiy kompyuter (root) tomonidan barcha kompyuterlarga bo'linadi. MPI\_Scatter funksiyasi bitta kompyuter tomonidan yuborilgan massivni barcha kompyuterlarga bo'lib taqsimlashni ta'minlaydi.

MPI\_Scatter(send\_data, send\_count, MPI\_INT, recv\_data, recv\_count, MPI\_INT, root\_rank, MPI\_COMM\_WORLD);

**MPI\_Gather -** MPI\_Scatterning qarshisi. Har bir kompyuter o'zida bo'lib taqsimlangan ma'lumotni asosiy kompyuter (root) tomonidan yig'ish uchun ishlatiladi. MPI\_Gather funksiyasi har bir kompyuter tomonidan bo'lib taqsimlangan ma'lumotni asosiy kompyuter tomonidan qabul qilishni ta'minlaydi.

MPI\_Gather(send\_data, send\_count, MPI\_INT, recv\_data, recv\_count, MPI\_INT, root\_rank, MPI\_COMM\_WORLD);

**MPI\_Reduce -** Barcha kompyuterlardagi ma'lumotlarni bitta joyda jamlash uchun ishlatiladi. MPI\_Reduce funksiyasi MPI kutubxonasi tomonidan taqdim etilgan yig'ish operatsiyasini (masalan, MPI\_SUM, MPI\_MAX, MPI\_MIN kabi) ishlatib, barcha kompyuterlardagi ma'lumotlarni jamlash uchun ishlatiladi.

MPI\_Reduce(send\_data, recv\_data, count, MPI\_INT, MPI\_SUM, root\_rank, MPI\_COMM\_WORLD);

**MPI\_Allreduce -** MPI\_Reducening barcha kompyuterlarga tarqatilgan varianti. Barcha kompyuterlardagi ma'lumotlarni bitta joyda jamlash va natijani barcha kompyuterlarga tarqatish uchun ishlatiladi.

MPI\_Allreduce(send\_data, recv\_data, count, MPI\_INT, MPI\_SUM, MPI\_COMM\_WORLD);

Jamoa aloqalari MPI kutubxonasi orqali ko'p qatorda ishlaydigan kompyuterlardagi ma'lumot almashuvini boshqarish uchun juda foydali bo'ladi. Har bir aloqa turi ma'lumotlarni tarqatish, bo'lib taqsimlash, jamlash va boshqa amallar uchun mo'ljallangan.

1. Blokirovkali xabar almashishni real muammolarni yechishga qoʻllanilishi.

MPI kutubxonasida blokirovkali xabar almashish uchun ikkita asosiy funksiya mavjud:

**MPI\_Send -** Bu funksiya orqali bitta kompyuterdan boshqa kompyuterga ma'lumotni blokirovkali tarzda yuboradi.

MPI\_Send(&data, 1, MPI\_INT, destination\_rank, tag, MPI\_COMM\_WORLD);

**MPI\_Recv** -Bu funksiya orqali bitta kompyuter boshqa kompyuterdan ma'lumotni blokirovkali tarzda qabul qiladi.

MPI\_Recv(&received\_data, 1, MPI\_INT, source\_rank, tag, MPI\_COMM\_WORLD, &status);

Blokirovkali xabar almashish, sinxronizatsiya muammolarini yechish uchun juda samarali bo'ladi. Misol uchun, birinchi kompyuter boshqa kompyuterlarga ma'lumot yuboradi va ularni o'zini qabul qilishini kutadi. Xuddi shu paytda, boshqa kompyuterlar ham bloklangan bo'lishi mumkin. Bu, ma'lumot almashishni bir qadamda o'tkazish va boshqa kompyuterlarning boshqa amallarini bajarishini ta'minlaydi. Blokirovkali xabar almashish orqali ma'lumot almashish va uni ishlovchi qismlarni yaxshi boshqarish mumkin. Bitta kompyuter ma'lumotni yuboradi, keyingi kompyuter uni qabul qilar va bloklangan qismni ishlovchi bo'lishi mumkin. Shunday qilib, ma'lumot almashish va ishlovchi qismlar bir xil bo'lib, muammolar paydo bo'lishini engillash uchun engoddiy va samarali usulni ta'minlaydi. Blokirovkali xabar almashish, oddiy va aniq kod yozish uchun juda qulaydir. Kod aniq va oson, ammo u bajarilgandan so'ng boshqa kompyuterlar ishga tushiriladi. Bu, kodning tushunishini va to'g'ri ishlayishini osonlashtiradi. Blokirovkali xabar almashish o'z ichiga yagona xabar almashishni qamrab olishni ta'minlaydi. Bitta kompyuter boshqa kompyuterlarga yuborgan ma'lumotni ko'rib chiqishi mumkin emas, chunki blokirovkali xabar almashishda birinchi kompyuter o'z bajarishini tugatgandan so'ng boshqa kompyuterlar ishga tushadi. Blokirovkali xabar almashish, sinxronizatsiya va ma'lumot almashishda amaliy muammolarni yechishda keng qo'llaniladi. Misollar shu jumladan, distributiv tizimlarda sinxronizatsiya talab etiladigan muammolar, katta ma'lumotlar bazalaridagi qatorlashgan amallar va boshqalar bo'lishi mumkin.

