МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Системы параллельной обработки данных»

Тема: СОЗДАНИЕ МАСШТАБИРУЕМЫХ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ

Студентка гр. 5303 _____ Допира В.Е.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург

2019

Формулировка задания

Задание:

1. Написать масштабируемую программу, моделирующую буфер типа FIFO ("первый пришел – первый вышел") на N сообщений, где N – число запущенных параллельных процессов. Таким образом, каждый процесс должен моделировать один регистр (ячейку памяти) буфера FIFO, хранящую одно сообщение. Программа должна моделировать управляющие сигналы буфера FIFO: буфер пуст и буфер полон.
2. Откомпилировать и запустить программу на одном, двух … N процессорах.
3. Предложить методику тестирования программы. Используя управляющие сигналы буфера FIFO (см. п.1), рассмотреть несколько режимов работы моделирующей программы: запись информации в пустой буфер до его заполнения (операция чтения не выполняется), чтение информации из полностью заполненного буфера (операция записи не выполняется), одновременная запись и чтение информации из буфера.

Описание алгоритма с использованием аппарата Сетей Петри

P_0 - начальное состояние. Нулевой процесс предлагает выбрать команду для работы с очередью буфера FIFO: добавить элемент, получить элемент и завершить выполнение программы. Передает следующему процессу команду и введенное значение (если операция push) для выполнения. Осуществляется переход t_1 из P_0. Аналогично выполняется выполнение других введенных команд и переход к состояниям P_1, P_2, P_3. Так как доступна также операция получения элемента из буфера, срабатывают переходы в обратную сторону. Программа работает, пока пользователь не выберет завершать выполнение, и нулевой процесс также подаст сигнал.

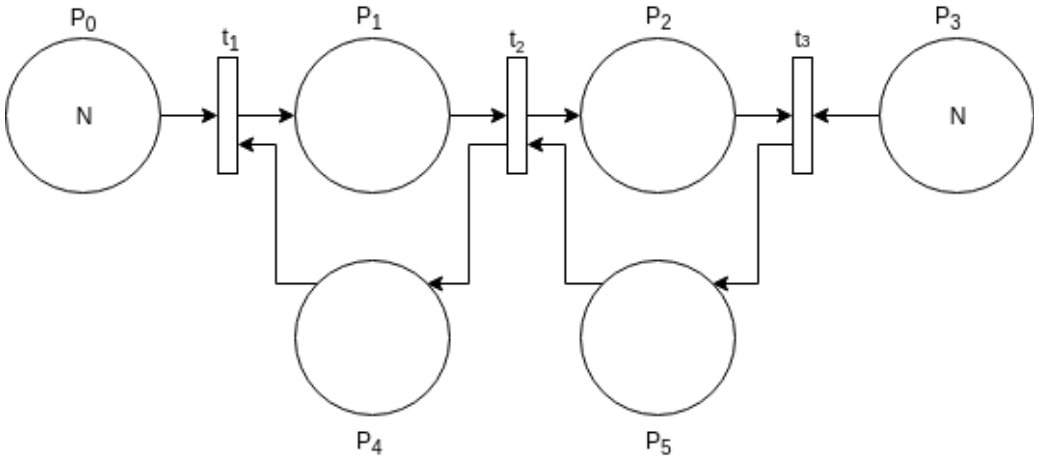

Рисунок 1 — Сеть Петри

Результаты работы программы на 1,2 …. N процессорах

```
$ mpirun 4.x -np 4
Please, select command
 Enter 1 to push element
 Enter 2 to pop element
 Enter 3 to close program
1
123
123 has been pushed
1
3456
3456 has been pushed
1
890
890 has been pushed
1
7
Queue is full
2
123 has been popped
2
3456 has been popped
2
890 has been popped
2
Fail! Queue is empty.
3
```

Вывод

В ходе выполнения лабораторной работы была написана масштабируемая программа с использованием MPI, моделирующую буфер типа FIFO.

Листинг программы

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

int main(int argc, char **argv) {
   int ProcNum, ProcRank;
   MPI_Status Status;
   MPI_Request request;

   //Tags:
   int tag_push = 0;
   int tag_element_pushed = 1;
   int tag_pop = 2;
   int tag_element_poped = 3;
   int tag_full = 4;
   int tag_empty = 5;
   int tag_end = 6;
   int tag_wait_to_push = 7;
   int tag_wait_to_pop = 8;
   int tag_not_wait_to_push = 9;
   int tag_not_wait_to_pop = 10;

   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
   MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);

   int NOTHING;

   if (ProcRank == 0)
   {
      int Size = 0;
      printf("Please, select command \n Enter 1 to push element\n
Enter 2 to pop element\n Enter 3 to close program\n");

      for (;;) {
         int command;
         scanf("%i", &command);

         if (command == 1)
         {
            int Element;
            scanf("%i", &Element);

            MPI_Send(&Element, 1, MPI_INT, 1, tag_push,
MPI_COMM_WORLD);
            MPI_Recv(&NOTHING, 1, MPI_INT, MPI_ANY_SOURCE,
MPI_ANY_TAG, MPI_COMM_WORLD, &Status);
```

```
            if (Status.MPI_TAG == tag_element_pushed)
            {
               Size++;
               printf("%i has been pushed\n", Element);
               // printf("Size of Queue: %i\n", Size);
            }
            else
            {
               printf("Queue is full\n");
            }

