МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Вятский государственный университет»
Факультет автоматики и вычислительной техники
Кафедра электронных вычислительных машин

Допущено к защите
Руководитель проекта
_____/Караваева О. В./
(подпись) (Ф.И.О)
«__» _____2019 г.

РАЗРАБОТКА ПРОГРАММНОЙ МОДЕЛИ ПЛАНИРОВЩИКА ПАМЯТИ

Пояснительная записка курсового проекта по дисциплине «Предметно-ориентированные автоматизированные информационные системы»
ТПЖА.09.03.01.066 ПЗ

Разработал студент группы ИВТ-42 _____/Рзаев А. Э./

Руководитель доцент кафедры ЭВМ _____/Караваева О. В./

Проект защищен с оценкой «_____» _____
(оценка) (дата)

Члены комиссии _____ / _____/
(подпись) (Ф.И.О)
_____ / _____/
(подпись) (Ф.И.О)
_____ / _____/
(подпись) (Ф.И.О)

Киров 2019

УДК 004.4

Реферат

Рзаев А. Э. Разработка программной модели планировщика памяти: ТПЖА.09.03.01.066 ПЗ: Курс. проект / ВятГУ, каф. ЭВМ; рук. Караваева О. В. – Киров, 2019. – Гр. ч. 4 л. ф. А1; ПЗ 48 с., 15 рис., 3 прил.

ЛАБОРАТОРНЫЙ ПРАКТИКУМ, УПРАВЛЕНИЕ ПАМЯТЬЮ, ДИСПЕТЧЕР ПАМЯТИ, ОПЕРАЦИОННЫЕ СИСТЕМЫ, ГЕНЕРАТОР ЗАДАНИЙ, QT, QMAKE, C++, JSON.

Цель курсового проекта – повышение качества обучения за счет использования программного комплекса моделирования работы диспетчера памяти в операционных системах при выполнении лабораторных работ по дисциплине «Операционные системы».

Программное обеспечение, разработанное в рамках данного курсового проекта – лабораторная установка «Модель диспетчера памяти операционной системы».

В ходе выполнения курсового проекта был выполнен анализ предметной области, проектирование и разработка программного обеспечения.

Содержание

Введение

В настоящее время в области образования все больше производится автоматизация контроля знаний учащихся и освоения нового материала.

Одним из способов автоматизации являются специальные программные модели, эмулирующие работу какой-либо системы. С их помощью студенты имеют возможность достаточно подробно изучить ее работу, принципы и особенности. В совокупности с теоретическим материалом это позволяет увеличить степень освоения новых знаний по данной дисциплине и повысить качество обучения в целом.

Такие программные модели достаточно широко используются при выполнении лабораторных работ по дисциплине «Операционные системы». К сожалению, качество некоторых приложений оставляет желать лучшего, что затрудняет изучение нового материала. Поэтому было принято решение выполнить анализ и доработку наиболее проблемной модели – диспетчера памяти операционной системы.

1 Анализ предметной области

На данном этапе работы необходимо провести обзор текущей программной модели диспетчера памяти, выяснить ее недостатки, обосновать актуальность разработки новой модели и сформировать требования к программному обеспечению в виде технического задания.

1.1 Обзор текущей программной модели

Текущая программная модель была разработана в 2002 году студентами Вятского государственного университета А. С. Гордиенко и М. Н. Томчуком; доработана в 2006 году студентами А. Ю. Соколовым и И. А. Брызгаловым.

1.1.1 Функции планировщика памяти

Функциями планировщика памяти в многозадачных ОС являются:

– отслеживание свободной и занятой памяти;

– первоначальное и динамическое выделение памяти процессам приложений и самой операционной системе и освобождение памяти по завершении процессов;

– настройка адресов программы на конкретную область физической памяти;

– полное или частичное вытеснение кодов и данных процессов из оперативной памяти (ОП) на диск, когда размеры ОП недостаточны для размещения всех процессов, и возвращение их в ОП;

– защита памяти, выделенной процессу, от возможных вмешательств со стороны других процессов;

– дефрагментация памяти.

Для изучения функций планировщика была разработана текущая программная модель. Студентам предлагается выполнить задание, состоящее из последовательности заявок, которые необходимо обработать, выступив в роли планировщика.

1.1.2 Модель памяти

В данной модели адресное пространство разбито на 256 страниц памяти по 4096 байт каждая. Наименьшая единица адресного пространства, доступная для выделения процессу – страница. Выделять можно только целое число страниц.

Процессам в данной модели могут присваиваться идентификаторы (PID) от 0 до 255 включительно.

Непрерывная область памяти, состоящая из одной и более страниц и имеющая начальный адрес и размер, называется блоком памяти.

Каждый блок памяти имеет следующие параметры:

- адрес начала блока;
- размер блока в страницах;
- идентификатор процесса, которому принадлежит данный блок.

Память процессам при каждом запросе выделяется в виде одного блока памяти.

Состояние памяти определяется совокупностью всех блоков памяти. Каждая страница адресного пространства должна находиться в одном и только одном блоке памяти, т. е. последовательность блоков памяти, упорядоченных по начальному адресу, должна полностью покрыть адресное пространство.

Под начальным состоянием памяти подразумевается такое состояние памяти, при котором все адресное пространство покрыто одним свободным блоком памяти с начальным адресом 0 и размером 256.

