Паралелно и дистрибуирано процесирање

Лабораториска вежба 1

Петар Атанасовски - 216052

Програмите се напишани во програмскиот јазик c

1. **Креирајте програма со четири процеси. Да се изминат сите процеси и да се испише нивниот ранк.**

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

int main(int argc, char** argv) {
   int rank, size;

   MPI_Init(&argc, &argv);

   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   MPI_Comm_size(MPI_COMM_WORLD, &size);

   printf("Process %d out of %d\n", rank, size);

   MPI_Finalize();

   return 0;
}
```

2. **Да се напише програма во која ќе постојат два процеси. Првиот ќе испраќа порака на вториот, а штом вториот ја прими, испраќа повратна порака на првиот.**

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

int main(int argc, char** argv) {
   int rank;
   MPI_Status status;
   char message[100];

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

   if (rank == 0) {
      strcpy(message, "Hello, Process 1!");
```

```
    MPI_Send(message, strlen(message)+1, MPI_CHAR, 1, 0, MPI_COMM_WORLD);
    printf("Process 0 sent message: %s\n", message);

    MPI_Recv(message, sizeof(message), MPI_CHAR, 1, 0, MPI_COMM_WORLD, &status);
    printf("Process 0 received message: %s\n", message);
  }
  else if (rank == 1) {

    MPI_Recv(message, sizeof(message), MPI_CHAR, 0, 0, MPI_COMM_WORLD, &status);
    printf("Process 1 received message: %s\n", message);

    strcpy(message, "Hi, Process 0!");
    MPI_Send(message, strlen(message)+1, MPI_CHAR, 0, 0, MPI_COMM_WORLD);
    printf("Process 1 sent message: %s\n", message);
  }

  MPI_Finalize();

  return 0;
}
```

3. **Да се напише програма во која ќе постојат три процеси. Првиот праќа порака на вториот додека вториот чека, а штом ќе ја прими испраќа порака на третиот којшто чека. Штом и третиот ја добие пораката, истата ја испраќа на првиот, а пораката се испишува на екран. Ваквиот циклус треба да се повторува 10 пати.**

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

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

  MPI_Init(NULL, NULL);
  int world_size;
  MPI_Comm_size(MPI_COMM_WORLD, &world_size);

  int processor_rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &processor_rank);

  const int BUFFER_SIZE = 128;
  char *message = (char *)malloc(BUFFER_SIZE * sizeof(char));

  for (int iter = 0; iter <= 10; iter++) {
    if (processor_rank == 0) {
      if (iter != 0) {
```

```
            MPI_Recv(message, BUFFER_SIZE, MPI_CHAR, 2, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
        }
        snprintf(message, BUFFER_SIZE, "Message %02d", iter);
        MPI_Send(message, BUFFER_SIZE, MPI_CHAR, 1, 0, MPI_COMM_WORLD);
      } else if (processor_rank == 1) {
        MPI_Recv(message, BUFFER_SIZE, MPI_CHAR, 0, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
        MPI_Send(message, BUFFER_SIZE, MPI_CHAR, 2, 0, MPI_COMM_WORLD);
      } else if (processor_rank == 2) {
        MPI_Recv(message, BUFFER_SIZE, MPI_CHAR, 1, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
        MPI_Send(message, BUFFER_SIZE, MPI_CHAR, 0, 0, MPI_COMM_WORLD);
      } else {
        fprintf(stderr, "The maximum number of processors for this task is 3\n");
      }
      printf("[P%d]\t%s\n", processor_rank, message);
   }

   free(message);

   MPI_Finalize();
   return 0;
}
```

4. **Нека постојат четири процеси. Иницијализирајте низа со должина делива со 4. Првиот процес нека ја испрати првата четвртина од низата на четвртиот процес, вториот втората четвртина на четвртиот процес и третиот третата четвртина на четвртиот процес. Четвртиот процес нека го испише збирот на сите елементи од низата.**

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

int main(int argc, char **argv)
{
   int NUMBER_OF_ELEMENTS_IN_ARRAY = 12;

   MPI_Init(NULL, NULL);

   int world_size;
   MPI_Comm_size(MPI_COMM_WORLD, &world_size);

   int processor_rank;
   MPI_Comm_rank(MPI_COMM_WORLD, &processor_rank);

   int array[12] = {11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22};
```

```
   int start = processor_rank * (NUMBER_OF_ELEMENTS_IN_ARRAY / 4);
   int end = start + (NUMBER_OF_ELEMENTS_IN_ARRAY / 4);
   int send_buffer[NUMBER_OF_ELEMENTS_IN_ARRAY / 4];

   int j = 0;
   for (int i = start; i < end; i++)
   {
      send_buffer[j] = array[i];
      j++;
   }

   MPI_Send(send_buffer, NUMBER_OF_ELEMENTS_IN_ARRAY / 4, MPI_INT, 3, 0,
MPI_COMM_WORLD);

   if (processor_rank == (world_size - 1))
   {
      int total_sum = 0;

      for (int rFrom = 0; rFrom <= (world_size - 1); rFrom++)
      {
         int receive_buffer[NUMBER_OF_ELEMENTS_IN_ARRAY / 4];
         MPI_Recv(receive_buffer, NUMBER_OF_ELEMENTS_IN_ARRAY / 4, MPI_INT, rFrom, 0,
MPI_COMM_WORLD, MPI_STATUS_IGNORE);

         for (int i = 0; i < NUMBER_OF_ELEMENTS_IN_ARRAY / 4; i++)
         {
            total_sum += receive_buffer[i];
         }
      }
      printf("Total sum: %d\n", total_sum);
   }

   MPI_Finalize();
   return 0;
}
```