Федеральное государственное автономное

образовательное учреждение

высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и информационных технологий

институт

Кафедра «Информатика»

кафедра

**ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ**

Лабораторная работа No 2. Синхронизация потоков в ОС GNU/Linux

тема

Преподаватель \_\_\_\_\_\_\_\_\_ А.С. Кузнецов

подпись, дата инициалы, фамилия

Студент КИ18-17/1б 031831229 \_\_\_\_\_\_\_\_\_ В.А. Прекель

номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2019

Содержание

[Содержание 2](#_Toc21371644)

[1 Цель и задача работы 4](#_Toc21371645)

[1.1 Цель работы 4](#_Toc21371646)

[1.2 Задача работы 4](#_Toc21371647)

[2 Описание и пояснение к работе 5](#_Toc21371648)

[3 Структура проекта и листинги 5](#_Toc21371649)

[3.1 Структура проекта 5](#_Toc21371650)

[3.2 CMakeLists.txt 6](#_Toc21371651)

[3.3 Doxyfile 7](#_Toc21371652)

[3.4 Lab\_02\_Interactive/CMakeLists.txt 7](#_Toc21371653)

[3.5 Lab\_02\_Interactive/MainInteractive.c 7](#_Toc21371654)

[3.6 Lab\_02\_InteractiveLib/CMakeLists.txt 10](#_Toc21371655)

[3.7 Lab\_02\_InteractiveLib/MainWindow.c 11](#_Toc21371656)

[3.8 Lab\_02\_InteractiveLib/MainWindow.h 15](#_Toc21371657)

[3.9 Lab\_02\_InteractiveLib/ProgramQuitThread.c 17](#_Toc21371658)

[3.10 Lab\_02\_InteractiveLib/ProgramQuitThread.h 19](#_Toc21371659)

[3.11 Lab\_02\_InteractiveLib/RendererThread.c 19](#_Toc21371660)

[3.12 Lab\_02\_InteractiveLib/RendererThread.h 23](#_Toc21371661)

[3.13 Lab\_02\_Lib/CMakeLists.txt 24](#_Toc21371662)

[3.14 Lab\_02\_Lib/Fork.c 25](#_Toc21371663)

[3.15 Lab\_02\_Lib/Fork.h 26](#_Toc21371664)

[3.16 Lab\_02\_Lib/Input.c 27](#_Toc21371665)

[3.17 Lab\_02\_Lib/Input.h 28](#_Toc21371666)

[3.18 Lab\_02\_Lib/Logger.c 29](#_Toc21371667)

[3.19 Lab\_02\_Lib/Logger.h 32](#_Toc21371668)

[3.20 Lab\_02\_Lib/Philosopher.c 33](#_Toc21371669)

[3.21 Lab\_02\_Lib/Philosopher.h 34](#_Toc21371670)

[3.22 Lab\_02\_Lib/PhilosopherEatingThread.c 36](#_Toc21371671)

[3.23 Lab\_02\_Lib/PhilosopherEatingThread.h 40](#_Toc21371672)

[3.24 Lab\_02\_Lib/PhilosophersSpawnerThread.c 41](#_Toc21371673)

[3.25 Lab\_02\_Lib/PhilosophersSpawnerThread.h 43](#_Toc21371674)

[3.26 Lab\_02\_Lib/PhilosophersWaiterThread.c 44](#_Toc21371675)

[3.27 Lab\_02\_Lib/PhilosophersWaiterThread.h 45](#_Toc21371676)

[3.28 Lab\_02\_Lib/Table.c 46](#_Toc21371677)

[3.29 Lab\_02\_Lib/Table.h 48](#_Toc21371678)

[3.30 Lab\_02\_Lib/Utils.c 49](#_Toc21371679)

[3.31 Lab\_02\_Lib/Utils.h 50](#_Toc21371680)

[4 Примеры использования 52](#_Toc21371681)

[4.1 Запуск №1 (5 философов, время приёма пищи от 1000 мс до 5000 мс, время появления от 1000 мс до 3000 мс, завершение закрытием окна) 52](#_Toc21371682)

[4.2 Запуск №2 (5 философов, время приёма пищи 5000 мс, время появления от 1000 мс до 2000 мс, завершение нажатием на ESC) 56](#_Toc21371683)

[4.3 Запуск №3 (5 философов, время приёма пищи от 1000 мс до 9000 мс, время появления от 500 мс до 1500 мс, завершение нажатием на ESC) 63](#_Toc21371684)

# Цель и задача работы

## Цель работы

Изучение программных средств синхронизации потоков в ОС.

## Задача работы

Требуется разработать программу в виде Linux-приложения, для выполнения различных частей которой создаются и запускается потоки управления, а для синхронизации доступа к требуемым ресурсам используются соответствующие объекты ОС. Результат выполнения выводится на терминал/консоль.

**Вариант 1.** «Обедающие философы 1». В пансионе отдыхают и предаются размышлениям 5 философов (потоки), пронумерованные от 1 до 5. В столовой расположен круглый стол, вокруг которого расставлены 5 стульев, также пронумерованные от 1 до 5. На столе находится одна большая тарелка со спагетти, которая пополняется бесконечно, также там расставлены 5 тарелок, в которые накладывается спагетти, и 5 вилок (разделяемые ресурсы), назначение которых очевидно. Для того чтобы пообедать, философ входит в столовую и садится на стул со своим номером. При этом есть философ сможет только в том случае, если свободны две вилки – справа и слева от его тарелки. При выполнении этого условия философ поднимает одновременно обе вилки и может поглощать пищу в течение какого-то заданного времени. В противном случае, философу приходится ждать освобождения обеих вилок. Пообедав, философ кладет обе вилки на стол одновременно и уходит .Величина временного промежутка для поглощения пищи устанавливается пользователем, а появление философа в столовой является случайной величиной с равномерным законом распределения.

# Описание и пояснение к работе

Используется система сборки CMake. Проект представляет из себя две библиотеки и исполняемую программу. Программа запрашивает данные у пользователя и затем запускает окно с графической визуализацией (с помощью библиотеки SDL) стола: большие квадраты – философы, маленькие – вилки. Так же в консоль выводится информация о каждом событии вместе с информацией о столе.

Для сборки требуется пакет libsdl2-dev и CMake версии не ниже 3.0.

Команды для сборки и запуска программы:

mkdir build  
cd build  
cmake ..  
make  
./Lab\_02\_Interactive/Lab\_02\_Interactive

Дополнительно поддерживается компилятор MinGW.

# Структура проекта и листинги

## Структура проекта

* CMakeLists.txt
* Doxyfile
* Lab\_02\_Interactive
  + CMakeLists.txt
  + MainInteractive.c
* Lab\_02\_InteractiveLib
  + CMakeLists.txt
  + MainWindow.c
  + MainWindow.h
  + ProgramQuitThread.c
  + ProgramQuitThread.h
  + RendererThread.c
  + RendererThread.h
* Lab\_02\_Lib
  + CMakeLists.txt
  + Fork.c
  + Fork.h
  + Input.c
  + Input.h
  + Logger.c
  + Logger.h
  + Philosopher.c
  + Philosopher.h
  + PhilosopherEatingThread.c
  + PhilosopherEatingThread.h
  + PhilosophersSpawnerThread.c
  + PhilosophersSpawnerThread.h
  + PhilosophersWaiterThread.c
  + PhilosophersWaiterThread.h
  + Table.c
  + Table.h
  + Utils.c
  + Utils.h

## CMakeLists.txt

cmake\_minimum\_required(**VERSION 3.0**)  
project(**Lab\_02**)  
  
set(**CMAKE\_C\_STANDARD 11**)  
  
add\_subdirectory(**Lab\_02\_Lib**)  
add\_subdirectory(**Lab\_02\_Interactive**)  
add\_subdirectory(**Lab\_02\_InteractiveLib**)

## Doxyfile

Слишком длинный, содержит конфигурацию генерируемой документации, содержится в архиве.

## Lab\_02\_Interactive/CMakeLists.txt

cmake\_minimum\_required(**VERSION 3.0**)  
project(**Lab\_02\_Interactive C**)  
  
set(**CMAKE\_C\_STANDARD 11**)  
  
add\_executable(**Lab\_02\_Interactive MainInteractive.c**)  
  
**if** (**UNIX**)  
 **if** (**TARGET SDL2**)  
 target\_link\_libraries(**Lab\_02\_Interactive PRIVATE SDL2main SDL2**)  
 **else** (**TARGET SDL2**)  
 find\_package(**SDL2 REQUIRED**)  
 target\_include\_directories(  
 **Lab\_02\_Interactive PRIVATE ${SDL2\_INCLUDE\_DIRS}**)  
 target\_link\_libraries(**Lab\_02\_Interactive PRIVATE ${SDL2\_LIBRARIES}**)  
 **endif** (**TARGET SDL2**)  
**endif** (**UNIX**)  
  
**if** (**MINGW**)  
 set(**CMAKE\_C\_FLAGS** "**${CMAKE\_C\_FLAGS} -mconsole -mwindows**")  
 target\_link\_libraries(**Lab\_02\_Interactive PRIVATE mingw32**)  
**endif** (**MINGW**)  
  
target\_link\_libraries(**Lab\_02\_Interactive PRIVATE Lab\_02\_InteractiveLib**)  
target\_link\_libraries(**Lab\_02\_Interactive PRIVATE Lab\_02\_Lib**)

## Lab\_02\_Interactive/MainInteractive.c

*/// \file  
/// \brief Главная функция  
/// \details Файл с главной функцией интерактивной программы.*#include **<stdio.h>**#include **<stdbool.h>**#include **<string.h>**#ifdef **\_\_MINGW32\_\_**#include **<windows.h>**#endif  
  
#include **"Input.h"**#include **"Table.h"**#include **"Logger.h"**#include **"MainWindow.h"***/// Ширина окна***const int** SCREEN\_WIDTH = 512;  
*/// Высота окна***const int** SCREEN\_HEIGHT = 512;  
  
*/// Максимальная длина для считывания целого числа***const int** MAX\_INT\_LENGTH = 20;  
  
*/// Проверяет введённое время на неотрицательность.  
///  
/// \param time Время для проверки в миллисекундах.  
/// \return Логическое значение - результат проверки.***bool** TimeChecker(**int** time)  
{  
 **return** 0 <= time;  
}  
  
*/// Проверяет введёное число философ чтобы было не меньше двух.  
///  
/// \param philosophersCount Число философов для проверки.  
/// \return Логическое значение - результат проверки.***bool** PhilosophersCountChecker(**int** philosophersCount)  
{  
 **return** 2 <= philosophersCount;  
}  
  
*/// Главная функция программы, считывающая данные и запускающая главное окно  
/// с главным циклом.  
///  
/// \param argc Число аргументов переданное программе.  
/// \param args Массив строк аргументов.  
/// \return Возвращает результат выполнения главного цикла - 0 в случае  
/// успешного завершения, 1 в случае принудительного.***int main**(**int** argc, **char**\*\* args)  
{  
#ifdef **\_\_MINGW32\_\_** SetConsoleOutputCP(**CP\_UTF8**);  
 SetConsoleCP(**CP\_UTF8**);  
#endif  
  
 printf(**"Обозначения: Большой квадрат - философ:\n"**);  
 printf(**" - тёмно-серый - поток ещё не запущен или "  
 "уже завершён;\n"**);  
 printf(**" \_ белый - ничего не делает;\n"**);  
 printf(**" = красный - ест;\n"**);  
 printf(**" ? зелёный - ожидает.\n"**);  
 printf(**" Маленький квадрат - вилка:\n"**);  
 printf(**" , оранжевый - занята;\n"**);  
 printf(**" . светло-серый - свободна.\n"**);  
 printf(**"\n"**);  
  
 printf(**"Управление: [1-9] - отправить философа "  
 "есть;\n"**);  
 printf(**" Alt+[1-9] - прекратить приём пищи или "  
 "ожидание;\n"**);  
 printf(**" Ctrl+[1-9] - переключение метки бесконечного "  
 "приёма пищи;\n"**);  
 printf(**" Esc - выход из программы с ожиданием "  
 "завершения всех потоков.\n"**);  
 printf(**"\n\n"**);  
  
 printf(**"Минимальное количество философов 2, "  
 "желательно не больше 9, рекомендуется 5.\n"**);  
 **int** philosophersCount =  
 CycleInputInt(**"Введите кол-во философов: "**,  
 MAX\_INT\_LENGTH,  
 PhilosophersCountChecker);  
 printf(**"\n"**);  
  
 printf(**"Время вводится в миллисекундах.\n"**);  
 printf(**"Разность между верхней границей и нижней должна быть "  
 "не меньше 0 и не больше %d.\n"**,  
 **RAND\_MAX**);  
 printf(**"Время будет генерироватся в полуинтервале "  
 "[нижняя граница; верхняя граница).\n\n"**);  
  
 **int** minDurationEat;  
 **int** maxDurationEat;  
 **do** {  
 printf(**"Для того, чтобы время приёма пищы было бесконечным, "  
 "введите 0 и 0.\n"**);  
 printf(**"Для того, чтобы было постоянным, введите одинаковые "  
 "числа.\n"**);  
 minDurationEat = CycleInputInt(  
 **"Введите нижнию границу времени приёма пищи "  
 "(например 1000): "**,  
 MAX\_INT\_LENGTH, TimeChecker);  
 maxDurationEat = CycleInputInt(  
 **"Введите верхнюю границу времени приёма пищи "  
 "(например 5000): "**,  
 MAX\_INT\_LENGTH, TimeChecker);  
 } **while** (maxDurationEat < minDurationEat ||  
 maxDurationEat - minDurationEat > **RAND\_MAX**);  
 **bool** isInfinityDuration = minDurationEat == 0 && maxDurationEat == 0;  
 printf(**"\n"**);  
  
 **int** minSendIntervalDuration;  
 **int** maxSendIntervalDuration;  
 **do** {  
 printf(**"Для того, философы не появлялись автоматически, "  
 "введите 0 и 0.\n"**);  
 printf(**"Для того, чтобы было постоянным, введите одинаковые "  
 "числа.\n"**);  
 minSendIntervalDuration = CycleInputInt(  
 **"Введите нижнию границу времени между "  
 "появлениями (например 500): "**,  
 MAX\_INT\_LENGTH, TimeChecker);  
 maxSendIntervalDuration = CycleInputInt(  
 **"Введите верхнюю границу времени между "  
 "появлениями (например 1500): "**,  
 MAX\_INT\_LENGTH, TimeChecker);  
 } **while** (maxSendIntervalDuration < minSendIntervalDuration ||  
 maxSendIntervalDuration - minSendIntervalDuration > **RAND\_MAX**);  
 **bool** isAutoSpawnDisabled =  
 minSendIntervalDuration == 0 && maxSendIntervalDuration == 0;  
 printf(**"\n"**);  
  
 Table\* pTable = CreateTable(philosophersCount, minDurationEat,  
 maxDurationEat,  
 isInfinityDuration);  
  
 *//InitLogger(pTable, stdout, false, fopen("5.txt", "w+"), false);* InitLogger(pTable, **stdout**, **true**, **NULL**, **false**);  
 **LOG**(**"Введены данные, создание объектов, запуск потоков"**);  
  
 MainWindow\* pMainWindow = CreateMainWindow(  
 SCREEN\_WIDTH,  
 SCREEN\_HEIGHT,  
 pTable,  
 minSendIntervalDuration,  
 maxSendIntervalDuration,  
 isAutoSpawnDisabled);  
  
 InitVideoMainWindow(pMainWindow);  
  
 StartThreadsMainWindow(pMainWindow);  
  
 **LOG**(**"Запуск главного цикла"**);  
 **return** MainCycleMainWindow(pMainWindow);  
}

## Lab\_02\_InteractiveLib/CMakeLists.txt

cmake\_minimum\_required(**VERSION 3.0**)  
project(**Lab\_02\_InteractiveLib C**)  
  
set(**CMAKE\_C\_STANDARD 11**)  
  
add\_library(**Lab\_02\_InteractiveLib  
 RendererThread.c RendererThread.h  
 ProgramQuitThread.c ProgramQuitThread.h  
 MainWindow.c MainWindow.h**)  
  
**if** (**UNIX**)  
 **if** (**TARGET SDL2**)  
 target\_link\_libraries(**Lab\_02\_InteractiveLib PRIVATE SDL2main SDL2**)  
 **else** (**TARGET SDL2**)  
 find\_package(**SDL2 REQUIRED**)  
 target\_include\_directories(  
 **Lab\_02\_InteractiveLib PRIVATE ${SDL2\_INCLUDE\_DIRS}**)  
 target\_link\_libraries(**Lab\_02\_InteractiveLib PRIVATE ${SDL2\_LIBRARIES}**)  
 **endif** (**TARGET SDL2**)  
**endif** (**UNIX**)  
  
**if** (**MINGW**)  
 set(**CMAKE\_C\_FLAGS** "**${CMAKE\_C\_FLAGS} -mconsole -mwindows**")  
 target\_link\_libraries(**Lab\_02\_InteractiveLib PRIVATE mingw32 SDL2main SDL2**)  
**endif** (**MINGW**)  
  
target\_link\_libraries(**Lab\_02\_InteractiveLib PRIVATE Lab\_02\_Lib**)  
  
target\_include\_directories(  
 **Lab\_02\_InteractiveLib PUBLIC ${CMAKE\_CURRENT\_SOURCE\_DIR}**)

## Lab\_02\_InteractiveLib/MainWindow.c

*/// \file  
/// \brief Реализация функций из MainWindow.h  
/// \details Реализация функций из MainWindow.h.*#include **<malloc.h>***//#include <sysexits.h>*#include **"Logger.h"**#include **"MainWindow.h"**#include **"ProgramQuitThread.h"**MainWindow\* CreateMainWindow(**int** screenWidth, **int** screenHeight, Table\* pTable,  
 **int** minSendIntervalDuration,  
 **int** maxSendIntervalDuration,  
 **bool** isAutoSpawnDisabled)  
{  
 MainWindow\* pMainWindow = (MainWindow\*) malloc(**sizeof**(MainWindow));  
 **FAILURE\_IF\_NULLPTR**(pMainWindow);  
  
 pMainWindow->ScreenWidth = screenWidth;  
 pMainWindow->ScreenHeight = screenHeight;  
  
 pMainWindow->pTable = pTable;  
  
 pMainWindow->MinSendIntervalDuration = minSendIntervalDuration;  
 pMainWindow->MaxSendIntervalDuration = maxSendIntervalDuration;  
  
 pMainWindow->IsAutoSpawnDisabled = isAutoSpawnDisabled;  
  
 pMainWindow->MainThreadId = pthread\_self();  
  
 **return** pMainWindow;  
}  
  
**int** InitVideoMainWindow(MainWindow\* pMainWindow)  
{  
 **LOG**(**"Инициализация SDL"**);  
 **if** (SDL\_Init(**SDL\_INIT\_VIDEO**) != 0)  
 {  
 fprintf(**stderr**, **"Не удаётся проинициализировать: %s\n"**,  
 SDL\_GetError());  
 **return** 1;  
 }  
  