1.1.3 Интерфейс пользователя

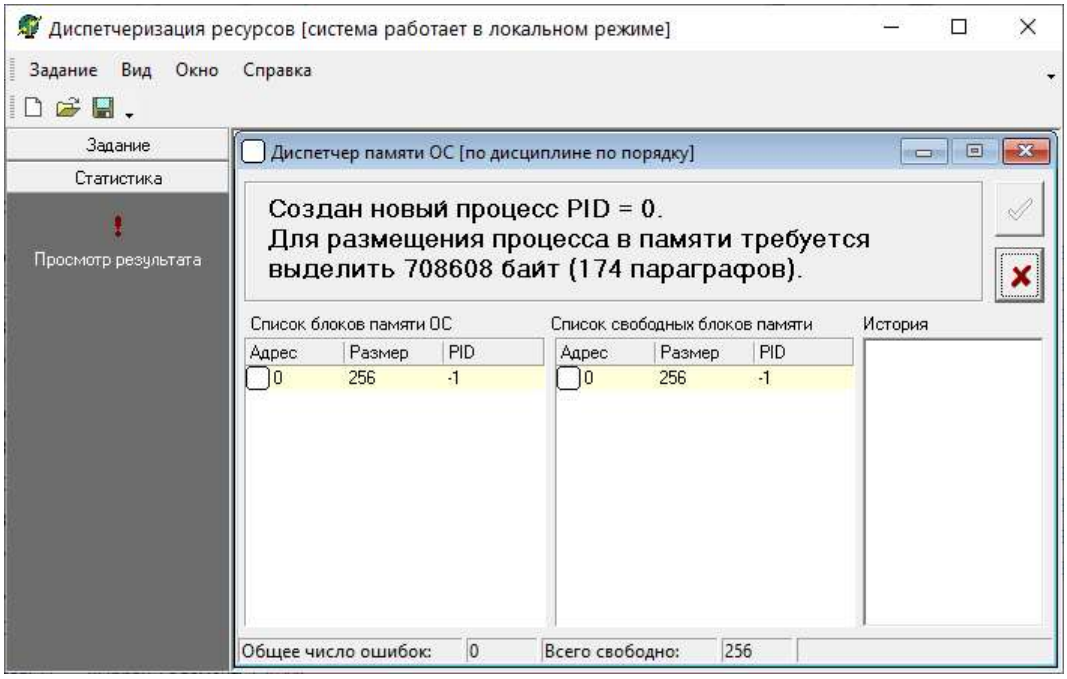


Рисунок 1 – Основное окно программы

Главное окно программы разделено на следующие области:

– заголовок с названием дисциплины;
– описание текущей заявки;
– список блоков памяти ОС;
– список свободных блоков памяти;
– блок кнопок: «Подтвердить» и «Отклонить»;
– список обработанных пользователем заявок;
– строка статуса.

В верхней части окна присутствует описание заявки и количество выполненных заявок. Задача пользователя – корректно обработать поступающие заявки.

В данной программной модели имеются следующие типы заявок:

– создание нового процесса;
– завершение процесса;

– выделение памяти существующему процессу;

– освобождение памяти, которой владеет процесс.

В левой части окна находится список блоков памяти. Для каждого блока указан его начальный адрес, размер в страницах, а также идентификатор процесса, владеющего данным блоком («-1» – нет владельца).

Пользователь имеет возможность выполнить следующие действия:

– ответить на запрос отказом, сразу нажав на кнопку "Отклонить";

– выделить память процессу. Для этого необходимо щелкнуть правой кнопкой мыши на свободном блоке в списке блоков памяти ОС и выбрать пункт «Выделить приложению». После этого в появившемся окне нужно ввести идентификатор процесса, которому выделяется память, а также количество выделяемых страниц;

– освободить память. Для этого необходимо щелкнуть правой кнопкой мыши на занятом блоке в списке блоков памяти ОС и выбрать пункт «Освободить»;

– объединить два свободных блока в один. Для этого необходимо щелкнуть правой кнопкой мыши на свободном блоке в списке блоков памяти ОС и выбрать пункт «Объединить со следующим», Данное действие необходимо выполнять каждый раз, когда образуются соседние свободные блоки;

– уплотнить (дефрагментировать память). Для этого необходимо щелкнуть правой кнопкой мыши на списке блоков памяти ОС и выбрать пункт «Уплотнение памяти»;

– подтвердить действия, нажав на кнопку "Подтвердить".

Основные дисциплины выбора свободного блока памяти:

– выбор первого подходящего блока. В этом случае список свободных блоков сортируется по адресу и выбирается первый блок, размер которого больше или равен требуемому размеру;

– выбор самого подходящего блока. Список свободных блоков сортируется по размеру (от меньшего к большему) и выбирается первый подходящий по размеру блок;

– выбор самого неподходящего блока. Самый неподходящий блок – это блок с максимальным размером. Список свободных блоков необходимо отсортировать в порядке убывания размера и выбрать первый блок.

В правой части экрана находится список выполненных шагов, которые хранят историю действий пользователя. Элемент в списке помечается красным цветом, если действия пользователя на данном шаге были некорректны. При наведении указателя мыши на элемент списка отображается информация о событии и действиях пользователя на данном шаге. Для отмены произвольного числа шагов необходимо щелкнуть мышью на элемент списка, до которого необходимо очистить историю. Отменённые шаги помечаются серым цветом. До тех пор, пока не выполнено какое-либо действие в окне диспетчера, можно восстановить действия пользователя, щелкнув на записи, соответствующей отмененному шагу.

В нижней части окна расположена строка статуса, в которой отображается общее число ошибок, которые допустил пользователь в процессе выполнения лабораторной работы, а также количество свободных страниц оперативной памяти.

По достижении заданного числа шагов выводится сообщение о выполнении лабораторной работы, а также количество неисправленных ошибок.

1.1.4 Недостатки

В текущей программной модели были найдены следующие недостатки:

– приложение доступно только для ОС Windows. Пользователи других операционных систем должны запускать Windows в виртуальной

машине либо использовать другие средства по запуску Windows-приложений в других ОС;

– нестабильность. В ходе выполнения лабораторной работы программная модель несколько раз аварийно завершалась, из-за чего результаты выполненной работы безвозвратно терялись;

– ошибки при проверке пользовательских действий. В некоторых случаях было замечено, что программная модель принимала правильную последовательность действий как ошибочную. Это в совокупности с нестабильностью программы усложняет изучение студентами программной модели и, как следствие, увеличивает время на выполнение лабораторной работы;

– нечеткость, «размытость» интерфейса на дисплеях со сверхвысоким разрешением (HiDPI);

– в связи с утерей исходного кода программы и сложностью дизассемблирования определить формат файла задания не представляется возможным, что препятствует разработке новых вариантов заданий.

1.2 Актуальность разработки

Лабораторная работа по данной теме (управление памятью в ОС) имеет важную роль в закреплении лекционного материала и предоставляет студентам возможность изучить более подробно устройство ОС.

Из-за того, что исходный код программной модели утерян, исправить ошибки и недостатки в текущей модели не представляется возможным. Поэтому было принято решение разработать новую программную модель, повторяющую функционал текущей, в которой будут исправлены вышеописанные ошибки и недостатки.

1.3 Техническое задание

1.3.1 Наименование программы

Наименование программы – "Модель диспетчера памяти операционный системы".

1.3.2 Краткая характеристика области применения

Программа предназначена для закрепления студентами лекционного материала по дисциплине «Операционные системы», а именно: управление памятью в операционных системах.

