Московский государственный университет имени М.В.Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра алгоритмических языков

Оуян Лэйло

Разработка компонентов электронного задачника по параллельному программированию в системе Unix

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Научный руководитель:
к.ф.-м.н., доцент
М. Э. Абрамян

Москва, 2023

Аннотация

Разработка компонентов электронного задачника
по параллельному программированию в системе Unix

Оуян Лэйло

В роботе описываются дополнительные компоненты электронного задачника Unix Taskbook [1], позволяющие адаптировать для него набор учебных заданий по параллельному программированию на базе технологии MPI-2, ранее реализованный в электронном задачнике Programming Taskbook for MPI-2 [2].

В ходе адаптации создана базовая динамическая библиотека, содержащая алгоритмы генерации всех учебных заданий по параллельному программированию, ранее разработанные для задачника Programming Taskbook. Кроме того, подготовлен специальный модуль, подключаемый к параллельному приложению и позволяющий каждому процессу считывать исходные данные, подготовленные задачником, и пересылать результаты на проверку.

Для каждой группы учебных заданий разработана динамическая библиотека, которая загружается из ядра задачника Unix Taskbook, использует базовую динамическую библиотеку для генерации всех необходимых данных, связанных с выполняемым заданием, управляет компиляцией и запуском учебной программы в параллельном режиме и обеспечивает проверку правильности полученных результатов.

Содержание

1 Введение

Традиционным способом создания практических заданий в области IT-дисциплин является разработка программ, связанных с изучаемыми технологиями, библиотеками и алгоритмами. Использование специализированных программных средств - электронных задачников [3] может значительно упростить и ускорить процесс решения и проверки задач как студентами, так и преподавателями.

Примером электронного задачника по программированию является Programming Taskbook. Хотя уже существуют группы задач по параллельному программированию в задачнике Programming Taskbook, в настоящее время он работает только на системах Windows. Поэтому, чтобы создать версию, которая будет работать под Unix-системами, была создана unixTaskbook [1]. Разработка unixTaskbook основана на принципы, лежащие в основе архитектуры задачника Programming Taskbook [4]. Затем поверх unixTaskbook можно легко перенести группы задач из задачника Programming Taskbook.

Задачник Unix Taskbook реализован на языке C++ для Unix и включает ядро, обеспечивающее его основную функциональность, и набор интерфейсов, которые позволяют нам легко расширить целевую группу Unix Taskbook.Эта диссертация использует этот набор интерфейсов для расширения динамической библиотеки групп задач по параллельной программированию для Unix Taskbook.

В дополнение к этим динамическим библиотекам, также был разработан специальный модуль, подключаемый к параллельному приложению и позволяющий каждому процессу считывать исходные данные, подготовленные задачником, и пересылать результаты на проверку. Прежде всего работа дается подробное описание особенностей архитектур задачников Programming Taskbook и Unix Taskbook. Во-вторых посвящен дополнительный модуль ut1.cpp, подключаемый к учебной программе. И потом, описывает реализацию набора динамических библиотек, подключаемых к задачнику Unix Taskbook. Наконец, показано практическое применение данного набора динамических библиотек.

2 Постановка задачи

Разработайте динамические библиотеки для пяти групп задач параллельного программирования и сделайте их загружаемыми в Unix Taskbook.

Для этого:

- Изучить архитектуру электронного задачника по программированию Programming Taskbook и Unix Taskbook;
- Разработка набора инструментов, обеспечивающих удобный интерфейс к динамической библиотеке каждой группы задач, позволяет динамической библиотеке управлять генерацией данных задачи, печатью информации о задаче, проверкой результатов выполнения задачи и другими функциями;
- Разработка специального модуля, подключаемого к параллельному приложению и позволяющий каждому процессу считывать исходные данные, подготовленные задачником, и пересылать результаты на проверку;
- Изучить, как запускать и компилировать программы MPI;
- Разработка пяти динамических библиотек для групп задач параллельного программирования на основе соответствующих характеристик задач MPI;

3 Особенности архитектур задачников Programming Taskbook и Unix Taskbook

Электронный задачник по программированию Unix Taskbook, разработанный Ли Шэнюем в рамках выполнения выпускной работы в 2022 году, использует ту же архитектуру, что и задачник Programming Taskbook. Она основана на применении динамических библиотек и позволяет расширять набор заданий без перекомпиляции ядра задачника. Каждая группа заданий реализуется в виде отдельной динамической библиотеки со стандартным набором функций, которые вызываются из ядра на различных этапах выполнения задания (основными этапами является этап инициализации, при котором генерируются исходные данные, передаваемые учебной программе, и этап итоговой проверки, на котором результаты, полученные программой, сравниваются с контрольными данными). Кроме того, в динамической библиотеке можно определить способ отображения на экране всей информации, связанной с заданием. Подобная расширяемая архитектура дает возможность разрабатывать задачи самого разного содержания, в том числе связанные с обработкой файлов и каталогов (которые автоматически создаются при инициализации задания) и с многопоточным программированием, что было также продемонстрировано в выпускной работе. Близость архитектур задачников Unix Taskbook и Programming Taskbook и их гибкость дает основания предполагать, что перенос уже имеющихся групп задач из задачника Programming Taskbook в задачник Unix Taskbook окажется не слишком сложным. Все задачи, входящие в задачник Programming Taskbook, подготовлены на основе конструктора учебных заданий TaskMaker, содержащего набор функций для генерации формулировок заданий, исходных и контрольных данных, настройки их отображения на экране, задания дополнительных свойств конкретной задачи (например, количества необходимых тестовых испытаний учебной программы). Конструктор TaskMaker реализован для нескольких языков (Pascal, C++, C#, PascalABC.NET), однако базовый набор из 1100 задач реализован на языке Pascal в среде Delphi (в которой было первоначально реализовано и ядро задачника Programming Taskbook). В то же время, задачник Unix Taskbook реализован на языке C++; именно это позволяет легко адаптировать его к различным операционным системам на базе Unix. Поэтому необходимо разработать механизм, который дал возможность переносить задания из базового набора (а также расширений) задачника

Programming Taskbook в задачник Unix Taskbook без переписывания кода, связанного с генерацией заданий.

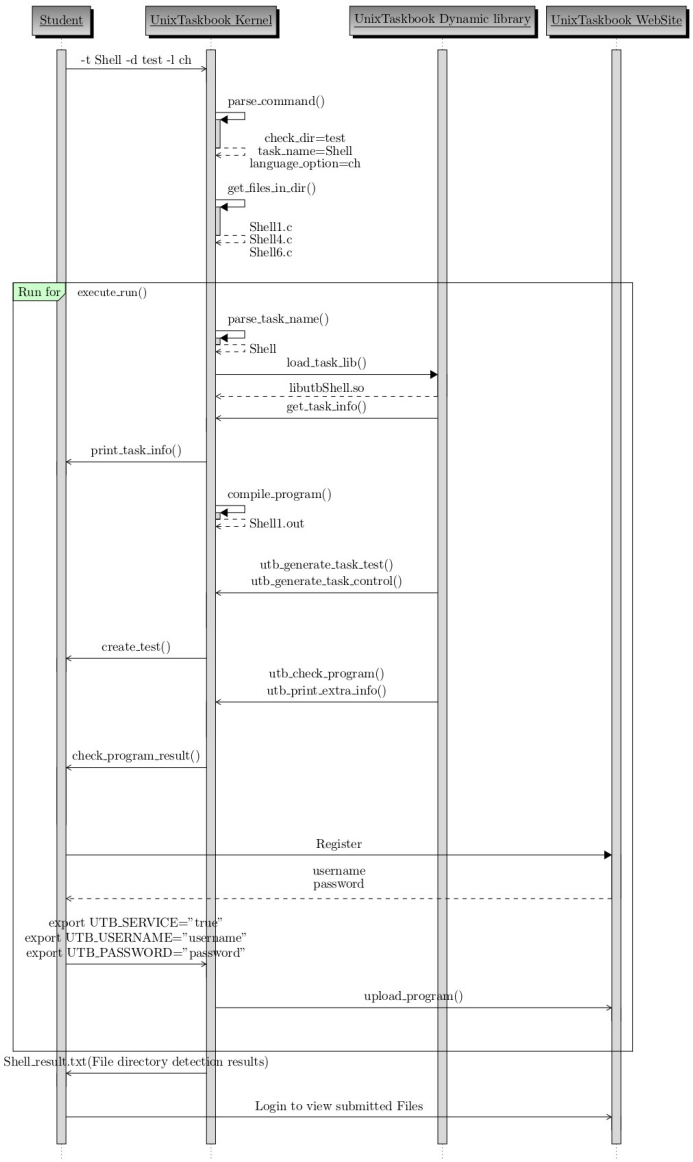


Рис. 1: архитектур Unix Taskbook [5]

4 Дополнительный модуль ut1.cpp, подключаемый к учебной программе

4.1 Схема использования данных, связанных с заданием, в учебной программе: дополнительный модуль ut1.cpp

Таким образом, при использовании описанной схемы выполнения заданий, импортированных из задачника Programming Taskbook, задачник Unix Taskbook должен выполнять следующую последовательность действий:

На следующем этапе реализации проекта переноса базового набора заданий задачника Programming Taskbook в задачник Unix Taskbook было необходимо разработать схему использования данных, подготовленных при инициализации задания, в учебной программе, выполняющей это задание. Задачник Programming Taskbook содержит встроенные средства для ввода исходных данных и вывода результатов; эти средства импортируются из ядра задачника и позволяют учебной программе получать исходные данные и передавать результаты задачнику на проверку. В задачнике Unix Taskbook подобных встроенных средств нет; это, в частности, означает, что при его использовании нет необходимости в подключении особых модулей к учебной программе. Например, можно реализовать группу заданий, в которой все исходные данные передаются в виде параметров командной строки, а результаты сохраняются в некотором файле. В этом случае учебная программа может разрабатываться как независимое приложение, которое можно запускать отдельно от задачника Unix Taskbook, явно указывая необходимые параметры командной строки и анализируя файлы, полученные в результате выполнения программы. Однако при ее запуске под управлением задачника появляются дополнительные возможности: задачник сам формирует набор параметров и сам проверяет правильность полученных файлов. Такая схема была использована при разработке групп заданий, связанных с различными темами курса по операционным системам. Аналогичную схему можно реализовать и для заданий, перенесенных из задачника Programming Taskbook. Однако при этом всё же требуется подключать к учебной программе дополнительный набор функций, которые позволяют при запуске программы под управлением задачника получать исходные, сгенерированные задачником, и передавать полученные результаты ему на проверку. Если же программа запуще-