Quyidagi dastur sinxronizatsiya va blokirovkali xabar almashishni amaliy muammolarni yechish uchun yaratilgan. Bitta kompyuter boshqa kompyuterlarga "Hello, MPI!" degan xabar yuboradi, keyingi barcha kompyuterlar xabarni qabul qiladi va bitta joyda to'plangan xabarni ekranga chiqaradi.

#include <iostream>

#include <cstring>

#include <mpi.h>

using namespace std;

int main(int argc, char \*\*argv)

{

    MPI\_Init(&argc, &argv);

    int size, rank;

    MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

    char message[100];

    if (rank == 0)

    {

        strcpy(message, "Hello, MPI!");

        for (int i = 1; i < size; i++)

        {

            MPI\_Send(message, strlen(message) + 1, MPI\_CHAR, i, 0, MPI\_COMM\_WORLD);

        }

    }

    else

    {

        MPI\_Recv(message, sizeof(message), MPI\_CHAR, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

        cout << rank << "- Jarayon qabul qilingan xabar: " << message << endl;

    }

    MPI\_Finalize();

    return 0;

}

// NATIJA:



1. MPI\_Wtick funksiyasi. MPI\_Probe funksiyasining muhim holatlarda qoʻllanilishi.

MPI\_Wtick funksiyasi MPI (Message Passing Interface) kutubxonasidagi vaqt o'lchovining yarim sari (half resolution)ni qaytaradi. U bu o'lchovni faqat o'zgaruvchanlar soni (ticks)ning aylanma bosqichi (cycle) qismini ifodalaydi. Shuningdek, MPI\_Wtick natijani o'lchovni aniqlash uchun ishlatilgan o'lchov turi (ticks per second) orqali ta'minlaydi.

Quyidagi oddiy dastur, MPI\_Wtick funksiyasini ishlatib, o'lchov turi (ticks per second)ni ekranga chiqaradi:

#include <iostream>

#include <mpi.h>

using namespace std;

int main(int argc, char \*\*argv)

{

    MPI\_Init(&argc, &argv);

    double tick = MPI\_Wtick();

    int rank;

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

    if (rank == 0)

    {

        cout << "MPI\_Wtick har bir belgi uchun soniya: " << tick << endl;

    }

    MPI\_Finalize();

    return 0;

}

// NATIJA:



MPI\_Probe funksiyasining muhim holatlarda qoʻllanilishi.

Probe funksiyani kelayotgan xabarni tekshirish uchun ishlatiladi. Prober ikki holatda ishlatiladi birinchi holat vaqtdan yutish uchun ikkinchi holat esa xotiradan yutish uchun.

**Birinchi holat:**

#include <iostream>

#include <mpi.h>

using namespace std;

int main(int argc, char \*\*argv)

{

    int rank;

    int a;

    MPI\_Status status, source;

    MPI\_Init(&argc, &argv);

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

    a = 0;

    if(rank == 1)

    {

        a = 55;

        MPI\_Send(&a, 1, MPI\_INT, 7, 1, MPI\_COMM\_WORLD);

    }

    if (rank == 2)

    {

        a = 77;

        MPI\_Send(&a, 1, MPI\_INT, 7, 1, MPI\_COMM\_WORLD);

    }

    if (rank == 7)

    {

        MPI\_Probe(MPI\_ANY\_SOURCE, 1, MPI\_COMM\_WORLD, &source);

        if(source.MPI\_SOURCE == 1)

        {

            MPI\_Recv(&a, 1, MPI\_INT, 1, 1, MPI\_COMM\_WORLD, &status);

            MPI\_Recv(&a, 1, MPI\_INT, 2, 1, MPI\_COMM\_WORLD, &status);

        }

        if(source.MPI\_SOURCE == 2)

        {

            MPI\_Recv(&a, 1, MPI\_INT, 2, 1, MPI\_COMM\_WORLD, &status);

