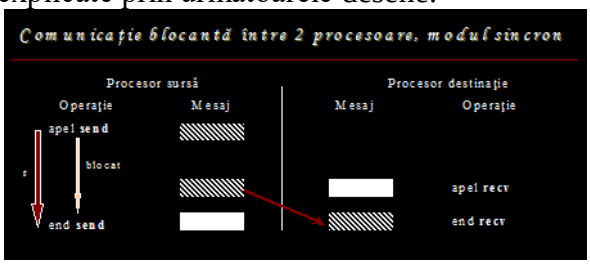
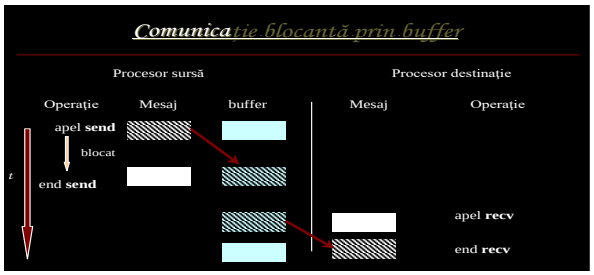
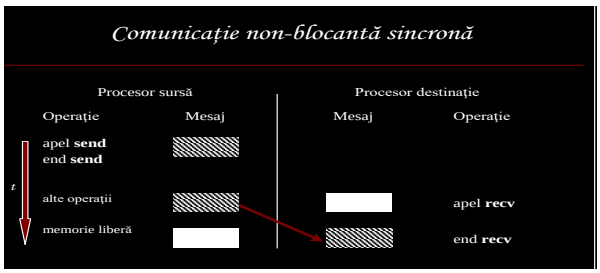
Lectia 3 COMUNICAREA DE TIP „PROCES-PROCES” (PUNCT LA PUNCT)

3.1 Operații de bază

Operațiile de bază pentru comunicarea punct la punct sunt **send** și **receive.** Modalitățile de transmitere a mesajelor pot fi explicate prin următoarele desene:







Considerăm mai întâi operația de transmitere de mesaje, în forma sa „uzuală”:

```
send(adresa,lungime,destinatie,tip)
```

unde

- **`adresa`** identifică începutul unei zone de memorie unde se află mesajul de transmis,
- **`lungime`** este lungimea în octeți a mesajului,
- **`destinatie`** este identificatorul procesului căruia i se trimite mesajul (uzual un număr întreg),
- **`tip(flag)`** este un întreg ne-negativ care restricționează recepția mesajului. Acest argument permite programatorului să rearanjeze mesajele în ordine, chiar dacă ele nu sosesc în secvența dorită. Acest set de parametri este un bun compromis între ceea ce utilizatorul dorește și ceea ce sistemul poate să ofere: transferul eficient al unei zone contigue de memorie de la un proces la altul. În particular, sistemul oferă mecanismele de păstrare în coadă a mesajelor, astfel că o operație de recepție **`recv(adresa,maxlung, sursa,tip,actlung)`** se execută cu succes doar dacă mesajul are tipul corect. Mesajele care nu corespund așteaptă în coadă. În cele mai multe sisteme, sursa este un argument de ieșire, care indică originea mesajului.

3.2 Funcţii MPI pentru transmiterea mesajelor

Funcția *MPI_Send*

În limbajul C această funcție are următorul prototip, valorile returnate reprezentând coduri de eroare:

```
int MPI_Send(void *buf,int count, MPI_Datatype datatype,int dest,int
    tag, MPI_Comm comm),
```

unde parametrii sunt:

IN **buf**	– adresa inițială a tamponului sursă;
IN **count**	– numărul de elemente (întreg ne-negativ);
IN **datatype**	– tipul fiecărui element;
IN **dest**	– numărul de ordine al destinatarului (întreg);
IN **tag**	– tipul mesajului (întreg);
IN **comm**	– comunicatorul implicat.

Astfel, procesul din mediul de comunicare **comm**, care execută funcția **MPI_Send**, expediază procesului **dest, count** date de tip **datatype** din memoria sa începând cu adresa **buf.** Acestor date li se atribuie eticheta **tag.**

Funcția *MPI_Recv*

Prototipul acestei funcții în limbajul C este

```
int MPI_Recv(void *buf,int count, MPI_Datatype datatype,int
    source,int tag,MPI_Comm comm, MPI_Status *status),
```

unde parametrii sunt:

OUT **buf**	– adresa inițială a tamponului destinatar;
IN **count**	– numărul de elemente (întreg ne-negativ);
IN **datatype**	– tipul fiecărui element;
IN **source**	– numărul de ordine al sursei (întreg);
IN **tag**	– tipul mesajului (întreg);
IN **comm**	– comunicatorul implicat;
OUT **status**	– starea, element (structură) ce indică sursa, tipul și contorul mesajului efectiv primit.

Astfel, procesul din mediul de comunicare **comm**, care execută funcția **MPI_Recv**, recepționează de la procesul **source, count** date de tip **datatype** care au eticheta **tag** și le salvează în memoria sa începând cu adresa **buf.**

Operația **send** este blocantă. Ea nu redă controlul apelantului până când mesajul nu a fost preluat din tamponul sursă, acesta din urmă putând fi reutilizat de transmițător. Mesajul poate fi copiat direct în tamponul destinatar (dacă se execută o operație **recv**) sau poate fi salvat într-un tampon temporar al sistemului. Memorarea temporară a mesajului decuplează operațiile **send** și **recv**, dar este consumatoare de resurse. Alegerea unei variante aparține implementatorului MPI. Și operația **recv** este blocantă: controlul este redat programului apelant doar când mesajul a fost recepționat.