 SDL\_version compiledVersion;  
 SDL\_version linkedVersion;  
 **SDL\_VERSION**(&compiledVersion);  
 SDL\_GetVersion(&linkedVersion);  
 **LOG**(**"Скомпилированная версия SDL %d.%d.%d"**,  
 compiledVersion.major,  
 compiledVersion.minor,  
 compiledVersion.patch);  
 **LOG**(**"Скомпонованная версия SDL %d.%d.%d"**,  
 linkedVersion.major,  
 linkedVersion.minor,  
 linkedVersion.patch);  
  
 **LOG**(**"Создание окна"**);  
 pMainWindow->pWindow = SDL\_CreateWindow(**"Обедающие философы"**,  
 **SDL\_WINDOWPOS\_UNDEFINED**,  
 **SDL\_WINDOWPOS\_UNDEFINED**,  
 pMainWindow->ScreenWidth,  
 pMainWindow->ScreenHeight,  
 ***SDL\_WINDOW\_SHOWN***);  
  
 **if** (pMainWindow->pWindow == **NULL**)  
 {  
 fprintf(**stderr**, **"Не удаётся создать окно: %s\n"**,  
 SDL\_GetError());  
 **return** 1;  
 }  
  
 **LOG**(**"Создание отрисовщика"**);  
 pMainWindow->pRenderer = SDL\_CreateRenderer(  
 pMainWindow->pWindow,  
 -1,  
 ***SDL\_RENDERER\_ACCELERATED***);  
 **if** (pMainWindow->pRenderer == **NULL**)  
 {  
 fprintf(**stderr**, **"Не удаётся создать отрисовщик: %s\n"**,  
 SDL\_GetError());  
 **return** 1;  
 }  
  
 **return** 0;  
}  
  
**void** StartThreadsMainWindow(MainWindow\* pMainWindow)  
{  
 LOG(**"Запуск потока отрисовщика"**);  
 pMainWindow->pRendererThreadOptions =  
 CreateRendererThreadOptions(pMainWindow->pTable,  
 pMainWindow->pRenderer,  
 pMainWindow->ScreenWidth,  
 pMainWindow->ScreenHeight);  
 pthread\_create(&pMainWindow->RendererThreadId,  
 NULL,  
 RendererThread,  
 pMainWindow->pRendererThreadOptions);  
  
 LOG(**"Запуск потоков-философов"**);  
 StartAllThreads(pMainWindow->pTable);  
  
 **if** (!pMainWindow->IsAutoSpawnDisabled)  
 {  
 LOG(**"Запуск потока, отправляющий философов есть"**);  
 pMainWindow->pPhilosophersSpawnerThreadOptions =  
 CreatePhilosophersSpawnerThreadOptions(  
 pMainWindow->pTable,  
 pMainWindow->MinSendIntervalDuration,  
 pMainWindow->MaxSendIntervalDuration);  
 pthread\_create(&pMainWindow->PhilosophersSpawnerThreadId,  
 NULL,  
 PhilosophersSpawnerThread,  
 pMainWindow->pPhilosophersSpawnerThreadOptions);  
 }  
  
 pMainWindow->pTable->IsEatingStarted = true;  
}  
  
**int** MainCycleMainWindow(MainWindow\* pMainWindow)  
{  
 LOG(**"Запуск главного цикла"**);  
  
 SDL\_Event event;  
  
 **while** (SDL\_WaitEvent(&event) != 0)  
 {  
 **if** (event.type == ***SDL\_QUIT***)  
 {  
 LOG(**"Главный цикл завершён событием"**);  
  
 **if** (!pMainWindow->pTable->IsEatingEnded)  
 {  
 pthread\_mutex\_lock(pMainWindow->pTable->pMutex);  
 LOG(**"Событие выхода из программы было послано без завершения "  
 "потоков и очистки"**);  
 LOG(**"Завершение программы с кодом 1 (EXIT\_FAILURE)"**);  
 **return** EXIT\_FAILURE;  
 }  
  
 StopThreadsMainWindow(pMainWindow);  
  
 QuitVideoMainWindow(pMainWindow);  
  
 LOG(**"Завершение программы с кодом 0 (EXIT\_SUCCESS)"**);  
  
 DestroyTable(pMainWindow->pTable);  
 DestroyMainWindow(pMainWindow);  
  
 fflush(stdout);  
 **return** EXIT\_SUCCESS;  
 }  
  
 **if** (event.type == ***SDL\_KEYDOWN***)  
 {  
 **if** (event.key.keysym.sym == ***SDLK\_ESCAPE*** &&  
 !pMainWindow->pTable->IsEatingMustEnd)  
 {  
 LOG(**"Начато завершение программы"**);  
  
 LOG(**"Запуск потока, который завершает потоки"**);  
  
 ProgramQuitThreadOptions\* pProgramQuitThreadOptions =  
 CreateProgramQuitThreadOptions(pMainWindow);  
 pthread\_t programQuitThreadId;  
 pthread\_create(&programQuitThreadId,  
 NULL,  
 ProgramQuitThread,  
 pProgramQuitThreadOptions);  
 }  
 **if** (event.key.keysym.mod & KMOD\_CTRL)  
 {  
 **char** button = event.key.keysym.sym;  
 **if** (**'1'** <= button && button <= **'9'**)  
 {  
 **int** philosopherId = (**int**) (button - **'0'**);  
 **int** phIndex = philosopherId - 1;  
 **if** (philosopherId <=  
 pMainWindow->pTable->PhilosophersCount)  
 {  
 pthread\_mutex\_lock(pMainWindow->pTable->pMutex);  
 Philosopher\* ph =  
 pMainWindow->pTable->ppPhilosophers[phIndex];  
 **if** (!ph->IsEating)  
 {  
 LOG(**"Переключение метки бесконечного времени "  
 "приёма пищи для философа с номером %d"**,  
 ph->PhilosopherId);  
 ph->IsInfinityDuration = !ph->IsInfinityDuration;  
 }  
 **else** {  
 LOG(**"Невозможно переключение метки бесконечного "  
 "времени приёма пищи для философа с "  
 "номером %d"**,  
 ph->PhilosopherId);  
 }  
 pthread\_mutex\_unlock(pMainWindow->pTable->pMutex);  
 }  
 }  
 }  
 **else if** (event.key.keysym.mod & KMOD\_ALT)  
 {  
 **char** button = event.key.keysym.sym;  
 **if** (**'1'** <= button && button <= **'9'**)  
 {  
 **int** philosopherId = (**int**) (button - **'0'**);  
 **int** phIndex = philosopherId - 1;  
 **if** (philosopherId <=  
 pMainWindow->pTable->PhilosophersCount)  
 {  
 Philosopher\* ph =  
 pMainWindow->pTable->ppPhilosophers[phIndex];  
 InterruptEating(ph, pMainWindow->pTable->pMutex);  
 }  
 }  
 }  
 **else if** (!pMainWindow->pTable->IsEatingMustEnd)  
 {  
 **char** button = event.key.keysym.sym;  
 **if** (**'1'** <= button && button <= **'9'**)  
 {  
 **int** philosopherId = (**int**) (button - **'0'**);  
 **int** phIndex = philosopherId - 1;  
 **if** (philosopherId <=  
 pMainWindow->pTable->PhilosophersCount)  
 {  
 Philosopher\* ph =  
 pMainWindow->pTable->ppPhilosophers[phIndex];  
 LOG(**"Философ с номером %d вручную отправлен есть"**,  
 ph->PhilosopherId);  
 SpawnPhilosopher(pMainWindow->pTable, ph);  
 }  
 }  
 }  
 }  
 }  
  
 LOG(**"Главный цикл завершён по неизвестной ошибке: %s"**, SDL\_GetError());  
 LOG(**"Завершение программы с кодом 70 (EX\_SOFTWARE)"**);  
  
 *//return EX\_SOFTWARE;* **return** 70;  
}  
  
**void** StopThreadsMainWindow(MainWindow\* pMainWindow)  
{  
 LOG(**"Завершение программы, завершение остальных потоков"**);  
  
 LOG(**"Ожидание завершения отрисовщика"**);  
 pthread\_join(pMainWindow->RendererThreadId, NULL);  
 DestroyRendererThreadOptions(pMainWindow->pRendererThreadOptions);  
}  
  
**int** QuitVideoMainWindow(MainWindow\* pMainWindow)  
{  
 LOG(**"Очистка и завершение отрисовщика, окна и SDL"**);  
  
 SDL\_DestroyRenderer(pMainWindow->pRenderer);  
  
 SDL\_DestroyWindow(pMainWindow->pWindow);  
  
 SDL\_Quit();  
  
 **return** 0;  
}  
  
**void** DestroyMainWindow(MainWindow\* pMainWindow)  
{  
 free(pMainWindow);  
}

## Lab\_02\_InteractiveLib/MainWindow.h

*/// \file  
/// \brief Главное окно  
/// \details Главное окно, функции создания, уничтожения, инициализации,  
/// главного цикла итд.*#ifndef **MAINWINDOW\_H**#define **MAINWINDOW\_H**#include **<pthread.h>**#include **<stdbool.h>**#include **<time.h>**#include **<SDL.h>**#include **"Table.h"**#include **"PhilosophersSpawnerThread.h"**#include **"RendererThread.h"***/// \struct MainWindow  
///  
/// Главное окно.***typedef struct**{  
 */// Ширина окна* **int** ScreenWidth;  
 */// Высота окна* **int** ScreenHeight;  
  
 */// Указатель на окно* SDL\_Window\* pWindow;  
 */// Указатель на отрисовщик* SDL\_Renderer\* pRenderer;  
  
 */// Указатель на параметры запуска потока, отправляющего философов* PhilosophersSpawnerThreadOptions\* pPhilosophersSpawnerThreadOptions;  
 */// Идентификатор потока, отправляющий философов* pthread\_t PhilosophersSpawnerThreadId;  
  
 */// Указатель на параметры запуска потока отрисовщика* RendererThreadOptions\* pRendererThreadOptions;  
 */// Идентификатор потока отрисовщика* pthread\_t RendererThreadId;  
  
 */// Идентификатор главного потока* pthread\_t MainThreadId;  
  
 */// Выключена ли автоматическая отправка философов* **bool** IsAutoSpawnDisabled;  
  
 */// Указатель на стол* Table\* pTable;  
  
 */// Нижная граница случайного времени между отправками в миллисекундах* **int** MinSendIntervalDuration;  
 */// Верхняя граница случайного времени между отправками в миллисекундах* **int** MaxSendIntervalDuration;  
} MainWindow;  
  
*/// Создаёт главное окно. Требуется очистка с помощью DestroyMainWindow.  
///  
/// \param screenWidth Ширина окна.  
/// \param screenHeight Высота окна.  
/// \param pTable Указатель на стол.  
/// \param minSendIntervalDuration Нижная граница случайного времени между  
/// отправками в миллисекундах.  
/// \param maxSendIntervalDuration Верхняя граница случайного времени между  
/// отправками в миллисекундах.  
/// \param isAutoSpawnDisabled Выключена ли автоматическая отправка философов.  
/// \return Указатель на созданное главное окно.*MainWindow\* CreateMainWindow(**int** screenWidth, **int** screenHeight, Table\* pTable,  
 **int** minSendIntervalDuration,  
 **int** maxSendIntervalDuration,  
 **bool** isAutoSpawnDisabled);  
  
*/// Инициализирует SDL, окно, отрисовщик  
///  
/// \param pMainWindow Указатель на главное окно.  
/// \return 0 в случае успеха, 1 если не удалось что-либо проинициализировать***int** InitVideoMainWindow(MainWindow\* pMainWindow);  
  
*/// Запускает потоки: поток отрисовщика и потоки философов  
///  
/// \param pMainWindow Указатель на главное окно.***void** StartThreadsMainWindow(MainWindow\* pMainWindow);  
  
*/// Главный цикл приложения. Обрабатывает события.  
///  
/// \param pMainWindow Указатель на главное окно.  
/// \return 0 в случае успешного завершения, 1 в случае принудительного,  
/// 70 в случае неизвестной ошибки***int** MainCycleMainWindow(MainWindow\* pMainWindow);  
  
*/// Завершает потоки (поток отрисовщика).  
///  
/// \param pMainWindow Указатель на главное окно.***void** StopThreadsMainWindow(MainWindow\* pMainWindow);  
  
*/// Завершает SDL, уничтожает окно и отрисовщик.  
///  
/// \param pMainWindow Указатель на главное окно.  
/// \return 0***int** QuitVideoMainWindow(MainWindow\* pMainWindow);  
  
*/// Уничтожает главное окно.  
///  
/// \param pMainWindow Указатель на главное окно.***void** DestroyMainWindow(MainWindow\* pMainWindow);  
  
#endif *//MAINWINDOW\_H*

## Lab\_02\_InteractiveLib/ProgramQuitThread.c

*/// \file  
/// \brief Реализация функций из ProgramQuitThread.h  
/// \details Реализация функций из ProgramQuitThread.h.*#include **<malloc.h>**#include **"Logger.h"**#include **"PhilosophersWaiterThread.h"**#include **"ProgramQuitThread.h"**ProgramQuitThreadOptions\* CreateProgramQuitThreadOptions(  
 MainWindow\* pMainWindow)  
{  
 ProgramQuitThreadOptions\* pOptions =  
 (ProgramQuitThreadOptions\*)  
 malloc(**sizeof**(ProgramQuitThreadOptions));  
 **FAILURE\_IF\_NULLPTR**(pOptions);  
  
 pOptions->pMainWindow = pMainWindow;  
  
 pOptions->pMutex = pMainWindow->pTable->pMutex;  
  
 **return** pOptions;  
}  
  
**void** DestroyProgramQuitThreadOptions(ProgramQuitThreadOptions\* pOptions)  
{  
 free(pOptions);  
}  
  
**void**\* ProgramQuitThread(**void**\* pProgramQuitThreadOptions)  
{  
 ProgramQuitThreadOptions\* pOptions =  
 (ProgramQuitThreadOptions\*) pProgramQuitThreadOptions;  
  
 **LOG**(**"Запуск потока"**);  
  
 pthread\_mutex\_lock(pOptions->pMainWindow->pTable->pMutex);  
 pOptions->pMainWindow->pTable->IsEatingMustEnd = **true**;  
  
 pthread\_mutex\_unlock(pOptions->pMainWindow->pTable->pMutex);  
  
 **LOG**(**"Запуск потока, который завершает потоки филосософ"**);  
 PhilosophersWaiterThreadOptions\*  
 pPhilosophersWaiterThreadOptions =  
 CreatePhilosophersWaiterThreadOptions(  
 pOptions->pMainWindow->pTable);  
 pthread\_t philosophersWaiterThreadId;  
 pthread\_create(&philosophersWaiterThreadId, **NULL**,  
 PhilosophersWaiterThread,  
 pPhilosophersWaiterThreadOptions);  
  
 **if** (!pOptions->pMainWindow->IsAutoSpawnDisabled)  
 {  
 **LOG**(**"Принудительная остановка потока-cпавнера"**);  
 sem\_post(pOptions->pMainWindow->pPhilosophersSpawnerThreadOptions->OnSemQuit);  
 }  
  
 **LOG**(**"Ожидание завершения потока, который завершает потоки филосософ"**);  
 pthread\_join(philosophersWaiterThreadId, **NULL**);  
 DestroyPhilosophersWaiterThreadOptions(pPhilosophersWaiterThreadOptions);  
  
 **if** (!pOptions->pMainWindow->IsAutoSpawnDisabled)  
 {  
 **LOG**(**"Ожидание завершения потока-cпавнера"**);  
 pthread\_join(pOptions->pMainWindow->PhilosophersSpawnerThreadId, **NULL**);  
 DestroyPhilosophersSpawnerThreadOptions(  
 pOptions->pMainWindow->pPhilosophersSpawnerThreadOptions);  
 }  
  
 **LOG**(**"Отправление события выхода главному циклу"**);  
 SDL\_Event event;  
 event.type = ***SDL\_QUIT***;  
 SDL\_PushEvent(&event);  
  
 DestroyProgramQuitThreadOptions(pOptions);  
 **LOG**(**"Завершение потока"**);  
  
 **return** pOptions;  
}

## Lab\_02\_InteractiveLib/ProgramQuitThread.h

*/// \file  
/// \brief Поток, завершающий программу  
/// \details Поток, завершающий программу, его параметры запуска итд.*#ifndef **PROGRAMQUITTHREAD\_H**#define **PROGRAMQUITTHREAD\_H**#include **<pthread.h>**#include **"MainWindow.h"***/// \struct ProgramQuitThreadOptions  
///  
/// Параметры запуска потока ProgramQuitThread.***typedef struct**{  
 */// Указатель на главное окно* MainWindow\* pMainWindow;  
 */// Указатель на главный мьютекс* pthread\_mutex\_t\* pMutex;  
} ProgramQuitThreadOptions;  
  
*/// Создаёт параметры запуска потока. Требуется очистка с  
/// помощью DestroyProgramQuitThreadOptions.  
///  
/// \param pMainWindow Указатель на главное окно.  
/// \return Созданые параметры запуска.*ProgramQuitThreadOptions\* CreateProgramQuitThreadOptions(  
 MainWindow\* pMainWindow);  
  
*/// Уничтожает параметры запуска потока.  
///  
/// \param pOptions***void** DestroyProgramQuitThreadOptions(ProgramQuitThreadOptions\* pOptions);  
  
*/// Функция потока, завершающего программу.  
///  
/// \param pProgramQuitThreadOptions Указатель на параметры запуска.  
/// \return Указатель на параметры, с которыми был запущен.***void**\* ProgramQuitThread(**void**\* pProgramQuitThreadOptions);  
  
#endif *//PROGRAMQUITTHREAD\_H*

## Lab\_02\_InteractiveLib/RendererThread.c

*/// \file  
/// \brief Реализация функций из RendererThread.h  
/// \details Реализация функций из RendererThread.h.*#include **<malloc.h>**#include **<math.h>**#include **<SDL.h>**#include **"Logger.h"**#include **"RendererThread.h"***/// Чёрный цвет (полная прозрачность)*#define **ZERO\_RGBA** 0  
*/// Белый цвет (полная непрозрачность)*#define **FULL\_RGBA** 255  
*/// Красная, зелёная, синяя составляющая цвета философа без потока*#define **THREAD\_NOT\_RUNNING\_RGB** 64  
*/// Красная составляющая цвета философа который ест*#define **EATING\_R** 255  
*/// Зелёная и синяя составляющая цвета философа который ест*#define **EATING\_GB** 64  
*/// Красная составляющая цвета философа который ожидает*#define **WAITING\_R** 32  
*/// Зелёная составляющая цвета философа который ожидает*#define **WAITING\_G** 255  
*/// Синяя составляющая цвета философа который ожидает*#define **WAITING\_B** 64  
*/// Красная, зелёная, синяя составляющая цвета философа который свободен*#define **FREE\_RGB** 255  
*/// Красная составляющая цвета занятой вилки*#define **USED\_R** 255  
*/// Зелёня составляющая цвета занятой вилки*#define **USED\_G** 128  
*/// Синяя составляющая цвета занятой вилки*#define **USED\_B** 64  
*/// Красная, зелёная, синяя составляющая цвета свободной вилки*#define **NOT\_USED\_RGB** 200  
  
