Lectia 10. GENERAREA DINAMICĂ A PROCESELOR MPI

10.1 Modalități de generare dinamică a proceselor

În toate exemplele descrise mai sus s-a folosit un singur mod de generare a proceselor MPI. Și anume prin utilizarea comenzii **mpirun.** Acest mod de generare a proceselor se numește static.

MPI presupune existenţa unui mediu de execuție a proceselor, cu care interacțiunea este păstrată la un nivel scăzut pentru a evita compromiterea portabilității. Interacțiunea se limitează la următoarele aspecte:

- ✓ Un proces poate porni dinamic alte procese prin funcția **MPI_Comm_spawn** și **MPI_Comm_spawn_multiple.**
- ✓ Poate comunica printr-un argument **info** informații despre unde și cum sa pornească procesul.
- ✓ Un atribut **MPI_UNIVERSE_SIZE** al **MPI_COMM_WORLD** precizează câte procese pot rula în total, deci câte pot fi pornite dinamic, în plus faţă de cele în execuție.

Funcția MPI_Comm_spawn

Această funcție se utilizează pentru generarea unui număr de procese MPI, fiecare dintre acestea va executa același cod de program. Prototipul funcției în limbajul C++ este

```
int MPI_Comm_spawn(char *command, char *argv[], int maxprocs,
    MPI_Info info, int root, MPI_Comm comm, MPI_Comm *intercomm,int
    array_of_errcodes[])
```

unde

IN **command**	– specifică numele codului de program care va fi executat de procesele MPI generate;
IN **argv[]**	– conține argumentele transmise programului în forma unui tablou de șiruri de caractere;
IN **maxprocs**	– numărul de procese generate care vor executa programul MPI specificat de **command**;
IN **info**	– conține informații adiționale pentru mediul de execuție în forma unor perechi de șiruri de caractere (cheie, valoare);
IN **root**	– rankul procesului pentru care sunt descrise argumentele anterioare (**info** etc.);
IN **comm**	– intracomunicatorul pentru grupul de procese care conține procesul generator (de procese MPI);
OUT **intercomm**	– un intercomunicator pentru comunicare între părinți și fii;
OUT **array_of_ errcodes[]**	– conține codurile de erori.

Astfel funcția **MPI_Comm_spawn** întoarce un intercomunicator pentru comunicare între procese „părinţi" și procese „fii". Acesta conține procesele „părinte" în grupul local și procesele „fii" în grupul distant.

Funcția MPI_Comm_ get_parent

Intercomunicatorul poate fi obținut de procesele „fii” apelând această funcție. Prototipul funcției în limbajul C++ este

```
int MPI_Comm_get_parent(MPI_Comm *parent)
```

unde

OUT **parent** – specifică numele intercomunicatorului „părinte” ;

În cazul în care un proces a fost generat utilizând funcția **MPI_Comm_spawn** sau **MPI_Comm_spawn_multiple**, atunci funcția **MPI_Comm_get_parent** returnează intercomunicatorul părinte pentru procesul curent. În cazul în care procesul nu a fost generat, **MPI_Comm_get_parent** returnează valoarea **MPI_COMM_NULL**.

Vom ilustra utilizarea rutinelor MPI descrise mai sus prin următorul exemplu.

Exemplul 10.1 *Să se elaboreze un program MPI în limbajul C++ pentru generarea dinamică a proceselor care, la rândul lor, vor executa același cod de program pe nodurile* `compute-0-0,compute-0-1` *și* `compute-0-3`.

Mai jos este prezentat codul programului în limbajul C++ în care se realizează cele indicate în exemplul 10.1. Codul programului **Exemplu_3_7_1.cpp** în care se generează procesele.

```
/* manager */
#include<string.h>
#include<stdlib.h>
#include<stdio.h>
#include <mpi.h>
int main(int argc, char *argv[])
{
 int world_size, universe_size, *universe_sizep, flag, err[4], namelen, rank;
MPI_Comm everyone;
 char worker_program[100]="./HB_MPI_Spawn_Worker.exe";
 char processor_name[MPI_MAX_ PROCESSOR_NAME];
MPI_Info hostinfo;
char *host =(char*)"host";
MPI_Init(&argc, &argv);
MPI_Get_processor_name(processor_name,&namelen);
MPI_Comm_size(MPI_COMM_WORLD, &world_size);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
if (world_size != 1)  printf("Top heavy with management");
MPI_Attr_get(MPI_COMM_WORLD, MPI_UNIVERSE_SIZE,&universe_sizep, &flag);
if (!flag) {
      printf("This MPI does not support UNIVERSE_SIZE. How many n processes total?\n");
      scanf("%d", &universe_size);
         } else universe_size = *universe_sizep;
if (universe_size == 1)  printf("No room to start workers");
MPI_Info_create(&hostinfo);
MPI_Info_set(hostinfo, host, "compute-0-1, compute-0-0,compute-0-3" );
 universe_size=9;
 MPI_Comm_spawn(worker_program, MPI_ARGV_NULL, universe_size-1,
      hostinfo,0,MPI_COMM_WORLD, &everyone,err);
printf("===I am Manager ('%s'), run on the node '%s' with rank %d and generate %d proceses that run
      the program '%s' ===\n",argv[0],processor_name,rank,universe_size-1,worker_program);
```

```
MPI_Finalize();
return 0;
}
```