            MPI\_Recv(&a, 1, MPI\_INT, 1, 1, MPI\_COMM\_WORLD, &status);

        }

    }

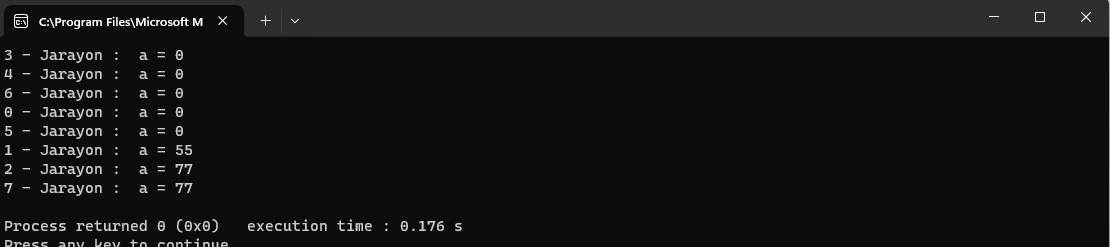
    cout << rank << " - Jarayon : " << " a = " << a << endl;

    MPI\_Finalize();

    return 0;

}

// NATIJA:



Bunda 1 – va 2 - jarayondan xabar yuboryapti va 7 – jarayonga qabul qilinyapti. Prob esa qaysi biri birinchi kelganini aniqlab shuni birinchi olyapti va bu orqali vaqtdan yutyapti.

**Ikkinchi holat:**

#include <iostream>

#include <mpi.h>

using namespace std;

int main(int argc, char \*\*argv)

{

    int rank;

    int a[5] = {0}, b[3] = {0};

    MPI\_Status status;

    MPI\_Init(&argc, &argv);

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

    if (rank == 1)

    {

        for (int i = 0; i < 5; i++)

            a[i] = 111;

        MPI\_Send(a, 5, MPI\_INT, 7, 1, MPI\_COMM\_WORLD);

    }

    if (rank == 2)

    {

        for (int i = 0; i < 3; i++)

            b[i] = 222;

        MPI\_Send(b, 3, MPI\_INT, 7, 1, MPI\_COMM\_WORLD);

    }

    if (rank == 7)

    {

        MPI\_Probe(MPI\_ANY\_SOURCE, 1, MPI\_COMM\_WORLD, &status);

        if (status.MPI\_SOURCE == 1)

        {

            int message\_count;

            MPI\_Get\_count(&status, MPI\_INT, &message\_count);

            if (message\_count == 5)

            {

                MPI\_Recv(a, 5, MPI\_INT, 1, 1, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

            }

            else if (message\_count == 3)

            {

                MPI\_Recv(b, 3, MPI\_INT, 2, 1, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

            }

        }

        if (status.MPI\_SOURCE == 2)

        {

            int message\_count;

            MPI\_Get\_count(&status, MPI\_INT, &message\_count);

            if (message\_count == 3)

            {

                MPI\_Recv(b, 3, MPI\_INT, 2, 1, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

            }

            else if (message\_count == 5)

            {

                MPI\_Recv(a, 5, MPI\_INT, 1, 1, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

            }

        }

    }

    cout << rank << " - Jarayon : a = ";

    for (int i = 0; i < 5; i++)

        cout << a[i] << " ";

    cout << "   b = ";

    for (int i = 0; i < 3; i++)

        cout << b[i] << " ";

    cout << endl;

    MPI\_Finalize();

    return 0;

}

// NATIJA:



Bu holatda 2 ta massif jonatilyapti. Va ularning o’lchami 2 xil birinchisi 5 ta elementdan iborat ikkinchisi 3 ta elemntdan iborat. Agar biz malumotni qabul qilayotganda kattaroq joy ajratsak yani ikkalasiga ham 5 elementdan iborat massif elon qilsak bunda 2 – ementda 2 ta joy bo’sh qoladi agarda kam joy ajratsak xatolik beradi. Shuning uchun get\_count() bilan massif o’lchamlarini olyapti va shunga mos Probe bilan jarayon raqamini tekshirib joy ajratyapti. Bu dasturda xotiradan yutilyapti.