на как отдельное приложение, то эти же функции позволяют ввести исходные данные с клавиатуры и вывести полученные результаты на экран. В набор дополнительных функций включены также функции отладочного вывода, которые упрощают вывод отладочных данных на экран как при запуске учебной программы под управлением задачника, так и при ее независимом запуске. Заметим, что данный набор функций оказывается особенно удобным для учебных программ, разрабатываемых на языке C, так как стандартные функции ввода-вывода в этом языке являются достаточно сложными. Признаком того, что учебная программа запущена под управлением задачника Unix Taskbook, является наличие в рабочем каталоге особого файла ut1inf.dat с основными характеристиками задания. При его наличии функции ввода получают информацию из еще одного дополнительного файла (одного или нескольких — в случае выполнения задания по параллельному программированию), а функции вывода и отладочного вывода выводят результаты не на экран, а в специальные файлы, которые в дальнейшем анализируются задачником. Кроме того, в особый файл также записывается информация об ошибках, которые можно обнаружить непосредственно при работе учебной программы (это ошибки, связанные с операциями ввода: ввод недостаточного или избыточного числа исходных данных или попытка ввода данных неверного типа). Заметим, что при обнаружении какой-либо ошибки учебная программа немедленно завершается. Окончательная проверка правильности решения выполняется задачником на заключительном этапе, когда анализируются полученные результаты, проверяется их количество, тип и соответствие контрольным значениям. В текущей реализации дополнительный файл ut1.h включает набор функций, позволяющий выполнять ввод, вывод и отладочный вывод на языках C и C++:

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <string.h>
4  #include <stdbool.h>
5
6  void ShowB(bool b);
7  void ShowN(int n);
8  void ShowD(double d);
9  void ShowC(char c);
10 void ShowS(const char *s);
11
12 void Show(const char *cmt);
```

```
13 void ShowW(const wchar_t *cmt);
14
15 void ShowLineB(bool b);
16 void ShowLineN(int n);
17 void ShowLineD(double d);
18 void ShowLineC(char c);
19 void ShowLineS(const char *s);
20
21 void ShowLine(const char *cmt);
22 void ShowLineW(const wchar_t *cmt);
23
24 void PutB(bool a);
25 void PutN(int a);
26 void PutD(double a);
27 void PutC(char a);
28 void PutS(const char *a);
29
30 void GetB(bool *a);
31 void GetN(int *a);
32 void GetD(double *a);
33 void GetC(char *a);
34 void GetS(char *a);
35
36 void SetPrecision(int n);
37 void SetWidth(int n);
```

Он соответствует набору, реализованному в задачнике Programming Taskbook версии 4.23 для языка C. На базе этого набора функций несложно реализовать более гибкий набор средств ввода-вывода для языка C++ (подобно тому, как это сделано в задачнике Programming Taskbook). Более того, поскольку исходный набор данных и другая исходная информация считывается из обычных текстовых файлов, а результаты и другие данные, связанные с выполнением задания (сообщения об ошибках и отладочные данные), также записываются в текстовые файлы, несложно разработать аналогичный набор функций и для других языков. В качестве иллюстрации того, как реализованы функции, описанные в файле ut1.h, приведем текст вспомогательной функции put_, используемой во всех функциях вывода, и пример ее использования в функции для вывода целых чисел PutN:

```
void put_(char id, double n, char c, const char *s)
{
  if (psize == -1)
    init_();
  if (psize == 0)
  {
    switch (id)
    {
    case 'b':
      ShowB((int)n);
      break;
    case 'i':
      ShowN((int)n);
      break;
    case 'r':
      ShowD(n);
      break;
    case 'c':
      ShowC(c);
      break;
    case 's':
      ShowS(s);
      break;
    }
  }
  else
  {
    FILE *f = fopen(outdat, "a");
    fprintf(f, "%c\n", id);
    switch (id)
    {
    case 'b':
      fprintf(f, "%d\n", (int)n);
      break;
    case 'i':
      fprintf(f, "%d\n", (int)n);
      break;
    case 'r':
      fprintf(f, "%0.14e\n", n);
      break;
    case 'c':
      fprintf(f, "%c\n", c);
      break;
    case 's':
      fprintf(f, "%s\n", s);
```

```
46          break;
47        }
48        fclose(f);
49      }
50  }
51
52  void PutN(int a)
53  {
54    put_('i', a, ' ', 0);
55  }
```

4.2 Порядок действий задачника Unix Taskbook при выполнении заданий, импортированных из задачника Programming Taskbook

1. определение группы задач и номера задачи в этой группе; эта информация передается в качестве параметра командной строки при запуске задачника Unix Taskbook;

2. компиляция учебной программы (если в ходе компиляции были обнаружены ошибки, то вывод соответствующего сообщения и немедленное завершение работы);

3. загрузка динамической библиотеки, связанной с указанной группой заданий и передача ей управления;

4. вызов из данной динамической библиотеки функций initgroup и inittask базовой динамической библиотеки, перенесенной из задачника Programming Taskbook (см. выше ее описание); в результате этого вызова на диске будет создан файл $$pt4dat$$.dat; анализ этого файла; в частности, определение количество требуемых тестовых запусков;

5. считывание данных из файла $$pt4dat$$.dat и генерация на их основе дополнительных файлов, которые будут использоваться при выполнении учебной программы; после этого файл $$pt4dat$$.dat желательно удалить;

6. запуск откомпилированной учебной программы и ожидание ее завершения;

7. анализ файлов, созданных при работе учебной программы, проверка правильности решения и отображение всей информации, связанной с решением, на экране;

8. при успешном прохождении текущего теста повторный вызов функции inittask для того же задания с целью генерации очередного набора тестовых данных и повторное выполнение шагов 5–8, пока не будут успешно пройдены все требуемые тесты или пока не будет обнаружена ошибка в решении.

5 Модуль, который отвечает за взаимодействие с базовой динамической библиотекой и проверкой решения

Базовая динамическая библиотека содержит весь текст и данные, необходимые для генерации задач.

Поскольку наша Unix Taskbook адаптирована из Programming Taskbook, удобно использовать базовые динамические библиотеки, используемые в Programming Taskbook напрямую, так что нам не нужно разрабатывать отдельный набор базовых динамических библиотек для UnixTaskbook.

Для того чтобы установить контакт с базовой динамической библиотекой, мы разработали интерактивный модуль p t4utilities, который в основном используется для загрузки базовой динамической библиотеки и проверки решения.

5.1 Функции, которые отвечают за взаимодействие с базовой динамической библиотекой

```
1  bool load_lib()
2  {
3      bool result = false;
4      free_lib();
5  #if defined __linux__
6      dylib_name += ".so";
7  #elif defined __APPLE__
8      dylib_name += ".dylib";
9  #endif
10     FLibHandle = dlopen(dylib_name.c_str(), RTLD_LAZY);
11     if (FLibHandle == nullptr)
12     {
13         return false;
14     }
15     InitGroup_ = (TInitGroup)dlsym(FLibHandle, "initgroup");
16     if (InitGroup_ == nullptr)
17     {
18         free_lib();
19         return false;
```

```
20      }
21      InitTask_ = (TInitTask)dlsym(FLibHandle, "inittask");
22      if (InitTask_ == nullptr)
23      {
24          free_lib();
25          return false;
26      }
27      return true;
28  }
```

load_lib - это функция, которая загружает базовую динамическую библиотеку. Сначала она определяет тип операционной системы, на которой запущена программа, а затем загружает динамическую библиотеку с суффиксом динамической библиотеки, соответствующей этой системе. Затем она получает функции **initgroup** и **inittask**, предоставляемые базовой динамической библиотекой.

Две другие функции в pt4utilities, **pt4_print_task_info** и **pt4_generate_task_test**, используют интерфейсы, предоставляемые initgroup и inittask.

5.2 Функции, которые отвечают за проверкой решения

```
1  int pt4_check_program(std::string task_group, int test_num)
2  {
3      bool result = false;
4      result = CheckAllResults();
5      return result ? 0 : 1;
6  }
```

Для проверки решения используется функция **pt4_check_program**, которая вызывает функцию **CheckAllResults** из другого модуля, который будет рассмотрен в следующей главе.

6 Модули для обработки и визуализации данных

С помощью предыдущего введения мы смогли сгенерировать необходимые данные и текст, но в настоящее время все данные и текст находятся в одном файле **$$pt4dat$$.dat**. Чтобы облегчить последующие манипуляции и визуализацию, мы также разработали два модуля **udata** и **uprint**.

6.1 Обработка данных

В модуле **udata** предусмотрен ряд функций для обработки данных и текста.

6.1.1 Загрузка данных и создание файлов данных

Функция **LoadData** считывает данные и текст из файла **$$pt4dat$$.dat** и сохраняет их в переменных.

```
1  void CreateAddFiles()
2  {
3      std::ofstream f("ut1inf.dat");
4      f << "11" << std::endl;              // Taskbook version
5      f << cursize << std::endl;           // number of processes
6      f << NumberOfUsedData << std::endl; // number of used data (in
           the process 0);
7      f.close();
8
9      for (int i = 0; i <= 100; i++)
10     {
11         if (std::filesystem::exists("ut1in" + std::to_string(i
               ) + ".dat"))
12         {
13             std::filesystem::remove("ut1in" + std::to_string(i
                   ) + ".dat");
14         }
15         if (std::filesystem::exists("ut1out" + std::to_string(
               i) + ".dat"))
16         {
17             std::filesystem::remove("ut1out" + std::to_string(
                   i) + ".dat");
18         }
19         if (std::filesystem::exists("ut1err" + std::to_string(
               i) + ".dat"))
```

```
20          {
21              std::filesystem::remove("ut1err" + std::to_string(
                    i) + ".dat");
22          }
23          if (std::filesystem::exists("ut1sh" + std::to_string(i
                ) + ".dat"))
24          {
25              std::filesystem::remove("ut1sh" + std::to_string(i
                    ) + ".dat");
26          }
27      }
28
29      std::ofstream file;
30      bool isOpen;
31      for (int i = 0; i < cursize; i++)
32      {
33          isOpen = false;
34          for (int j = 0; j < nidata; j++)
35          {
36              if (idata[j].r == i)
37              {
38                  if (!isOpen)
39                  {
40                      file.open("ut1in" + std::to_string(i) + ".
                            dat");
41                      isOpen = true;
42                  }
43                  file << idata[j].id << std::endl;
44                  switch (idata[j].id)
45                  {
46                  case 'b':
47                  case 'i':
48                      file << std::round(idata[j].n) << std::
                            endl; // TODO: this 'round' is dirfferent
                            from the one in pascal
49                      break;
50                  case 'r':
51                      file << idata[j].n << std::endl;
52                      break;
53                  case 'c':
54                  case 's':
55                      file << idata[j].s << std::endl;
56                      break;
57                  }
58              }
59          }
```

```
60          if (isOpen)
61          {
62              file.close();
63          }
64      }
65  }
```

CreateAddFiles затем записывает эти данные во вспомогательные файлы, которые используются при проверке решения.

- **ut1in** - Хранение входных данных.
- **ut1out** - Хранение данных, полученных в результате работы учебных программ.
- **ut1err** - Хранение сообщений об ошибках.
- **ut1sh** - Хранение отладочной информации для учебных программ.

6.1.2 Проверка данных решения