*/// Радиус от центра экрана до середины квадрата философа*#define **PHILOSOPHER\_RADIUS** 200  
*/// Ширина квадрата философа*#define **PHILOSOPHER\_WIDTH** 60  
*/// Радиус от центра экрана до середины квадрата вилки*#define **FORK\_RADIUS** 160  
*/// Ширина квадрата вилки*#define **FORK\_WIDTH** 30  
  
*/// Угол на полный оборот*#define **FULL\_ANGLE** 360.0  
*/// Развёрнутый угол*#define **WIDE\_ANGLE** 180.0  
*/// Прямой угол*#define **RIGHT\_ANGLE** 90.0  
  
*/// Число миллисекунд на 1 кадр, если в 1 секунде 60 кадров*#define **VSYNCMS** 16  
  
RendererThreadOptions\* CreateRendererThreadOptions(  
 Table\* pTable, SDL\_Renderer\* pRenderer,  
 **int** screenWidth, **int** screenHeight)  
{  
 RendererThreadOptions\* pOptions =  
 (RendererThreadOptions\*) malloc(**sizeof**(RendererThreadOptions));  
 **FAILURE\_IF\_NULLPTR**(pOptions);  
  
 pOptions->pTable = pTable;  
 pOptions->pRenderer = pRenderer;  
 pOptions->ScreenWidth = screenWidth;  
 pOptions->ScreenHeight = screenHeight;  
  
 pOptions->pMutex = pTable->pMutex;  
  
 **return** pOptions;  
}  
  
**void** DestroyRendererThreadOptions(RendererThreadOptions\* pOptions)  
{  
 free(pOptions);  
}  
  
*/// Вычисляет X-координату сдвинутого на заданный радиус и угол центра экрана.  
///  
/// \param screenWidth Ширина окна.  
/// \param angle Угол.  
/// \param r Радиус.  
/// \return Вычисляемая координата.***static inline int** CenterCircleX(**int** screenWidth, **double** angle, **double** r)  
{  
 **return** screenWidth / 2 + (**int**) (cos(angle \* **M\_PI** / **WIDE\_ANGLE**) \* r);  
}  
  
*/// Вычисляет Y-координату сдвинутого на заданный радиус и угол центра экрана.  
///  
/// \param screenHeight Высота окна.  
/// \param angle Угол.  
/// \param r Радиус.  
/// \return Вычисляемая координата.***static inline int** CenterCircleY(**int** screenHeight, **double** angle, **double** r)  
{  
 **return** screenHeight / 2 + (**int**) (sin(angle \* **M\_PI** / **WIDE\_ANGLE**) \* r);  
}  
  
**void** DrawSquare(  
 SDL\_Renderer\* pRenderer, **int** screenWidth, **int** screenHeight,  
 **int** width, **int** r, **double** angle)  
{  
 SDL\_Rect rect = {  
 CenterCircleX(screenWidth, angle, r) - width / 2,  
 CenterCircleY(screenHeight, angle, r) - width / 2,  
 width,  
 width};  
 SDL\_RenderFillRect(pRenderer, &rect);  
}  
  
**void**\* RendererThread(**void**\* pRendererThreadOptions)  
{  
 **LOG**(**"Запуск потока"**);  
  
 RendererThreadOptions\* pOptions = (RendererThreadOptions\*)  
 pRendererThreadOptions;  
  
 **unsigned int** ticks1 = SDL\_GetTicks();  
  
 **while** (**true**)  
 {  
 **if** (pOptions->pTable->IsEatingEnded) **break**;  
  
 SDL\_SetRenderDrawColor(pOptions->pRenderer,  
 **ZERO\_RGBA**, **ZERO\_RGBA**, **ZERO\_RGBA**, **ZERO\_RGBA**);  
 SDL\_RenderClear(pOptions->pRenderer);  
 SDL\_SetRenderDrawColor(pOptions->pRenderer,  
 **FULL\_RGBA**, **FULL\_RGBA**, **FULL\_RGBA**, **FULL\_RGBA**);  
  
  
 **for** (**int** i = 0; i < pOptions->pTable->PhilosophersCount; i++)  
 {  
 **if** (!pOptions->pTable->ppPhilosophers[i]->IsThreadRunning)  
 {  
 SDL\_SetRenderDrawColor(pOptions->pRenderer,  
 **THREAD\_NOT\_RUNNING\_RGB**,  
 **THREAD\_NOT\_RUNNING\_RGB**,  
 **THREAD\_NOT\_RUNNING\_RGB**,  
 **FULL\_RGBA**);  
 }  
 **else if** (pOptions->pTable->ppPhilosophers[i]->IsEating)  
 {  
 SDL\_SetRenderDrawColor(pOptions->pRenderer,  
 **EATING\_R**, **EATING\_GB**, **EATING\_GB**,  
 **FULL\_RGBA**);  
 }  
 **else if** (pOptions->pTable->ppPhilosophers[i]->IsWaiting)  
 {  
 SDL\_SetRenderDrawColor(pOptions->pRenderer,  
 **WAITING\_R**, **WAITING\_G**, **WAITING\_B**,  
 **FULL\_RGBA**);  
 }  
 **else if** (!pOptions->pTable->ppPhilosophers[i]->IsEating)  
 {  
 SDL\_SetRenderDrawColor(pOptions->pRenderer,  
 **FREE\_RGB**, **FREE\_RGB**, **FREE\_RGB**,  
 **FULL\_RGBA**);  
 }  
  
 **double** angle =  
 **FULL\_ANGLE** / pOptions->pTable->PhilosophersCount \* i -  
 **RIGHT\_ANGLE**;  
  
 DrawSquare(pOptions->pRenderer,  
 pOptions->ScreenWidth,  
 pOptions->ScreenHeight,  
 **PHILOSOPHER\_WIDTH**,  
 **PHILOSOPHER\_RADIUS**,  
 angle);  
 }  
  
  
 **for** (**int** i = 0; i < pOptions->pTable->PhilosophersCount; i++)  
 {  
 **if** (pOptions->pTable->ppForks[i]->IsInUse)  
 {  
 SDL\_SetRenderDrawColor(pOptions->pRenderer,  
 **USED\_R**, **USED\_G**, **USED\_B**, **FULL\_RGBA**);  
 }  
 **else** {  
 SDL\_SetRenderDrawColor(pOptions->pRenderer,  
 **NOT\_USED\_RGB**, **NOT\_USED\_RGB**,  
 **NOT\_USED\_RGB**, **FULL\_RGBA**);  
 }  
  
 **double** angle =  
 **FULL\_ANGLE** / pOptions->pTable->PhilosophersCount \* i -  
 (**RIGHT\_ANGLE** - (**FULL\_ANGLE** /  
 (pOptions->pTable->PhilosophersCount \*  
 2)));  
  
 DrawSquare(pOptions->pRenderer,  
 pOptions->ScreenWidth,  
 pOptions->ScreenHeight,  
 **FORK\_WIDTH**,  
 **FORK\_RADIUS**,  
 angle);  
 }  
  
 **unsigned int** frameMs = SDL\_GetTicks() - ticks1;  
  
 ticks1 = SDL\_GetTicks();  
  
 SDL\_RenderPresent(pOptions->pRenderer);  
  
 **if** (frameMs < **VSYNCMS**) SDL\_Delay(**VSYNCMS** - frameMs);  
 }  
  
 **LOG**(**"Завершение потока"**);  
  
 **return NULL**;  
}

## Lab\_02\_InteractiveLib/RendererThread.h

*/// \file  
/// \brief Поток отрисовщика  
/// \details Поток отрисовщикка, параметры его запуска итд.*#ifndef **RENDERERTHREAD\_H**#define **RENDERERTHREAD\_H**#include **<SDL.h>**#include **"Table.h"***/// \struct  
///  
/// Параметры запуска RendererThread.***typedef struct**{  
 */// Указатель на стол* Table\* pTable;  
 */// Указатель на отрисовщик* SDL\_Renderer\* pRenderer;  
 */// Ширина окна* **int** ScreenWidth;  
 */// Высота окна* **int** ScreenHeight;  
 */// Указатель на главный мьютекс* pthread\_mutex\_t\* pMutex;  
} RendererThreadOptions;  
  
*/// Создаёт параметры запуска потока. Требуется очистка с  
/// помощью DestroyRendererThreadOptions.  
///  
/// \param pTable Указатель на стол.  
/// \param pRenderer Указатель на отрисовщик.  
/// \param screenWidth Ширина окна.  
/// \param screenHeight Высота окна.  
/// \return Указатель на созданные параметры потока.*RendererThreadOptions\* CreateRendererThreadOptions(  
 Table\* pTable, SDL\_Renderer\* pRenderer,  
 **int** screenWidth, **int** screenHeight);  
  
*/// Уничтожает параметры потока.  
///  
/// \param pOptions Указатель на парамеры потока.***void** DestroyRendererThreadOptions(RendererThreadOptions\* pOptions);  
  
*/// Функция потока отрисовщика. Цикл, рисующий состояние стола примерно  
/// 60 раз в секунду. Тёмно-серый цвет - философ с незапущенным потоком,  
/// красный - философ который ест, зелёный - ожидает, белый - свободен;  
/// светло-серый - свободная вилка, оранжевый - занятая вилка.  
///  
/// \param pRendererThreadOptions Указатель на параметры запуска потока.  
/// \return Указатель на параметры с которыми был запущен.***void**\* RendererThread(**void**\* pRendererThreadOptions);  
  
#endif *//RENDERERTHREAD\_H*

## Lab\_02\_Lib/CMakeLists.txt

cmake\_minimum\_required(**VERSION 3.0**)  
project(**Lab\_02\_Lib C**)  
  
set(**CMAKE\_C\_STANDARD 11**)  
  
add\_library(**Lab\_02\_Lib  
 Input.c Input.h  
 Philosopher.c Philosopher.h  
 Fork.c Fork.h  
 Table.c Table.h  
 Utils.c Utils.h  
 PhilosopherEatingThread.c PhilosopherEatingThread.h  
 PhilosophersSpawnerThread.c PhilosophersSpawnerThread.h  
 Logger.c Logger.h  
 PhilosophersWaiterThread.c PhilosophersWaiterThread.h**)  
  
target\_link\_libraries(**Lab\_02\_Lib PRIVATE pthread**)  
  
target\_link\_libraries(**Lab\_02\_Lib PRIVATE m**)  
  
target\_include\_directories(**Lab\_02\_Lib PUBLIC ${CMAKE\_CURRENT\_SOURCE\_DIR}**)

## Lab\_02\_Lib/Fork.c

*/// \file  
/// \brief Реализация функций из Fork.h  
/// \details Реализация функций из Fork.h.*#include **<malloc.h>**#include **<pthread.h>**#include **<stdio.h>**#include **"Fork.h"**#include **"Logger.h"**Fork\* CreateFork(**int** id)  
{  
 Fork\* pFork = (Fork\*) malloc(**sizeof**(Fork));  
 **FAILURE\_IF\_NULLPTR**(pFork);  
  
 pFork->ForkId = id;  
 pFork->IsInUse = **false**;  
  
 pFork->CondSignalOnRelease = (pthread\_cond\_t\*) malloc(  
 **sizeof**(pthread\_cond\_t));  
 **FAILURE\_IF\_NULLPTR**(pFork->CondSignalOnRelease);  
 pthread\_cond\_init(pFork->CondSignalOnRelease, **NULL**);  
  
 **return** pFork;  
}  
  
**void** TakeOnFork(Fork\* pFork)  
{  
 pFork->IsInUse = **true**;  
  
 **LOG**(**"Занятие вилки с номером %d"**, pFork->ForkId);  
}  
  
**void** TakeOffFork(Fork\* pFork)  
{  
 pFork->IsInUse = **false**;  
  
 pthread\_cond\_signal(pFork->CondSignalOnRelease);  
  
 **LOG**(**"Освобождение вилки с номером %d"**, pFork->ForkId);  
}  
  
**void** DestroyFork(Fork\* pFork)  
{  
 pthread\_cond\_destroy(pFork->CondSignalOnRelease);  
 free(pFork->CondSignalOnRelease);  
 free(pFork);  
}

## Lab\_02\_Lib/Fork.h

*/// \file  
/// \brief Вилка  
/// \details Вилка, функции для её создания, уничтожения и взаимодействия*#ifndef **FORK\_H**#define **FORK\_H**#include **<stdbool.h>**#include **<pthread.h>**#include **<semaphore.h>***/// \struct Fork  
///  
/// Вилка.***typedef struct**{  
 */// Номер вилки* **int** ForkId;  
 */// Занята ли вилка* **bool** IsInUse;  
 */// Указатель на условную переменную сигнализирующую о том, что вилка  
 /// освободилась* pthread\_cond\_t\* CondSignalOnRelease;  
} Fork;  
  
*/// Создаёт вилку. Требуется очистка с помощью DestroyFork.  
///  
/// \param id Номер вилки.  
/// \return Указатель на созданную вилку.*Fork\* CreateFork(**int** id);  
  
*/// Взять вилку. Главный мьютекс должен быть заблокирован.  
///  
/// \param pFork Указатель на вилку, которую нужно взять.***void** TakeOnFork(Fork\* pFork);  
  
*/// Положить вилку. Главный мьютекс должен быть заблокирован.  
///  
/// \param pFork Указатель на вилку, которую требуется положить.***void** TakeOffFork(Fork\* pFork);  
  
*/// Уничтожает вилку.  
///  
/// \param pFork Указатель на вилку, которую требудется уничтожить.***void** DestroyFork(Fork\* pFork);  
  
#endif *//FORK\_H*

## Lab\_02\_Lib/Input.c

*/// \file  
/// \brief Реализация функций из Input.h  
/// \details Реализация функций из Input.h.*#include **<stdbool.h>**#include **<stdio.h>**#include **<string.h>**#include **<stdlib.h>**#include **<malloc.h>**#include **"Input.h"**#include **"Utils.h"  
  
int** InputLine(**char**\* stringToInput, **int** maxStringLength)  
{  
 **unsigned int** errorCode = 0;  
 **unsigned long** stringLength = 0;  
 **while** (errorCode == 0 || errorCode == 1)  
 {  
 **char**\* fgetsReturn = fgets(stringToInput, maxStringLength, **stdin**);  
  
 **int** isEof = feof(**stdin**);  
 **int** isErr = ferror(**stdin**);  
  
 **if** (isEof == 1)  
 {  
 errorCode |= 2u;  
 }  
 **if** (isErr == 1)  
 {  
 errorCode |= 4u;  
 }  
 **if** (fgetsReturn == **NULL**)  
 {  
 errorCode |= 8u;  
 }  
 **if** (errorCode > 1)  
 {  
 **break**;  
 }  
  
 stringLength = strlen(stringToInput);  
  
 **if** (stringToInput[stringLength - 1] != **'\n'**)  
 {  
 errorCode |= 1u;  
 }  
 **else** {  
 **break**;  
 }  
 }  
 **if** (errorCode == 0)  
 {  
 stringToInput[stringLength - 1] = **'\0'**;  
 **return** (**int**) stringLength - 1;  
 }  
 **return** -(**int**) errorCode;  
}  
  
**int** CycleInputInt(**char**\* stringToOutput, **int** maxStringLength,  
 **bool**(\* pChecker)(**int**))  
{  
 **int** number = -1;  
 **int** position = -1;  
 **char**\* stringNumber = (**char**\*) malloc(maxStringLength \* **sizeof**(**char**));  
 **FAILURE\_IF\_NULLPTR**(stringNumber);  
 **while** (**true**)  
 {  
 printf(**"%s"**, stringToOutput);  
 fflush(**stdout**);  
  
 **int** inputLineCode = InputLine(stringNumber, maxStringLength);  
 **if** (inputLineCode == -1) **continue**;  
 **if** (inputLineCode < 0)  
 {  
 fprintf(**stderr**, **"Ошибка при вводе\n"**);  
 fflush(**stderr**);  
 exit(**EXIT\_FAILURE**);  
 }  
 **int** sscanfCode = sscanf(stringNumber, **"%d%n"**, &number, &position);  
 **if** (position != inputLineCode) **continue**;  
 **if** (pChecker != **NULL** && !pChecker(number)) **continue**;  
 **if** (sscanfCode > 0) **break**;  
 }  
 free(stringNumber);  
 **return** number;  
}

## Lab\_02\_Lib/Input.h

*/// \file  
/// \brief Функции для ввода с проверкой  
/// \details Функции для ввода с проверками.*#ifndef **INPUT\_H**#define **INPUT\_H**#include **<stdbool.h>***/// Считывает строку до перевода строки.  
///  
/// \param stringToInput Указатель на считываемую строку.  
/// \param maxStringLength Размер максимально возможной для использования  
/// памяти через указатель.  
/// \return Неотрицательное число в случае успеха - длина строки,  
/// -1 если строка не влезла в maxStringLength,  
/// число меньшее -1 если EOF или другая ошибка.***int** InputLine(**char**\* stringToInput, **int** maxStringLength);  
  
*/// Выводит выходную фразу и считывает целое число. Если из считанной  
/// строки не получается получить число, или оно не удотворяет условию чекера  
/// pChecker, то число считывается занаго. Если был получен EOF или другая  
/// ошибка, программа аварийно завершается.  
///  
/// \param stringToOutput Строка для вывода перед вводом.  
/// \param maxStringLength Максимальная длина считываемого числа.  
/// \param pChecker Указател на функцию-чекер. Если равер NULL, проверка через  
/// чекер не проводится.  
/// \return Считанное число.***int** CycleInputInt(**char**\* stringToOutput, **int** maxStringLength,  
 **bool**(\* pChecker)(**int**));  
  
#endif *//INPUT\_H*

## Lab\_02\_Lib/Logger.c

*/// \file  
/// \brief Реализация функций из Logger.h  
/// \details Реализация функций из Logger.h.*#include **<malloc.h>**#include **<stdio.h>**#include **<stdarg.h>**#include **"Logger.h"***/// Логгируемый стол***static** Table\* g\_pLoggingTable;  
  
*/// Инициализирон ли логгер***static bool** g\_IsLoggerInitialized = **false**;  
*/// Основной поток для вывода***static** FILE\* g\_pMainOutputStream;  
*/// Требуется ли выводить информацию о столе в основной поток***static bool** g\_IsMainTableInfoEnabled;  
*/// Дополнителый поток для вывода***static** FILE\* g\_pSecondaryOutputStream;  
*/// Требуется ли выводить информацию о столе в дополнительный поток***static bool** g\_IsSecondaryTableInfoEnabled;  
*/// Требуется ли генерировать информацию о столе***static bool** g\_IsTableInfoRequired;  
*/// Дополнительный мьютекс (не главный) для логгирования***static** pthread\_mutex\_t g\_pLoggerMutex;  
  
*/// Преобразует вилку в символ.  
///  
/// \param pFork Указатель на вилку.  
/// \return Занятая вилка - ',', свободная - '.'.***char** ForkToChar(Fork\* pFork)  
{  
 **if** (pFork->IsInUse)  
 {  
 **return ','**;  
 }  
 **else** {  
 **return '.'**;  
 }  
}  
  