1.3.3 Назначение разработки

Функциональным назначением программы является предоставление студентам возможности изучить работу диспетчера памяти во время выполнения лабораторных работ.

Программа должна эксплуатироваться на ПК студентов, преподавателей и на ПК, установленных в учебных аудиториях Вятского государственного университета. Особые требования к конечному пользователю не предъявляются.

1.3.4 Требования к программе

Программа должна обеспечивать возможность выполнения перечисленных ниже функций:

1) функции генерации задания;

2) функции сохранения текущего прогресса выполнения задания в файл;
3) функции загрузки задания с сохраненным прогрессом выполнения из файла;
4) функции подсчета количества ошибок, сделанных в ходе выполнения задания;
5) функции просмотра и отмены действий, выполненных в ходе прохождения задания.

Надежное (устойчивое) выполнение программы должно быть обеспечено выполнением пользователем совокупности организационно-технических мероприятий, перечень которых приведен ниже:

1) организацией бесперебойного питания технических средств;
2) использованием лицензионного программного обеспечения.

Отказы программы возможны вследствие некорректных действий пользователя при взаимодействии с операционной системой. Во избежание возникновения отказов программы по указанной выше причине следует обеспечить работу конечного пользователя без предоставления ему административных привилегий.

В состав технических средств должен входить IBM-совместимый персональный компьютер, включающий в себя:

1) x86-совместимый процессор с тактовой частотой не меньше 1.0 ГГц;
2) дисплей с разрешением не меньше, чем 1024x768;
3) не менее 1 гигабайта оперативной памяти;
4) не менее 100 мегабайт свободного дискового пространства;
5) клавиатура, мышь.

Системные программные средства, используемые программой, должны быть представлены следующими операционными системами:

– 64-разрядная ОС Windows 7/8/8.1/10;

– 64-разрядная ОС Ubuntu 16.04 и выше.

Программа должна обеспечивать взаимодействие с пользователем посредством графического пользовательского интерфейса и предоставлять возможность выполнять наиболее частые операции с помощью сочетаний клавиш на клавиатуре.

1.3.5 Требования к программной документации

Состав программной документации должен включать в себя:

1) техническое задание;
2) программу и методики испытаний;
3) руководство пользователя;
4) техническую документацию;
5) исходный код.

1.3.6 Стадии и этапы разработки

Разработка должна быть проведена в три стадии:

1) разработка технического задания;
2) рабочее проектирование;
3) внедрение.

На стадии разработки технического задания должен быть выполнен этап разработки, согласования и утверждения настоящего технического задания.

На стадии рабочего проектирования должны быть выполнены перечисленные ниже этапы работ:

1) разработка программы;
2) разработка программной документации;
3) испытания программы.

На стадии внедрения должен быть выполнен этап подготовки и передачи программы Заказчику.

На этапе разработки технического задания должны быть выполнены перечисленные ниже работы:

1) постановка задачи;
2) определение и уточнение требований к техническим средствам;
3) определение требований к программе;
4) определение стадий, этапов и сроков разработки программы и документации на неё;
5) согласование и утверждение технического задания.

На этапе разработки программы должна быть выполнена работа по программированию (кодированию) и отладке программы.

На этапе разработки программной документации должна быть выполнена разработка программных документов в соответствии с требованиями ГОСТ 19.101-77 с требованием п. Предварительный состав программной документации настоящего технического задания.

На этапе испытаний программы должны быть выполнены перечисленные ниже виды работ:

1) разработка, согласование, утверждение программы и методики испытаний;
2) проведение приемо-сдаточных испытаний;
3) корректировка программы и программной документации по результатам испытаний.

На этапе подготовки и передачи программы должна быть выполнена работа по подготовке и передаче программы и программной документации в эксплуатацию Заказчику.

2 Разработка структуры приложения

На данном этапе работы необходимо в соответствии с требованиями, поставленными в техническом задании, разработать алгоритмы функционирования и модульную структуру приложения.

2.1 Разработка алгоритмов функционирования

В данной программной модели состояние памяти определяется списком всех блоков памяти, упорядоченных по начальному адресу, а также списком свободных блоков памяти.

Во время обработки заявки пользователь выполняет некоторую последовательность действий из шагов определенного типа. Данные шаги можно представить как операции, применяемые к состоянию памяти.

Для проверки действий, выполненных пользователем над состоянием памяти, необходимо разработать алгоритмы обработки поступающих заявок.

2.1.1 Операции изменения состояния памяти

Каждая операция, описанная ниже, выполняет определенные преобразования над состоянием памяти. Результатом операции является новое состояние памяти.

Операция выделения памяти процессу в заданном блоке памяти:

1) рассчитать параметры для нового блока памяти, который будет выделен процессу: идентификатор процесса, размер, адрес;
2) рассчитать параметры для нового свободного блока памяти: размер, адрес;
3) удалить заданный блок памяти из списка блоков памяти и из списка свободных блоков памяти;

4) добавить в список блоков памяти новый блок памяти, выделенный процессу;
5) если размер нового свободного блока памяти больше 0, то добавить новый свободный блок памяти в список свободных блоков памяти.

Пример применения данной операции представлен на рисунке 2.