Codul programului **`HB_MPI_Spawn_Worker.cpp`** care va fi executat de procesele generate:

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
int size,size1,rank,namelen,t,incep=3;
char processor_name[MPI_MAX_PROCESSOR_NAME];
MPI_Comm parent;
MPI_Status status;
MPI_Init(&argc, &argv);
MPI_Comm_get_parent(&parent);
MPI_Get_processor_name(processor_name,&namelen);
if (parent == MPI_COMM_NULL)
printf("=== Intercomunicatorul parinte nu a fost creat!\n");
MPI_Comm_remote_size(parent, &size);
MPI_Comm_size(MPI_COMM_WORLD,
      &size1);
MPI_Comm_rank(MPI_COMM_WORLD,
      &rank);
if (size != 1) printf("Something's wrong with the parent");
printf("Module '%s'. Start on the processor rank %d of the node name '%s'of world_size  %d \n",argv[0],
      rank, processor_name, size);
MPI_Barrier(MPI_COMM_WORLD);
if(rank==incep)
{
MPI_Send(&rank,1,MPI_INT, (rank + 1) % size1, 10, MPI_COMM_WORLD);
MPI_Recv(&t,1,MPI_INT, (rank+size1-1) % size1,10,MPI_COMM_WORLD,&status);
}
else
{
MPI_Recv(&t,1,MPI_INT, (rank+size1-1)%size1, 10, MPI_COMM_WORLD, &status);
MPI_Send(&rank,1,MPI_INT,(rank+1)%size1,10,MPI_COMM_WORLD);
}
printf("proc num %d@%s rcvd=%d from %d\n",rank, processor_name, t, t);
MPI_Finalize();
return 0;
}
```

Rezultatele posibile ale executării programului:

```
[Hancu_B_S@hpc]$ /opt/openmpi/bin/mpiCC -o Exemplu _3_7_1.exe  Exemplu _3_7_1.cpp[1]
[Hancu_B_S@hpc]$ /opt/openmpi/bin/mpiCC -o HB_MPI_Spawn_Worker.exe
HB_MPI_Spawn_Worker.cpp
[Hancu_B_S@hpc]$ /opt/openmpi/bin/mpirun -n 1 -machinefile ~/nodes Exemplu _3_7_1.exe
```

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

```
===I am Manager (' Exemplu _3_7_1.exe'), run on the node 'compute-0-0.local' with rank 0 and ===I am Manager ('Exemplu_3_7_1.exe'), run on the node 'compute-0-0.local' with rank 0 and generate 8 proceses that run the program './HB_MPI_Spawn_Worker.exe' ===
Module './HB_MPI_Spawn_Worker.exe'. Start on the processor rank 2 of the node name 'compute-0-0.local'of world_size  1
Module './HB_MPI_Spawn_Worker.exe'. Start on the processor rank 5 of the node name 'compute-0-0.local'of world_size  1
Module './HB_MPI_Spawn_Worker.exe'. Start on the processor rank 0 of the node name 'compute-0-3.local'of world_size  1
Module './HB_MPI_Spawn_Worker.exe'. Start on the processor rank 3 of the node name 'compute-0-3.local'of world_size  1
Module './HB_MPI_Spawn_Worker.exe'. Start on the processor rank 6 of the node name 'compute-0-3.local'of world_size  1
Module './HB_MPI_Spawn_Worker.exe'. Start on the processor rank 1 of the node name 'compute-0-1.local'of world_size  1
Module './HB_MPI_Spawn_Worker.exe'. Start on the processor rank 4 of the node name 'compute-0-1.local'of world_size  1
Module './HB_MPI_Spawn_Worker.exe'. Start on the processor rank 7 of the node name 'compute-0-1.local'of world_size  1
proc num 4@compute-0-1.local rcvd=3 from 3
proc num 5@compute-0-0.local rcvd=4 from 4
proc num 6@compute-0-3.local rcvd=5 from 5
proc num 7@compute-0-1.local rcvd=6 from 6
proc num 0@compute-0-3.local rcvd=7 from 7
proc num 1@compute-0-1.local rcvd=0 from 0
proc num 2@compute-0-0.local rcvd=1 from 1
proc num 3@compute-0-3.local rcvd=2 from 2
```

Funcția MPI_Comm_spawn_multiple

Această funcție se utilizează pentru generarea unui număr de procese MPI, fiecare dintre acestea pot executa coduri diferite de program. Prototipul funcției în limbajul C++ este