         }
         else if (command == 2)
         {
            int RecvElement;
            MPI_Send(&NOTHING, 1, MPI_INT, ProcNum - 1, tag_pop,
MPI_COMM_WORLD);
            MPI_Recv(&RecvElement, 1, MPI_INT, MPI_ANY_SOURCE,
MPI_ANY_TAG, MPI_COMM_WORLD, &Status);

            if (Status.MPI_TAG == tag_element_poped)
            {
               Size--;
               printf("%i has been popped\n", RecvElement);
               // printf("Size of Queue: %i\n", Size);
            }
            else
            {
               printf("Fail! Queue is empty.\n");
            }

         }
         else if (command == 3)
         {
            for (int i = 1; i < ProcNum; i++)
            {
               MPI_Send(&NOTHING, 1, MPI_INT, i, tag_end,
MPI_COMM_WORLD);
            }
            MPI_Finalize();
            return 0;
         }
         else
         {
            printf("Wrong input! Try again\n");
         }

      }
   }
```

```
   else
   {
      int containsElement = false;
      int is_wait_to_push;
      int is_wait_to_pop = false;

      if (ProcRank == ProcNum - 1)
      {
         is_wait_to_push = true;
      }
      else
      {
         is_wait_to_push = false;
      }

      int Element;
      int RecvElement;

      for (;;)
      {
         MPI_Recv(&RecvElement, 1, MPI_INT, MPI_ANY_SOURCE,
MPI_ANY_TAG, MPI_COMM_WORLD, &Status);

         if (Status.MPI_TAG == tag_push)
         {
            if (containsElement)
            {
               MPI_Send(&NOTHING, 1, MPI_INT, 0, tag_full,
MPI_COMM_WORLD);
            }

            else if (is_wait_to_push)
            {
               Element = RecvElement;
               containsElement = true;
               is_wait_to_push = false;
               is_wait_to_pop = true;

               if (ProcRank == 1) {
                  MPI_Send(&Element, 1, MPI_INT, 0,
tag_element_pushed, MPI_COMM_WORLD);
               }
               else
               {
                  MPI_Send(&Element, 1, MPI_INT, ProcRank - 1,
tag_wait_to_push, MPI_COMM_WORLD);
               }

               if (ProcRank != ProcNum - 1)
               {
```

```
                MPI_Send(&NOTHING, 1, MPI_INT, ProcRank + 1,
tag_not_wait_to_pop, MPI_COMM_WORLD);
             }

          }

          else
          {
             MPI_Send(&RecvElement, 1, MPI_INT, ProcRank + 1,
tag_push, MPI_COMM_WORLD);
          }

       }
       else if (Status.MPI_TAG == tag_wait_to_push)
       {
          if (Status.MPI_TAG == tag_wait_to_push)
          {
             is_wait_to_push = true;
          }
          MPI_Send(&RecvElement, 1, MPI_INT, ProcRank - 1,
tag_element_pushed, MPI_COMM_WORLD);
       }
       else if (Status.MPI_TAG == tag_not_wait_to_push)
       {
          is_wait_to_push = false;
       }
       else if (Status.MPI_TAG == tag_element_pushed)
       {
          MPI_Send(&Element, 1, MPI_INT, ProcRank - 1,
tag_element_pushed, MPI_COMM_WORLD);
       }
       else if (Status.MPI_TAG == tag_pop)
       {
          if (!containsElement)
          {
             MPI_Send(&NOTHING, 1, MPI_INT, 0, tag_empty,
MPI_COMM_WORLD);
          }
          else if (is_wait_to_pop)
          {
             containsElement = false;
             is_wait_to_push = true;
             is_wait_to_pop = false;
             if(ProcRank == ProcNum - 1)
             {
                MPI_Send(&Element, 1, MPI_INT, 0,
tag_element_poped, MPI_COMM_WORLD);
             }
             else
             {
```

```
                  MPI_Send(&Element, 1, MPI_INT, ProcRank + 1,
tag_wait_to_pop, MPI_COMM_WORLD);
               }

               if (ProcRank != 1)
               {
                  MPI_Send(&NOTHING, 1, MPI_INT, ProcRank - 1,
tag_not_wait_to_push, MPI_COMM_WORLD);
               }

            }
            else
            {
               MPI_Send(&NOTHING, 1, MPI_INT, ProcRank - 1,
tag_pop, MPI_COMM_WORLD);
            }
         }

         else if (Status.MPI_TAG == tag_wait_to_pop)
         {
            is_wait_to_pop = true;
            MPI_Send(&Element, 1, MPI_INT, (ProcRank + 1) %
ProcNum, tag_element_poped, MPI_COMM_WORLD);
            Element = RecvElement;
         }

         else if (Status.MPI_TAG == tag_not_wait_to_pop)
         {
            is_wait_to_pop = false;
         }
         else if (Status.MPI_TAG == tag_element_poped)
         {
            MPI_Send(&Element, 1, MPI_INT, (ProcRank + 1) %
ProcNum, tag_element_poped, MPI_COMM_WORLD);
            Element = RecvElement;
         }
         else
         {
            MPI_Finalize();
            return 0;
         }
      }
   }
   return 0;
}
```