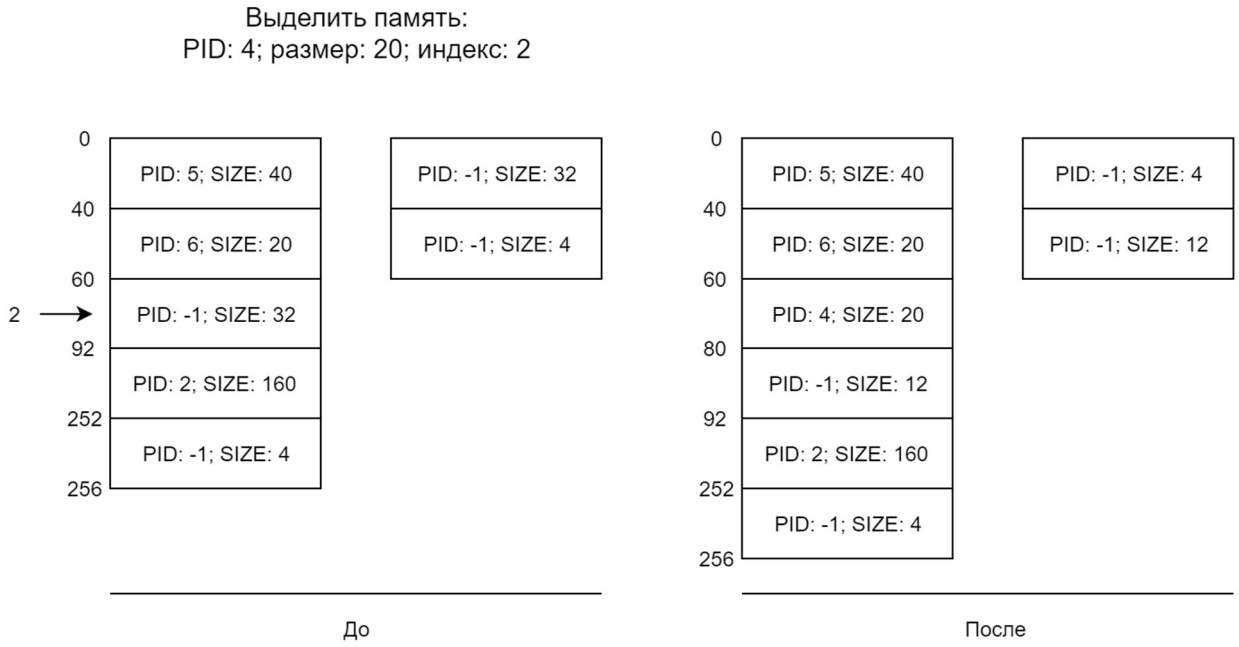


Рисунок 2 – Пример операции выделения памяти

Операция освобождения блока памяти, принадлежащего процессу:

1) параметру «Идентификатор процесса» заданного блока памяти присвоить новое значение: -1;
2) добавить данный блок памяти в список свободных блоков памяти.

Пример применения данной операции представлен на рисунке 3.

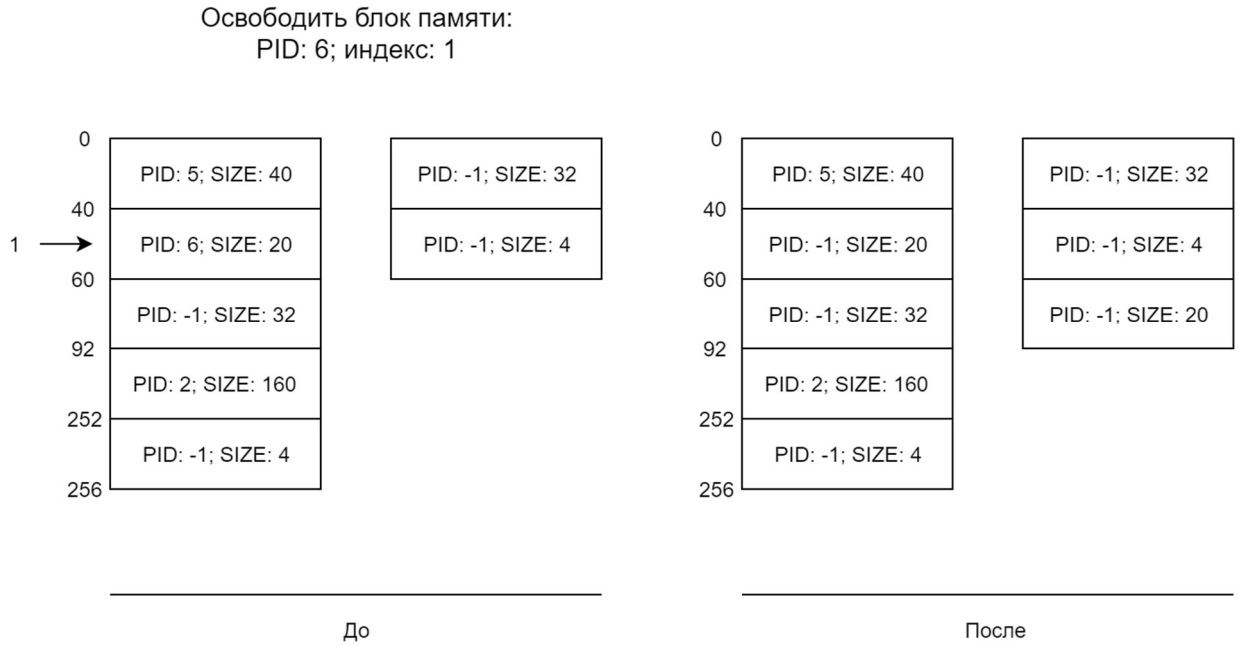


Рисунок 3 – Пример операции освобождения блока памяти

Операция дефрагментации памяти:

1) вычислить суммарный объем свободной памяти;
2) перенести все занятые блоки памяти в новый список;
3) заново рассчитать для них начальные адреса;
4) создать новый список свободных блоков памяти, добавить в него единственный блок с размером равным суммарному объему свободной памяти;
5) добавить новый свободный блок памяти в список свободных блоков памяти.

Пример применения данной операции представлен на рисунке 4.