1. Blokirovkasiz xabar almashish mohiyati va uning qoʻllanilish sohalari.

Blokirovkasiz (asinxron) xabar almashish, MPI (Message Passing Interface) kutubxonasida amaliyotda ishlatiladigan qo'llanma xususiyatidir. Bu, xabar almashish operatsiyalarini boshqa amallar bajarilayotgan holda davom ettirishga imkon beradi. Blokirovkasiz xabar almashish funksiyalari bajarilayotgan xabarlar qabul qilinmagan bo'lganida ham dastur davom ettirib boradi. Blokirovkasiz xabar almashish, dastur davom etish imkoniyatini beradi, xabar almashish jarayoni tugaganini kuzatib boradi. Bu, parallel ishlovchilar yordamida xar doim xabar almashishni tekshirish va dasturni davom ettirishga imkon beradi. Blokirovkasiz xabar almashish, xabar almashish operatsiyasi tugab qolganida boshqa amallarni bajarishga imkon beradi. Bu, ishlovchilar orasida parallel amallarni yaxshiroq taqsimlashga va dasturni optimallashtirishga imkon beradi.

Quyidagi oddiy dastur, blokirovkasiz xabar almashishning mohiyatini va uning qo'llanilish sohalarini ko'rsatadi:

#include <iostream>

#include <mpi.h>

using namespace std;

int main(int argc, char \*\*argv)

{

    MPI\_Init(&argc, &argv);

    int size, rank, data;

    MPI\_Request request;

    MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

    if (rank == 0)

    {

        data = 777;

        MPI\_Isend(&data, 1, MPI\_INT, 1, 0, MPI\_COMM\_WORLD, &request);

        cout << " 0 - Jarayondan yuborildi: " << data << endl;

    }

    else if (rank == 1)

    {

        MPI\_Irecv(&data, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, &request);

        MPI\_Wait(&request, MPI\_STATUS\_IGNORE);

        cout << " 1 - Jarayonga qabul qilindi: " << data << endl;

        int flag = 0;

        MPI\_Test(&request, &flag, MPI\_STATUS\_IGNORE);

        if (flag)

        {

            cout << " Malumot qabul qilindi! " << endl;

        }

    }

    MPI\_Finalize();

    return 0;

}

// NATIJA:



1. Blokirovkasiz xabar almashish funksiyalari.

Blokirovkasiz xabar almashish funksiyalari MPI kutubxonasida tushunchalarga asoslangan amallardir. Bu funksiyalar, xabar almashish operatsiyalarini to'xtatmaydigan va dasturning boshqa qismlari bilan parallel amallar bajarishga imkon beradigan asosiy MPI funksiyalardir. Quyidagi funksiyalar Blokirovkasi xabar almashish uchun ishlatiladi:

**MPI\_Isend** - MPI kutubxonasida blokirovkasiz xabar jo'natish operatsiyasini bajarish uchun ishlatiladi.

int MPI\_Isend(const void \*buf, int count, MPI\_Datatype datatype, int dest, int tag, MPI\_Comm comm, MPI\_Request \*request);

* buf: Xabar jo'natiladigan ma'lumotlarni o'zlashtiruvchi joy.
* count: Jo'natilayotgan elementlar soni.
* datatype: Ma'lumot turi.
* dest: Xabar qabul qiluvchi ishlovchi raqami.
* tag: Xabar tegii.
* comm: Kommunikator.
* request: Xabar jarayonini identifikatsiyalash.

**MPI\_Irecv:** - MPI kutubxonasida blokirovkasiz xabar qabul qilish uchun ishlatiladi. Bu funksiya dastur davom etayotganda xabar qabul qilingandan so'ng natijaning o'zlashtirilayotganligini ta'minlaydi. Ushbu operatsiya dastur davom etgandan so'ng xabar qabul qilinganligini tekshirish uchun MPI\_Wait yoki MPI\_Test funksiyalari bilan ishlatiladi.

int MPI\_Irecv(void \*buf, int count, MPI\_Datatype datatype, int source, int tag, MPI\_Comm comm, MPI\_Request \*request);

* buf: Xabar qabul qilingan ma'lumotlarni o'zlashtiruvchi joy.
* count: Qabul qilinayotgan elementlar soni.
* datatype: Ma'lumot turi.
* source: Xabar jo'natuvchi ishlovchi raqami yoki MPI\_ANY\_SOURCE (har qanday ishlovchidan kelgan xabarni qabul qilish uchun).
* tag: Xabar tegii yoki MPI\_ANY\_TAG (har qanday tagli xabarni qabul qilish uchun).
* comm: Kommunikator.
* request: Xabar jarayonini identifikatsiyalash.

Wait funksiyalari

**MPI\_Wait** - MPI kutubxonasida blokirovkasiz xabar almashish operatsiyalari tugaganligini kutish uchun ishlatiladi. MPI\_Wait bu faqat bitta jarayonni kutadi.

int MPI\_Wait(MPI\_Request \*request, MPI\_Status \*status);

* request: Xabar jarayonini identifikatsiyalash.
* status: Natijaning holati.