*/// Преобразует философа в символ.  
///  
/// \param pPhilosopher Указатель на философа.  
/// \return Философ без запущенного потока - '-', обедающий филосов - '=',  
/// ожидающий - '?', свободный - '\_'.***char** PhilosopherToChar(Philosopher\* pPhilosopher)  
{  
 **if** (!pPhilosopher->IsThreadRunning)  
 {  
 **return '-'**;  
 }  
 **else if** (pPhilosopher->IsEating)  
 {  
 **return '='**;  
 }  
 **else if** (pPhilosopher->IsWaiting)  
 {  
 **return '?'**;  
 }  
 **else** {  
 **return '\_'**;  
 }  
}  
  
**void** InitLogger(Table\* pTable, FILE\* pMainOutputStream,  
 **bool** isMainTableInfoEnabled, FILE\* pSecondaryOutputStream,  
 **bool** isSecondaryTableInfoEnabled)  
{  
 **if** (g\_IsLoggerInitialized)  
 {  
 **return**;  
 }  
  
 g\_pLoggingTable = pTable;  
  
 g\_pMainOutputStream = pMainOutputStream;  
 g\_IsMainTableInfoEnabled = isMainTableInfoEnabled;  
  
 g\_pSecondaryOutputStream = pSecondaryOutputStream;  
 g\_IsSecondaryTableInfoEnabled = isSecondaryTableInfoEnabled;  
  
 g\_IsTableInfoRequired =  
 isMainTableInfoEnabled || isSecondaryTableInfoEnabled;  
  
 g\_IsLoggerInitialized = **true**;  
  
 **if** (g\_pMainOutputStream == **NULL** && g\_pSecondaryOutputStream == **NULL**)  
 {  
 g\_IsLoggerInitialized = **false**;  
 }  
  
 pthread\_mutex\_init(&g\_pLoggerMutex, **NULL**);  
}  
  
**void** Log(**char**\* format, ...)  
{  
 **if** (!g\_IsLoggerInitialized)  
 {  
 **return**;  
 }  
  
 **int** tableInfoLength = g\_IsTableInfoRequired ?  
 g\_pLoggingTable->PhilosophersCount \* 2 + 1 : 1;  
 **char** result[tableInfoLength];  
 **if** (g\_IsTableInfoRequired)  
 {  
 **for** (**int** i = 0; i < g\_pLoggingTable->PhilosophersCount; i++)  
 {  
 result[i \* 2] = PhilosopherToChar(  
 g\_pLoggingTable->ppPhilosophers[i]);  
 result[i \* 2 + 1] = ForkToChar(g\_pLoggingTable->ppForks[i]);  
 }  
 }  
 result[tableInfoLength - 1] = **'\0'**;  
  
 **char** empty[] = **""**;  
 **char**\* res1 = g\_IsMainTableInfoEnabled ? result : empty;  
 **char**\* res2 = g\_IsSecondaryTableInfoEnabled ? result : empty;  
  
 pthread\_mutex\_lock(&g\_pLoggerMutex);  
  
#ifdef **\_\_MINGW32\_\_  
 if** (g\_pMainOutputStream) fprintf(g\_pMainOutputStream, **"[%s][tid: 0x%08llx]"**, res1, pthread\_self());  
 **if** (g\_pSecondaryOutputStream) fprintf(g\_pSecondaryOutputStream, **"[%s][tid: 0x%08llx]"**, res2, pthread\_self());  
#else  
 **if** (g\_pMainOutputStream)  
 {  
 fprintf(g\_pMainOutputStream, **"[%s][tid: 0x%08lx]"**, res1,  
 pthread\_self());  
 }  
 **if** (g\_pSecondaryOutputStream)  
 {  
 fprintf(g\_pSecondaryOutputStream, **"[%s][tid: 0x%08lx]"**, res2,  
 pthread\_self());  
 }  
#endif  
  
 va\_list argPtr;  
 **va\_start**(argPtr, format);  
 **if** (g\_pMainOutputStream) vfprintf(g\_pMainOutputStream, format, argPtr);  
 **if** (g\_pMainOutputStream) fprintf(g\_pMainOutputStream, **"\n"**);  
 **va\_end**(argPtr);  
  
 **va\_start**(argPtr, format);  
 **if** (g\_pSecondaryOutputStream)  
 {  
 vfprintf(g\_pSecondaryOutputStream, format, argPtr);  
 }  
 **if** (g\_pSecondaryOutputStream) fprintf(g\_pSecondaryOutputStream, **"\n"**);  
 **va\_end**(argPtr);  
  
 fflush(g\_pMainOutputStream);  
  
 pthread\_mutex\_unlock(&g\_pLoggerMutex);  
}

## Lab\_02\_Lib/Logger.h

*/// \file  
/// \brief Функции и макросы для логгирования  
/// \details Функции и макросы для логгирования.*#ifndef **LOGGER\_H**#define **LOGGER\_H**#include **<time.h>**#include **<string.h>**#include **"Fork.h"**#include **"Table.h"**#include **"Philosopher.h"**#include **"Utils.h"***/// Ширина в логе для имени файла*#define **LOG\_FILE\_WIDTH** 32  
*/// Формат аргументов для логгера генерируемых макросом*#define **LOG\_FMT "[%\*s:%d] "***/// Аргументы для логгера генерируемые макросом*#define **LOG\_ARGS** (**LOG\_FILE\_WIDTH** - **sizeof**(**STRINGIZE**(\_\_LINE\_\_))), **\_FILE**, **\_\_LINE\_\_***/// Помощник для макроса-логгера*#define **LOG\_HELPER**(format, ...) Log(**LOG\_FMT** format, \_\_VA\_ARGS\_\_)  
  
*/// Логгирует сообщение с соответствующим форматом. Будет выведена информация  
/// о столе (если не выключена), идентификатор потока, имя файла со строкой  
/// и сообщение  
///  
/// \param message Сообщение (формат) для логгирования.  
/// \param args Аргументы для вывода.*#define **LOG**(message, args...) **LOG\_HELPER**(message, LOG\_ARGS, ## args)  
  
*/// Инициализирует логгер.  
///  
/// \param pTable Стол.  
/// \param pMainOutputStream Основной поток для вывода логов.  
/// Например stdout или NULL.  
/// \param isMainTableInfoEnabled Треуется ли выводить информацию о столе  
/// в оснойной поток.  
/// \param pSecondaryOutputStream Дополнительный поток для вывода логов.  
/// Например fopen("1.log", "w+") или NULL.  
/// \param isSecondaryTableInfoEnabled Треуется ли выводить информацию о столе  
/// в дополнительный поток.***void** InitLogger(Table\* pTable, FILE\* pMainOutputStream,  
 **bool** isMainTableInfoEnabled, FILE\* pSecondaryOutputStream,  
 **bool** isSecondaryTableInfoEnabled);  
  
*/// Логгирует сообщение, рекомендуется использовать макрос  
/// LOG(message, args...).  
///  
/// \param format Сообщение (формат) для логгирования.  
/// \param ... Аргументы для вывода.***void** Log(**char**\* format, ...);  
  
#endif *//LOGGER\_H*

## Lab\_02\_Lib/Philosopher.c

*/// \file  
/// \brief Реализация функций из Philosopher.h  
/// \details Реализация функций из Philosopher.h.*#include **<malloc.h>**#include **<stdbool.h>**#include **<pthread.h>**#include **<semaphore.h>**#include **<stdio.h>**#include **"Logger.h"**#include **"Philosopher.h"**Philosopher\* CreatePhilosopher(**int** id, Fork\* leftFork, Fork\* rightFork,  
 **int** minDurationEatMs, **int** maxDurationEatMs,  
 **bool** isInfinityDuration)  
{  
 Philosopher\* pPhilosopher = (Philosopher\*) malloc(**sizeof**(Philosopher));  
 **FAILURE\_IF\_NULLPTR**(pPhilosopher);  
  
 pPhilosopher->PhilosopherId = id;  
 pPhilosopher->pLeftFork = leftFork;  
 pPhilosopher->pRightFork = rightFork;  
  
 pPhilosopher->IsEating = **false**;  
 pPhilosopher->IsWaiting = **false**;  
 pPhilosopher->IsThreadRunning = **false**;  
 pPhilosopher->IsWaitingLeftFork = **false**;  
 pPhilosopher->IsWaitingRightFork = **false**;  
  
 pPhilosopher->MinDurationEatMs = minDurationEatMs;  
 pPhilosopher->MaxDurationEatMs = maxDurationEatMs;  
 pPhilosopher->IsInfinityDuration = isInfinityDuration;  
  
  
 pPhilosopher->pSemOnGoingToEat = (sem\_t\*) malloc(**sizeof**(sem\_t));  
 **FAILURE\_IF\_NULLPTR**(pPhilosopher->pSemOnGoingToEat);  
 sem\_init(pPhilosopher->pSemOnGoingToEat, 0, 0);  
  
 pPhilosopher->pSemOnWaitingEnding = (sem\_t\*) malloc(**sizeof**(sem\_t));  
 **FAILURE\_IF\_NULLPTR**(pPhilosopher->pSemOnWaitingEnding);  
 sem\_init(pPhilosopher->pSemOnWaitingEnding, 0, 0);  
  
 **return** pPhilosopher;  
}  
  
**void** DestroyPhilosopher(Philosopher\* pPhilosopher)  
{  
 sem\_destroy(pPhilosopher->pSemOnGoingToEat);  
 free(pPhilosopher->pSemOnGoingToEat);  
 sem\_destroy(pPhilosopher->pSemOnWaitingEnding);  
 free(pPhilosopher->pSemOnWaitingEnding);  
 free(pPhilosopher);  
}  
  
**int** InterruptEating(Philosopher\* pPhilosopher, pthread\_mutex\_t\* pMutex)  
{  
 **LOG**(**"Попытка прервать приём пищи философа с номером %d"**,  
 pPhilosopher->PhilosopherId);  
  
 pthread\_mutex\_lock(pMutex);  
 **if** (pPhilosopher->IsEating)  
 {  
 **LOG**(**"Приём пищи философа с номером %d прерван"**,  
 pPhilosopher->PhilosopherId);  
 sem\_post(pPhilosopher->pSemOnWaitingEnding);  
 pthread\_mutex\_unlock(pMutex);  
 **return** 0;  
 }  
 **else if** (pPhilosopher->IsWaitingLeftFork)  
 {  
 **LOG**(**"Ожидание левой вилки философом с номером %d прервано"**,  
 pPhilosopher->PhilosopherId);  
 pthread\_cond\_signal(pPhilosopher->pLeftFork->CondSignalOnRelease);  
 pthread\_mutex\_unlock(pMutex);  
 **return** 0;  
 }  
 **else if** (pPhilosopher->IsWaitingRightFork)  
 {  
 **LOG**(**"Ожидание правой вилки философом с номером %d прервано"**,  
 pPhilosopher->PhilosopherId);  
 pthread\_cond\_signal(pPhilosopher->pRightFork->CondSignalOnRelease);  
 pthread\_mutex\_unlock(pMutex);  
 **return** 0;  
 }  
  
 **LOG**(**"Философ с номером %d не ест"**, pPhilosopher->PhilosopherId);  
 pthread\_mutex\_unlock(pMutex);  
  
 **return** 1;  
}

## Lab\_02\_Lib/Philosopher.h

*/// \file  
/// \brief Философ  
/// \details Философ, функции для создания, уничтожения итд.*#ifndef **PHILOSOPHER\_H**#define **PHILOSOPHER\_H**#include **<stdbool.h>**#include **<pthread.h>**#include **<semaphore.h>**#include **"Fork.h"***/// \struct Philosopher  
///  
/// Философ.***typedef struct**{  
 */// Номер философа* **int** PhilosopherId;  
 */// Указатель на правую филку* Fork\* pRightFork;  
 */// Указатель на левую вилку* Fork\* pLeftFork;  
 */// Ест ли философ* **bool** IsEating;  
 */// Ожидает ли философ* **bool** IsWaiting;  
 */// Запущен ли поток философа* **bool** IsThreadRunning;  
 */// Идентификатор потока философа* pthread\_t pThread;  
 */// Указатель на семафор, сигнализирующий о том, что нужно начать есть* sem\_t\* pSemOnGoingToEat;  
 */// Указатель на семафор, сигнализирующий что требуется перестать есть  
 /// или ждать* sem\_t\* pSemOnWaitingEnding;  
 */// Ждёт ли философ левую вилку.* **bool** IsWaitingLeftFork;  
 */// Ждёт ли философ правую вилку.* **bool** IsWaitingRightFork;  
 */// Нижняя граница случайного времени приёма пищи в миллисекундах* **int** MinDurationEatMs;  
 */// Верхняя граница случайного времени приёма пищи в миллисекундах* **int** MaxDurationEatMs;  
 */// Бесконечен ли приём пищи* **bool** IsInfinityDuration;  
} Philosopher;  
  
*/// Создаёт философа. Требудется очистка с помощью DestroyPhilosopher.  
///  
/// \param id Номер философа.  
/// \param leftFork Указатель на левую вилку.  
/// \param rightFork Указатель на правую вилку.  
/// \param minDurationEatMs Нижняя граница случайного времени приёма пищи  
/// в миллисекундах.  
/// \param maxDurationEatMs Верхняя граница случайного времени приёма пищи  
/// в миллисекундах.  
/// \param isInfinityDuration Бесконечен ли приём пищи.  
/// \return Указатель на созданного филисофа.*Philosopher\* CreatePhilosopher(**int** id, Fork\* leftFork, Fork\* rightFork,  
 **int** minDurationEatMs, **int** maxDurationEatMs,  
 **bool** isInfinityDuration);  
  
*/// Уничтожает философа.  
///  
/// \param pPhilosopher Указатель на уничтожаемого философа.***void** DestroyPhilosopher(Philosopher\* pPhilosopher);  
  
*/// Заставляет философа перестать есть или ожидать свободных вилок.  
/// Мьютекс должен быть разблокированным.  
///  
/// \param pPhilosopher Указатель на философа.  
/// \param pMutex Указатель на главный мьютекс.  
/// \return 0 если удачно прервано, 1 если философ ни ест, ни ожидает.***int** InterruptEating(Philosopher\* pPhilosopher, pthread\_mutex\_t\* pMutex);  
  
#endif *//PHILOSOPHER\_H*

## Lab\_02\_Lib/PhilosopherEatingThread.c

*/// \file  
/// \brief Реализация функций из PhilosopherEatingThread.h  
/// \details Реализация функций из PhilosopherEatingThread.h.*#include **<malloc.h>**#include **<errno.h>**#include **<stdio.h>**#include **<signal.h>**#include **<limits.h>**#include **"Utils.h"**#include **"PhilosopherEatingThread.h"**#include **"Logger.h"**#include **"Philosopher.h"**PhilosopherEatingThreadOptions\* CreatePhilosopherEatingThreadOptions(  
 Table\* pTable, Philosopher\* pPhilosopher, pthread\_mutex\_t\* pMutex)  
{  
 PhilosopherEatingThreadOptions\* pOptions =  
 (PhilosopherEatingThreadOptions\*) malloc(  
 **sizeof**(PhilosopherEatingThreadOptions));  
 **FAILURE\_IF\_NULLPTR**(pOptions);  
  
 pOptions->pTable = pTable;  
 pOptions->pPhilosopher = pPhilosopher;  
 pOptions->pMutex = pMutex;  
  
 **return** pOptions;  
}  
  
**void** DestroyPhilosopherEatingThreadOptions(  
 PhilosopherEatingThreadOptions\* pOptions)  
{  
 free(pOptions);  
}  
  
**void**\* PhilosopherEatingThread(**void**\* pEatThreadOptions)  
{  
 **LOG**(**"Запуск потока"**);  
  
 srand(time(**NULL**));  
  
 PhilosopherEatingThreadOptions\* pOptions =  
 (PhilosopherEatingThreadOptions\*) pEatThreadOptions;  
  
 Philosopher\* pPhilosopher = pOptions->pPhilosopher;  
  
 pthread\_mutex\_t\* pMutex = pOptions->pMutex;  
  
 pthread\_mutex\_lock(pMutex);  
 pPhilosopher->IsThreadRunning = **true**;  
 pthread\_mutex\_unlock(pMutex);  
  
 **while** (!pOptions->pTable->IsEatingMustEnd)  
 {  
 sem\_wait(pPhilosopher->pSemOnGoingToEat);  
  
 pthread\_mutex\_lock(pMutex);  
  
 **if** (pOptions->pTable->IsEatingMustEnd)  
 {  
 **LOG**(**"Поток для философа %d завершается"**,  
 pPhilosopher->PhilosopherId);  
 **break**;  
 }  
  
 **struct** timespec pDurationEat = RandomTimeMs(  
 pPhilosopher->MinDurationEatMs,  
 pPhilosopher->MaxDurationEatMs);  
  
  
 **LOG**(**"Философ с номером %d начинает есть, смотрит на вилки"**,  
 pPhilosopher->PhilosopherId);  
  
 **if** (pPhilosopher->pLeftFork->IsInUse == **false** &&  
 pPhilosopher->pRightFork->IsInUse == **false**)  
 {  
 **LOG**(**"Вилки свободны для философа с номером %d, начинает есть"**,  
 pPhilosopher->PhilosopherId);  
  
 pPhilosopher->IsEating = **true**;  
  
 TakeOnFork(pPhilosopher->pLeftFork);  
  
 TakeOnFork(pPhilosopher->pRightFork);  
  
 **LOG**(**"Философ с номером %d начал есть %lf секунд"**,  
 pPhilosopher->PhilosopherId,  
 TimespecToDouble(pDurationEat,  
 pOptions->pPhilosopher->IsInfinityDuration));  
  
 pthread\_mutex\_unlock(pMutex);  
  
 **if** (SleepOrWaitSem(pOptions->pPhilosopher->pSemOnWaitingEnding,  
 pDurationEat,  
 pOptions->pPhilosopher->IsInfinityDuration))  
 {  
 **LOG**(**"Приём пищи принудительно завершён заранее"**);  
 }  
  
 pthread\_mutex\_lock(pMutex);  
  
 **LOG**(**"Закончил есть философ с номером %d"**,  
 pPhilosopher->PhilosopherId);  
  
 TakeOffFork(pPhilosopher->pLeftFork);  
  
 TakeOffFork(pPhilosopher->pRightFork);  
  
 pPhilosopher->IsEating = **false**;  
  
 **LOG**(**"Поел, уходит философ с номером %d"**,  
 pPhilosopher->PhilosopherId);  
  
 pthread\_mutex\_unlock(pMutex);  
 }  
 **else** {  
 pPhilosopher->IsWaiting = **true**;  
  
 **if** (pPhilosopher->pLeftFork->IsInUse)  
 {  
 pPhilosopher->IsWaitingLeftFork = **true**;  
  
 **LOG**(**"Левая вилка для философа с номером %d несвободна, "  
 "ожидание"**,  
 pPhilosopher->PhilosopherId);  
  
 pthread\_cond\_wait(  
 pPhilosopher->pLeftFork->CondSignalOnRelease, pMutex);  
  