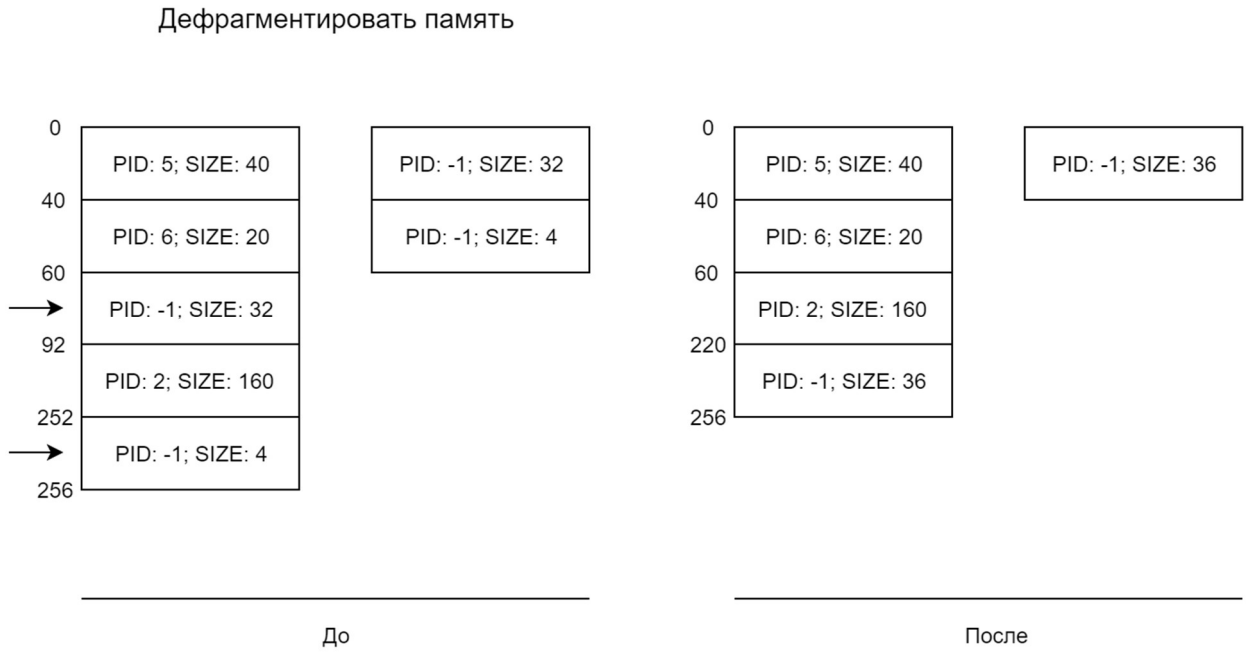


Рисунок 4 – Пример операции дефрагментации памяти

Операция сжатия памяти:

1) выделить последовательность непрерывно идущих друг за другом свободных блоков памяти, начиная с заданного номера;
2) вычислить суммарный объем свободной памяти в данной последовательности;
3) удалить из списка свободных блоков памяти и списка всех блоков памяти входящие в эту последовательность свободные блоки;
4) добавить новый свободный блок памяти с размером, равным объему свободной памяти в данной последовательности, в список свободных блоков памяти и в список всех блоков памяти.

Пример применения данной операции представлен на рисунке 5.

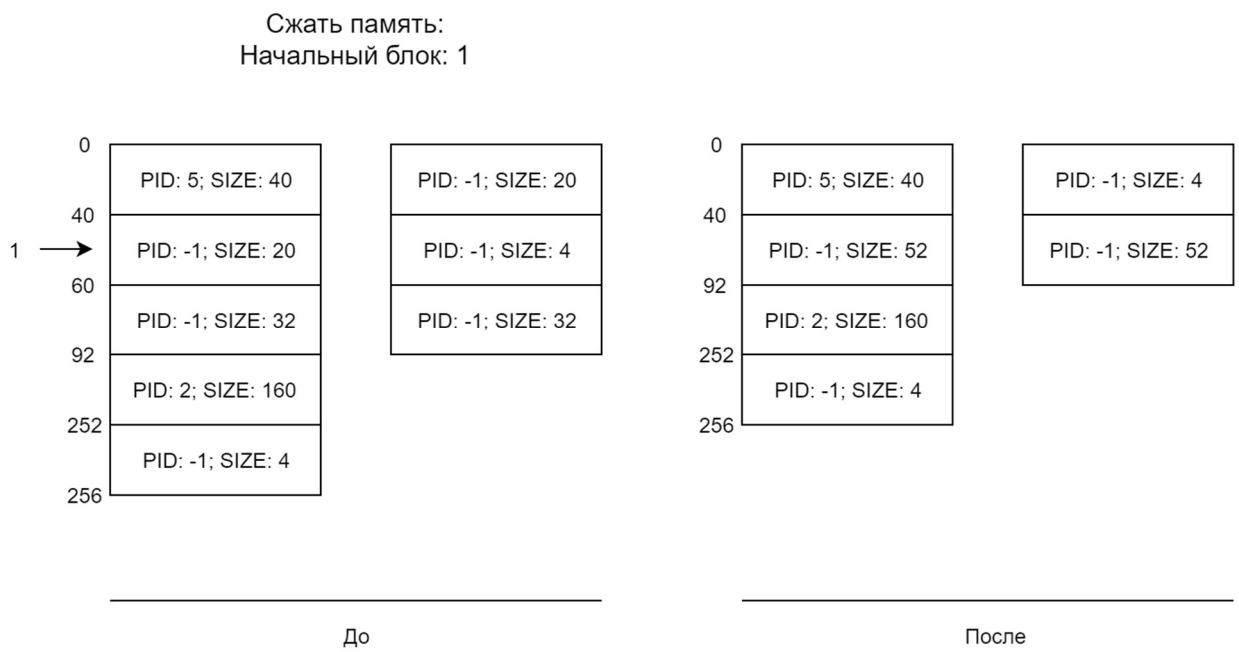


Рисунок 5 – Пример операции дефрагментации памяти

2.1.2 Алгоритмы обработки поступающих заявок

Алгоритм обработки заявки «Создан новый процесс»:

1) сначала нужно удостовериться в том, что процессу не выделены какие-либо блоки памяти:
   1.1) если процессу выделены блоки памяти, то считается, что процесс уже создан, а следовательно, заявка некорректная. Никаких действий над памятью не выполнять; алгоритм завершен;
   1.2) в противном случае перейти к пункту 2;
2) найти в соответствии с дисциплиной свободный блок памяти с размером не меньше, чем запрашивается;
3) если такой блок есть, то выделить в нем память процессу и перейти к пункту 6;

4) если такого блока нет, но суммарно свободной памяти достаточно, то выполнить дефрагментацию памяти, в новом свободном блоке выделить память процессу и перейти к пункту 6;
5) если свободной памяти недостаточно, то никаких действий над памятью не выполнять; алгоритм завершен;
6) упорядочить список свободных блоков памяти в соответствии с дисциплиной.

Схема алгоритма обработки заявки «Создан новый процесс» представлена на рисунке 6.