**MPI\_Waitall** - Berilgan hamma jarayondagi xabar olinishini kutadi.

int MPI\_Waitall(int count, MPI\_Request array\_of\_requests[], MPI\_Status array\_of\_statuses[]);

**MPI\_Waitany** – Istalgan bitta jarayon yakunlanishini kutadi.

int MPI\_Waitany(int count, MPI\_Request array\_of\_requests[], int \*index, MPI\_Status \*status);

**MPI\_Waitsome** – Istalgan bir nechta jarayon yakunlanishini kutadi.

int MPI\_Waitsome(int incount, MPI\_Request array\_of\_requests[], int \*outcount, int array\_of\_indices[], MPI\_Status array\_of\_statuses[]);

Test funksiyalari

**MPI\_Test** - Vazifa: Berilgan so'rovnoma tugagan bo'lsa, flag o'zgaruvchisi true ga, aks holda false ga tenglanadi.

int MPI\_Test(MPI\_Request \*request, int \*flag, MPI\_Status \*status);

**MPI\_Testall** - Berilgan hamma so'rovnoma tugagan bo'lsa, flag o'zgaruvchisi true ga, aks holda false ga tenglanadi.

int MPI\_Testall(int count, MPI\_Request array\_of\_requests[], int \*flag, MPI\_Status array\_of\_statuses[]);

**MPI\_Testany** - Berilgan so'rovnoma ro'yxatida biror xabar almashish operatsiyasi tugagan bo'lsa, index o'zgaruvchisiga tugagan operatsiyaning indeksi, flag o'zgaruvchisiga true, va status o'zgaruvchisiga tugagan operatsiyaning holati qaytariladi.

int MPI\_Testany(int count, MPI\_Request array\_of\_requests[], int \*index, int \*flag, MPI\_Status \*status);

**MPI\_Testsome** - Berilgan so'rovnoma ro'yxatida bir nechta xabar almashish operatsiyalari tugagan bo'lsa, outcount o'zgaruvchisiga tugagan operatsiyalar soni, array\_of\_indices o'zgaruvchisiga tugagan operatsiyalar indekslari, va array\_of\_statuses o'zgaruvchisiga tugagan operatsiyalar holati qaytariladi.

int MPI\_Testsome(int incount, MPI\_Request array\_of\_requests[], int \*outcount, int array\_of\_indices[], MPI\_Status array\_of\_statuses[]);

1. Blokirovkasiz xabar almashish holatini tekshirish funksiyalari va ularni amalda qoʻllanilishi.

Blakirovkasiz (asinxron) xabar almashishda xabar qabul qilinganligini tekshirish uchun Test funksiyalaridan foydalaniladi. Bular MPI\_Test, MPI\_Testall, MPI\_Testany, MPI\_Testsome, funkdiyalaridan foydalaniladi.

Test funksiyalari

**MPI\_Test** - Vazifa: Berilgan so'rovnoma tugagan bo'lsa, flag o'zgaruvchisi true ga, aks holda false ga tenglanadi.

int MPI\_Test(MPI\_Request \*request, int \*flag, MPI\_Status \*status);

**MPI\_Testall** - Berilgan hamma so'rovnoma tugagan bo'lsa, flag o'zgaruvchisi true ga, aks holda false ga tenglanadi.

int MPI\_Testall(int count, MPI\_Request array\_of\_requests[], int \*flag, MPI\_Status array\_of\_statuses[]);

**MPI\_Testany** - Berilgan so'rovnoma ro'yxatida biror xabar almashish operatsiyasi tugagan bo'lsa, index o'zgaruvchisiga tugagan operatsiyaning indeksi, flag o'zgaruvchisiga true, va status o'zgaruvchisiga tugagan operatsiyaning holati qaytariladi.

int MPI\_Testany(int count, MPI\_Request array\_of\_requests[], int \*index, int \*flag, MPI\_Status \*status);

**MPI\_Testsome** - Berilgan so'rovnoma ro'yxatida bir nechta xabar almashish operatsiyalari tugagan bo'lsa, outcount o'zgaruvchisiga tugagan operatsiyalar soni, array\_of\_indices o'zgaruvchisiga tugagan operatsiyalar indekslari, va array\_of\_statuses o'zgaruvchisiga tugagan operatsiyalar holati qaytariladi.

int MPI\_Testsome(int incount, MPI\_Request array\_of\_requests[], int \*outcount, int array\_of\_indices[], MPI\_Status array\_of\_statuses[]);