```
1  bool CheckAllResults()
2  {
3      nrdata = 0;
4      check_result = true;
5      headerinfos.assign(cursize, "");
6      odata_procs.resize(cursize);
7      for (int i = 0; i < cursize; ++i)
8      {
9          std::vector<TData> odata_i;
10         for (int j = 0; j < nodata; ++j)
11         {
12             if (odata[j].r == i)
13             {
14                 odata_i.emplace_back(odata[j]);
15             }
16         }
17         odata_procs[i] = odata_i;
18         if (i == 0)
19             continue;
20         check_result = CheckResults(i) && check_result;
21     }
22
23     if (check_result)
```

```
24          check_result = CheckResults(0);
25      else
26      {
27          CheckResults(0);
28          headerinfos[0] = WrongSolutionMsg;
29      }
30
31      return check_result;
32  }
```

Функция **CheckAllResults** вызывается pt4utilities и проверяет решение каждого процесса.

Функция CheckResults вызывается для проверки решения отдельного процесса. Она будет сравнивать один за другим данные, выведенные учебной программой, с данными правильного решения, если они все одинаковые, то оно правильное, если в нем разные данные, то учебная программа ошибается.

6.2 Визуализация данных

Визуализация данных имеет решающее значение для обеспечения хорошего интерактивного интерфейса для пользователей. Модули udata и uprint предоставляют ряд функций для достижения этой цели.

6.2.1 Основные функции визуализации

Эта часть была выполнена другим автором, Чжан Инцин, который построил основные рамки интерфейса программы, такие как границы интерфейса. Далее приведен пример одной из таких функций.

```
1  // print header info
2  void PrintHeader(const std::string RunInfo)
3  {
4      std::vector<std::string> s(2);
5      std::string s1 = InitString(' ');
6      std::string s0 = InitString(box_drawings::LIGHT_HORIZONTAL
           );
7      PrintCmt(s0, GrTopic + std::to_string(TaskNumber) + " [" +
           GrDescr + "] ", 4);
8      PrintCmt(s0, '(' + std::to_string(TestNumber) + ')', 75);
9      s[0] = s0;
```

```
10      std::cout << theme_bg << border_color << box_drawings::
           LIGHT_DOWN_AND_RIGHT << s[0]
11                << box_drawings::LIGHT_DOWN_AND_LEFT << colors::
                     RESET << std::endl;
12      s[1] = s1;
13      PrintCmt(s[1], RunInfo, 2);
14      std::string bg_color = check_bg(RunInfo);
15      std::cout << theme_bg << border_color << box_drawings::
           LIGHT_VERTICAL << bg_color << colors::WHITE << s[1]
16                << colors::RESET << theme_bg << border_color <<
                     box_drawings::LIGHT_VERTICAL << colors::RESET
                      << std::endl;
17  }
```

Функция **PrintHeader** используется для печати заголовочной информации интерфейса.

Чтобы добиться эффекта непрерывной границы, мы используем для печати границ кодировку Unicode, определенную в заголовочном файле **box_drawing**.

Функция PrintCmt определена и реализована в uprint, и она гарантирует, что каждая строка вывода на терминал будет одинаковой длины.

6.2.2 Обработка цвета

Чтобы сделать интерфейс более эстетически приятным, цвета всего интерфейса были тщательно обработаны.

Сначала мы разделим все цвета в интерфейсе на цвет фона, цвет границы, цвет текста и цвет данных.

```
1   std::string Colorize(const std::string &text)
2   {
3
4       std::regex special_pattern(R"(\s*(\d+)\|\s*(\d+)\>\s*(.*))
           ");
5       if (std::regex_match(text, special_pattern))
6       {
7           return text;
8       }
9
10      std::regex pattern(R"((-?(true|false|\d+(\.\d+)?)))");
11
```

```
12      std::string colored_text = std::regex_replace(text,
            pattern, colors::RESET + theme_bg + data_color + "$&" +
             colors::RESET + theme_bg + text_color);
13
14      return ProcessString(colored_text);
15  }
```

Функция **Colorize** используется для установки цвета данных, где регулярное выражение используется для поиска данных, цвет которых необходимо изменить.

- **theme_bg** - Определяет ASCII-код для цвета фона.
- **text_color** - Определяет ASCII-код цвета текста.
- **data_color** - Определяет ASCII-код цвета данных.
- **border_color** - Определяет ASCII-код для цвета границы.

Функции, используемые для вывода остальной части интерфейса, также используют коды ASCII для указания всех цветов. Эти цвета определяются перед сгенерированным интерфейсом и позволяют пользователю настраивать эти цвета.

Чтобы реализовать функцию пользовательского цвета, мы добавили файл конфигурации **config.yml**.

```
1  # This is a configuration file for the app's colors.
2  # The file specifies the RGB values for each of the five colors used in
       the app.
3
4  # Define the RGB value for the background color.
5  theme_bg: [0, 0, 50] # Black
6
7  # Define the RGB value for the data color.
8  data_color: [255, 255, 200] # Yellow
9
10 # Define the RGB value for the text color.
11 text_color: [128, 128, 200] # Gray
12
13 # Define the RGB value for the border color.
14 border_color: [255, 255, 255] # White
```

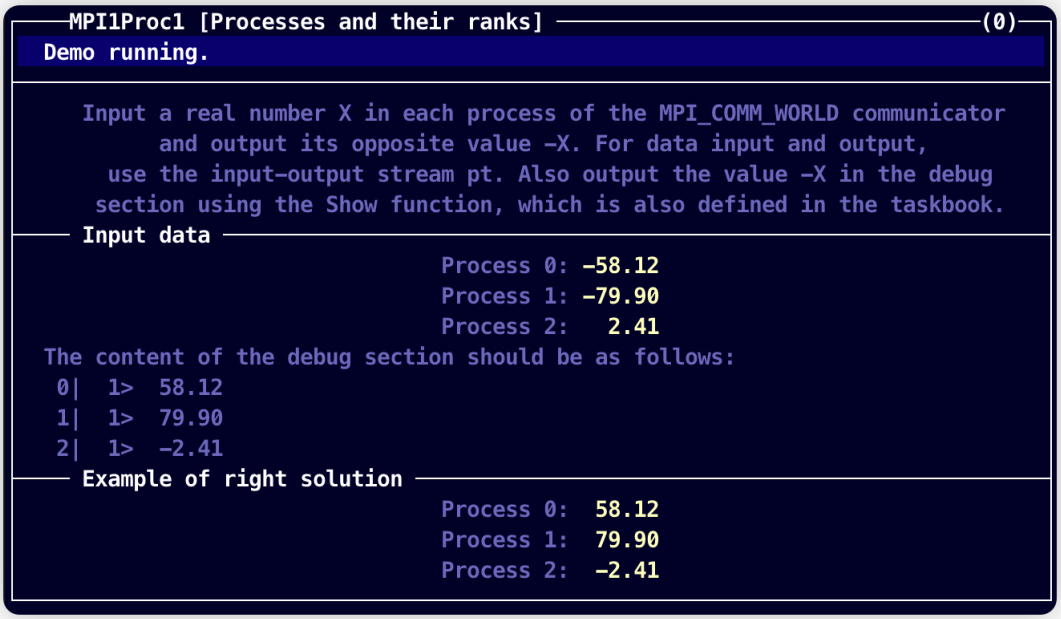


Рис. 2: Примеры тем.

Перед созданием интерфейс программа считывает RGB-коды для этих цветов из этого профиля. Затем эти значения присваиваются соответствующим цветовым переменным.

Это изображение2 демонстрирует эффект, достигаемый с помощью данного профиля.

Однако следует отметить, что если вы хотите, чтобы пользовательская тема вступила в силу, необходимо добавить параметр **-c custom** после команды unixTaskbook в файле **runtest.cmd**.

7 Реализация набора динамических библиотек, подключаемых к задачнику Unix Taskbook

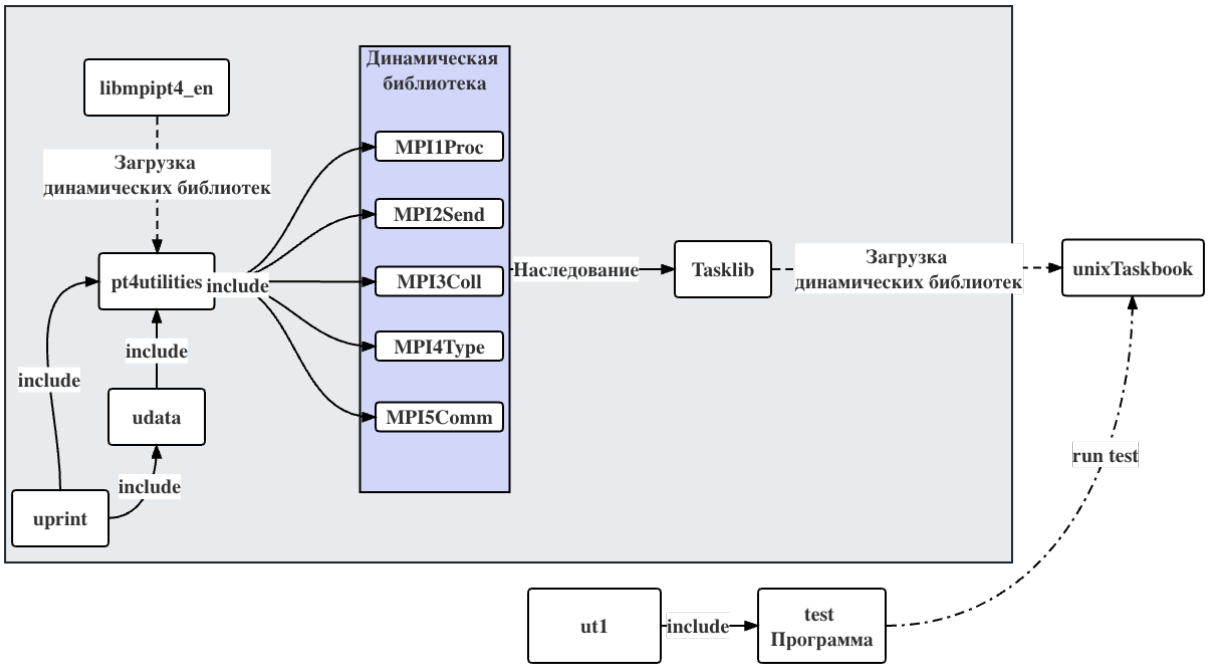


Рис. 3: Отношения между динамическими библиотеками и компонентами.

7.1 Проектные решения для динамических библиотек

Учитывая различия между разными группами задач, например, они могут иметь разные методы компиляции, параметры компиляции, количество задач и т.д., для совместимости с разными группами задач нам необходимо разработать интерфейс для регулирования различных реализаций динамических библиотек. То есть, все эти динамические библиотеки будут реализовывать этот интерфейс, поэтому все динамические библиотеки будут иметь некоторые общие поля или методы, но эти поля и методы могут иметь различные значения или реализации.

7.1.1 Интерфейс TaskLib