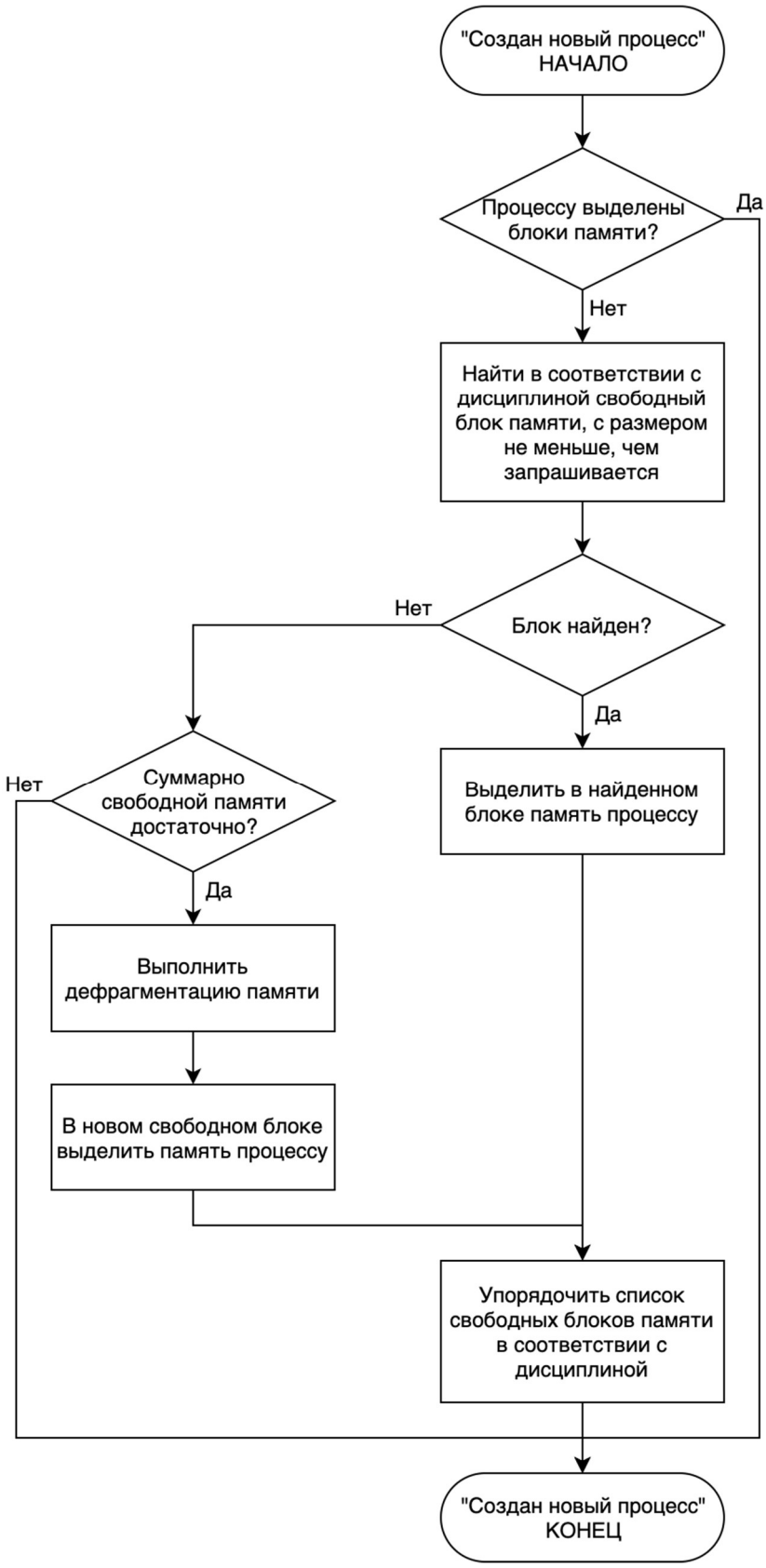


Рисунок 6 – Схема алгоритма обработки заявки «Создан новый процесс»

Алгоритм обработки заявки «Завершение процесса»:

1) проверить, выделены ли данному процессу какие-либо блоки памяти:
   1.1) если процессу не выделено никаких блоков памяти, то считается, что процесс не существует, а следовательно, заявка некорректная. Никаких действий над памятью не выполнять; алгоритм завершен;
   1.2) в противном случае перейти к пункту 2;
2) освободить все блоки памяти, принадлежащие данному процессу;
3) объединить соседние свободные блоки памяти;
4) упорядочить список свободных блоков памяти в соответствии с дисциплиной.

Алгоритм обработки заявки «Выделение памяти существующему процессу»:

1) сначала нужно удостовериться в том, что процессу выделены какие-либо блоки памяти:
   1.1) если процессу не выделены блоки памяти, то считается, что процесс не существует, а следовательно, заявка некорректная. Никаких действий над памятью не выполнять; алгоритм завершен;
   1.2) в противном случае перейти к пункту 2;
2) найти в соответствии с дисциплиной свободный блок памяти с размером не меньше, чем запрашивается;
3) если такой блок есть, то выделить в нем память процессу и перейти к пункту 6;
4) если такого блока нет, но суммарно свободной памяти достаточно, то выполнить дефрагментацию памяти, в новом свободном блоке выделить память процессу и перейти к пункту 6;

5) если свободной памяти недостаточно, то никаких действий над памятью не выполнять; алгоритм завершен;
6) упорядочить список свободных блоков памяти в соответствии с дисциплиной.

Схема алгоритма обработки заявки «Выделение памяти существующему процессу» представлен на рисунке 7.