Quyidagi misol blokirovkasiz xabar almashish operatsiyalari va MPI\_Test funksiyasi ishlatilgan dasturni ko'rsatadi:

#include <iostream>

#include <mpi.h>

using namespace std;

int main(int argc, char \*\*argv)

{

    int rank;

    int a = 0, flag = 0;

    MPI\_Status status;

    MPI\_Request request;

    MPI\_Init(&argc, &argv);

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

    if (rank == 2)

    {

        a = 77;

        MPI\_Isend(&a, 1, MPI\_INT, 7, 11, MPI\_COMM\_WORLD, &request);

    }

    if (rank == 7)

    {

        MPI\_Irecv(&a, 1, MPI\_INT, 2, 11, MPI\_COMM\_WORLD, &request);

        // Xabar qabul qilishni kutamiz

        MPI\_Wait(&request, &status);

        MPI\_Test(&request, &flag, &status);

        if (flag)

        {

            cout << " Xabar qabul qilindi! " << endl;

        }

    }

    cout << rank << " - Jarayon :  a = " << a << endl;

    MPI\_Finalize();

    return 0;

}

// NATIJA:



Bu dasturda 1 ta xabar yuborilyapti va bu xabar qabul qilinishini MPI\_Wait orqali kuttiryapti va MPI\_Test bilan qabul qilinganligini tekshiryapti.

1. Blokirovkali va blokirovkasiz xabar almashish oʻrtasidagi asosiy farqlar, afzallik va kamchiliklari.

**Blokirovkali Xabar Almashish**:

* **Farqlar:** Blokirovkali xabar almashishda xabarni yuboruvchi va qabul qiluvchi to'plamlar o'zlarining xabar almashish amallarini bajarib ketgandagina kutishadi. Xabar qabul qilingan vaqtgacha dastur to'xtaydi. Xabar yuboruvchi to'plamning (MPI\_Send funksiyasi yoki boshqa o'xshash funksiya) boshqa to'plamdan olingan xabarni qabul qilguncha kutishadi. Xabar almashish jarayonida "blocking" so'zi qo'llaniladi, chunki xabar almashish jarayoni to'xtagan paytda dastur to'xtab qoladi.
* **Afzalliklar:** Oson va aniq: Blokirovkali xabar almashish aniq va oson. Xabarlar bo'yicha sinxronizatsiya avtomatik ravishda amalga oshiriladi.
* **Kamchiliklar:** Dastur "blocking" qilib qoladi, ya'ni xabar almashish jarayonida kutishuchida dastur to'xtaydi.

**Blokirovkasiz Xabar Almashish:**

* Farqlar: Blokirovkasiz xabar almashishda xabarni yuboruvchi to'plam xabar almashishni boshlashdan so'ng dasturni to'xtatmasdan davom ettirishi mumkin. Bu jarayonning blokirovkasiz yoki asinxron (non-blocking or asynchronous) ekanligi bilan bog'liq. MPI\_Isend (blokirovkasiz yuborish) va MPI\_Irecv (blokirovkasiz qabul qilish) funksiyalari xabar almashishni asinxron ravishda boshlash uchun ishlatiladi.
* **Afzalliklar:** Qo'shimcha amallar olib borilishi mumkin va dasturni to'xtatmasdan davom ettirilishi mumkin. Xabar yuboruvchi va qabul qiluvchi to'plamlar to'xtashmasdan, dastur o'zini davom ettirishi mumkin.
* **Kamchiliklar:** Dastur joriy ishlayotgan xabar almashish jarayonida boshqa ishlarni bajarishi mumkin, ammo xabarni qabul qilib bo'lguncha aniqlash jarayoni qayta yuborish uchun kutilishi kerak.Xabar qabul qilingan bo'lsa, uni aniqlash uchun MPI\_Wait yoki boshqa blokirovka yordamchi funksiyalardan foydalanish kerak.

Blokirovkali xabar almashish dasturni to'xtatib qo'yadi va kutish uchun vaqt sarflashi mumkin.

Blokirovkasiz xabar almashish esa dastur davom etishiga imkoniyat tanlashi mumkin, ammo dastur uchun qo'shimcha tuzatishlar (masalan, kutish funksiyalari) kerak bo'ladi.

Asosiy tanlov dasturingizning talablari va xususiyatlari boyicha bog'liq. Blokirovkali xabar almashish bajarilishi oson va to'g'ri, lekin kamchiliklarga olib kelishi mumkin. Blokirovkasiz xabar almashish esa dasturning aniq vaqtda davom etish imkoniyatini beradi, lekin dasturda qo'shimcha kodlarni ishlatish kerak.