```cpp
#ifndef TASKLIB_HPP
#define TASKLIB_HPP

#include "header.hpp"

class TaskLib
{
protected:
    std::vector<std::string> task_text_russian;
    std::vector<std::string> task_text_chinese;
    std::vector<std::string> task_text_english;

    std::vector<std::string> compile_argv;

    std::vector<std::string> execute_argv;

    std::vector<std::string> test_files;
    std::string control_file = "_control.tst";
    std::string result_file;

    std::string compiler;

    std::string library_name;

    int task_count;
    int output = 1;
    int f_control;
    int total_test_count;

public:
    bool print_file = true;
    bool print_task_info = false;
    // Public real function
    // Implemented by tasklib itself
    TaskLib() {}
    int get_task_count() const;
    std::vector<std::string> get_execute_argv() const;
    std::string get_task_info(int task_num, int
        language_option) const;
    // The virtual function (interface)
    // Implemented by each tasklib itself
    virtual void utb_print_task_info(int task_num, int
        language_option) {}
    virtual void utb_generate_task_test(int task_num, int
        test_num) = 0;
```

```
43      virtual void utb_generate_task_control(int task_num) = 0;
44      virtual void utb_print_extra_info(int task_num) = 0;
45      virtual int utb_check_program(int test_num) const = 0;
46      virtual ~TaskLib() {}
47
48      // friend class
49      friend class UnixTaskbook;
50  };
51
52  // the types of the class factories
53  typedef TaskLib *create_t();
54  typedef void destroy_t(TaskLib *);
55
56  #endif
```

Интерфейс Tasklib был реализован в рамках дипломного проекта [1] Ли Шэнюй.

TaskLib является основой для создания библиотек разных задач на программирования и предоставляет удобный интерфейс для работы с задачами.

7.1.2 Особенности динамической библиотеки параллельного программирования MPI

- MPICH является высокопроизводительной и широко переносимой реализацией стандарта MPI, который представляет собой интерфейс передачи сообщений для распределенных приложений, использующих параллельные вычисления. MPICH и его производные формируют самые широко используемые реализации MPI в мире.

  MPICH поддерживает последнюю версию стандарта MPI-2 и может быть установлен и запущен на платформах Windows и UNIX. MPICH предоставляет компоненты, среды выполнения и инструменты, необходимые для компиляции и запуска программ MPI [6] [7]. MPICH также обладает рядом преимуществ, таких как оптимизация коллективных операций связи [8], эффективная поддержка многопоточной связи MPI и совместимость с другими реализациями MPI.

- mpicc является оберткой над базовым компилятором C, которая автоматически добавляет необходимые опции компилятора и линковщика для MPI-программ. Использование mpicc упрощает компиляцию и сборку MPI-программ, без необходимости указывать расположение заголовочных и библиотечных файлов MPI

[9] [10]. mpicc является инструментом, предоставляемым реализацией MPI (например, MPICH или Open MPI), который может адаптироваться к различным реализациям MPI и платформам [11] [12].

Для большинства задач в группе задач по программированию мы просто компилируем тестируемый файл с помощью gcc/g++ для создания исполняемого файла <имя_файла>.out и затем выполняем его с помощью . /<имя_файла>.out, чтобы выполнить его непосредственно из командной строки.

Но в группах параллельных программ мы необходимо использовать MPICH для выполнения параллельных вычислительных задач.

В этом случае, если мы используем gcc для компиляции тестового файла, нам придется вручную добавлять опции компилятора и компоновщика, связанные с MPI, что, конечно, обременительно, поэтому для классов, относящихся к группе задач параллельного программирования, нам нужно наследовать класс TaskLib и установить поле **compiler** в "mpicc".

Для компиляции файла на языке C с помощью mpicc необходимо указать следующие параметры:

- <имя_файла>.c - задает имя или путь исходного файла на языке C;
- -o <имя_файла> - задает имя или путь исполняемого файла, который будет создан при компиляции;
- -c - задает режим компиляции без сборки;

Например, следующая команда компилирует файл MPI1Proc7.c и создает исполняемый файл MPI1Proc7.out:

mpicc MPI1Proc7.c -o MPI1Proc7.out

При тестировании задачи в группах параллельных программ мы используем утилиту mpiexec для запуска программы. mpiexec является универсальной утилитой для запуска MPI-приложений, которая поддерживает различные реализации MPI и различные среды выполнения [13].

Для запуска MPI-приложения с помощью mpiexec необходимо указать следующие параметры:

- -np <число> - задает количество MPI-процессов для запуска;
- -f <имя_файла> ;

Например, следующая команда запускает MPI1Proc7.out на 8 MPI-процессах:

mpiexec -np 8 ./MPI1Proc7.out

Таким образом, чтобы реализовать динамическую библиотеку для групп задач параллельного программирования, мы должны наследовать TaskLib и указать поле **executor** как "mpiexec".

7.2 Реализованные динамические библиотеки

Для реализации динамической библиотеки для групп задач параллельного программирования необходимо написать файлы классов, соответствующие каждой группе задач, и эти классы, являющиеся подклассами класса TaskLib, реализуют интерфейсы, объявленные классом TaskLib.

В дополнение к этому, поскольку мы подключаем существующую группу задач из задачника Programming Taskbook в Unix Taskbook, нам также необходимо реализовать инструментальную библиотеку, pt4utilities, для подключения к существующей группе задач, так что динамическая библиотека, которую мы реализуем, фактически действует как промежуточное программное обеспечение, передавая управление генерацией информации о задаче, представлением информации о задаче и проверкой файла результатов существующей группе задач.

Далее мы опишем интерфейсы, предоставляемые pt4utilities для динамических библиотек.

```
1 #ifndef PT4UTILITIES_HPP
2 #define PT4UTILITIES_HPP
3 #include <string>
4
5 typedef void __attribute__((stdcall)) (*TInitGroup)(const char
      *);
6 typedef void __attribute__((stdcall)) (*TInitTask)(int, int);
7
8 void pt4_print_task_info(std::string task_group, int task_num,
      int language_option);
```

```
9  void pt4_generate_task_test(std::string task_group, int
       task_num, int test_num);
10 void pt4_print_extra_info(std::string task_group, int task_num
       );
11 int pt4_check_program(std::string task_group, int test_num);
12 int pt4_mpi_get_size();
13
14 #endif
```

pt4utilities содержит объявления для всех функций, необходимых для работы с группами задач в задачнике Programming Taskbook.

- **void pt4_print_task_info(std::string task_group, int task_num, int language_option)** - выводит информацию о задаче, включая ее название, описание и примеры входных и выходных данных;
- **void pt4_generate_task_test(std::string task_group, int task_num, int test_num)** - генерирует тестовые данные для задачи;
- **void pt4_print_extra_info(std::string task_group, int task_num)** - выводит информацию о результатах выполнения текущего задания;
- **int pt4_check_program(std::string task_group, int test_num)** - проверяет программу на соответствие ожидаемому результату для заданного теста;
- **int pt4_mpi_get_size()** - Возвращает количество процессов, необходимых для выполнения текущего тестового задания;

Вышеуказанные функции будут доступны для всех динамических библиотек, которым необходим доступ к уже существующим группам задач задачника Programming Taskbook, чтобы обеспечить управление этими группами задач.

Стоит отметить, что для компиляции pt4utilities необходимо указать, что g++ использует стандарт c++17 и выше. Это связано с тем, что в этом файле и других файлах, которые он представил были представлены некоторые новые возможности c++17. Например, std::filesystem(Файловая система (filesystem) - это библиотека стандартной библиотеки C++17, которая предоставляет возможности для выполнения операций с файловыми системами и их компонентами, такими как пути, обычные файлы и каталоги [14])

Основные используемые функции - std::filesystem::exists и std::filesystem::remove.

Функция **std::filesystem::exists** используется для проверки существования файла или директории в файловой системе. Она принимает в качестве аргумента путь к файлу или директории и возвращает **true**, если файл или директория существует, и **false**, если файла или директории не существует.

Функция **std::filesystem::remove** используется для удаления файла или директории из файловой системы. Она принимает в качестве аргумента путь к файлу или директории и удаляет его из файловой системы.

```
1  for (int i = 0; i <= 100; i++)
2  {
3      if (std::filesystem::exists("ut1in" + std::to_string(i) +
           ".dat"))
4      {
5          std::filesystem::remove("ut1in" + std::to_string(i) +
              ".dat");
6      }
7      if (std::filesystem::exists("ut1out" + std::to_string(i) +
           ".dat"))
8      {
9          std::filesystem::remove("ut1out" + std::to_string(i) +
               ".dat");
10     }
11     if (std::filesystem::exists("ut1err" + std::to_string(i) +
           ".dat"))
12     {
13         std::filesystem::remove("ut1err" + std::to_string(i) +
               ".dat");
14     }
15     if (std::filesystem::exists("ut1sh" + std::to_string(i) +
           ".dat"))
16     {
17         std::filesystem::remove("ut1sh" + std::to_string(i) +
              ".dat");
18     }
19 }
```

Это один из фрагментов кода, основной функцией которого является очистка остаточных вспомогательных файлов.

Имея pt4utilities в качестве необходимого условия, мы можем приступить к разработке динамических библиотек.

7.2.1 MPI1Proc

```
1  #include "tasklib.hpp"
2  #include "pt4utilities.hpp"
3
4  class utbMPI1Proc : public TaskLib
5  {
6  private:
7      std::string task_group;
8
9  public:
10     utbMPI1Proc();
11     virtual ~utbMPI1Proc() {}
12
13     virtual void utb_print_task_info(int task_num, int
           language_option)
14     {
15         pt4_print_task_info(task_group, task_num,
               language_option);
16     }
17
18     virtual void utb_generate_task_test(int task_num, int
           test_num)
19     {
20         pt4_generate_task_test(task_group, task_num, test_num)
               ;
21     }
22
23     virtual void utb_generate_task_control(int task_num) {}
24
25     virtual void utb_print_extra_info(int task_num)
26     {
27         pt4_print_extra_info(task_group, task_num);
28     }
29
30     virtual int utb_check_program(int test_num) const
31     {
32         return pt4_check_program(task_group, test_num);
33     }
34
35     // friend class
36     friend class UnixTaskbook;
37 };
38
```