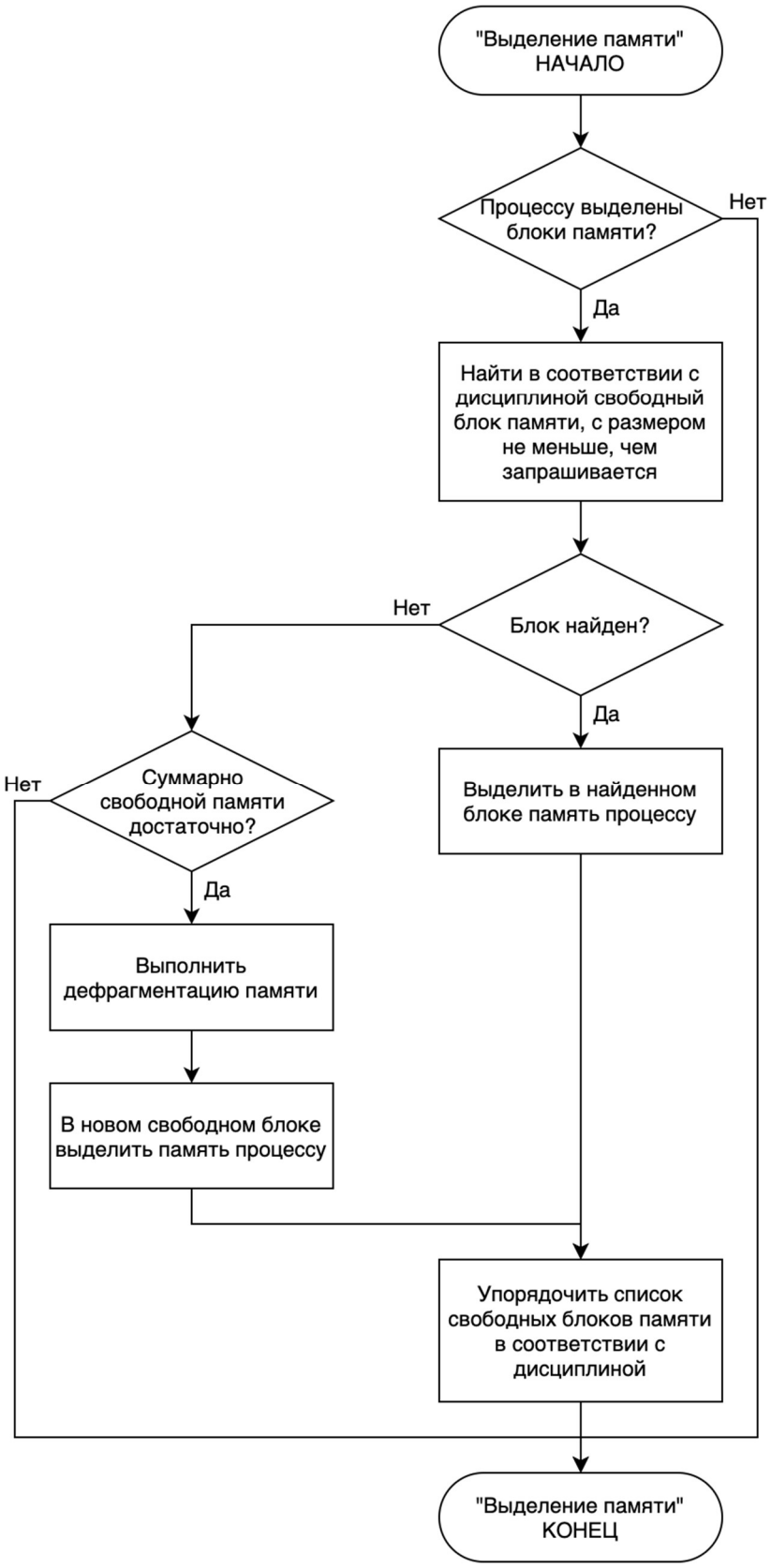


Рисунок 7 – Схема алгоритма обработки заявки «Выделение памяти существующему процессу»

Алгоритм обработки заявки «Освобождение памяти, которой владеет процесс»:

1) найти блок с заданным начальным адресом;
2) обработать следующие исключительные ситуации:
   2.1) блок не найден; алгоритм завершен;
   2.2) блок выделен другому процессу; алгоритм завершен;
3) освободить найденный блок памяти;
4) объединить соседние свободные блоки памяти;
5) упорядочить список свободных блоков памяти в соответствии с дисциплиной.

2.1.3 Алгоритм генерации заданий

Основная идея алгоритма генерации заданий заключается в том, что каждая следующая заявка создается на основе текущего состояния памяти (первая заявка генерируется на основе начального состояния памяти). При этом тип дисциплины, тип заявки, корректность или некорректность заявки, а также параметры заявки определяются случайным образом.

Алгоритм генерации заданий состоит из следующих шагов:

1) выбрать случайным образом дисциплину;
2) создать пустой список заявок задания;
3) на основе текущего состояния памяти сгенерировать корректные и некорректные заявки всех типов (по одной на каждый тип);
4) случайным образом определить должна ли текущая заявка быть корректной или нет;
5) если текущая заявка должна быть корректной, то из списка сгенерированных корректных заявок случайным образом выбрать одну и добавить ее в список заявок задания. В противном случае использовать список сгенерированных некорректных заявок;

6) если сгенерировано достаточное количество заявок, то завершить алгоритм, в противном случае перейти к пункту 7;
7) обновить текущее состояние памяти в соответствии с выбранной заявкой и дисциплиной, перейти к пункту 3.

Схема алгоритма генерации задания представлена на рисунке 8.