1. Jarayonlar oʻrtasida ikki tomonlama xabar almashish uchun ishlatiladigan funksiyalar.

Jarayonlar oʻrtasida ikki tomonlama xabar almashish uchun MPI\_Sendrecv va MPI\_Sendrecv\_replace

ishlatiladi.

**MPI\_Sendrecv** funksiyasi ikki tomonlama xabar almashishni bitta operatsiya orqali bajaradi.

**MPI\_Sendrecv\_replace** esa qabul qilingan xabarni saqlash uchun alohida o'zgaruvchi ishlatmasdan ikki tomonlama xabar almashishni bajaradi.

Bu funksiyalar MPI dasturlashda ikki tomonlama xabar almashishni osonlashtirish uchun yordam beradi. O'z navbatida, xabarlar turlari, xabar almashish qoidalari va dastur strukturalari bo'yicha detalarni o'rganish foydali bo'ladi.

**MPI\_Sendrecv:** - funksiyasi ikki tomonlama xabar almashishni bitta operatsiya orqali bajaradi. Bu funksiya MPI\_Send va MPI\_Recv ni birlashmasi hisoblanadi.

int MPI\_Sendrecv(const void \*sendbuf, int sendcount, MPI\_Datatype sendtype,int dest, int sendtag, void \*recvbuf, int recvcount,MPI\_Datatype recvtype, int source, int recvtag, MPI\_Comm comm, MPI\_Status \*status);

Parameterlar:

* sendbuf: Yuboriladigan xabarlar massivi.
* sendcount: Yuboriladigan elementlar soni.
* sendtype: Yuboriladigan elementlar turi.
* dest: Yuboriladigan tomon (process) raqami.
* sendtag: Yuboriladigan xabarlarning turi (tag).
* recvbuf: Qabul qilingan xabarlar massivi.
* recvcount: Qabul qilingan elementlar soni.
* recvtype: Qabul qilingan elementlar turi.
* source: Qabul qilingan tomon (process) raqami.
* recvtag: Qabul qilingan xabarlarning turi (tag).
* comm: Kommunikatsiya orqali aloqador to'plam (communicator).
* status: Operatsiyadagi status ma'lumotlarni saqlaydi.

**MPI\_Sendrecv\_replace -** funksiyasi ham ikki tomonlama xabar almashishni bajaradi, ammo qabul qilingan xabarni saqlash uchun yangi o'zgaruvchi zarur bo'lmaydi. U MPI\_Sendrecv bilan bir xil xizmatni ko'rsatadi, lekin qabul qilingan xabarni saqlash uchun alohida o'zgaruvchiga ega emas.

int MPI\_Sendrecv\_replace(void \*buf, int count, MPI\_Datatype datatype, int dest, int sendtag, int source, int recvtag, MPI\_Comm comm, MPI\_Status \*status);

* Parameterlar:
* buf: Xabarlar massivi.
* count: Elementlar soni.
* datatype: Elementlar turi.
* dest: Yuboriladigan tomon (process) raqami.
* sendtag: Yuboriladigan xabarlarning turi (tag).
* source: Qabul qilingan tomon (process) raqami.
* recvtag: Qabul qilingan xabarlarning turi (tag).
* comm: Kommunikatsiya orqali aloqador to'plam (communicator).
* status: Operatsiyadagi status ma'lumotlarni saqlaydi.

#include <iostream>

#include <mpi.h>

using namespace std;

int main(int argc, char \*\*argv)

{

    MPI\_Init(&argc, &argv);

    int rank, size;

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

    MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

    int send\_data;

    int recv\_data;

    if (rank == 0)

    {

        send\_data = 777;

        MPI\_Sendrecv(&send\_data, 1, MPI\_INT, 1, 0, &recv\_data, 1, MPI\_INT, 1, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

    }

    if (rank == 1)

    {

        MPI\_Sendrecv\_replace(&send\_data, 1, MPI\_INT, 0, 0, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