```
39 extern "C" TaskLib *create()
40 {
41     return new utbMPI1Proc;
42 }
43
44 extern "C" void destroy(TaskLib *t)
45 {
46     delete t;
47 }
48
49 utbMPI1Proc::utbMPI1Proc()
50 {
51 #if defined __linux__
52     library_name = "libutbMPI1Proc.so";
53 #elif defined __APPLE__
54     library_name = "libutbMPI1Proc.dylib";
55 #endif
56
57     compiler = "mpicc";
58     compile_argv = {compiler, "-Wall", "-w", "", "ut1.c", "-o"
        };
59
60     execute_argv = {"mpiexec", "-np", std::to_string(
        pt4_mpi_get_size())};
61
62     task_group = "MPI1Proc";
63     task_count = 10;
64     total_test_count = 3;
65
66     print_file = false;
67     print_task_info = true;
68 }
```

Задачи этой группы [15] по MPI предназначены для тестирования умений студентов работать с параллельными вычислениями, а также для проверки их знаний по работе с входными и выходными данными в MPI. Кроме того, задачи требуют от студентов умения работать с функциями ввода-вывода и отладочными функциями в MPI.

В задачах используются различные функции MPI, такие как MPI_Comm_rank, MPI_Comm_size, MPI_Send и MPI_Recv. Задачи покрывают различные аспекты параллельных вычислений, такие как ввод и вывод данных, вычисления в зависимости от номера процесса и обмен данными между процессами.

Решение этих задач требует от студентов умения анализировать поставленную зада-

чу и правильно применять соответствующие функции MPI. Это также требует от них умения работать с различными типами данных и умения разрабатывать эффективные алгоритмы для параллельных вычислений.

7.2.2 Другие динамические библиотеки о группах задач по параллельному программированию

Реализация нескольких других классов(MPI2Send, MPI3Coll, MPI4Type, MPI5Comm) динамической библиотеки похожа на реализацию MPI1Proc, основное отличие заключается в различии некоторых переменных-членов. Например, количество задач может быть разным для разных групп задач. Конечно, имена конструкторов и деструкторов также различны.

8 Руководство по работе с Unix Taskbook

8.1 Запуск задачника

8.1.1 Установка MPICH в системе unix

Ниже описаны шаги по установке MPICH на различных системах UNIX:

- Чтобы установить MPICH на Debian или Ubuntu:

```
1       sudo apt-get update
2       sudo apt-get install mpich
```

- Чтобы установить MPICH на Fedora или CentOS:

```
1       sudo dnf update
2       sudo dnf install mpich
```

- Чтобы установить MPICH на macOS:

```
1       brew update
2       brew install mpich
```

8.1.2 Компиляция динамических библиотек

Чтобы получить динамические библиотеки, которые могут быть загружены ядром, мы сначала должны скомпилировать файлы классов отдельных динамических библиотек, которые мы написали.

Но когда количество динамических библиотек растет, компилировать их по отдельности и вручную может быть хлопотно. Поэтому мы написали файл runlib.cmd для компиляции всех необходимых динамических библиотек в одном файле.

runlib.cmd

```
#!/bin/bash
if ! [ -d ./utblib ]; then mkdir ./utblib; fi
g++ -fPIC -std=c++17 -Wno-attributes src/utbDir.cpp src/utilities.cpp -shared -o ./utblib/libutbDir.so
g++ -fPIC -std=c++17 -Wno-attributes src/utbFile.cpp src/utilities.cpp -shared -o ./utblib/libutbFile.so
g++ -fPIC -std=c++17 -Wno-attributes src/utbText.cpp src/utilities.cpp -shared -o ./utblib/libutbText.so
g++ -fPIC -std=c++17 -Wno-attributes src/utbShell.cpp src/utilities.cpp -shared -o ./utblib/libutbShell.so
g++ -fPIC -std=c++17 -Wno-attributes src/utbBegin.cpp src/pt4utilities.cpp src/udata.cpp src/uprint.cpp -shared -o ./utblib/libutbBegin.so
g++ -fPIC -std=c++17 -Wno-attributes src/utbMPI1Proc.cpp src/pt4utilities.cpp src/udata.cpp src/uprint.cpp -shared -o ./utblib/libutbMPI1Proc.so
g++ -fPIC -std=c++17 -Wno-attributes src/utbMPI2Send.cpp src/pt4utilities.cpp src/udata.cpp src/uprint.cpp -shared -o ./utblib/libutbMPI2Send.so
g++ -fPIC -std=c++17 -Wno-attributes src/utbMPI3Coll.cpp src/pt4utilities.cpp src/udata.cpp src/uprint.cpp -shared -o ./utblib/libutbMPI3Coll.so
g++ -fPIC -std=c++17 -Wno-attributes src/utbMPI4Type.cpp src/pt4utilities.cpp src/udata.cpp src/uprint.cpp -shared -o ./utblib/libutbMPI4Type.so
g++ -fPIC -std=c++17 -Wno-attributes src/utbMPI5Comm.cpp src/pt4utilities.cpp src/udata.cpp src/uprint.cpp -shared -o ./utblib/libutbMPI5Comm.so
```

Перед компиляцией мы проверим, существует ли каталог utblib в каталоге текущего пользователя, если нет, то он будет создан автоматически, и все динамические библиотеки будут автоматически сохранены в этом каталоге после компиляции, чтобы мы могли легко управлять и находить динамические библиотеки в будущем.

Чтобы запустить cmd-файл, нам нужно убедиться, что все файлы классов динамической библиотеки находятся в том же каталоге, что и cmd-файл, а затем выполнить команду **./runlib.cmd**

8.1.3 Компиляция ядра

Следующим шагом будет компиляция для получения исполняемого файла unixTaskbook.

Из-за большого количества файлов и параметров компиляции, необходимых для компиляции, мы написали файл run.cmd, чтобы упростить команду компиляции.

run.cmd

```
1 #!/bin/bash
2 g++ main.cpp unixTaskbook.cpp utilities.cpp tasklib.cpp error.
     cpp service.cpp -o unixTaskbook -ldl
```

Чтобы запустить cmd-файл, нам нужно убедиться, что все файлы классов динамической библиотеки находятся в том же каталоге, что и cmd-файл, а затем выполнить команду **./run.cmd**

8.1.4 Выполнение тестов

После компиляции всех необходимых файлов, чтобы начать использовать unixTaskbook, нам нужно написать тестовый файл, выбрать задачу, которую мы хотим выполнить, создать .c файл с именем задачи и передать имя файла в качестве параметра выполнения в файл runtest.cmd, и мы можем начать выполнение теста.

Например, мы хотим выполнить задачу 7 в группе задач MPI1Proc:

./runtest.cmd MPI1Proc7.c

runtest.cmd