```
int MPI_Comm_spawn_multiple(int count, char *array_of_commands[],
    char **array_of_argv[], int array_of_maxprocs[], MPI_Info
    array_of_info[], int root, MPI_Comm comm, MPI_Comm *intercomm,
    int array_of_errcodes[])
```

unde

IN **count**	– numărul de programe care urmează a fi executate (este relevant numai pentru procesul **root**);
IN **array_of_commands[ ]**	– vector de lungimea **count** în care elementul **i** specifică numele codului de program care vor fi executate de procesele MPI generate (este relevant numai pentru procesul **root**);
IN **array_of_argv[]**	– vector de lungimea **count** în care se specifică argumentele transmise programelor, în forma unui tablou de șiruri de caractere

	(este relevant numai pentru procesul **root**);
IN **array_of_ maxprocs[]**	– vector de lungimea **count** în care elementul **i** specifică numărul maximal de procese care vor executa programul indicat de vectorul **array_of_commands[]** (este relevant numai pentru procesul **root**);
IN **array_of_info[]**	– vector de lungimea **count** în care elementul **i** conține informații adiționale pentru mediul de execuție în forma unor perechi de șiruri de caractere (cheie, valoare) (este relevant numai pentru procesul **root**) ;
IN **root**	– rankul procesului pentru care sunt descrise argumentele anterioare;
IN **comm**	– intracomunicatorul pentru grupul de procese care conține procesul generator (de procese MPI) ;
OUT **intercomm**	– un intercomunicator pentru comunicare între părinți și fii.
OUT **array_of_ errcodes[]**	– conține codurile de erori

Vom ilustra utilizarea rutinei **MPI_Comm_spawn_multiple** prin următorul exemplu.

Exemplul 10.2 *Să se elaboreze un program MPI în limbajul C++ pentru generarea dinamică a proceselor, care la rândul lor, vor executa coduri diferite de program în următorul mod*

- *Programul HB_MPI_Spawn_Worker1.exe se va executa pe 8 procesoare ale nodurilor compute-0-0 şi compute-0-1.*
- *Programul HB_MPI_Spawn_Worker2.exe se va executa pe 4 procesoare ale nodului compute-0-3.*
- *Programul HB_MPI_Spawn_Worker3.exe se va executa pe 3 procesoare ale nodului compute-0-11.*

Mai jos este prezentat codul programului în limbajul C++ în care se realizează cele indicate în exemplul 10.2.

Codul programului *exemplul_3_7_2.cpp* în care se generează procesele:

```
#include <mpi.h>
#include <stdio.h>
#include<stdlib.h>
int main(int argc, char *argv[])
{
    int world_size, *universe_sizep, flag, err[4],namelen,rank;
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    MPI_Comm everyone;          /* intercommunicator */
    const int count = 3;
    int universe_size[count] = {8, 4, 3};
    char *worker_program[count] = {"./HB_MPI_Spawn_Worker1.exe",
    "./HB_MPI_Spawn_Worker2.exe", "./HB_MPI_Spawn_Worker3.exe"};
```