    }

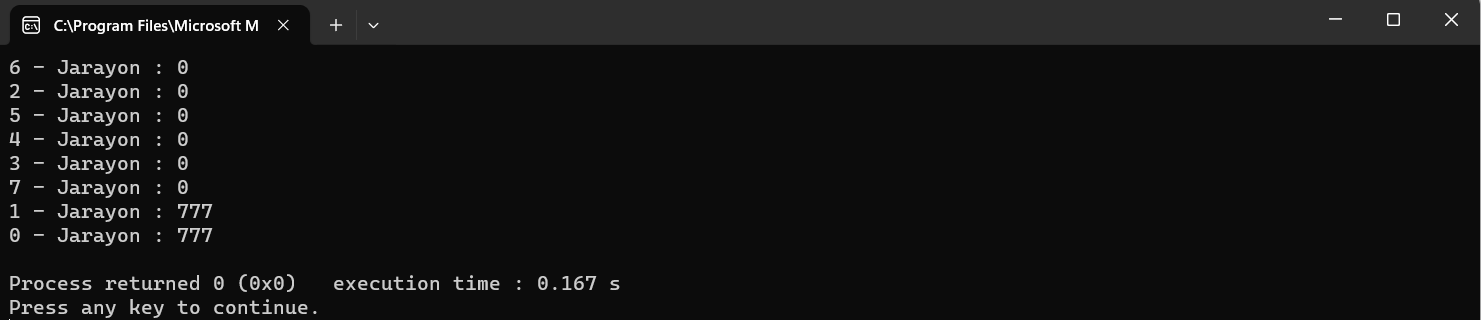
    cout << rank << " - Jarayon : " << send\_data << endl;

    MPI\_Finalize();

    return 0;

}

// NATIJA:



Bu dasturda, ikki MPI protsessi ishlaydi: protsess 0 va protsess 1. Dastur quyidagi jarayonlarni amalga oshiradi: rank == 0 (Process 0): bo’lganda send\_data o'zgaruvchisi 777 qiymatiga tenglanadi.

MPI\_Sendrecv funksiyasi orqali Process 1 ga send\_data xabarni yuboradi va Process 1 dan recv\_data o'zgaruvchisiga xabar qabul qiladi.

MPI\_Sendrecv funksiyasi Process 0 dan Process 1 ga xabar yuborish va u holda qabul qilish uchun ishlatiladi.

rank == 1 (Process 1): bo’lganda MPI\_Sendrecv\_replace funksiyasi orqali Process 0 dan send\_data xabarni qabul qilib, Process 0 ga yangi send\_data qiymatini yuboradi.

MPI\_Sendrecv\_replace funksiyasi Process 1 dan Process 0 ga xabar yuborish va u holda qabul qilish uchun ishlatiladi.

Natijada, dastur chiqish qatordan oldin har bir protsessning rank qiymatini va o'zgartirilgan send\_data qiymatini ekranga chiqaradi. Ikki protsess orasidagi xabar almashish MPI funksiyalari orqali amalga oshiriladi.

1. Jarayonlar oʻrtasida jamoaviy (kollektiv) xabar almashish.

amoaviy (kollektiv) xabar almashish, odam yoki guruh iste'molchilarining oʻrtasida xabar almashish va oʻzaro muloqot qilishini anglatadi. Bu turi xabar almashish jarayonlari bir guruh, jamoa yoki kollektiv isteʼmolchilarini qoʻllab-quvvatlash, aʼlo tadbirlarini tashkil etish yoki maʼlumot almashish maqsadida amalga oshirilishi mumkin.

Jamoaviy xabar almashishning muhim qismlaridan biri kommunikatsiya va birlikda ishlash imkoniyatini oshirishdir. Unda har bir a'zo oʻz fikrini bayon etish, maslahatlar berish va jamoa a'zolari bilan fikr almashish imkoniyati mavjud boʻladi.

Jamoaviy xabar almashishni amalga oshirish uchun quyidagi vositalardan foydalanish mumkin:

Elektron pochta: Guruh yoki jamoaga aloqalar va xabarlarni joʻnatish uchun elektron pochta tizimi.

Qoʻllaniladigan xabar almashish tizimlari: Telegram, Slack, Microsoft Teams kabi xabar almashish tizimlari, guruh a'zolari oʻrtasida tez, samarali va jamoaviy xabar almashish imkoniyatini taʼminlaydi.

Virtual konferensiyalar va videomuloqotlar: Zoom, Microsoft Teams, Google Meet kabi virtual platfomalarda jamoaviy muloqotlar oʻtkazish, tashkilot ichida birlikda ishlashni oshiradi.

Elektron hodisalar: Jamoaviy xabar almashish va birlikda ishlashni taʼminlash maqsadida veb-saytlar, bloglar, forumlar orqali elektron hodisalar tashkil etish.

Sotsial tarmoqlar: Sotsial tarmoqlar orqali jamoaviy aloqa oʻrnatish, jamoaga axborot, tadbir va yangiliklar joʻnatish mumkin.