```
1 #!/bin/bash
2 LD_LIBRARY_PATH=~/utblib
3 export LD_LIBRARY_PATH
4 ../unixTaskbook -t $1
```

В файле runtest.cmd мы экспортируем переменную LD_LIBRARY_PATH, которая сообщает компилятору, где находятся все динамические библиотеки.

Во время выполнения написанные нами тестовые файлы компилируются, и если в процессе компиляции что-то пойдет не так, на экран будет выведено сообщение об ошибке 4.

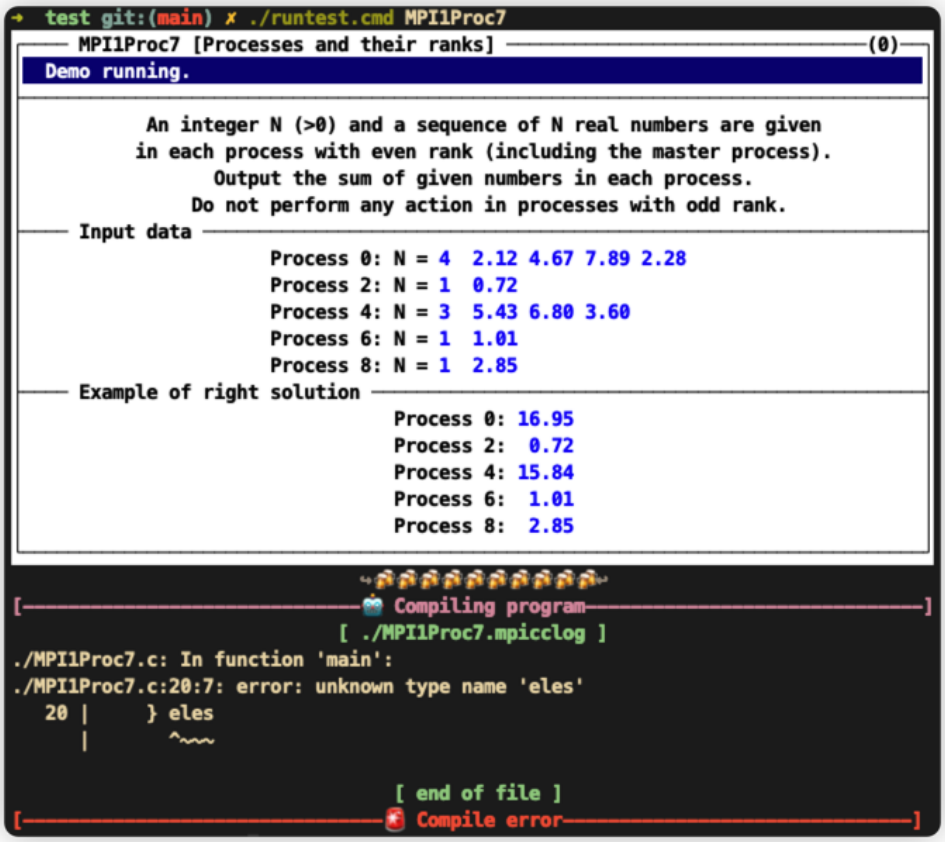


Рис. 4: Log info.

8.2 Примеры использования разработанных групп заданий

(a) demo running

(b) right solution

(c) wrong solution

(d) invalid data

(e) superfluous data

(f) lack of data

Рис. 5: Task info.

- Демонстрация задания выводится перед выполнением теста, его логотип - **Demo running** 5a;
- Если результаты всех процессов верны, на экране появляется **Right solution** 5b;
- Если выход процесса не соответствует правильному ответу, на экране появляется **Wrong solution** 5c, где число в начале обозначает номер процесса;

- Если процесс пытается ввести или вывести данные, которые не соответствуют запрошенному типу, на экране появляется сообщение **Invalid type is used for an input/output data item** 5d, где число в начале обозначает номер процесса;
- Если процесс пытается ввести или вывести лишние данные, на экране появляется **An attempt to input/output superfluous data** 5e, где число в начале обозначает номер процесса;
- Если данные в процессе не вводятся или не выводятся, на экране появляется **Data are not output/input** или **Some required data are not output/input**, где число в начале обозначает номер процесса;

Прежде чем приступить к написанию программы-задачи, мы можем выполнить **./runtest.cmd <имя_задачи>**, чтобы посмотреть информацию о задаче.

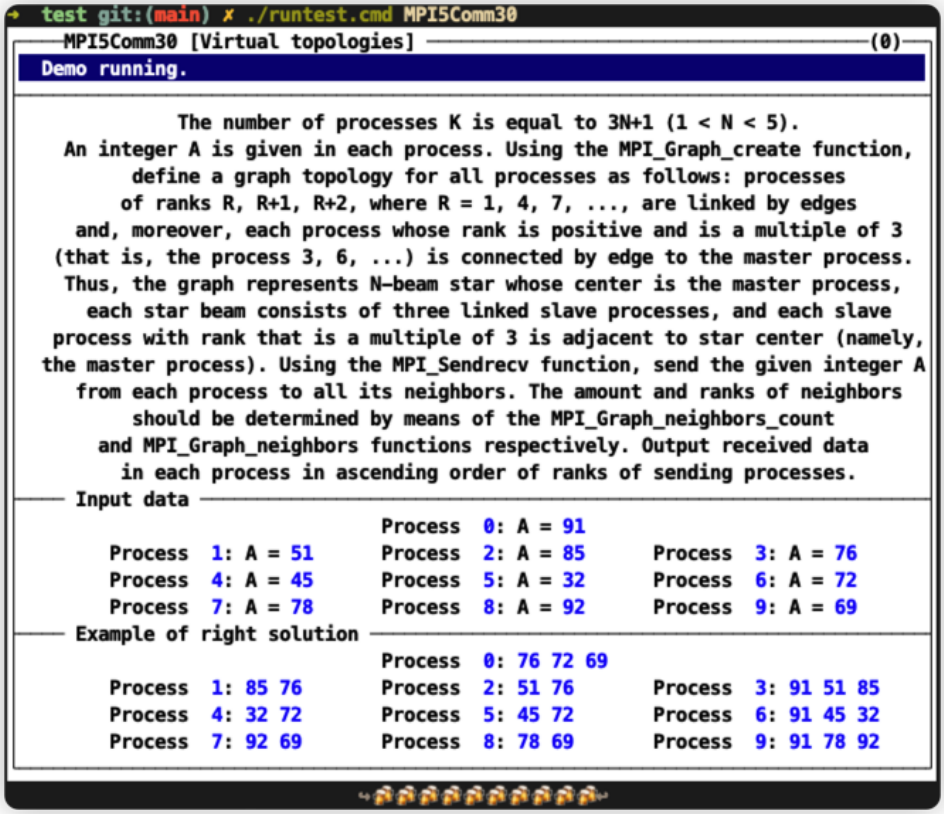
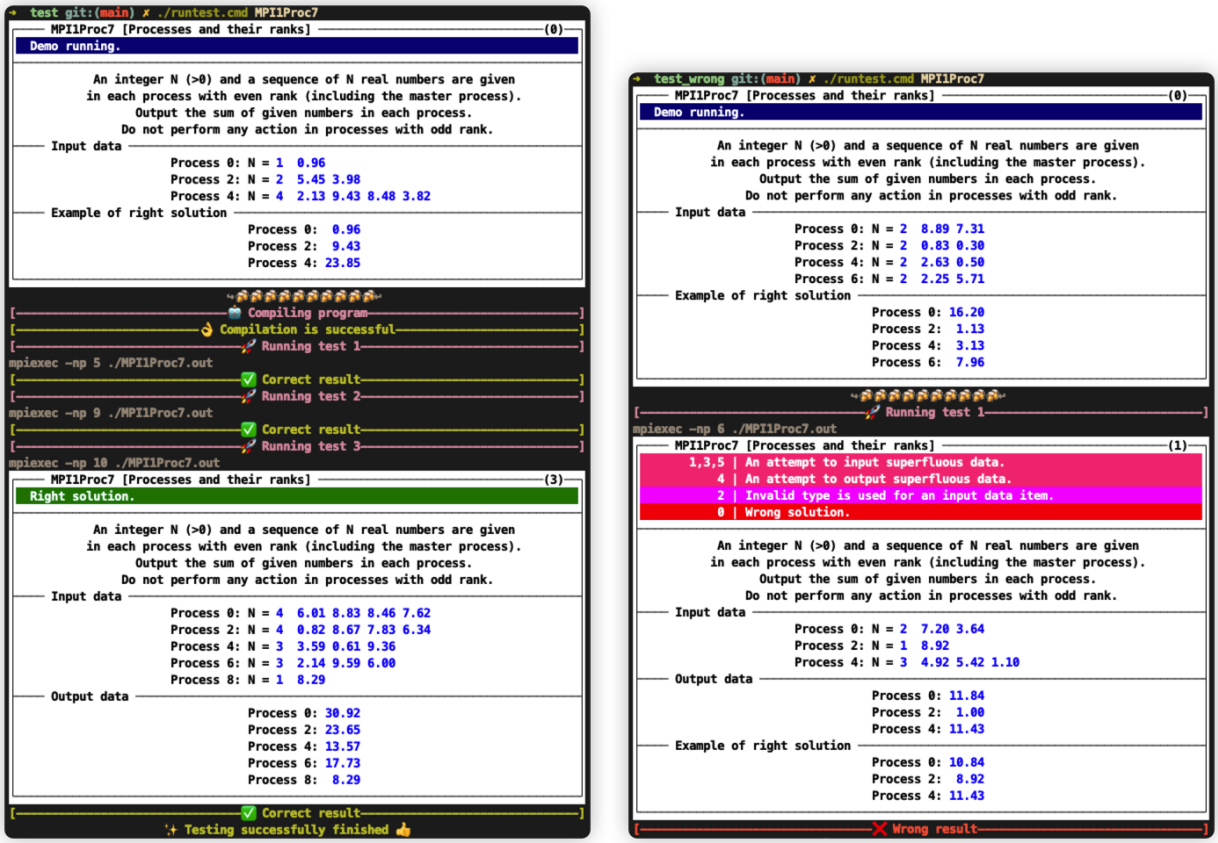


Рис. 6: info demo.

8.2.1 MPI1Proc7



(a) Правильно (b) Неисправность

Рис. 7: MPI1Proc.

```
1  #include "ut1.h"
2  int main()
3  {
4      MPI_Init(NULL, NULL);
5      int rank, size;
6      MPI_Comm_size(MPI_COMM_WORLD, &size);
7      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
8      if (!(rank % 2)) {
9          int n;
10         double num;
11         double sum = 0;
12         GetN(&n);
13         for (int i = 0; i < n; ++i) {
```

```
14              GetD(&num);
15              sum += num;
16          }
17          PutD(sum);
18      }
19      MPI_Finalize();
20  }
```

Код выводит правильный ответ 7a.

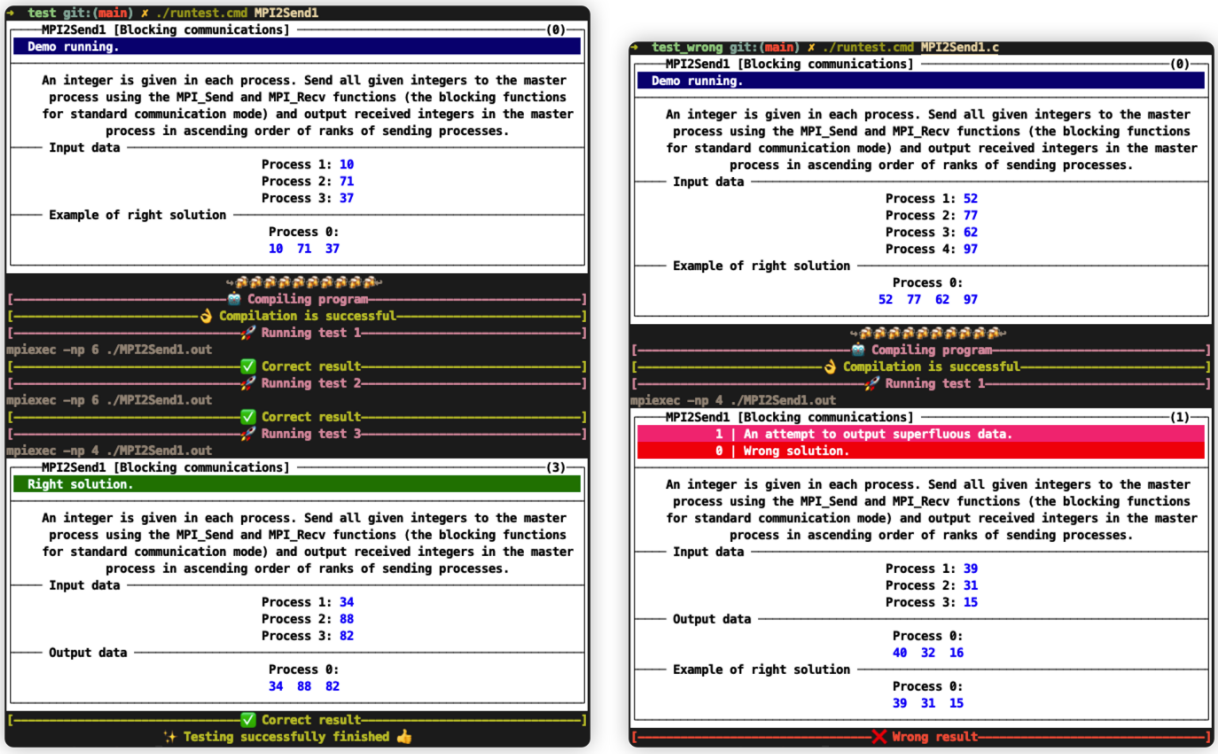
```
1   #include "ut1.h"
2   int main()
3   {
4       MPI_Init(NULL, NULL);
5       int rank, size;
6       MPI_Comm_size(MPI_COMM_WORLD, &size);
7       MPI_Comm_rank(MPI_COMM_WORLD, &rank);
8       if (rank == 0) {
9           int n;
10          double num;
11          double sum = 0;
12          GetN(&n);
13          for (int i = 0; i < n; ++i) {
14              GetD(&num);
15              sum += num;
16          }
17          PutD(sum + 1);
18      }
19
20      if (rank == 1) {
21          int n;
22          double num;
23          double sum = 0;
24          GetN(&n);
25          for (int i = 0; i < n; ++i) {
26              sum += num;
27          }
28          PutD(sum);
29      }
30
31      if (rank == 2) {
32          int n;
33          int num;
34          double sum = 0;
35          GetN(&n);
```

```
36          for (int i = 0; i < n; ++i) {
37              GetN(&num);
38              sum += num;
39          }
40          PutD(sum + 1);
41      }
42
43      if (rank == 3) {
44          int n;
45          double num;
46          double sum = 0;
47          GetN(&n);
48          GetN(&n);
49          for (int i = 0; i < n; ++i) {
50              GetD(&num);
51              sum += num;
52          }
53          PutD(sum);
54      }
55
56      if (rank == 4) {
57          int n;
58          double num;
59          double sum = 0;
60          GetN(&n);
61          for (int i = 0; i < n; ++i) {
62              GetD(&num);
63              sum += num;
64          }
65          PutD(sum);
66          PutD(sum);
67      }
68
69      if (rank == 5) {
70          int n;
71          double num;
72          double sum = 0;
73          GetN(&n);
74          for (int i = 0; i < n; ++i) {
75              GetD(&num);
76              sum += num;
77          }
78          PutD(sum);
79      }
80
```

```
81      if (rank == 6) {
82          int n;
83          double num;
84          double sum = 0;
85          GetN(&n);
86          for (int i = 0; i < n; ++i) {
87              GetD(&num);
88              sum += num;
89          }
90          PutD(sum);
91      }
92
93      if (rank == 7) {
94          int n;
95          double num;
96          double sum = 0;
97          GetN(&n);
98          for (int i = 0; i < n; ++i) {
99              GetD(&num);
100             sum += num;
101         }
102         PutD(sum);
103     }
104
105
106     MPI_Finalize();
107 }
```

Мы намеренно создали ряд ошибок в различных процессах, в результате чего в конце было выведено несколько различных сообщений об ошибках 7b.

8.2.2 MPI2Send1



(a) Правильно (b) Неисправность

Рис. 8: MPI2Send.