 **if** (pPhilosopher->pLeftFork->IsInUse)  
 {  
 **LOG**(**"Ожидание левой вилки для философа с номером %d "  
 "принудительно прервано"**,  
 pPhilosopher->PhilosopherId);  
  
 pPhilosopher->IsEating = **false**;  
 pPhilosopher->IsWaiting = **false**;  
 pPhilosopher->IsWaitingLeftFork = **false**;  
 pPhilosopher->IsWaitingRightFork = **false**;  
  
 **LOG**(**"Философ с номером %d принудительно уходит после "  
 "ожидания левой вилки"**,  
 pPhilosopher->PhilosopherId);  
 pthread\_mutex\_unlock(pMutex);  
 **continue**;  
 }  
  
 **LOG**(**"Освободилась левая вилка для философа с номером %d"**,  
 pPhilosopher->PhilosopherId);  
  
 pPhilosopher->IsWaitingLeftFork = **false**;  
 }  
  
 **LOG**(**"Занятие левой вилки для философа с номером %d"**,  
 pPhilosopher->PhilosopherId);  
  
 TakeOnFork(pPhilosopher->pLeftFork);  
  
 **if** (pPhilosopher->pRightFork->IsInUse)  
 {  
 pPhilosopher->IsWaitingRightFork = **true**;  
  
 **LOG**(**"Правая вилка для философа с номером %d несвободна, "  
 "ожидание"**,  
 pPhilosopher->PhilosopherId);  
  
 pthread\_cond\_wait(  
 pPhilosopher->pRightFork->CondSignalOnRelease,  
 pMutex);  
  
 **if** (pPhilosopher->pRightFork->IsInUse)  
 {  
 **LOG**(**"Ожидание правой вилки для философа с номером %d "  
 "принудительно прервано"**,  
 pPhilosopher->PhilosopherId);  
  
 TakeOffFork(pPhilosopher->pLeftFork);  
  
 pPhilosopher->IsEating = **false**;  
 pPhilosopher->IsWaiting = **false**;  
 pPhilosopher->IsWaitingLeftFork = **false**;  
 pPhilosopher->IsWaitingRightFork = **false**;  
  
 **LOG**(**"Философ с номером %d принудительно уходит после "  
 "ожидания левой вилки"**,  
 pPhilosopher->PhilosopherId);  
 pthread\_mutex\_unlock(pMutex);  
 **continue**;  
 }  
  
 **LOG**(**"Освободилась правая вилка для философа с номером %d"**,  
 pPhilosopher->PhilosopherId);  
  
 pPhilosopher->IsWaitingRightFork = **false**;  
 }  
 **LOG**(**"Занятие правой вилки для философа с номером %d"**,  
 pPhilosopher->PhilosopherId);  
  
 TakeOnFork(pPhilosopher->pRightFork);  
  
 **LOG**(**"Философ с номером %d ест после ожидания %lf сек."**,  
 pPhilosopher->PhilosopherId,  
 TimespecToDouble(pDurationEat,  
 pOptions->pPhilosopher->IsInfinityDuration));  
  
 pPhilosopher->IsWaiting = **false**;  
 pPhilosopher->IsEating = **true**;  
  
 pthread\_mutex\_unlock(pMutex);  
  
 **if** (SleepOrWaitSem(pOptions->pPhilosopher->pSemOnWaitingEnding,  
 pDurationEat,  
 pOptions->pPhilosopher->IsInfinityDuration))  
 {  
 **LOG**(**"Приём пищи после ожидания для философа с номером %d "  
 "завершён заранее сигналом"**,  
 pPhilosopher->PhilosopherId);  
 }  
  
 pthread\_mutex\_lock(pMutex);  
  
 **LOG**(**"Философ с номером %d закончил есть"**,  
 pPhilosopher->PhilosopherId);  
  
 TakeOffFork(pPhilosopher->pLeftFork);  
  
 TakeOffFork(pPhilosopher->pRightFork);  
  
 pPhilosopher->IsEating = **false**;  
  
 **LOG**(**"Философ с номером %d поел после ожидания, уходит"**,  
 pPhilosopher->PhilosopherId);  
  
 pthread\_mutex\_unlock(pMutex);  
 }  
 }  
  
 pPhilosopher->IsThreadRunning = **false**;  
 pthread\_mutex\_unlock(pMutex);  
  
 **LOG**(**"Завершение потока"**);  
  
 **return** pOptions;  
}

## Lab\_02\_Lib/PhilosopherEatingThread.h

*/// \file  
/// \brief Поток философа  
/// \details Поток философа, его конфигурация, функции для её создания  
/// уничтожения и тд.*#ifndef **PHILOSOPHEREATINGTHREAD\_H**#define **PHILOSOPHEREATINGTHREAD\_H**#include **<pthread.h>**#include **<semaphore.h>**#include **"Table.h"**#include **"Philosopher.h"***/// \struct PhilosopherEatingThreadOptions  
///  
/// Параметры запуска потока PhilosopherEatingThread.***typedef struct**{  
 */// Указатель на стол* Table\* pTable;  
 */// Указатель на философа* Philosopher\* pPhilosopher;  
 */// Указатель на главный мьютекс* pthread\_mutex\_t\* pMutex;  
} PhilosopherEatingThreadOptions;  
  
*/// Создаёт параметры запуска потока PhilosopherEatingThread.  
/// Требуется очистка с помощью DestroyPhilosopherEatingThreadOptions.  
///  
/// \param pTable Указатель на стол.  
/// \param pPhilosopher Указатель на философа.  
/// \param pMutex Указатель на глаынй мьютекс.  
/// \return Указатель на созданные параметры запуска.*PhilosopherEatingThreadOptions\* CreatePhilosopherEatingThreadOptions(  
 Table\* pTable, Philosopher\* pPhilosopher, pthread\_mutex\_t\* pMutex);  
  
*/// Уничтожает параметры запуска потока PhilosopherEatingThread.  
///  
/// \param pOptions Указатель на параметры запуска потока.***void** DestroyPhilosopherEatingThreadOptions(  
 PhilosopherEatingThreadOptions\* pOptions);  
  
*/// Функция потока философа. Ожидает пока философа отправят есть, ждёт  
/// свободных вилок если они не свободны, ест в течении заданного случайного  
/// времени, ждёт пока отправят есть... Пока не получит каманду завершения.  
///  
/// \param pEatThreadOptions Указатель на параметры запуска потока  
/// \return Возвращает указатель на параметры запуска с какими был запущен.***void**\* PhilosopherEatingThread(**void**\* pEatThreadOptions);  
  
#endif *//PHILOSOPHEREATINGTHREAD\_H*

## Lab\_02\_Lib/PhilosophersSpawnerThread.c

*/// \file  
/// \brief Реализация функций из PhilosophersWaiterThread.h  
/// \details Реализация функций из PhilosophersWaiterThread.h.*#include **<malloc.h>**#include **<time.h>**#include **<stdlib.h>**#include **<stdio.h>**#include **"PhilosophersSpawnerThread.h"**#include **"Utils.h"**#include **"Logger.h"**PhilosophersSpawnerThreadOptions\* CreatePhilosophersSpawnerThreadOptions(  
 Table\* pTable,  
 **int** minSendIntervalDuration,  
 **int** maxSendIntervalDuration)  
{  
 PhilosophersSpawnerThreadOptions\* pOptions =  
 (PhilosophersSpawnerThreadOptions\*)  
 malloc(**sizeof**(PhilosophersSpawnerThreadOptions));  
 **FAILURE\_IF\_NULLPTR**(pOptions);  
  
 pOptions->pTable = pTable;  
  
 pOptions->MinSendIntervalDuration = minSendIntervalDuration;  
 pOptions->MaxSendIntervalDuration = maxSendIntervalDuration;  
  
 pOptions->pMutex = pTable->pMutex;  
  
 pOptions->OnSemQuit = (sem\_t\*) malloc(**sizeof**(sem\_t));  
 **FAILURE\_IF\_NULLPTR**(pOptions->OnSemQuit);  
 sem\_init(pOptions->OnSemQuit, 0, 0);  
  
 **return** pOptions;  
}  
  
**void** DestroyPhilosophersSpawnerThreadOptions(  
 PhilosophersSpawnerThreadOptions\* pOptions)  
{  
 sem\_destroy(pOptions->OnSemQuit);  
 free(pOptions->OnSemQuit);  
 free(pOptions);  
}  
  
**int** SpawnPhilosopher(Table\* pTable, Philosopher\* pPhilosopher)  
{  
 pthread\_mutex\_lock(pTable->pMutex);  
 **if** (pPhilosopher->IsEating == **true**)  
 {  
 **LOG**(**"Философ с номером %d уже ест"**, pPhilosopher->PhilosopherId);  
 pthread\_mutex\_unlock(pTable->pMutex);  
 **return** 1;  
 }  
 **if** (pPhilosopher->IsWaiting == **true**)  
 {  
 **LOG**(**"Философ с номером %d ещё ожидает"**, pPhilosopher->PhilosopherId);  
 pthread\_mutex\_unlock(pTable->pMutex);  
 **return** 1;  
 }  
  
 pthread\_mutex\_unlock(pTable->pMutex);  
  
 sem\_post(pPhilosopher->pSemOnGoingToEat);  
  
 **return** 0;  
}  
  
**void**\* PhilosophersSpawnerThread(**void**\* pAutoEatThreadOptions)  
{  
 **LOG**(**"Запуск потока"**);  
 srand(time(**NULL**));  
  
 PhilosophersSpawnerThreadOptions\* pOptions =  
 (PhilosophersSpawnerThreadOptions\*) pAutoEatThreadOptions;  
  
 pOptions->pTable->IsEatingStarted = **true**;  
  
 **while** (!pOptions->pTable->IsEatingMustEnd)  
 {  
 **struct** timespec randomWaitTime = RandomTimeMs(  
 pOptions->MinSendIntervalDuration,  
 pOptions->MaxSendIntervalDuration);  
  
 **int** c = RandomInterval(0, pOptions->pTable->PhilosophersCount);  
 Philosopher\* pPhilosopher = pOptions->pTable->ppPhilosophers[c];  
  
 **LOG**(**"Философ с номером %d отправлен есть"**,  
 pPhilosopher->PhilosopherId);  
  
 **if** (SpawnPhilosopher(pOptions->pTable, pPhilosopher) == 1)  
 {  
 *//LOG("Не удалось отправить есть философа с номером %d",  
 // pPhilosopher->PhilosopherId);* }  
  
 **LOG**(**"После отправки философа с номером %d задержка перед отправкой "  
 "следующего %lf сек."**,  
 pPhilosopher->PhilosopherId,  
 TimespecToDouble(randomWaitTime, false));  
  
 **if** (SleepOrWaitSem(pOptions->OnSemQuit, randomWaitTime, **false**))  
 {  
 **LOG**(**"Принудительная остановка потока-спавнера"**);  
 **break**;  
 }  
 }  
  
 **LOG**(**"Завершение потока"**);  
  
 **return** pOptions;  
}

## Lab\_02\_Lib/PhilosophersSpawnerThread.h

*/// \file  
/// \brief Поток, посылающий философов есть  
/// \details Поток, посылающий философов есть, параметры его запуска, итд.*#ifndef **PHILOSOPHERSSPAWNERTHREAD\_H**#define **PHILOSOPHERSSPAWNERTHREAD\_H**#include **<stdbool.h>**#include **"Table.h"***/// \struct  
///  
/// Параметры запуска потока PhilosophersSpawnerThread.***typedef struct**{  
 Table\* pTable;  
 **int** MinSendIntervalDuration;  
 **int** MaxSendIntervalDuration;  
 pthread\_mutex\_t\* pMutex;  
 sem\_t\* OnSemQuit;  
} PhilosophersSpawnerThreadOptions;  
  
*/// Создаёт параметры запуска потока PhilosophersSpawnerThread. Требуется  
/// очистка с помощью DestroyPhilosophersSpawnerThreadOptions.  
///  
/// \param pTable Указатель на стол.  
/// \param minSendIntervalDuration Нижняя граница случайного времени в  
/// миллисекундах между отправками.  
/// \param maxSendIntervalDuration Верхняя граница случайного времени в  
/// миллисекундах между отправками.  
/// \return Указатель на созданные параметры.*PhilosophersSpawnerThreadOptions\* CreatePhilosophersSpawnerThreadOptions(  
 Table\* pTable,  
 **int** minSendIntervalDuration,  
 **int** maxSendIntervalDuration);  
  
*/// Уничтожает параметры запуска потока PhilosophersSpawnerThread.  
///  
/// \param pOptions Указатель на параметры запуска потока.***void** DestroyPhilosophersSpawnerThreadOptions(  
 PhilosophersSpawnerThreadOptions\* pOptions);  
  
*/// Посылает философа есть.  
///  
/// \param pTable Указатель на стол.  
/// \param pPhilosopher Указатель на философа.  
/// \return 1 если философ уже есть или ожидает, 0 если свободен и  
/// успешно отправлен.***int** SpawnPhilosopher(Table\* pTable, Philosopher\* pPhilosopher);  
  
*/// Функция потока, отправляющего есть философов каждое случайное время.  
///  
/// \param pAutoEatThreadOptions Указатель на параметры запуска потока.  
/// \return Указатель на параметры запуска с какими был запущен.***void**\* PhilosophersSpawnerThread(**void**\* pAutoEatThreadOptions);  
  
#endif *//PHILOSOPHERSSPAWNERTHREAD\_H*

## Lab\_02\_Lib/PhilosophersWaiterThread.c

*/// \file  
/// \brief Реализация функций из PhilosophersWaiterThread.h  
/// \details Реализация функций из PhilosophersWaiterThread.h.*#include **<malloc.h>**#include **<pthread.h>**#include **<stdio.h>**#include **<stdlib.h>**#include **"PhilosophersWaiterThread.h"**#include **"Logger.h"**#include **"PhilosopherEatingThread.h"**PhilosophersWaiterThreadOptions\* CreatePhilosophersWaiterThreadOptions(  
 Table\* pTable)  
{  
 PhilosophersWaiterThreadOptions\* pOptions =  
 (PhilosophersWaiterThreadOptions\*)  
 malloc(**sizeof**(PhilosophersWaiterThreadOptions));  
 **FAILURE\_IF\_NULLPTR**(pOptions);  
  
 pOptions->pTable = pTable;  
  
 pOptions->pMutex = pTable->pMutex;  
  
 **return** pOptions;  
}  
  
**void** DestroyPhilosophersWaiterThreadOptions(  
 PhilosophersWaiterThreadOptions\* pOptions)  
{  
 free(pOptions);  
}  
  
**void**\* PhilosophersWaiterThread(**void**\* pPhilosophersWaiterThreadOptions)  
{  
 **LOG**(**"Запуск потока"**);  
  
 PhilosophersWaiterThreadOptions\* pOptions =  
 (PhilosophersWaiterThreadOptions\*)  
 pPhilosophersWaiterThreadOptions;  
  
 **for** (**int** i = 0; i < pOptions->pTable->PhilosophersCount; i++)  
 {  
 pthread\_mutex\_lock(pOptions->pTable->pMutex);  
 **if** (pOptions->pTable->ppPhilosophers[i]->IsThreadRunning)  
 {  
 **LOG**(**"Ожидание завершения потока философа %d"**,  
 pOptions->pTable->ppPhilosophers[i]->PhilosopherId);  
  
 sem\_post(pOptions->pTable->ppPhilosophers[i]->pSemOnGoingToEat);  
  
 pthread\_mutex\_unlock(pOptions->pTable->pMutex);  
  
 **void**\* pReturned;  
 pthread\_join(pOptions->pTable->ppPhilosophers[i]->pThread,  
 &pReturned);  
 PhilosopherEatingThreadOptions\* pReturnedOptions =  
 (PhilosopherEatingThreadOptions\*) pReturned;  
 DestroyPhilosopherEatingThreadOptions(pReturnedOptions);  
  
 **continue**;  
 }  
 pthread\_mutex\_unlock(pOptions->pTable->pMutex);  
 }  
  
 pthread\_mutex\_lock(pOptions->pTable->pMutex);  
 pOptions->pTable->IsEatingEnded = **true**;  
 pthread\_mutex\_unlock(pOptions->pTable->pMutex);  
  
 **LOG**(**"Завершение потока"**);  
  
 **return** pOptions;  
}

## Lab\_02\_Lib/PhilosophersWaiterThread.h

*/// \file  
/// \brief Поток, завершающий потоки философов  
/// \details Поток, завершающий потоки философов, его параметры запуска итд.*#ifndef **PHILOSOPHERSWAITERTHREAD\_H**#define **PHILOSOPHERSWAITERTHREAD\_H**#include **<pthread.h>**#include **"Table.h"***/// \struct PhilosophersWaiterThreadOptions  
///  
/// Параметры запуска потока PhilosophersWaiterThread.***typedef struct**{  
 */// Указатель на стол* Table\* pTable;  
 */// Указатель на главный мьтекс* pthread\_mutex\_t\* pMutex;  
} PhilosophersWaiterThreadOptions;  
  
*/// Создаёт параметры запуска потока PhilosophersWaiterThread. Требуется  
/// очистка с помощью DestroyPhilosophersWaiterThreadOptions.  
///  
/// \param pTable Указатель на стол.  
/// \return Указатель на созданные параметры запуска.*PhilosophersWaiterThreadOptions\* CreatePhilosophersWaiterThreadOptions(  
 Table\* pTable);  
  
*/// Уничтожает параметры запуска потока.  
///  
/// \param pOptions Указатель на параметры запуска потока.***void** DestroyPhilosophersWaiterThreadOptions  
 (PhilosophersWaiterThreadOptions\* pOptions);  
  
*/// Функция потока, завершающего потоки философов.  
///  
/// \param pPhilosophersWaiterThreadOptions Указатель на параметры запуска.  
/// \return Указатель на параметры, с какими был запущен.***void**\* PhilosophersWaiterThread(**void**\* pPhilosophersWaiterThreadOptions);  
  
#endif *//PHILOSOPHERSWAITERTHREAD\_H*

## Lab\_02\_Lib/Table.c

*/// \file  
/// \brief Реализация функций из Table.h  
/// \details Реализация функций из Table.h.*#include **<malloc.h>**#include **<pthread.h>**#include **<stdio.h>**#include **<stdlib.h>**#include **"Table.h"**#include **"Input.h"**#include **"PhilosopherEatingThread.h"**#include **"Logger.h"**Table\* CreateTable(**int** philosophersCount, **int** minDurationEatMs,  
 **int** maxDurationEatMs, **bool** isInfinityDuration)  
{  
 Table\* pTable = (Table\*) malloc(**sizeof**(Table));  
 **FAILURE\_IF\_NULLPTR**(pTable);  
  
 pTable->PhilosophersCount = philosophersCount;  
  
 pTable->ppForks =  
 (Fork\*\*) malloc(pTable->PhilosophersCount \* **sizeof**(Fork\*));  
 **FAILURE\_IF\_NULLPTR**(pTable->ppForks);  
 **for** (**int** i = 0; i < pTable->PhilosophersCount; i++)  
 {  
 pTable->ppForks[i] = CreateFork(i + 1);  
 }  
  
 pTable->ppPhilosophers = (Philosopher\*\*) malloc(  
 pTable->PhilosophersCount \* **sizeof**(Philosopher\*));  
 **FAILURE\_IF\_NULLPTR**(pTable->ppPhilosophers);  
  