```
      char **args[count];
      char *argv0[] = {NULL};
      char *argv1[] = {NULL};
      char *argv2[] = {NULL};
      args[0] = argv0;
      args[1] = argv1;
      args[2] = argv2;
      MPI_Info hostinfo[count];
char *host =(char*)"host";
MPI_Init(&argc, &argv);
MPI_Get_processor_name(processor_name,&namelen);
MPI_Comm_size(MPI_COMM_WORLD, &world_size);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
for(int i = 0; i < count; i++)
MPI_Info_create(&hostinfo[i]);
MPI_Info_set(hostinfo[0], host,"compute-0-0, compute-0-1" );
MPI_Info_set(hostinfo[1], host,"compute-0-3" );
MPI_Info_set(hostinfo[2], host,"compute-0-11" );
if (world_size != 1)  printf("Top heavy with management");
      printf("===I am Manager ('%s'), run on the node '%s' with rank %d and generate the following
      proceses: \n",argv[0],processor_name,rank);
for(int i = 0; i < count; i++)
{
printf("%d proceses run the module '%s'\n", universe_size[i],worker_program[i]);
}
printf("===\n");
MPI_Comm_spawn_multiple(count, worker_program, args, universe_size, hostinfo,
      0,MPI_COMM_SELF, &everyone,err);
MPI_Finalize();
return 0;
}
```

Codul programului *HB_MPI_Spawn_Worker1.cpp* care va fi executat de procesele generate. Programele *HB_MPI_Spawn_Worker2.cpp*, *HB_MPI_Spawn_Worker3.cpp* sunt similare.

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
int size,size1,rank,namelen,t,incep=3;
char processor_name[MPI_MAX_PROCESSOR_NAME];
MPI_Comm parent;
MPI_Status status;
MPI_Init(&argc, &argv);
MPI_Comm_get_parent(&parent);
MPI_Get_processor_name(processor_name,&namelen);
if (parent == MPI_COMM_NULL)
printf("=== Intercomunicatorul parinte nu a fost creat!\n");
MPI_Comm_remote_size(parent, &size);
```

```
MPI_Comm_size(MPI_COMM_WORLD,
      &size1);
MPI_Comm_rank(MPI_COMM_WORLD,
      &rank);
if (size != 1) printf("Something's wrong with the parent");
printf("Module '%s'. Start on the processor rank %d of the node name '%s'of world_size  %d \n",argv[0],
      rank, processor_name, size);
MPI_Barrier(MPI_COMM_WORLD);
if(rank==incep)
{
MPI_Send(&rank,1,MPI_INT, (rank + 1) % size1, 10, MPI_COMM_WORLD);
MPI_Recv(&t,1,MPI_INT, (rank+size1-1) % size1,10,MPI_COMM_WORLD,&status);
}
else
{
MPI_Recv(&t,1,MPI_INT, (rank+size1-1)%size1, 10, MPI_COMM_WORLD, &status);
MPI_Send(&rank,1,MPI_INT,(rank+1)%size1,10,MPI_COMM_WORLD);
}
printf("proc num %d@%s rcvd=%d from %d\n",rank, processor_name, t, t);
MPI_Finalize();
return 0;
}
```

Rezultatele posibile ale executării programului:

```
 [Hancu_B_S@hpc]$ /opt/openmpi/bin/mpiCC -o Exemplu _3_7_2.exe  Exemplu _3_7_2.cpp[2]
[Hancu_B_S@hpc]$ /opt/openmpi/bin/mpiCC -o HB_MPI_Spawn_Worker1.exe
HB_MPI_Spawn_Worker1.cpp
[Hancu_B_S@hpc]$ /opt/openmpi/bin/mpiCC  -o HB_MPI_Spawn_Worker2.exe
HB_MPI_Spawn_Worker2.cpp
[Hancu_B_S@hpc]$ /opt/openmpi/bin/mpiCC  -o HB_MPI_Spawn_Worker3.exe
HB_MPI_Spawn_Worker3.cpp
[Hancu_B_S@hpc]$ /opt/openmpi/bin/mpirun -n 1 -machinefile ~/nodes Exemplu _3_7_2.exe
[Hancu_B_S@hpc Notate_Exemple]$ /opt/openmpi/bin/mpirun -n 1  -machinefile ~/nodes6
Exemplu_3_7_2.exe
===I am Manager ('Exemplu_3_7_2.exe'), run on the node 'compute-0-0.local' with rank 0 and generate
the following proceses:
8 proceses run the module './HB_MPI_Spawn_Worker1.exe'
4 proceses run the module './HB_MPI_Spawn_Worker2.exe'
3 proceses run the module './HB_MPI_Spawn_Worker3.exe'
===
Module './HB_MPI_Spawn_Worker2.exe'. Start  on the processor rank 8 of the node name 'compute-0-
3.local'of world_size  1
Module './HB_MPI_Spawn_Worker1.exe'. Start  on the processor rank 0 of the node name 'compute-0-
1.local'of world_size  1
Module './HB_MPI_Spawn_Worker1.exe'. Start  on the processor rank 2 of the node name 'compute-0-
1.local'of world_size  1
```