```
1  #include "ut1.h"
2  int main()
3  {
4      MPI_Init(NULL, NULL);
5      int rank, size;
6      MPI_Comm_size(MPI_COMM_WORLD, &size);
7      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
8      int n;
9
10     if (!rank) {
11         for (int i = 1; i < size; ++i) {
12             MPI_Recv(&n, 1, MPI_INT, i, 0, MPI_COMM_WORLD,
                   MPI_STATUS_IGNORE);
13             PutN(n);
14         }
15     }
```

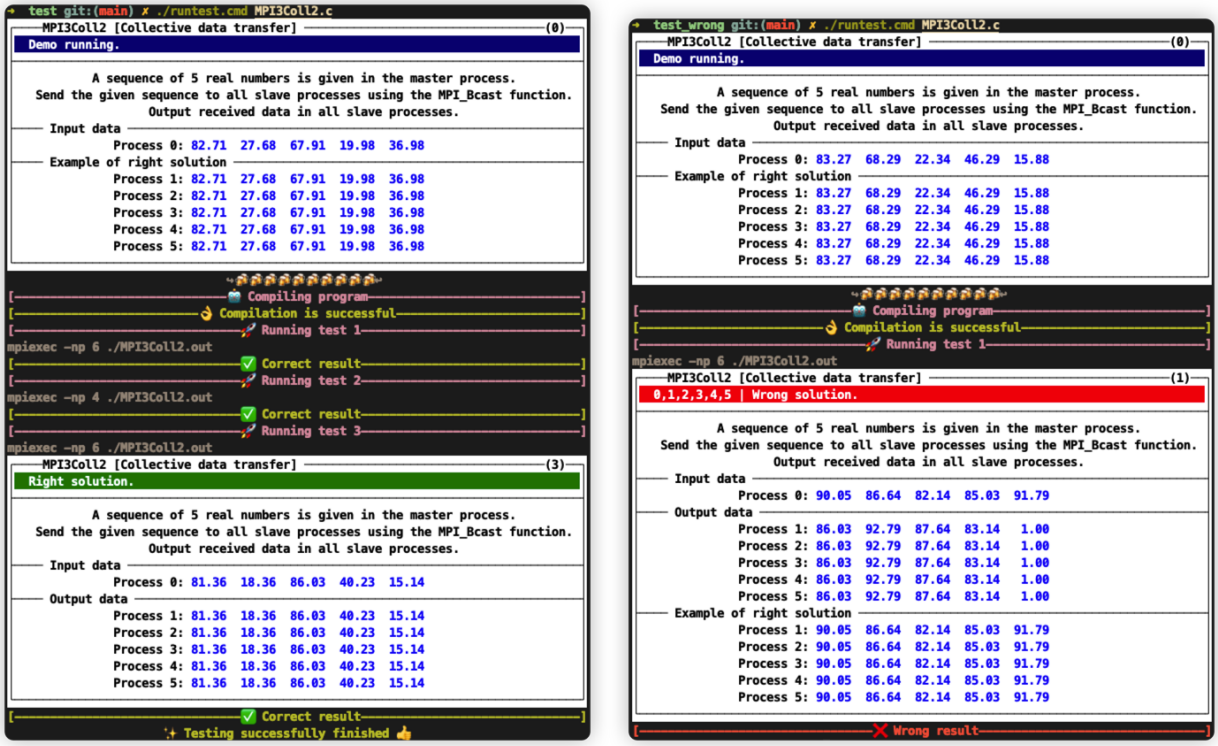
```
16      else {
17          GetN(&n);
18          MPI_Send(&n, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
19      }
20      MPI_Finalize();
21  }
```

Код выводит правильный ответ 8a.

```
1   #include "ut1.h"
2   int main()
3   {
4       MPI_Init(NULL, NULL);
5       int rank, size;
6       MPI_Comm_size(MPI_COMM_WORLD, &size);
7       MPI_Comm_rank(MPI_COMM_WORLD, &rank);
8       int n;
9
10      if (!rank) {
11          for (int i = 1; i < size; ++i) {
12              MPI_Recv(&n, 1, MPI_INT, i, 0, MPI_COMM_WORLD,
                    MPI_STATUS_IGNORE);
13              PutN(n + 1);
14          }
15      }
16      else {
17          GetN(&n);
18          MPI_Send(&n, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
19          if (rank == 1)
20              PutN(n);
21      }
22      MPI_Finalize();
23  }
```

Мы намеренно создали ряд ошибок в различных процессах, в результате чего в конце было выведено несколько различных сообщений об ошибках 8b.

8.2.3 MPI3Coll2



(a) Правильно (b) Неисправность

Рис. 9: MPI3Coll.

```
#include "ut1.h"
int main()
{
    MPI_Init(NULL, NULL);
    int rank, size;
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    double numbers[5];
    if (rank == 0)
        for (int i = 0; i < 5; ++i)
            GetD(&numbers[i]);
    MPI_Bcast(&numbers, 5, MPI_DOUBLE, rank, MPI_COMM_WORLD);
    if (rank != 0)
        for (int i = 0; i < 5; ++i)
            PutD(numbers[i]);
```

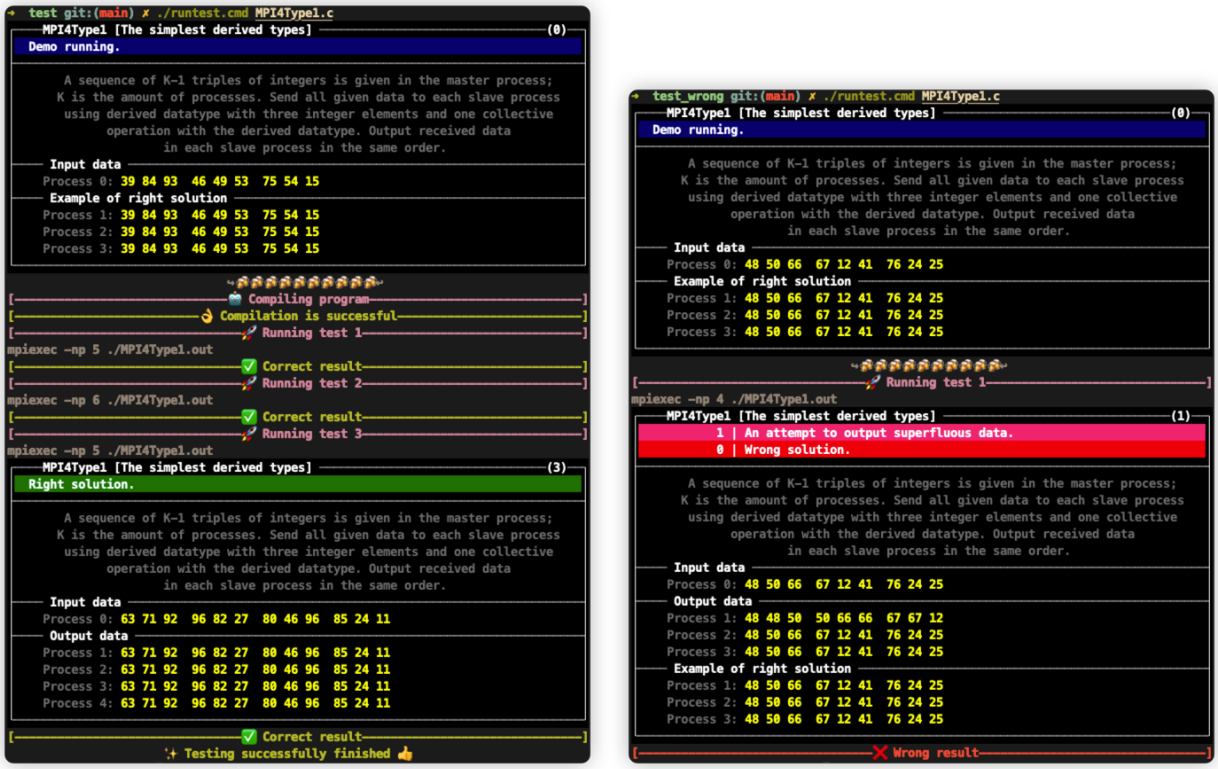
```
16
17      MPI_Finalize();
18  }
```

Код выводит правильный ответ 9a.

```
1  #include "ut1.h"
2  int main()
3  {
4      MPI_Init(NULL, NULL);
5      int rank, size;
6      MPI_Comm_size(MPI_COMM_WORLD, &size);
7      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
8      double numbers[5];
9      if (rank == 0)
10         for (int i = 0; i < 5; ++i)
11             GetD(&numbers[(i+1)%4]);
12     MPI_Bcast(&numbers, 5, MPI_DOUBLE, rank, MPI_COMM_WORLD);
13     if (rank != 0)
14         for (int i = 0; i < 5; ++i)
15             PutD(numbers[i]+1);
16
17     MPI_Finalize();
18 }
```

Мы намеренно создали ряд ошибок в различных процессах, в результате чего в конце было выведено несколько различных сообщений об ошибках 9b.

8.2.4 MPI4Type1



(a) Правильно (b) Неисправность

Рис. 10: MPI4Type.