 **for** (**int** i = 0; i < pTable->PhilosophersCount; i++)  
 {  
 Fork\* lFork = pTable->ppForks[i == 0 ?  
 pTable->PhilosophersCount - 1 : i - 1];  
 pTable->ppPhilosophers[i] =  
 CreatePhilosopher(i + 1,  
 lFork,  
 pTable->ppForks[i],  
 minDurationEatMs,  
 maxDurationEatMs,  
 isInfinityDuration);  
 }  
  
 pTable->IsEatingStarted = **false**;  
 pTable->IsEatingEnded = **false**;  
 pTable->IsEatingMustEnd = **false**;  
  
 pTable->pMutex = (pthread\_mutex\_t\*) malloc(**sizeof**(pthread\_mutex\_t));  
 **FAILURE\_IF\_NULLPTR**(pTable->pMutex);  
 pthread\_mutex\_init(pTable->pMutex, **NULL**);  
  
 **return** pTable;  
}  
  
**void** StartAllThreads(Table\* pTable)  
{  
 **for** (**int** i = 0; i < pTable->PhilosophersCount; i++)  
 {  
 PhilosopherEatingThreadOptions\* options  
 = CreatePhilosopherEatingThreadOptions(  
 pTable,  
 pTable->ppPhilosophers[i],  
 pTable->pMutex);  
  
 **LOG**(**"Создан поток для философа %d"**,  
 pTable->ppPhilosophers[i]->PhilosopherId);  
  
 pthread\_create(&pTable->ppPhilosophers[i]->pThread, **NULL**,  
 PhilosopherEatingThread, options);  
 }  
}  
  
**void** DestroyTable(Table\* pTable)  
{  
 **for** (**int** i = 0; i < pTable->PhilosophersCount; i++)  
 {  
 DestroyFork(pTable->ppForks[i]);  
 DestroyPhilosopher(pTable->ppPhilosophers[i]);  
 }  
 pthread\_mutex\_destroy(pTable->pMutex);  
 free(pTable->pMutex);  
 free(pTable->ppForks);  
 free(pTable->ppPhilosophers);  
 free(pTable);  
}

## Lab\_02\_Lib/Table.h

*/// \file  
/// \brief Стол  
/// \details Стол, функции для его создания, уничтожения, старта потоков и  
/// отправки философов.*#ifndef **TABLE\_H**#define **TABLE\_H**#include **<stdbool.h>**#include **<pthread.h>**#include **<semaphore.h>**#include **"Philosopher.h"**#include **"Fork.h"***/// \struct Table  
///  
/// Стол.***typedef struct**{  
 */// Число философов* **int** PhilosophersCount;  
 */// Массив философов* Philosopher\*\* ppPhilosophers;  
 */// Массив вилок* Fork\*\* ppForks;  
 */// Начата ли трапеза* **bool** IsEatingStarted;  
 */// Кончилась ли трапеза* **bool** IsEatingEnded;  
 */// Должна ли кончится трапеза* **bool** IsEatingMustEnd;  
 */// Главный мьютекс* pthread\_mutex\_t\* pMutex;  
} Table;  
  
*/// Создаёт стол. Требуется очистка с помощью DestroyTable.  
///  
/// \param philosophersCount Число философов.  
/// \param minDurationEatMs Нижняя граница случайного времени  
/// для приёма пищи.  
/// \param maxDurationEatMs Верхняя граница случайного времени  
/// для приёма пищи.  
/// \param isInfinityDuration Бесконечен ли приём пищи.  
/// \return Указатель на созданный стол.*Table\* CreateTable(**int** philosophersCount, **int** minDurationEatMs,  
 **int** maxDurationEatMs, **bool** isInfinityDuration);  
  
*/// Запускает все потоки философов.  
///  
/// \param pTable Указатель на стол.***void** StartAllThreads(Table\* pTable);  
  
*/// Уничтожает стол.  
///  
/// \param pTable Указатель на стол.***void** DestroyTable(Table\* pTable);  
  
#endif *//TABLE\_H*

## Lab\_02\_Lib/Utils.c

*/// \file  
/// \brief Реализация функций из Utils.h  
/// \details Реализация функций из Utils.h.*#include **<stdlib.h>**#include **<time.h>**#include **<assert.h>**#include **<stdbool.h>**#include **<math.h>**#include **<limits.h>**#include **<errno.h>**#include **"Utils.h"  
  
int** RandomInterval(**int** min, **int** max)  
{  
 **assert**(max >= min);  
 **assert**(max - min <= RAND\_MAX);  
  
 **if** (min == max)  
 {  
 **return** min;  
 }  
 **return** rand() % (max - min) + min;  
}  
  
**struct** timespec RandomTimeMs(**int** minMs, **int** maxMs)  
{  
 **assert**(maxMs >= minMs);  
  
 **struct** timespec tw;  
 **if** (maxMs == minMs)  
 {  
 tw.tv\_sec = minMs / MS\_IN\_S;  
 tw.tv\_nsec = minMs % MS\_IN\_S \* NS\_IN\_MS;  
 }  
 **else** {  
 **int** randomMs = RandomInterval(minMs, maxMs);  
  
 tw.tv\_sec = randomMs / MS\_IN\_S;  
  
 tw.tv\_nsec =  
 randomMs % MS\_IN\_S \* NS\_IN\_MS +  
 RandomInterval(0, 1000) \* MS\_IN\_S +  
 RandomInterval(0, 1000);  
 }  
 **return** tw;  
}  
  
**double** TimespecToDouble(**struct** timespec duration, **bool** isInfinityTime)  
{  
 **if** (isInfinityTime)  
 {  
 **return INFINITY**;  
 }  
 **return** duration.tv\_sec \* 1.0 + duration.tv\_nsec \* 1.0 / NS\_IN\_S;  
}  
  
**int** SleepOrWaitSem(sem\_t\* pSemOnWaitingEnding, **struct** timespec duration,  
 **bool** isInfinityDuration)  
{  
 **if** (isInfinityDuration)  
 {  
 **struct** timespec infinityTime = {**INT\_MAX**, NS\_IN\_S - 1};  
 **int** timedwaitReturns = sem\_timedwait(  
 pSemOnWaitingEnding,  
 &infinityTime);  
 **return** timedwaitReturns == 0;  
 }  
 **else** {  
 **struct** timespec currentTime;  
 clock\_gettime(**CLOCK\_REALTIME**, &currentTime);  
  
 **struct** timespec endTime = {  
 currentTime.tv\_sec + duration.tv\_sec,  
 currentTime.tv\_nsec + duration.tv\_nsec};  
 **if** (endTime.tv\_nsec >= NS\_IN\_S)  
 {  
 endTime.tv\_sec++;  
 endTime.tv\_nsec -= NS\_IN\_S;  
 }  
  
 **int** timedwaitReturns = sem\_timedwait(  
 pSemOnWaitingEnding,  
 &endTime);  
 **return** timedwaitReturns == 0;  
 *//return errno == ETIMEDOUT;* }  
}

## Lab\_02\_Lib/Utils.h

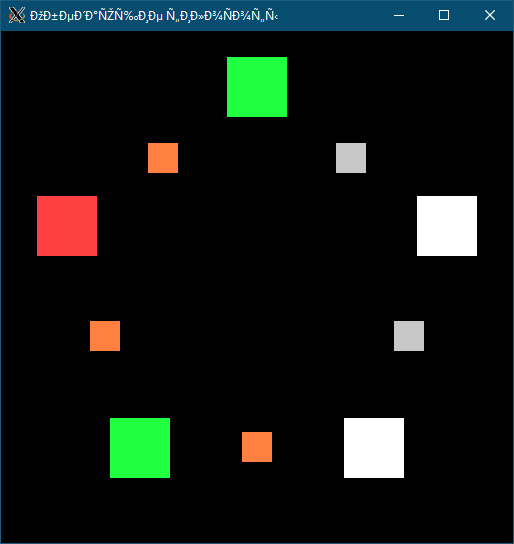
*/// \file  
/// \brief Прочие функции  
/// \details Прочие функции, такие как для генерации случайных чисел  
/// или времени в интервале, макросы итд.*#ifndef **UTILS\_H**#define **UTILS\_H**#include **<stdlib.h>**#include **<stdio.h>**#include **<time.h>**#include **<stdbool.h>**#include **"Philosopher.h"***/// Макрос для проверки указателя на NULL. Программа аварийно завершается с  
/// кодом 0 (EXIT\_FAILURE) если ptr == NULL.  
/// \param ptr Указатель для проверки*#define **FAILURE\_IF\_NULLPTR**(ptr) **do** { \  
 **if**((ptr) == **NULL**) { \  
 fprintf(**stderr**, **"Ошибка при выделении памяти\n"**); \  
 exit(**EXIT\_FAILURE**); \  
 } \  
} **while**(0)  
  
*/// Макрос, формиирующий строку с именем файла  
/// (например, для этого файла будет "Macro.h")*#define **\_FILE** (strrchr(**\_\_FILE\_\_**, **'/'**) ? strrchr(**\_\_FILE\_\_**, **'/'**) + 1 : (strrchr(**\_\_FILE\_\_**, **'\\'**) ? strrchr(**\_\_FILE\_\_**, **'\\'**) + 1 : **\_\_FILE\_\_**))  
  
*/// Макрос нужен для STRINGIZE(A)*#define **STRINGIZE\_NX**(A) #A  
  
*/// Макрос преобразующий не-строковое определение в строковое.  
/// Например STRINGIZE(\_\_LINE\_\_) будет "26".  
/// \param A Определение для преобразования*#define **STRINGIZE**(A) **STRINGIZE\_NX**(A)  
  
*/// Число миллисекунд в секундах***static const int** MS\_IN\_S = 1000;  
*/// Число наносекунд в миллисекундах***static const int** NS\_IN\_MS = 1000000;  
*/// Число наносекунд в секундах***static const int** NS\_IN\_S = 1000000000;  
  
*/// Генерирует случайное число используя rand в полуинтервале [min, max).  
/// Разность max - min не должна быть больше RAND\_MAX и не меньше 0.  
/// Если min == max, то возвращается min.  
///  
/// \param min Нижняя граница генеруремого случайного числа.  
/// \param max Верхняя граница генеруремого случайного числа.  
/// \return Случайное число.***int** RandomInterval(**int** min, **int** max);  
  
*/// Генерирует случайное время используя rand в полуинтервале [minMs, maxMs).  
/// Разность maxMs - minMs не должна быть больше RAND\_MAX и не меньше 0.  
/// Если minMs == maxMs, то возвращается minMs.  
///  
/// \param minMs Нижняя граница генерируемого случайного времени  
/// в миллисекундах.  
/// \param maxMs Верхняя граница генерируемого случайного времени  
/// в миллисекундах.  
/// \return Случайное время с точностью до наносекунды.***struct** timespec RandomTimeMs(**int** minMs, **int** maxMs);  
  
*/// Переводит время в наносекундах в рациональное число секунд.  
/// Если помечено, что время бесконечно, возвращается INFINITY.  
///  
/// \param time Время для перевода.  
/// \param isInfinityTime Бесконечно ли время.  
/// \return Время в секундах в рациональном виде.***double** TimespecToDouble(**struct** timespec time, **bool** isInfinityTime);  
  
*/// Ожидает семафора до заданного времени.  
///  
/// \param pSemOnWaitingEnding Ожидаемый семафор.  
/// \param duration Время ожидания.  
/// \param isInfinityDuration Бесконечно ли время ожидания.  
/// \return 0 если время закончилось быстрее, 1 если семафор быстрее времени.***int** SleepOrWaitSem(sem\_t\* pSemOnWaitingEnding, **struct** timespec duration,  
 **bool** isInfinityDuration);  
  
#endif *//UTILS\_H*

# Примеры использования

## Запуск №1 (5 философов, время приёма пищи от 1000 мс до 5000 мс, время появления от 1000 мс до 3000 мс, завершение закрытием окна)

WSL Ubuntu 18.04, компилятор Clang 8.0.0

/mnt/c/Users/vladislav/Projects/SystemProgramming/cmake-build-release-wsl-clang/0x04/Lab\_02/Lab\_02\_Interactive/Lab\_02\_Interactive  
Обозначения: Большой квадрат - философ:  
 - тёмно-серый - поток ещё не запущен или уже завершён;  
 \_ белый - ничего не делает;  
 = красный - ест;  
 ? зелёный - ожидает.  
 Маленький квадрат - вилка:  
 , оранжевый - занята;  
 . светло-серый - свободна.  
  
Управление: [1-9] - отправить философа есть;  
 Alt+[1-9] - прекратить приём пищи или ожидание;  
 Ctrl+[1-9] - переключение метки бесконечного приёма пищи;  
 Esc - выход из программы с ожиданием завершения всех потоков.  
  
  
Минимальное количество философов 2, желательно не больше 9, рекомендуется 5.  
Введите кол-во философов: 5  
  
Время вводится в миллисекундах.  
Разность между верхней границей и нижней должна быть не меньше 0 и не больше 2147483647.  
Время будет генерироватся в полуинтервале [нижняя граница; верхняя граница).  
  
Для того, чтобы время приёма пищы было бесконечным, введите 0 и 0.  
Для того, чтобы было постоянным, введите одинаковые числа.  
Введите нижнию границу времени приёма пищи (например 1000): 1000  
Введите верхнюю границу времени приёма пищи (например 5000): 5000  
  
Для того, философы не появлялись автоматически, введите 0 и 0.  
Для того, чтобы было постоянным, введите одинаковые числа.  
Введите нижнию границу времени между появлениями (например 500): 1000  
Введите верхнюю границу времени между появлениями (например 1500): 3000  
  