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

```
Module './HB_MPI_Spawn_Worker1.exe'. Start  on the processor rank 4 of the node name 'compute-0-1.local'of world_size  1
Module './HB_MPI_Spawn_Worker1.exe'. Start  on the processor rank 6 of the node name 'compute-0-1.local'of world_size  1
Module './HB_MPI_Spawn_Worker1.exe'. Start  on the processor rank 1 of the node name 'compute-0-0.local'of world_size  1
Module './HB_MPI_Spawn_Worker1.exe'. Start  on the processor rank 3 of the node name 'compute-0-0.local'of world_size  1
Module './HB_MPI_Spawn_Worker1.exe'. Start  on the processor rank 5 of the node name 'compute-0-0.local'of world_size  1
Module './HB_MPI_Spawn_Worker3.exe'. Start  on the processor rank 14 of the node name 'compute-0-11.local'of world_size  1
Module './HB_MPI_Spawn_Worker3.exe'. Start  on the processor rank 12 of the node name 'compute-0-11.local'of world_size  1
Module './HB_MPI_Spawn_Worker3.exe'. Start  on the processor rank 13 of the node name 'compute-0-11.local'of world_size  1
Module './HB_MPI_Spawn_Worker2.exe'. Start  on the processor rank 9 of the node name 'compute-0-3.local'of world_size  1
Module './HB_MPI_Spawn_Worker2.exe'. Start  on the processor rank 10 of the node name 'compute-0-3.local'of world_size  1
Module './HB_MPI_Spawn_Worker2.exe'. Start  on the processor rank 11 of the node name 'compute-0-3.local'of world_size  1
Module './HB_MPI_Spawn_Worker1.exe'. Start  on the processor rank 7 of the node name 'compute-0-0.local'of world_size  1
==Module './HB_MPI_Spawn_Worker1.exe'. proc num 4 @compute-0-1.local rcvd=3 from 3
==Module './HB_MPI_Spawn_Worker1.exe'. proc num 6 @compute-0-1.local rcvd=5 from 5
==Module './HB_MPI_Spawn_Worker1.exe'. proc num 5 @compute-0-0.local rcvd=4 from 4
==Module './HB_MPI_Spawn_Worker1.exe'. proc num 7 @compute-0-0.local rcvd=6 from 6
==Module './HB_MPI_Spawn_Worker3.exe'. proc num 14 @compute-0-11.local rcvd=13 from 13
==Module './HB_MPI_Spawn_Worker3.exe'. proc num 12 @compute-0-11.local rcvd=11 from 11
==Module './HB_MPI_Spawn_Worker2.exe'. proc num 8 @compute-0-3.local rcvd=7 from 7
==Module './HB_MPI_Spawn_Worker1.exe'. proc num 0 @compute-0-1.local rcvd=14 from 14
==Module './HB_MPI_Spawn_Worker2.exe'. proc num 9 @compute-0-3.local rcvd=8 from 8
==Module './HB_MPI_Spawn_Worker3.exe'. proc num 13 @compute-0-11.local rcvd=12 from 12
==Module './HB_MPI_Spawn_Worker2.exe'. proc num 10 @compute-0-3.local rcvd=9 from 9
==Module './HB_MPI_Spawn_Worker1.exe'. proc num 1 @compute-0-0.local rcvd=0 from 0
==Module './HB_MPI_Spawn_Worker2.exe'. proc num 11 @compute-0-3.local rcvd=10 from 10
==Module './HB_MPI_Spawn_Worker1.exe'. proc num 3 @compute-0-0.local rcvd=2 from 2
==Module './HB_MPI_Spawn_Worker1.exe'. proc num 2 @compute-0-1.local rcvd=1 from 1
```

10.2 Comunicarea între procesele generate dinamic

MPI permite stabilirea unor „canale” de comunicare între procese, chiar dacă acestea nu împart un comunicator comun. Aceasta este utilă în următoarele situații:

- ✓Două părți ale unei aplicații, pornite independent, trebuie să comunice între ele.
- ✓Un instrument de vizualizare vrea să se atașeze la un proces în execuție.
- ✓Un server vrea să accepte conexiuni de la mai mulți clienți; serverul și clienții pot fi programe paralele.

Mai jos vom prezenta modalitățile de comunicare între procesul părinte și procesele fii generate.

Vom exemplifica comunicarea între procesul părinte şi procesele fiu prin intermediul intercomunicatorului (adică ei fac parte din grupuri diferite de procese) în cele ce urmează.