```c
#include "ut1.h"
int main()
{
    MPI_Init(NULL, NULL);
    int rank, size;
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    MPI_Datatype newtype;
    MPI_Type_contiguous(3, MPI_INT, &newtype);
    MPI_Type_commit(&newtype);

    int *numbers = (int*)malloc((size - 1) * 3 * sizeof(int));

    if (rank == 0)
```

```
16          for (int i = 0; i < (size - 1) * 3; ++i)
17              GetN(&numbers[i]);
18
19      MPI_Bcast(numbers, size - 1, newtype, 0, MPI_COMM_WORLD);
20
21      if (rank != 0)
22          for (int i = 0; i < (size - 1) * 3; ++i)
23              PutN(numbers[i]);
24
25      MPI_Type_free(&newtype);
26      free(numbers);
27      numbers = NULL;
28      MPI_Finalize();
29  }
```

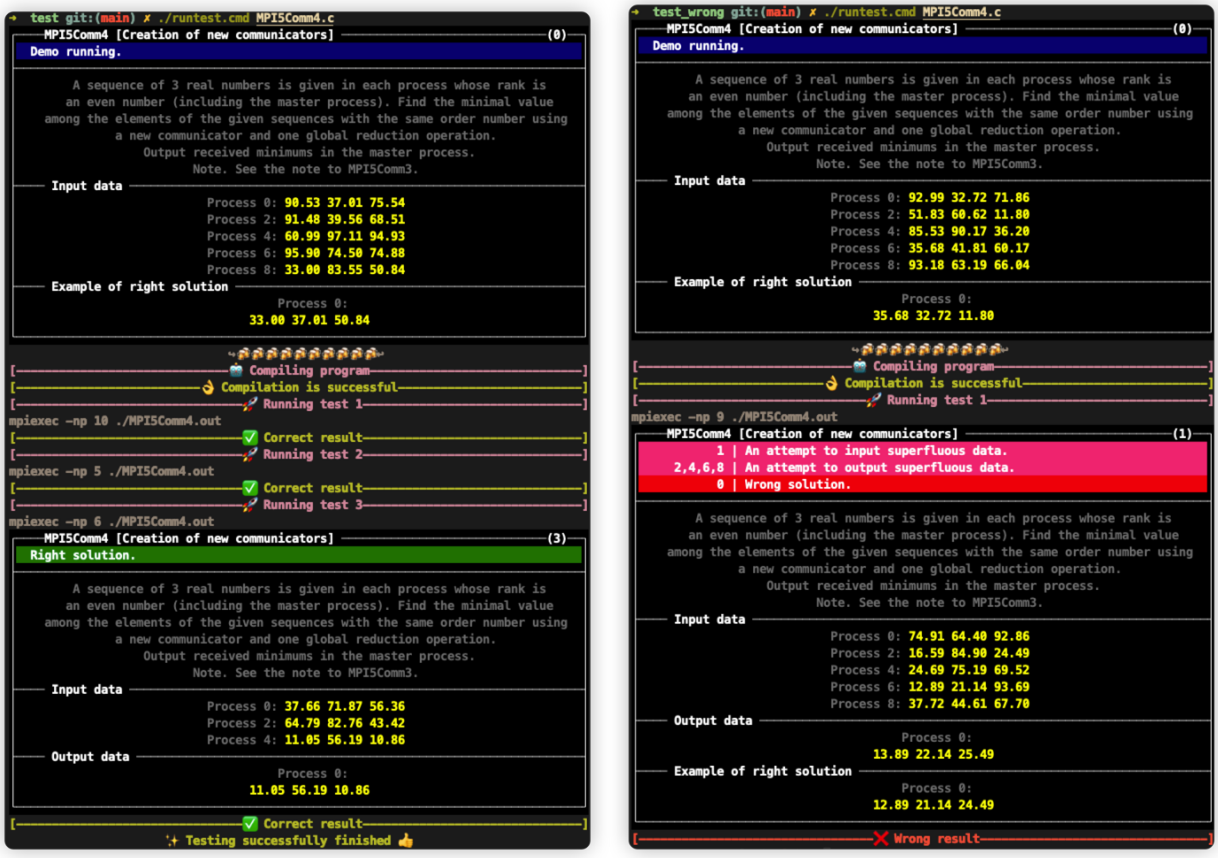
Код выводит правильный ответ 10a.

```
1   #include "ut1.h"
2   int main()
3   {
4       MPI_Init(NULL, NULL);
5       int rank, size;
6       MPI_Comm_size(MPI_COMM_WORLD, &size);
7       MPI_Comm_rank(MPI_COMM_WORLD, &rank);
8
9       MPI_Datatype newtype;
10      MPI_Type_contiguous(3, MPI_INT, &newtype);
11      MPI_Type_commit(&newtype);
12
13      int *numbers = (int*)malloc((size - 1) * 3 * sizeof(int));
14
15      if (rank == 0)
16          for (int i = 0; i < (size - 1) * 3 + 1; ++i)
17              GetN(&numbers[i]);
18
19      MPI_Bcast(numbers, size - 1, newtype, 0, MPI_COMM_WORLD);
20
21      if (rank != 0)
22          for (int i = 0; i < (size - 1) * 3; ++i)
23          {
24              PutN(numbers[i]);
25              if (rank == 1)
26                  PutN(numbers[i]);
27          }
28
```

```
29      MPI_Type_free(&newtype);
30      free(numbers);
31      numbers = NULL;
32      MPI_Finalize();
33  }
```

Мы намеренно создали ряд ошибок в различных процессах, в результате чего в конце было выведено несколько различных сообщений об ошибках 10b.

8.2.5 MPI5Comm4



(a) Правильно

(b) Неисправность

Рис. 11: MPI5Comm.

```
#include "ut1.h"
int main()
{
    MPI_Init(NULL, NULL);
    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    MPI_Group even;
    MPI_Comm even_comm;
    int color = rank % 2;
    MPI_Comm_split(MPI_COMM_WORLD, color, rank, &even_comm);

    if (rank % 2 == 0)
    {
        double nums[3];
        for (int i = 0; i < 3; ++i)
            GetD(&nums[i]);
        double result[3];
        MPI_Reduce(nums, result, 3, MPI_DOUBLE, MPI_MIN, 0,
           even_comm);
        if (rank == 0)
            for (int i = 0; i < 3; ++i)
                PutD(result[i]);
    }
    MPI_Finalize();
}
```

Код выводит правильный ответ 11a.

```
#include "ut1.h"
int main()
{
    MPI_Init(NULL, NULL);
    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    MPI_Group even;
    MPI_Comm even_comm;
    int color = rank % 2;
    MPI_Comm_split(MPI_COMM_WORLD, color, rank, &even_comm);

    if (rank == 1)
```

```
15      {
16          double a;
17          GetD(&a);
18      }
19      if (rank % 2 == 0)
20      {
21          double nums[3];
22          for (int i = 0; i < 3; ++i)
23              GetD(&nums[i]);
24          if (rank != 0)
25              PutD(nums[0]);
26          double result[3];
27          MPI_Reduce(nums, result, 3, MPI_DOUBLE, MPI_MIN, 0,
                even_comm);
28          if (rank == 0)
29              for (int i = 0; i < 3; ++i)
30                  PutD(result[i] + 1);
31      }
32      MPI_Finalize();
33  }
```

Мы намеренно создали ряд ошибок в различных процессах, в результате чего в конце было выведено несколько различных сообщений об ошибках 11b.

9 Заключение

В процессе выполнения поставленной задачи были разработаны следующие компоненты:

- Динамическая библиотека MPI1Proc;
- Динамическая библиотека MPI2Send;
- Динамическая библиотека MPI3Coll;
- Динамическая библиотека MPI4Type;
- Динамическая библиотека MPI5Comm;
- Модуль для работы с данными udata;
- Модуль для управления стилем вывода uprint;
- Промежуточный компонент pt4utilities, соединяющий динамическую библиотеку для параллельного программирования и группу задач по параллельному программированию задачника Programming Taskbook;

Ядро Unix Taskbook также было изменено, чтобы сделать его более совместимым с группами задач параллельного программирования и сделать интерфейс вывода более привлекательным.

Список литературы

[1] *Э., Абрамян М.* Электронный задачник по курсу «Операционные системы» / Современные информационные технологии: тенденции и перспективы развития. Материалы XXIX научной конференции / Абрамян М. Э., Ли Шэнюй. — Ростов н/Д, Таганрог: Изд-во ЮФУ. — P. 22–24.

[2] *Э, Абрамян М.* Электронный задачник по параллельному программированию на базе интерфейса MPI стандарта 2.0 // Современные информационные технологии и ИТ-образование. 2017 / Абрамян М. Э // *Том.* — Vol. 13, no. 4. С. — P. 91–104.

[3] *Абрамян, М. Э.* Инструменты и методы разработки электронных образовательных ресурсов по компьютерным наукам: монография / М. Э. Абрамян. — Ростов н/Д, Таганрог: Изд-во ЮФУ, 2018. — P. 260.

[4] *Абрамян, М. Э.* Об архитектуре универсального электронного задачника по программированию / М. Э. Абрамян // *Информатизация образования и науки.* — 2015. — no. 3 (27). — P. 134–150.

[5] *Э., Абрамян М.* Электронный задачник по курсу «Операционные системы» / Современные информационные технологии: тенденции и перспективы развития. Материалы XXIX научной конференции / Абрамян М. Э., Ли Шэнюй. — Ростов н/Д, Таганрог: Изд-во ЮФУ. — P. 9–11.

[6] *Gropp, William.* Using MPI-2: advanced features of the message-passing interface / William Gropp, Ewing Lusk, Rajeev Thakur. — MIT press, 1999.

[7] *Balaji, Pavan.* Mpi on millions of cores / Pavan Balaji, Darius Buntinas, David Goodell, William Gropp, Torsten Hoefler, Sameer Kumar, Ewing Lusk, Guillaume Mercier, Ken Raffenetti, Rajeev Thakur // *Parallel Processing Letters.* — 2009. — Vol. 19, no. 01. — Pp. 45–60.

[8] *Thakur, Rajeev.* Optimization of collective communication operations in MPICH / Rajeev Thakur, Rolf Rabenseifner, William Gropp // *The International Journal of High Performance Computing Applications.* — 2005. — Vol. 19, no. 1. — Pp. 49–66.

[9] *Gropp, William.* Using MPI: portable parallel programming with the message-passing interface / William Gropp, Ewing Lusk, Rajeev Thakur. — MIT press, 2006.

[10] *user1260391.* How to compile MPI with gcc? - Stack Overflow / user1260391 et al. — `https://stackoverflow.com/questions/11312719/how-to-compile-mpi-with-gcc`. — 2012.

[11] Open MPI Project. — mpicc(1) man page (version 3.0.6) - Open MPI, 2020.

[12] *Computing, USC Research.* Open MPI Tutorial - GitHub Pages. — `https://usc-rc.github.io/tutorials/open-mpi`.

[13] *Gropp, William.* Using MPI: portable parallel programming with the message-passing interface / William Gropp, Ewing Lusk, Rajeev Thakur. — MIT press, 2006.

[14] *Cppreference.* Filesystem library (since C++17). — 2021. — [Online; accessed 6-April-2021]. `https://en.cppreference.com/w/cpp/filesystem`.

[15] *Э, Абрамян М.* Programming Taskbook - MPI1Proc. — `https://ptaskbook.com/en/ptformpi2/mpi1proc.php`.