[-.-.-.-.-.][tid: 0x7fe43eab1580][ MainInteractive.c:144] Введены данные, создание объектов, запуск потоков  
[-.-.-.-.-.][tid: 0x7fe43eab1580][ MainWindow.c:38] Инициализация SDL  
[-.-.-.-.-.][tid: 0x7fe43eab1580][ MainWindow.c:53] Скомпилированная версия SDL 2.0.11  
[-.-.-.-.-.][tid: 0x7fe43eab1580][ MainWindow.c:57] Скомпонованная версия SDL 2.0.11  
[-.-.-.-.-.][tid: 0x7fe43eab1580][ MainWindow.c:59] Создание окна  
[-.-.-.-.-.][tid: 0x7fe43eab1580][ MainWindow.c:74] Создание отрисовщика  
[-.-.-.-.-.][tid: 0x7fe43eab1580][ MainWindow.c:91] Запуск потока отрисовщика  
[-.-.-.-.-.][tid: 0x7fe43eab1580][ MainWindow.c:102] Запуск потоков-философов  
[-.-.-.-.-.][tid: 0x7fe43eab1580][ Table.c:70] Создан поток для философа 1  
[-.-.-.-.-.][tid: 0x7fe42dc50700][ RendererThread.c:119] Запуск потока  
[-.-.-.-.-.][tid: 0x7fe42d440700][ PhilosopherEatingThread.c:39] Запуск потока  
[-.-.-.-.-.][tid: 0x7fe43eab1580][ Table.c:70] Создан поток для философа 2  
[\_.-.-.-.-.][tid: 0x7fe43eab1580][ Table.c:70] Создан поток для философа 3  
[\_.-.-.-.-.][tid: 0x7fe42cc30700][ PhilosopherEatingThread.c:39] Запуск потока  
[\_.\_.-.-.-.][tid: 0x7fe43eab1580][ Table.c:70] Создан поток для философа 4  
[\_.\_.-.-.-.][tid: 0x7fe427ff0700][ PhilosopherEatingThread.c:39] Запуск потока  
[\_.\_.\_.-.-.][tid: 0x7fe4277e0700][ PhilosopherEatingThread.c:39] Запуск потока  
[\_.\_.\_.-.-.][tid: 0x7fe43eab1580][ Table.c:70] Создан поток для философа 5  
[\_.\_.\_.\_.-.][tid: 0x7fe43eab1580][ MainWindow.c:107] Запуск потока, отправляющий философов есть  
[\_.\_.\_.\_.-.][tid: 0x7fe426fd0700][ PhilosopherEatingThread.c:39] Запуск потока  
[\_.\_.\_.\_.\_.][tid: 0x7fe43eab1580][ MainInteractive.c:158] Запуск главного цикла  
[\_.\_.\_.\_.\_.][tid: 0x7fe43eab1580][ MainWindow.c:124] Запуск главного цикла  
[\_.\_.\_.\_.\_.][tid: 0x7fe4267c0700][ PhilosophersSpawnerThread.c:71] Запуск потока  
[\_.\_.\_.\_.\_.][tid: 0x7fe4267c0700][ PhilosophersSpawnerThread.c:89] Философ с номером 4 отправлен есть  
[\_.\_.\_.\_.\_.][tid: 0x7fe4267c0700][ PhilosophersSpawnerThread.c:100] После отправки философа с номером 4 задержка перед отправкой следующего 1.898850 сек.  
[\_.\_.\_.\_.\_.][tid: 0x7fe4277e0700][ PhilosopherEatingThread.c:73] Философ с номером 4 начинает есть, смотрит на вилки  
[\_.\_.\_.\_.\_.][tid: 0x7fe4277e0700][ PhilosopherEatingThread.c:79] Вилки свободны для философа с номером 4, начинает есть  
[\_.\_.\_,=.\_.][tid: 0x7fe4277e0700][ Fork.c:32] Занятие вилки с номером 3  
[\_.\_.\_,=,\_.][tid: 0x7fe4277e0700][ Fork.c:32] Занятие вилки с номером 4  
[\_.\_.\_,=,\_.][tid: 0x7fe4277e0700][ PhilosopherEatingThread.c:90] Философ с номером 4 начал есть 2.570396 секунд  
[\_.\_.\_,=,\_.][tid: 0x7fe4267c0700][ PhilosophersSpawnerThread.c:89] Философ с номером 3 отправлен есть  
[\_.\_.\_,=,\_.][tid: 0x7fe4267c0700][ PhilosophersSpawnerThread.c:100] После отправки философа с номером 3 задержка перед отправкой следующего 2.686560 сек.  
[\_.\_.\_,=,\_.][tid: 0x7fe427ff0700][ PhilosopherEatingThread.c:73] Философ с номером 3 начинает есть, смотрит на вилки  
[\_.\_.?,=,\_.][tid: 0x7fe427ff0700][ PhilosopherEatingThread.c:157] Занятие левой вилки для философа с номером 3  
[\_.\_,?,=,\_.][tid: 0x7fe427ff0700][ Fork.c:32] Занятие вилки с номером 2  
[\_.\_,?,=,\_.][tid: 0x7fe427ff0700][ PhilosopherEatingThread.c:167] Правая вилка для философа с номером 3 несвободна, ожидание  
[\_.\_,?,=,\_.][tid: 0x7fe4277e0700][ PhilosopherEatingThread.c:104] Закончил есть философ с номером 4  
[\_.\_,?.=,\_.][tid: 0x7fe4277e0700][ Fork.c:41] Освобождение вилки с номером 3  
[\_.\_,?.=.\_.][tid: 0x7fe4277e0700][ Fork.c:41] Освобождение вилки с номером 4  
[\_.\_,?.\_.\_.][tid: 0x7fe4277e0700][ PhilosopherEatingThread.c:113] Поел, уходит философ с номером 4  
[\_.\_,?.\_.\_.][tid: 0x7fe427ff0700][ PhilosopherEatingThread.c:194] Освободилась правая вилка для философа с номером 3  
[\_.\_,?.\_.\_.][tid: 0x7fe427ff0700][ PhilosopherEatingThread.c:199] Занятие правой вилки для философа с номером 3  
[\_.\_,?,\_.\_.][tid: 0x7fe427ff0700][ Fork.c:32] Занятие вилки с номером 3  
[\_.\_,?,\_.\_.][tid: 0x7fe427ff0700][ PhilosopherEatingThread.c:206] Философ с номером 3 ест после ожидания 3.696436 сек.  
[\_.\_,=,\_.\_.][tid: 0x7fe4267c0700][ PhilosophersSpawnerThread.c:89] Философ с номером 1 отправлен есть  
[\_.\_,=,\_.\_.][tid: 0x7fe4267c0700][ PhilosophersSpawnerThread.c:100] После отправки философа с номером 1 задержка перед отправкой следующего 2.814579 сек.  
[\_.\_,=,\_.\_.][tid: 0x7fe42d440700][ PhilosopherEatingThread.c:73] Философ с номером 1 начинает есть, смотрит на вилки  
[\_.\_,=,\_.\_.][tid: 0x7fe42d440700][ PhilosopherEatingThread.c:79] Вилки свободны для философа с номером 1, начинает есть  
[=.\_,=,\_.\_,][tid: 0x7fe42d440700][ Fork.c:32] Занятие вилки с номером 5  
[=,\_,=,\_.\_,][tid: 0x7fe42d440700][ Fork.c:32] Занятие вилки с номером 1  
[=,\_,=,\_.\_,][tid: 0x7fe42d440700][ PhilosopherEatingThread.c:90] Философ с номером 1 начал есть 4.987595 секунд  
[=,\_,=,\_.\_,][tid: 0x7fe427ff0700][ PhilosopherEatingThread.c:225] Философ с номером 3 закончил есть  
[=,\_.=,\_.\_,][tid: 0x7fe427ff0700][ Fork.c:41] Освобождение вилки с номером 2  
[=,\_.=.\_.\_,][tid: 0x7fe427ff0700][ Fork.c:41] Освобождение вилки с номером 3  
[=,\_.\_.\_.\_,][tid: 0x7fe427ff0700][ PhilosopherEatingThread.c:234] Философ с номером 3 поел после ожидания, уходит  
[=,\_.\_.\_.\_,][tid: 0x7fe4267c0700][ PhilosophersSpawnerThread.c:89] Философ с номером 5 отправлен есть  
[=,\_.\_.\_.\_,][tid: 0x7fe4267c0700][ PhilosophersSpawnerThread.c:100] После отправки философа с номером 5 задержка перед отправкой следующего 2.963363 сек.  
[=,\_.\_.\_.\_,][tid: 0x7fe426fd0700][ PhilosopherEatingThread.c:73] Философ с номером 5 начинает есть, смотрит на вилки  
[=,\_.\_.\_.?,][tid: 0x7fe426fd0700][ PhilosopherEatingThread.c:157] Занятие левой вилки для философа с номером 5  
[=,\_.\_.\_,?,][tid: 0x7fe426fd0700][ Fork.c:32] Занятие вилки с номером 4  
[=,\_.\_.\_,?,][tid: 0x7fe426fd0700][ PhilosopherEatingThread.c:167] Правая вилка для философа с номером 5 несвободна, ожидание  
[=,\_.\_.\_,?,][tid: 0x7fe42d440700][ PhilosopherEatingThread.c:104] Закончил есть философ с номером 1  
[=,\_.\_.\_,?.][tid: 0x7fe42d440700][ Fork.c:41] Освобождение вилки с номером 5  
[=.\_.\_.\_,?.][tid: 0x7fe42d440700][ Fork.c:41] Освобождение вилки с номером 1  
[\_.\_.\_.\_,?.][tid: 0x7fe42d440700][ PhilosopherEatingThread.c:113] Поел, уходит философ с номером 1  
[\_.\_.\_.\_,?.][tid: 0x7fe426fd0700][ PhilosopherEatingThread.c:194] Освободилась правая вилка для философа с номером 5  
[\_.\_.\_.\_,?.][tid: 0x7fe426fd0700][ PhilosopherEatingThread.c:199] Занятие правой вилки для философа с номером 5  
[\_.\_.\_.\_,?,][tid: 0x7fe426fd0700][ Fork.c:32] Занятие вилки с номером 5  
[\_.\_.\_.\_,?,][tid: 0x7fe426fd0700][ PhilosopherEatingThread.c:206] Философ с номером 5 ест после ожидания 4.940518 сек.  
[\_.\_.\_.\_,=,][tid: 0x7fe4267c0700][ PhilosophersSpawnerThread.c:89] Философ с номером 4 отправлен есть  
[\_.\_.\_.\_,=,][tid: 0x7fe4267c0700][ PhilosophersSpawnerThread.c:100] После отправки философа с номером 4 задержка перед отправкой следующего 2.650975 сек.  
[\_.\_.\_.\_,=,][tid: 0x7fe4277e0700][ PhilosopherEatingThread.c:73] Философ с номером 4 начинает есть, смотрит на вилки  
[\_.\_.\_.?,=,][tid: 0x7fe4277e0700][ PhilosopherEatingThread.c:157] Занятие левой вилки для философа с номером 4  
[\_.\_.\_,?,=,][tid: 0x7fe4277e0700][ Fork.c:32] Занятие вилки с номером 3  
[\_.\_.\_,?,=,][tid: 0x7fe4277e0700][ PhilosopherEatingThread.c:167] Правая вилка для философа с номером 4 несвободна, ожидание  
[\_.\_.\_,?,=,][tid: 0x7fe4267c0700][ PhilosophersSpawnerThread.c:89] Философ с номером 1 отправлен есть  
[\_.\_.\_,?,=,][tid: 0x7fe4267c0700][ PhilosophersSpawnerThread.c:100] После отправки философа с номером 1 задержка перед отправкой следующего 2.747332 сек.  
[\_.\_.\_,?,=,][tid: 0x7fe42d440700][ PhilosopherEatingThread.c:73] Философ с номером 1 начинает есть, смотрит на вилки  
[?.\_.\_,?,=,][tid: 0x7fe42d440700][ PhilosopherEatingThread.c:127] Левая вилка для философа с номером 1 несвободна, ожидание  
[?.\_.\_,?,=,][tid: 0x7fe426fd0700][ PhilosopherEatingThread.c:225] Философ с номером 5 закончил есть  
[?.\_.\_,?.=,][tid: 0x7fe426fd0700][ Fork.c:41] Освобождение вилки с номером 4  
[?.\_.\_,?.=.][tid: 0x7fe426fd0700][ Fork.c:41] Освобождение вилки с номером 5



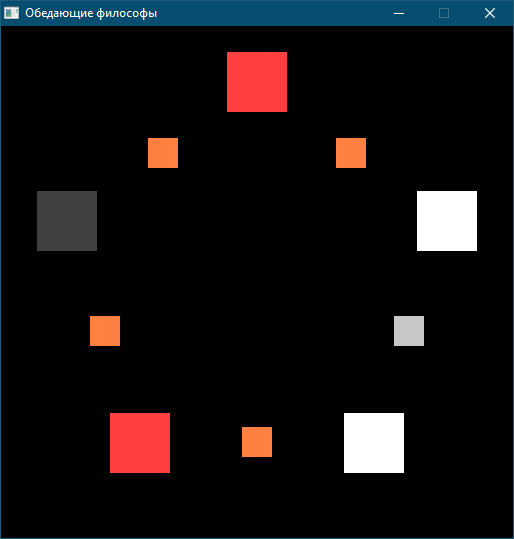
1. Окно визуализации когда текстовая визуализация [?.\_.\_,?.=.]

[?.\_.\_,?.\_.][tid: 0x7fe426fd0700][ PhilosopherEatingThread.c:234] Философ с номером 5 поел после ожидания, уходит  
[?.\_.\_,?.\_.][tid: 0x7fe4277e0700][ PhilosopherEatingThread.c:194] Освободилась правая вилка для философа с номером 4  
[?.\_.\_,?.\_.][tid: 0x7fe4277e0700][ PhilosopherEatingThread.c:199] Занятие правой вилки для философа с номером 4  
[?.\_.\_,?,\_.][tid: 0x7fe4277e0700][ Fork.c:32] Занятие вилки с номером 4  
[?.\_.\_,?,\_.][tid: 0x7fe4277e0700][ PhilosopherEatingThread.c:206] Философ с номером 4 ест после ожидания 2.825936 сек.  
[?.\_.\_,=,\_.][tid: 0x7fe42d440700][ PhilosopherEatingThread.c:151] Освободилась левая вилка для философа с номером 1  
[?.\_.\_,=,\_.][tid: 0x7fe42d440700][ PhilosopherEatingThread.c:157] Занятие левой вилки для философа с номером 1  
[?.\_.\_,=,\_,][tid: 0x7fe42d440700][ Fork.c:32] Занятие вилки с номером 5  
[?.\_.\_,=,\_,][tid: 0x7fe42d440700][ PhilosopherEatingThread.c:199] Занятие правой вилки для философа с номером 1  
[?,\_.\_,=,\_,][tid: 0x7fe42d440700][ Fork.c:32] Занятие вилки с номером 1  
[?,\_.\_,=,\_,][tid: 0x7fe42d440700][ PhilosopherEatingThread.c:206] Философ с номером 1 ест после ожидания 3.244373 сек.  
[=,\_.\_,=,\_,][tid: 0x7fe4267c0700][ PhilosophersSpawnerThread.c:89] Философ с номером 3 отправлен есть  
[=,\_.\_,=,\_,][tid: 0x7fe4267c0700][ PhilosophersSpawnerThread.c:100] После отправки философа с номером 3 задержка перед отправкой следующего 1.941162 сек.  
[=,\_.\_,=,\_,][tid: 0x7fe427ff0700][ PhilosopherEatingThread.c:73] Философ с номером 3 начинает есть, смотрит на вилки  
[=,\_.?,=,\_,][tid: 0x7fe427ff0700][ PhilosopherEatingThread.c:157] Занятие левой вилки для философа с номером 3  
[=,\_,?,=,\_,][tid: 0x7fe427ff0700][ Fork.c:32] Занятие вилки с номером 2  
[=,\_,?,=,\_,][tid: 0x7fe427ff0700][ PhilosopherEatingThread.c:167] Правая вилка для философа с номером 3 несвободна, ожидание  
[=,\_,?,=,\_,][tid: 0x7fe43eab1580][ MainWindow.c:132] Главный цикл завершён событием  
[=,\_,?,=,\_,][tid: 0x7fe43eab1580][ MainWindow.c:138] Событие выхода из программы было послано без завершения потоков и очистки  
[=,\_,?,=,\_,][tid: 0x7fe43eab1580][ MainWindow.c:139] Завершение программы с кодом 1 (EXIT\_FAILURE)  
  
Process finished with exit code 1

## Запуск №2 (5 философов, время приёма пищи 5000 мс, время появления от 1000 мс до 2000 мс, завершение нажатием на ESC)

Windows 10, компилятор MinGW-w64 8.1.0

C:\Users\vladislav\Projects\SystemProgramming\cmake-build-release-mingw\0x04\Lab\_02\Lab\_02\_Interactive\Lab\_02\_Interactive.exe  
Обозначения: Большой квадрат - философ:  
 - тёмно-серый - поток ещё не запущен или уже завершён;  
 \_ белый - ничего не делает;  
 = красный - ест;  
 ? зелёный - ожидает.  
 Маленький квадрат - вилка:  
 , оранжевый - занята;  
 . светло-серый - свободна.  
  
Управление: [1-9] - отправить философа есть;  
 Alt+[1-9] - прекратить приём пищи или ожидание;  
 Ctrl+[1-9] - переключение метки бесконечного приёма пищи;  
 Esc - выход из программы с ожиданием завершения всех потоков.  
  
  
Минимальное количество философов 2, желательно не больше 9, рекомендуется 5.  
Введите кол-во философов: 5  
  
Время вводится в миллисекундах.  
Разность между верхней границей и нижней должна быть не меньше 0 и не больше 32767.  
Время будет генерироватся в полуинтервале [нижняя граница; верхняя граница).  
  
Для того, чтобы время приёма пищы было бесконечным, введите 0 и 0.  
Для того, чтобы было постоянным, введите одинаковые числа.  
Введите нижнию границу времени приёма пищи (например 1000): 5000  
Введите верхнюю границу времени приёма пищи (например 5000): 5000  
  
Для того, философы не появлялись автоматически, введите 0 и 0.  
Для того, чтобы было постоянным, введите одинаковые числа.  
Введите нижнию границу времени между появлениями (например 500): 1000  
Введите верхнюю границу времени между появлениями (например 1500): 2000  
  