Exemplul 10.3 *Să se elaboreze un program MPI în limbajul C++ pentru generarea dinamică a proceselor, astfel încât fiecare proces fiu va trimite rankul său procesului părinte utilizând un mediu de comunicare de tip intercomunicator. Procesele generate se vor executa pe nodurile* compute-0-1, compute-0-0 şi compute-0-3.

Mai jos este prezentat codul programului în limbajul C++ în care se realizează cele indicate în exemplul 10.3.

Codul programului e*xemplu_3_7_3.cpp* în care se generează procesele.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mpi.h>
int main(int argc, char** argv)
{
int rank, size, namelen, version, subversion, universe_size;
MPI_Comm family_comm;
char processor_name[MPI_MAX_PROCESSOR_NAME],
worker_program[100];
int rank_from_child,ich;
MPI_Info hostinfo;
MPI_Status status;
MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Get_processor_name(processor_name,&namelen);
MPI_Get_version(&version,&subversion);
printf("I'm manager %d of %d on %s running MPI %d.%d\n", rank, size, processor_ name, version,
    subversion);
if (size != 1) printf("Error: Only one manager process should be running, but %d were started.\n", size);
universe_size = 12;
strcpy(worker_program,"./HB_Woker_Comunication.exe");
MPI_Info_create(&hostinfo);
MPI_Info_set(hostinfo, "host","compute-0-1, compute-0-0,compute-0-3" );
printf("Spawning %d worker processes running %s\n", universe_size-1, worker_program);
MPI_Comm_spawn(worker_program, MPI_ARGV_NULL,universe_size-1,
    hostinfo,0,MPI_COMM_SELF,&family_comm, MPI_ERRCODES_IGNORE);
for(ich=0;ich<(universe_size-1);ich++)
{
MPI_Recv(&rank_from_child,1,MPI_INT,ich,0,family_comm,MPI_STATUS_IGNORE);
printf("Received rank %d from child %d \n",rank_from_child,ich);
}
MPI_Bcast(&rank,1,MPI_INT,MPI_ROOT, family_comm);
MPI_Comm_disconnect(&family_comm);
MPI_Finalize();
return 0;
}
```

Codul programului *HB_Woker_Comunication.cpp* care va fi executat de procesele generate.

```
#include <mpi.h>
#include <stdio.h>
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
int main(int argc, char** argv)
{
int rank, size, namelen, version, subversion, psize;
int parent_rank;
MPI_Comm parent;
char processor_name[MPI_MAX_PROCESSOR_NAME];
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD,
      &rank);
MPI_Comm_size(MPI_COMM_WORLD,
      &size);
MPI_Get_processor_name(processor_name,&namelen);
MPI_Get_version(&version,&subversion);
printf("I'm worker %d of %d on %s running MPI %d.%d\n", rank, size, processor_name, version,
      subversion);
MPI_Comm_get_parent(&parent);
if (parent == MPI_COMM_NULL) { printf("Error: no parent process found!\n");
exit(1);
}
MPI_Comm_remote_size(parent,&psize);
if
(psize!=1)
{
printf("Error: number of parents (%d) should be 1.\n", psize);
exit(2);
}
/* comunicae cu procesul parinte */
 Int sendrank=rank;
printf("Worker %d:Success!\n", rank);
MPI_Send(&rank,1,MPI_INT,0,0,parent);
MPI_Bcast(&parent_rank,1,MPI_INT,0,
      parent);
printf("For Woker %d value of rank received from parent is %d \n", rank, parent_rank);
MPI_Comm_disconnect(&parent);
MPI_Finalize();
return 0;
}
```

Rezultatele posibile ale executării programului:

```
[Hancu_B_S@hpc]$ /opt/openmpi/bin/mpiCC -o Exemplu _3_7_3.exe Exemplu _3_7_3.cpp
[Hancu_B_S@hpc]$ /opt/openmpi/bin/mpiCC -o HB_Spawn_Comunication.exe
HB_Spawn_Comunication.cpp
[Hancu_B_S@hpc]$ /opt/openmpi/bin/mpirun -n 1 -machinefile ~/nodes Exemplu _3_7_3.exe
I'm manager 0 of 1 on compute-0-0.local running MPI 2.1
```