Funcția *MPI_Sendrecv*

În situațiile în care se dorește realizarea unui schimb reciproc de date între procese, mai sigur este de a utiliza o funcție combinată **MPI_Sendrecv**.

Prototipul acestei funcții în limbajul C este

```
int MPI_Sendrecv(void *sendbuf, int sendcount, MPI_Datatype
    sendtype, int dest, int sendtag, void *recvbuf, int recvcount,
    MPI_Datatype recvtype, int source, MPI_Datatypea recvtag,
    MPI_Comm comm, MPI_Status *status),
```

unde parametrii sunt:

IN **sendbuf**	– adresa inițială a tamponului sursă;
IN **sendcount**	– numărul de elemente transmise (întreg ne-negativ);
IN **sendtype**	– tipul fiecărui element trimis;
IN **dest**	– numărul de ordine al destinatarului (întreg);
IN **sendtag**	– identificatorul mesajului trimis;

OUT **recvbuf**	– adresa inițială a tamponului destinatar;
IN **recvcount**	– numărul maximal de elemente primite;
IN **recvtype**	– tipul fiecărui element primit;
IN **source**	– numărul de ordine al sursei (întreg);
IN **recvtag**	– identificatorul mesajului primit;
IN **comm**	– comunicatorul implicat;
OUT **status**	– starea, element (structura) ce indică sursa, tipul și contorul mesajului efectiv primit.

Există și alte funcții MPI care permit programatorului să controleze modul de comunicare de tip proces-proces. În tabelul de mai jos prezentăm aceste funcții.

Funcțiile de transmitere a mesajelor de tip „proces-proces”

Modul de transmitere	**Cu blocare**	**Cu non-blocare**
Transmitere standart	MPI_Send	MPI_Isend
Transmitere sincron	MPI_Ssend	MPI_Issend
Transmitere prin tampon	MPI_Bsend	MPI_Ibsend
Transmitere coordonată	MPI_Rsend	MPI_Irsend
Recepționare mesaje	MPI_Recv	MPI_Irecv

Vom ilustra utilizarea funcțiilor **MPI_Send** și **MPI_Recv** prin următorul exemplu.

Exemplul 3.1.1 *Să se elaboreze un program MPI în limbajul C++ în care se realizează transmiterea mesajelor pe cerc începând cu procesul „încep”. Valoarea variabilei „încep” este inițializată de toate procesele și se află în diapazonul 0,…,size-1.*

Mai jos este prezentat codul programului în care se realizează condițiile enunțate în exemplul 3.1.1.

```
#include<stdio.h>
#include <iostream>
#include<mpi.h>
using namespace std;
int main(int argc,char *argv[])
{
int size,rank,t,namelen;
int incep=8;
char processor_name[MPI_MAX_PROCESSOR_NAME];
MPI_Status status;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&size);
MPI_Comm_rank(MPI_COMM_WORLD,
     &rank);
MPI_Get_processor_name(processor_name,
     &namelen);
if (rank ==0)
printf("\n=====REZULTATUL PROGRAMULUI '%s' \n",argv[0]);
if(rank==incep)
  {
MPI_Send(&rank,1,MPI_INT, (rank + 1) % size, 10, MPI_COMM_WORLD);
MPI_Recv(&t,1,MPI_INT, (rank+size-1) % size,10,MPI_COMM_WORLD,&status);
  }
```

```
else
   {
MPI_Recv(&t,1,MPI_INT, (rank+size-1)%size, 10,MPI_COMM_WORLD,&status);
MPI_Send(&rank,1,MPI_INT,(rank+1)%size,10,MPI_COMM_WORLD);
   }
printf("Procesul cu rancul %d al nodului '%s' a primit valoarea %d de la procesul cu rancul %d\n", rank,
      processor_name, t, t);
MPI_Finalize();
return 0;
}
```

Rezultatele posibile ale executării programului:

```
[Hancu_B_S@hpc Finale]$ /opt/openmpi/bin/mpiCC -o Exemplu_3_3_1.exe Exemplu_3_3_1.cpp[1]
[Hancu_B_S@hpc Finale]$ /opt/openmpi/bin/mpirun -n 9 -machinefile ~/nodes Exemplu_3_3_1.exe

=====REZULTATUL PROGRAMULUI ' Exemplu_3_3_1.exe'
Procesul cu rancul 0 al nodului 'compute-0-0.local' a primit valoarea 8 de la procesul cu rancul 8
Procesul cu rancul 1 al nodului 'compute-0-0.local' a primit valoarea 0 de la procesul cu rancul 0
Procesul cu rancul 2 al nodului 'compute-0-0.local' a primit valoarea 1 de la procesul cu rancul 1
Procesul cu rancul 3 al nodului 'compute-0-0.local' a primit valoarea 2 de la procesul cu rancul 2
Procesul cu rancul 4 al nodului 'compute-0-1.local' a primit valoarea 3 de la procesul cu rancul 3
Procesul cu rancul 8 al nodului 'compute-0-2.local' a primit valoarea 7 de la procesul cu rancul 7
Procesul cu rancul 5 al nodului 'compute-0-1.local' a primit valoarea 4 de la procesul cu rancul 4
Procesul cu rancul 6 al nodului 'compute-0-1.local' a primit valoarea 5 de la procesul cu rancul 5
Procesul cu rancul 7 al nodului 'compute-0-1.local' a primit valoarea 6 de la procesul cu rancul 6
[Hancu_B_S@hpc Finale]$
```

[1] Numele programului corespunde numelui exemplului din notele de curs Boris HÎNCU, Elena CALMÎŞ "MODELE DE PROGRAMARE PARALELĂ PE CLUSTERE. PARTEA I. PROGRAMARE MPI". Chisinau 2016.