[-.-.-.-.-.][tid: 0x00000001][ MainInteractive.c:144] Введены данные, создание объектов, запуск потоков  
[-.-.-.-.-.][tid: 0x00000001][ MainWindow.c:38] Инициализация SDL  
[-.-.-.-.-.][tid: 0x00000001][ MainWindow.c:53] Скомпилированная версия SDL 2.0.11  
[-.-.-.-.-.][tid: 0x00000001][ MainWindow.c:57] Скомпонованная версия SDL 2.0.11  
[-.-.-.-.-.][tid: 0x00000001][ MainWindow.c:59] Создание окна  
[-.-.-.-.-.][tid: 0x00000001][ MainWindow.c:74] Создание отрисовщика  
[-.-.-.-.-.][tid: 0x00000001][ MainWindow.c:91] Запуск потока отрисовщика  
[-.-.-.-.-.][tid: 0x00000001][ MainWindow.c:102] Запуск потоков-философов  
[-.-.-.-.-.][tid: 0x00000002][ RendererThread.c:119] Запуск потока  
[-.-.-.-.-.][tid: 0x00000001][ Table.c:70] Создан поток для философа 1  
[-.-.-.-.-.][tid: 0x00000001][ Table.c:70] Создан поток для философа 2  
[-.-.-.-.-.][tid: 0x00000003][ PhilosopherEatingThread.c:39] Запуск потока  
[\_.-.-.-.-.][tid: 0x00000001][ Table.c:70] Создан поток для философа 3  
[\_.-.-.-.-.][tid: 0x00000001][ Table.c:70] Создан поток для философа 4  
[\_.-.-.-.-.][tid: 0x00000001][ Table.c:70] Создан поток для философа 5  
[\_.-.-.-.-.][tid: 0x00000001][ MainWindow.c:107] Запуск потока, отправляющий философов есть  
[\_.-.-.-.-.][tid: 0x00000001][ MainInteractive.c:158] Запуск главного цикла  
[\_.-.-.-.-.][tid: 0x00000001][ MainWindow.c:124] Запуск главного цикла  
[\_.-.-.-.-.][tid: 0x00000004][ PhilosopherEatingThread.c:39] Запуск потока  
[\_.-.-.-.-.][tid: 0x00000005][ PhilosopherEatingThread.c:39] Запуск потока  
[\_.-.-.-.-.][tid: 0x00000006][ PhilosopherEatingThread.c:39] Запуск потока  
[\_.-.-.-.-.][tid: 0x00000007][ PhilosopherEatingThread.c:39] Запуск потока  
[\_.-.-.-.-.][tid: 0x00000008][ PhilosophersSpawnerThread.c:71] Запуск потока  
[\_.\_.\_.\_.\_.][tid: 0x00000008][ PhilosophersSpawnerThread.c:89] Философ с номером 3 отправлен есть  
[\_.\_.\_.\_.\_.][tid: 0x00000008][ PhilosophersSpawnerThread.c:100] После отправки философа с номером 3 задержка перед отправкой следующего 1.369883 сек.  
[\_.\_.\_.\_.\_.][tid: 0x00000005][ PhilosopherEatingThread.c:73] Философ с номером 3 начинает есть, смотрит на вилки  
[\_.\_.\_.\_.\_.][tid: 0x00000005][ PhilosopherEatingThread.c:79] Вилки свободны для философа с номером 3, начинает есть  
[\_.\_,=.\_.\_.][tid: 0x00000005][ Fork.c:32] Занятие вилки с номером 2  
[\_.\_,=,\_.\_.][tid: 0x00000005][ Fork.c:32] Занятие вилки с номером 3  
[\_.\_,=,\_.\_.][tid: 0x00000005][ PhilosopherEatingThread.c:90] Философ с номером 3 начал есть 5.000000 секунд  
[\_.\_,=,\_.\_.][tid: 0x00000008][ PhilosophersSpawnerThread.c:89] Философ с номером 4 отправлен есть  
[\_.\_,=,\_.\_.][tid: 0x00000008][ PhilosophersSpawnerThread.c:100] После отправки философа с номером 4 задержка перед отправкой следующего 1.686176 сек.  
[\_.\_,=,\_.\_.][tid: 0x00000006][ PhilosopherEatingThread.c:73] Философ с номером 4 начинает есть, смотрит на вилки  
[\_.\_,=,?.\_.][tid: 0x00000006][ PhilosopherEatingThread.c:127] Левая вилка для философа с номером 4 несвободна, ожидание  
[\_.\_,=,?.\_.][tid: 0x00000008][ PhilosophersSpawnerThread.c:89] Философ с номером 2 отправлен есть  
[\_.\_,=,?.\_.][tid: 0x00000008][ PhilosophersSpawnerThread.c:100] После отправки философа с номером 2 задержка перед отправкой следующего 1.057252 сек.  
[\_.\_,=,?.\_.][tid: 0x00000004][ PhilosopherEatingThread.c:73] Философ с номером 2 начинает есть, смотрит на вилки  
[\_.?,=,?.\_.][tid: 0x00000004][ PhilosopherEatingThread.c:157] Занятие левой вилки для философа с номером 2  
[\_,?,=,?.\_.][tid: 0x00000004][ Fork.c:32] Занятие вилки с номером 1  
[\_,?,=,?.\_.][tid: 0x00000004][ PhilosopherEatingThread.c:167] Правая вилка для философа с номером 2 несвободна, ожидание  
[\_,?,=,?.\_.][tid: 0x00000008][ PhilosophersSpawnerThread.c:89] Философ с номером 1 отправлен есть  
[\_,?,=,?.\_.][tid: 0x00000008][ PhilosophersSpawnerThread.c:100] После отправки философа с номером 1 задержка перед отправкой следующего 1.863177 сек.  
[\_,?,=,?.\_.][tid: 0x00000003][ PhilosopherEatingThread.c:73] Философ с номером 1 начинает есть, смотрит на вилки  
[?,?,=,?.\_.][tid: 0x00000003][ PhilosopherEatingThread.c:157] Занятие левой вилки для философа с номером 1  
[?,?,=,?.\_,][tid: 0x00000003][ Fork.c:32] Занятие вилки с номером 5  
[?,?,=,?.\_,][tid: 0x00000003][ PhilosopherEatingThread.c:167] Правая вилка для философа с номером 1 несвободна, ожидание  
[?,?,=,?.\_,][tid: 0x00000005][ PhilosopherEatingThread.c:104] Закончил есть философ с номером 3  
[?,?.=,?.\_,][tid: 0x00000005][ Fork.c:41] Освобождение вилки с номером 2  
[?,?.=.?.\_,][tid: 0x00000005][ Fork.c:41] Освобождение вилки с номером 3  
[?,?.\_.?.\_,][tid: 0x00000005][ PhilosopherEatingThread.c:113] Поел, уходит философ с номером 3  
[?,?.\_.?.\_,][tid: 0x00000004][ PhilosopherEatingThread.c:194] Освободилась правая вилка для философа с номером 2  
[?,?.\_.?.\_,][tid: 0x00000004][ PhilosopherEatingThread.c:199] Занятие правой вилки для философа с номером 2  
[?,?,\_.?.\_,][tid: 0x00000004][ Fork.c:32] Занятие вилки с номером 2  
[?,?,\_.?.\_,][tid: 0x00000004][ PhilosopherEatingThread.c:206] Философ с номером 2 ест после ожидания 5.000000 сек.  
[?,=,\_.?.\_,][tid: 0x00000006][ PhilosopherEatingThread.c:151] Освободилась левая вилка для философа с номером 4  
[?,=,\_.?.\_,][tid: 0x00000006][ PhilosopherEatingThread.c:157] Занятие левой вилки для философа с номером 4  
[?,=,\_,?.\_,][tid: 0x00000006][ Fork.c:32] Занятие вилки с номером 3  
[?,=,\_,?.\_,][tid: 0x00000006][ PhilosopherEatingThread.c:199] Занятие правой вилки для философа с номером 4  
[?,=,\_,?,\_,][tid: 0x00000006][ Fork.c:32] Занятие вилки с номером 4  
[?,=,\_,?,\_,][tid: 0x00000006][ PhilosopherEatingThread.c:206] Философ с номером 4 ест после ожидания 5.000000 сек.  
[?,=,\_,=,\_,][tid: 0x00000008][ PhilosophersSpawnerThread.c:89] Философ с номером 4 отправлен есть  
[?,=,\_,=,\_,][tid: 0x00000008][ PhilosophersSpawnerThread.c:51] Философ с номером 4 уже ест  
[?,=,\_,=,\_,][tid: 0x00000008][ PhilosophersSpawnerThread.c:100] После отправки философа с номером 4 задержка перед отправкой следующего 1.704364 сек.  
[?,=,\_,=,\_,][tid: 0x00000008][ PhilosophersSpawnerThread.c:89] Философ с номером 5 отправлен есть  
[?,=,\_,=,\_,][tid: 0x00000008][ PhilosophersSpawnerThread.c:100] После отправки философа с номером 5 задержка перед отправкой следующего 1.199271 сек.  
[?,=,\_,=,\_,][tid: 0x00000007][ PhilosopherEatingThread.c:73] Философ с номером 5 начинает есть, смотрит на вилки  
[?,=,\_,=,?,][tid: 0x00000007][ PhilosopherEatingThread.c:127] Левая вилка для философа с номером 5 несвободна, ожидание  
[?,=,\_,=,?,][tid: 0x00000008][ PhilosophersSpawnerThread.c:89] Философ с номером 3 отправлен есть  
[?,=,\_,=,?,][tid: 0x00000008][ PhilosophersSpawnerThread.c:100] После отправки философа с номером 3 задержка перед отправкой следующего 1.000222 сек.  
[?,=,\_,=,?,][tid: 0x00000005][ PhilosopherEatingThread.c:73] Философ с номером 3 начинает есть, смотрит на вилки  
[?,=,?,=,?,][tid: 0x00000005][ PhilosopherEatingThread.c:127] Левая вилка для философа с номером 3 несвободна, ожидание  
[?,=,?,=,?,][tid: 0x00000008][ PhilosophersSpawnerThread.c:89] Философ с номером 1 отправлен есть  
[?,=,?,=,?,][tid: 0x00000008][ PhilosophersSpawnerThread.c:57] Философ с номером 1 ещё ожидает  
[?,=,?,=,?,][tid: 0x00000008][ PhilosophersSpawnerThread.c:100] После отправки философа с номером 1 задержка перед отправкой следующего 1.170148 сек.  
[?,=,?,=,?,][tid: 0x00000004][ PhilosopherEatingThread.c:225] Философ с номером 2 закончил есть  
[?.=,?,=,?,][tid: 0x00000004][ Fork.c:41] Освобождение вилки с номером 1  
[?.=.?,=,?,][tid: 0x00000004][ Fork.c:41] Освобождение вилки с номером 2  
[?.\_.?,=,?,][tid: 0x00000004][ PhilosopherEatingThread.c:234] Философ с номером 2 поел после ожидания, уходит  
[?.\_.?,=,?,][tid: 0x00000006][ PhilosopherEatingThread.c:225] Философ с номером 4 закончил есть  
[?.\_.?.=,?,][tid: 0x00000006][ Fork.c:41] Освобождение вилки с номером 3  
[?.\_.?.=.?,][tid: 0x00000006][ Fork.c:41] Освобождение вилки с номером 4  
[?.\_.?.\_.?,][tid: 0x00000006][ PhilosopherEatingThread.c:234] Философ с номером 4 поел после ожидания, уходит  
[?.\_.?.\_.?,][tid: 0x00000003][ PhilosopherEatingThread.c:194] Освободилась правая вилка для философа с номером 1  
[?.\_.?.\_.?,][tid: 0x00000003][ PhilosopherEatingThread.c:199] Занятие правой вилки для философа с номером 1  
[?,\_.?.\_.?,][tid: 0x00000003][ Fork.c:32] Занятие вилки с номером 1  
[?,\_.?.\_.?,][tid: 0x00000003][ PhilosopherEatingThread.c:206] Философ с номером 1 ест после ожидания 5.000000 сек.  
[=,\_.?.\_.?,][tid: 0x00000005][ PhilosopherEatingThread.c:151] Освободилась левая вилка для философа с номером 3  
[=,\_.?.\_.?,][tid: 0x00000005][ PhilosopherEatingThread.c:157] Занятие левой вилки для философа с номером 3  
[=,\_,?.\_.?,][tid: 0x00000005][ Fork.c:32] Занятие вилки с номером 2  
[=,\_,?.\_.?,][tid: 0x00000005][ PhilosopherEatingThread.c:199] Занятие правой вилки для философа с номером 3  
[=,\_,?,\_.?,][tid: 0x00000005][ Fork.c:32] Занятие вилки с номером 3  
[=,\_,?,\_.?,][tid: 0x00000005][ PhilosopherEatingThread.c:206] Философ с номером 3 ест после ожидания 5.000000 сек.  
[=,\_,=,\_.?,][tid: 0x00000007][ PhilosopherEatingThread.c:151] Освободилась левая вилка для философа с номером 5  
[=,\_,=,\_.?,][tid: 0x00000007][ PhilosopherEatingThread.c:157] Занятие левой вилки для философа с номером 5  
[=,\_,=,\_,?,][tid: 0x00000007][ Fork.c:32] Занятие вилки с номером 4  
[=,\_,=,\_,?,][tid: 0x00000007][ PhilosopherEatingThread.c:167] Правая вилка для философа с номером 5 несвободна, ожидание  
[=,\_,=,\_,?,][tid: 0x00000008][ PhilosophersSpawnerThread.c:89] Философ с номером 4 отправлен есть  
[=,\_,=,\_,?,][tid: 0x00000008][ PhilosophersSpawnerThread.c:100] После отправки философа с номером 4 задержка перед отправкой следующего 1.692982 сек.  
[=,\_,=,\_,?,][tid: 0x00000006][ PhilosopherEatingThread.c:73] Философ с номером 4 начинает есть, смотрит на вилки  
[=,\_,=,?,?,][tid: 0x00000006][ PhilosopherEatingThread.c:127] Левая вилка для философа с номером 4 несвободна, ожидание  
[=,\_,=,?,?,][tid: 0x00000008][ PhilosophersSpawnerThread.c:89] Философ с номером 4 отправлен есть  
[=,\_,=,?,?,][tid: 0x00000008][ PhilosophersSpawnerThread.c:57] Философ с номером 4 ещё ожидает  
[=,\_,=,?,?,][tid: 0x00000008][ PhilosophersSpawnerThread.c:100] После отправки философа с номером 4 задержка перед отправкой следующего 1.749410 сек.  
[=,\_,=,?,?,][tid: 0x00000008][ PhilosophersSpawnerThread.c:89] Философ с номером 3 отправлен есть  
[=,\_,=,?,?,][tid: 0x00000008][ PhilosophersSpawnerThread.c:51] Философ с номером 3 уже ест  
[=,\_,=,?,?,][tid: 0x00000008][ PhilosophersSpawnerThread.c:100] После отправки философа с номером 3 задержка перед отправкой следующего 1.214073 сек.  
[=,\_,=,?,?,][tid: 0x00000003][ PhilosopherEatingThread.c:225] Философ с номером 1 закончил есть  
[=,\_,=,?,?.][tid: 0x00000003][ Fork.c:41] Освобождение вилки с номером 5  
[=.\_,=,?,?.][tid: 0x00000003][ Fork.c:41] Освобождение вилки с номером 1  
[\_.\_,=,?,?.][tid: 0x00000003][ PhilosopherEatingThread.c:234] Философ с номером 1 поел после ожидания, уходит  
[\_.\_,=,?,?.][tid: 0x00000005][ PhilosopherEatingThread.c:225] Философ с номером 3 закончил есть  
[\_.\_.=,?,?.][tid: 0x00000005][ Fork.c:41] Освобождение вилки с номером 2  
[\_.\_.=.?,?.][tid: 0x00000005][ Fork.c:41] Освобождение вилки с номером 3  
[\_.\_.\_.?,?.][tid: 0x00000005][ PhilosopherEatingThread.c:234] Философ с номером 3 поел после ожидания, уходит  
[\_.\_.\_.?,?.][tid: 0x00000007][ PhilosopherEatingThread.c:194] Освободилась правая вилка для философа с номером 5  
[\_.\_.\_.?,?.][tid: 0x00000007][ PhilosopherEatingThread.c:199] Занятие правой вилки для философа с номером 5  
[\_.\_.\_.?,?,][tid: 0x00000007][ Fork.c:32] Занятие вилки с номером 5  
[\_.\_.\_.?,?,][tid: 0x00000007][ PhilosopherEatingThread.c:206] Философ с номером 5 ест после ожидания 5.000000 сек.  
[\_.\_.\_.?,=,][tid: 0x00000006][ PhilosopherEatingThread.c:151] Освободилась левая вилка для философа с номером 4  
[\_.\_.\_.?,=,][tid: 0x00000006][ PhilosopherEatingThread.c:157] Занятие левой вилки для философа с номером 4  
[\_.\_.\_,?,=,][tid: 0x00000006][ Fork.c:32] Занятие вилки с номером 3  
[\_.\_.\_,?,=,][tid: 0x00000006][ PhilosopherEatingThread.c:167] Правая вилка для философа с номером 4 несвободна, ожидание  
[\_.\_.\_,?,=,][tid: 0x00000008][ PhilosophersSpawnerThread.c:89] Философ с номером 1 отправлен есть  
[\_.\_.\_,?,=,][tid: 0x00000008][ PhilosophersSpawnerThread.c:100] После отправки философа с номером 1 задержка перед отправкой следующего 1.923258 сек.  
[\_.\_.\_,?,=,][tid: 0x00000003][ PhilosopherEatingThread.c:73] Философ с номером 1 начинает есть, смотрит на вилки  
[?.\_.\_,?,=,][tid: 0x00000003][ PhilosopherEatingThread.c:127] Левая вилка для философа с номером 1 несвободна, ожидание  
[?.\_.\_,?,=,][tid: 0x00000001][ MainWindow.c:161] Начато завершение программы  
[?.\_.\_,?,=,][tid: 0x00000001][ MainWindow.c:163] Запуск потока, который завершает потоки  
[?.\_.\_,?,=,][tid: 0x00000009][ ProgramQuitThread.c:37] Запуск потока  
[?.\_.\_,?,=,][tid: 0x00000009][ ProgramQuitThread.c:44] Запуск потока, который завершает потоки филосософ  
[?.\_.\_,?,=,][tid: 0x00000009][ ProgramQuitThread.c:56] Принудительная остановка потока-cпавнера  
[?.\_.\_,?,=,][tid: 0x00000009][ ProgramQuitThread.c:60] Ожидание завершения потока, который завершает потоки филосософ  
[?.\_.\_,?,=,][tid: 0x00000008][ PhilosophersSpawnerThread.c:104] Принудительная остановка потока-спавнера  
[?.\_.\_,?,=,][tid: 0x00000008][ PhilosophersSpawnerThread.c:109] Завершение потока  
[?.\_.\_,?,=,][tid: 0x0000000a][ PhilosophersWaiterThread.c:37] Запуск потока  
[?.\_.\_,?,=,][tid: 0x0000000a][ PhilosophersWaiterThread.c:49] Ожидание завершения потока философа 1  
[?.\_.\_,?,=,][tid: 0x00000007][ PhilosopherEatingThread.c:225] Философ с номером 5 закончил есть  
[?.\_.\_,?.=,][tid: 0x00000007][ Fork.c:41] Освобождение вилки с номером 4  
[?.\_.\_,?.=.][tid: 0x00000007][ Fork.c:41] Освобождение вилки с номером 5  
[?.\_.\_,?.\_.][tid: 0x00000007][ PhilosopherEatingThread.c:234] Философ с номером 5 поел после ожидания, уходит  
[?.\_.\_,?.-.][tid: 0x00000007][ PhilosopherEatingThread.c:243] Завершение потока  
[?.\_.\_,?.-.][tid: 0x00000006][ PhilosopherEatingThread.c:194] Освободилась правая вилка для философа с номером 4  
[?.\_.\_,?.-.][tid: 0x00000006][ PhilosopherEatingThread.c:199] Занятие правой вилки для философа с номером 4  
[?.\_.\_,?,-.][tid: 0x00000006][ Fork.c:32] Занятие вилки с номером 4  
[?.\_.\_,?,-.][tid: 0x00000006][ PhilosopherEatingThread.c:206] Философ с номером 4 ест после ожидания 5.000000 сек.  
[?.\_.\_,=,-.][tid: 0x00000003][ PhilosopherEatingThread.c:151] Освободилась левая вилка для философа с номером 1  
[?.\_.\_,=,-.][tid: 0x00000003][ PhilosopherEatingThread.c:157] Занятие левой вилки для философа с номером 1  
[?.\_.\_,=,-,][tid: 0x00000003][ Fork.c:32] Занятие вилки с номером 5  
[?.\_.\_,=,-,][tid: 0x00000003][ PhilosopherEatingThread.c:199] Занятие правой вилки для философа с номером 1  
[?,\_.\_,=,-,][tid: 0x00000003][ Fork.c:32] Занятие вилки с номером 1  
[?,\_.\_,=,-,][tid: 0x00000003][ PhilosopherEatingThread.c:206] Философ с номером 1 ест после ожидания 5.000000 сек.  
[=,\_.\_,=,-,][tid: 0x00000006][ PhilosopherEatingThread.c:225] Философ с номером 4 закончил есть  
[=,\_.\_.=,-,][tid: 0x00000006][ Fork.c:41] Освобождение вилки с номером 3  
[=,\_.\_.=.-,][tid: 0x00000006][ Fork.c:41] Освобождение вилки с номером 4



1. Окно визуализации когда текстовая визуализация [=,\_.\_,=,-,]

[=,\_.\_.\_.-,][tid: 0x00000006][ PhilosopherEatingThread.c:234] Философ с номером 4 поел после ожидания, уходит  
[=,\_.\_.-.-,][tid: 0x00000006][ PhilosopherEatingThread.c:243] Завершение потока  
[=,\_.\_.-.-,][tid: 0x00000003][ PhilosopherEatingThread.c:225] Философ с номером 1 закончил есть  
[=,\_.\_.-.-.][tid: 0x00000003][ Fork.c:41] Освобождение вилки с номером 5  
[=.\_.\_.-.-.][tid: 0x00000003][ Fork.c:41] Освобождение вилки с номером 1  
[\_.\_.\_.-.-.][tid: 0x00000003][ PhilosopherEatingThread.c:234] Философ с номером 1 поел после ожидания, уходит  
[-.\_.\_.-.-.][tid: 0x00000003][ PhilosopherEatingThread.c:243] Завершение потока  
[-.\_.\_.-.-.][tid: 0x0000000a][ PhilosophersWaiterThread.c:49] Ожидание завершения потока философа 2  
[-.\_.\_.-.-.][tid: 0x00000004][ PhilosopherEatingThread.c:63] Поток для философа 2 завершается  
[-.-.\_.-.-.][tid: 0x00000004][ PhilosopherEatingThread.c:243] Завершение потока  
[-.-.\_.-.-.][tid: 0x0000000a][ PhilosophersWaiterThread.c:49] Ожидание завершения потока философа 3  
[-.-.\_.-.-.][tid: 0x00000005][ PhilosopherEatingThread.c:63] Поток для философа 3 завершается  
[-.-.-.-.-.][tid: 0x00000005][ PhilosopherEatingThread.c:243] Завершение потока  
[-.-.-.-.-.][tid: 0x0000000a][ PhilosophersWaiterThread.c:71] Завершение потока  
[-.-.-.-.-.][tid: 0x00000002][ RendererThread.c:216] Завершение потока  
[-.-.-.-.-.][tid: 0x00000009][ ProgramQuitThread.c:66] Ожидание завершения потока-cпавнера  
[-.-.-.-.-.][tid: 0x00000009][ ProgramQuitThread.c:72] Отправление события выхода главному циклу  
[-.-.-.-.-.][tid: 0x00000009][ ProgramQuitThread.c:78] Завершение потока  
[-.-.-.-.-.][tid: 0x00000001][ MainWindow.c:132] Главный цикл завершён событием  
[-.-.-.-.-.][tid: 0x00000001][ MainWindow.c:250] Завершение программы, завершение остальных потоков  
[-.-.-.-.-.][tid: 0x00000001][ MainWindow.c:252] Ожидание завершения отрисовщика  
[-.-.-.-.-.][tid: 0x00000001][ MainWindow.c:259] Очистка и завершение отрисовщика, окна и SDL  
[-.-.-.-.-.][tid: 0x00000001][ MainWindow.c:147] Завершение программы с кодом 0 (EXIT\_SUCCESS)  
  
Process finished with exit code 0

## Запуск №3 (5 философов, время приёма пищи от 1000 мс до 9000 мс, время появления от 500 мс до 1500 мс, завершение нажатием на ESC)

Xubuntu 19.04, компилятор GCC 8.3.0