```
Spawning 11 worker processes running ./HB_Woker_Comunication.exe
I'm worker 5 of 11 on compute-0-0.local running MPI 2.1
Worker 5:Success!
I'm worker 0 of 11 on compute-0-3.local running MPI 2.1
Worker 0:Success!
I'm worker 8 of 11 on compute-0-0.local running MPI 2.1
Worker 8:Success!
I'm worker 1 of 11 on compute-0-1.local running MPI 2.1
Worker 1:Success!
I'm worker 3 of 11 on compute-0-3.local running MPI 2.1
Worker 3:Success!
I'm worker 6 of 11 on compute-0-3.local running MPI 2.1
Worker 6:Success!
Received rank 0 from child 0
Received rank 1 from child 1
Received rank 2 from child 2
I'm worker 4 of 11 on compute-0-1.local running MPI 2.1
Worker 4:Success!
I'm worker 7 of 11 on compute-0-1.local running MPI 2.1
Worker 7:Success!
I'm worker 9 of 11 on compute-0-3.local running MPI 2.1
Worker 9:Success!
I'm worker 2 of 11 on compute-0-0.local running MPI 2.1
Worker 2:Success!
I'm worker 10 of 11 on compute-0-1.local running MPI 2.1
Worker 10:Success!
Received rank 3 from child 3
Received rank 4 from child 4
Received rank 5 from child 5
Received rank 6 from child 6
Received rank 7 from child 7
Received rank 8 from child 8
Received rank 9 from child 9
Received rank 10 from child 10
For Woker 5 value of rank received from parent is 0
For Woker 9 value of rank received from parent is 0
For Woker 8 value of rank received from parent is 0
For Woker 2 value of rank received from parent is 0
For Woker 0 value of rank received from parent is 0
For Woker 3 value of rank received from parent is 0
For Woker 6 value of rank received from parent is 0
For Woker 10 value of rank received from parent is 0
For Woker 1 value of rank received from parent is 0
For Woker 4 value of rank received from parent is 0
For Woker 7 value of rank received from parent is 0
[Hancu_B_S@hpc]$
```

Pentru realizarea comunicării între procesul „părinte” și procesele „fii” prin intermediul uni mediu de comunicare de tip intracomunicator trebuie utilizată funcția **MPI_Intercomm_merge.** Prototipul funcției în limbajul C++ este

```
int MPI_Intercomm_merge(MPI_Comm intercomm, int high,MPI_Comm
    *newintracomm)
```

unde

IN **intercomm**	– numele intercomunicatorului;
IN **high**	– o variabilă logică care indică modul de reuniune a grupelor de procese;
OUT **newintracomm**	– numele intracomunicatorului.

Vom ilustra comunicarea între procesul părinte și procesele fii prin intermediul intracomunicatorului (adică ei fac parte din același grup de procese) prin următorul exemplu.

Exemplul 10.4 *Să se elaboreze un program MPI în limbajul C++ pentru generarea dinamică a proceselor, astfel încât procesul „fiu” va trimite rankul său procesului „părinte” utilizând un mediu de comunicare de tip intracomunicator. Procesele generate se vor executa pe nodurile* compute-0-1, compute-0-0 şi compute-0-3.

Mai jos este prezentat codul programului în limbajul C++ în care se realizează cele indicate în exemplul 10.4.

Codul programului *exemplu_3_7_4.cpp* în care se generează procesele.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mpi.h>
int main(int argc, char** argv)
{
int rank, size, namelen, version, subversion, universe_size;
int globalrank,sumrank;
MPI_Comm family_comm,allcomm;
char processor_name[MPI_MAX_PROCESSOR_NAME],
worker_program[100];
int rank_from_child,ich;
MPI_Status status;
 MPI_Info hostinfo;
MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Get_processor_name(processor_name,&namelen);
MPI_Get_version(&version,&subversion);
printf("I'm manager %d of %d on %s running MPI %d.%d\n", rank, size, processor_name, version,
    subversion);
if (size != 1) printf("Error: Only one manager process should be running, but %d were started.\n", size);
universe_size = 12;
MPI_Info_create(&hostinfo);
MPI_Info_set(hostinfo, "host", "compute-0-1,compute-0-0,compute-0-3" );
strcpy(worker_program,"./HB_Woker_Comunication_V1.exe");
printf("Spawning %d worker processes running %s\n", universe_size-1, worker_program);
MPI_Comm_spawn(worker_program, MPI_ARGV_NULL,universe_size-1,
    hostinfo,0,MPI_COMM_SELF,&family_comm, MPI_ERRCODES_IGNORE);
```

```
MPI_Intercomm_merge(family_comm,1, &allcomm);
MPI_Comm_rank(allcomm, &globalrank);
printf("manager: global rank is %d,rank is %d \n",globalrank,rank);
MPI_Allreduce(&globalrank,&sumrank,1, MPI_INT,MPI_SUM,allcomm);
printf("sumrank after allreduce on process %d is %d \n", rank,sumrank);
MPI_Comm_disconnect(&family_comm);
MPI_Finalize();
return 0;
}
```

Codul programului *HB_Woker_Comunication_V1.cpp* care va fi executat de procesele generate.

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
int main(int argc, char** argv)
{
int rank, size, namelen, version, subversion, psize;
int parent_rank;
int globalrank,sumrank;
MPI_Comm parent,allcom;
char processor_name[MPI_MAX_PROCESSOR_
      NAME];
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD,
      &rank);
MPI_Comm_size(MPI_COMM_WORLD,
      &size);
MPI_Get_processor_name(processor_name,&namelen);
MPI_Get_version(&version,&subversion);
printf("I'm worker %d of %d on %s running MPI %d.%d\n", rank, size, processor_name, version,
      subversion);
MPI_Comm_get_parent(&parent);
if (parent == MPI_COMM_NULL)
{
printf("Error: no parent process found!\n");
exit(1);
}
MPI_Comm_remote_size(parent,&psize);
if
(psize!=1)
{
printf("Error: number of parents (%d) should be 1.\n", psize);
exit(2);
}
MPI_Intercomm_merge(parent,1,&allcom);
MPI_Comm_rank(allcom,      &globalrank);
 printf("worker: globalrank is %d,rank is %d \n",globalrank, rank);
MPI_Allreduce(&globalrank,&sumrank,1, MPI_INT,MPI_SUM,allcom);
```

```
 printf("sumrank after allreduce on  process  %d  is  %d  \n", rank,sumrank);
MPI_Comm_disconnect(&parent);
MPI_Finalize();
return 0;
}
```

Rezultatele posibile ale executării programului:

```
[Hancu_B_S@hpc]$ /opt/openmpi/bin/mpiCC -o Exemplu _3_7_4.exe Exemplu _3_7_4.cpp
[Hancu_B_S@hpc]$ /opt/openmpi/bin/mpiCC -o HB_Spawn_Comunication_V1.exe
HB_Spawn_Comunication_V1.cpp
[Hancu_B_S@hpc]$ /opt/openmpi/bin/mpirun -n 1 -machinefile ~/nodes Exemplu _3_7_4.exe
I'm manager 0 of 1 on compute-0-0.local running MPI 2.1
Spawning 11 worker processes running ./HB_Woker_Comunication_V1.exe
I'm worker 0 of 11 on compute-0-3.local running MPI 2.1
I'm worker 3 of 11 on compute-0-3.local running MPI 2.1
I'm worker 6 of 11 on compute-0-3.local running MPI 2.1
I'm worker 9 of 11 on compute-0-3.local running MPI 2.1
I'm worker 8 of 11 on compute-0-0.local running MPI 2.1
I'm worker 1 of 11 on compute-0-1.local running MPI 2.1
I'm worker 2 of 11 on compute-0-0.local running MPI 2.1
I'm worker 4 of 11 on compute-0-1.local running MPI 2.1
I'm worker 5 of 11 on compute-0-0.local running MPI 2.1
I'm worker 7 of 11 on compute-0-1.local running MPI 2.1
I'm worker 10 of 11 on compute-0-1.local running MPI 2.1
manager: global rank is 0,rank is 0
worker: globalrank is 9,rank is 8
worker: globalrank is 3,rank is 2
worker: globalrank is 6,rank is 5
worker: globalrank is 2,rank is 1
worker: globalrank is 5,rank is 4
worker: globalrank is 8,rank is 7
worker: globalrank is 11,rank is 10
worker: globalrank is 10,rank is 9
worker: globalrank is 1,rank is 0
worker: globalrank is 4,rank is 3
worker: globalrank is 7,rank is 6
sumrank after allreduce on  process  6  is  66
sumrank after allreduce on  process  1  is  66
sumrank after allreduce on  process  10  is  66
sumrank after allreduce on  process  2  is  66
sumrank after allreduce on  process  8  is  66
sumrank after allreduce on  process  5  is  66
sumrank after allreduce on  process  9  is  66
sumrank after  allreduce  on  process  0  is  66
sumrank after allreduce on  process  0  is  66
sumrank after allreduce on  process  3  is  66
sumrank after allreduce on  process  7  is  66
sumrank after allreduce on  process  4  is  66
[Hancu_B_S@hpc Finale]$
```