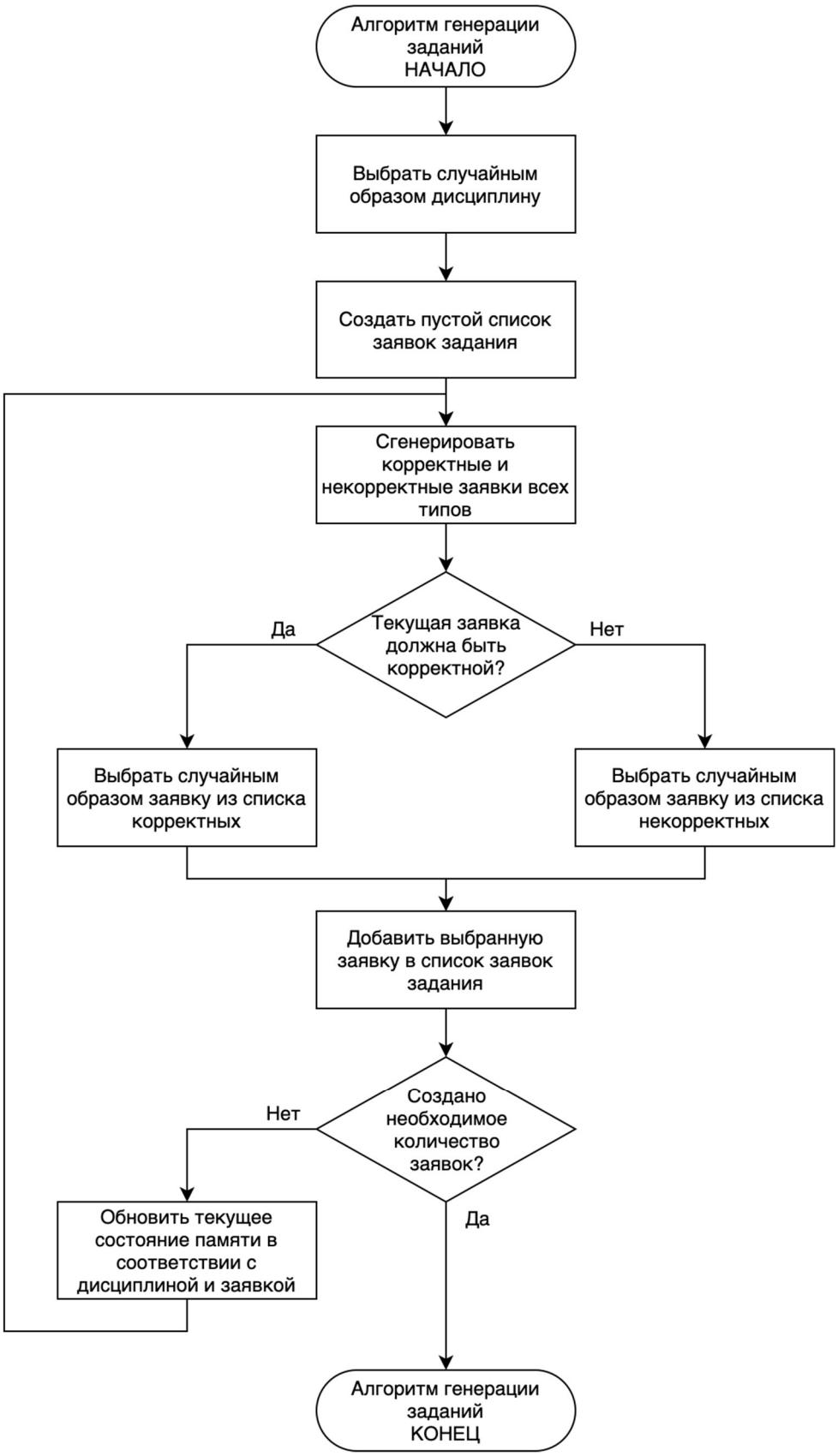


Рисунок 8 – Схема алгоритма генерации задания

2.2 Разработка модульной структуры приложения

Для обеспечения функционирования системы разработана обобщенная структура программного продукта, представляющая собой набор взаимосвязанных модулей, которые реализуют используемые алгоритмы функционирования. Модульная структура приложения представлена на рисунке 9.

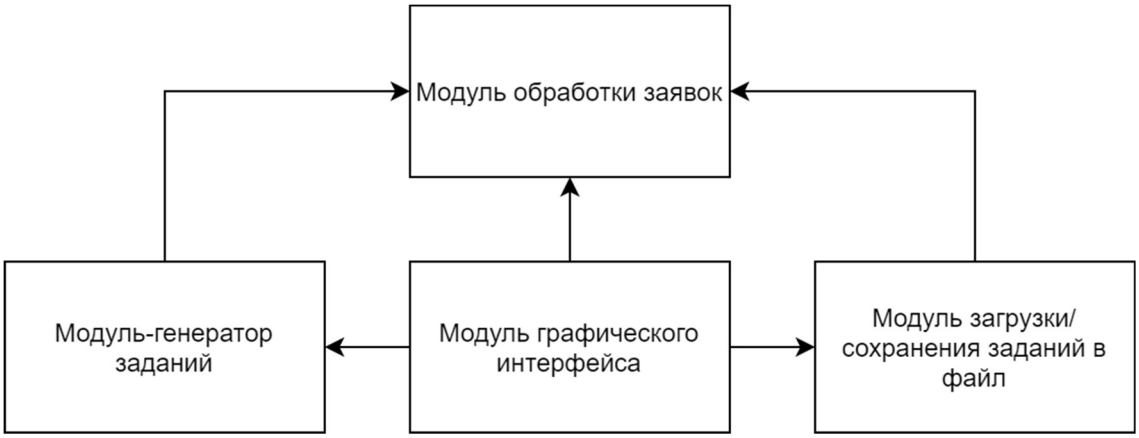


Рисунок 9 – Модульная структура приложения

Каждый из модулей имеет следующее назначение и функционал:

– модуль обработки заявок. Определяет алгоритмы обработки заявок и проверки выполненных пользователем действий, а также определяет необходимые типы данных для представления заявок, состояния памяти и собственно задания на лабораторную работы;

– модуль-генератор заданий. Определяет алгоритмы генерации заданий. Обеспечивает достаточную степень уникальности задания для каждого пользователя без необходимости разработки заданий вручную;

– модуль загрузки и сохранения заданий в файл. Предоставляет возможность сохранять текущее состояние выполняемого задания между запусками приложения, а также обеспечивает защиту от непредвиденного изменения структуры файла;

– модуль графического интерфейса. Является связующим звеном между приложением и пользователем; отображает данные о ходе выполнения задания в текстовом и графическом виде.

3 Программная реализация

На данном этапе работы необходимо выбрать инструменты для кодирования приложения (язык программирования, библиотеки), выполнить реализацию разработанных ранее модулей и алгоритмов, а также разработать графический интерфейс пользователя.

3.1 Выбор инструментов разработки

Так как приложение должно запускаться на различных ОС, а также быть простым в установке и запуске, было принято решение выбрать компилируемый язык программирования с возможностью создания родных для платформы исполняемых файлов. Помимо прочего, для реализации графического интерфейса необходимо выбрать кроссплатформенную библиотеку для его построения.

Среди доступных языков программирования имеются:

– Golang;

– C++;

– Rust;

– C# (.NET Core 3).

Среди кроссплатформенных библиотек для построения графического интерфейса имеются:

– GTK (от сокращения GIMP Toolkit) – кроссплатформенная библиотека элементов интерфейса. Написана на C, доступны интерфейсы для других языков программирования, в том числе для C++, C#, Python. Для проектирования графического интерфейса доступно приложение Glade;

– Qt. Кроссплатформенный фреймворк для разработки программного обеспечения на языке программирования C++. Включает в себя все основные классы, которые могут потребоваться при разработке

прикладного программного обеспечения, начиная от элементов графического интерфейса и заканчивая классами для работы с сетью, базами данных и XML. Комплектуется визуальной средой разработки графического интерфейса Qt Designer, позволяющей создавать диалоги и формы в режиме WYSIWYG;

– wxWidgets. Кроссплатформенная библиотека инструментов с открытым исходным кодом для разработки кроссплатформенных на уровне исходного кода приложений. Основным применением wxWidgets является построение графического интерфейса пользователя, однако библиотека включает большое количество других функций и используется для создания весьма разнообразного ПО. Важная особенность wxWidgets: в отличие от некоторых других библиотек (GTK, Qt и др.), она максимально использует «родные» графические элементы интерфейса операционной системы всюду, где это возможно. Это существенное преимущество для многих пользователей, поскольку они привыкают работать в конкретной среде, и изменения интерфейса программ часто вызывают затруднения в их работе. Написана на C++. Для проектирования графического интерфейса доступно приложение wxGlade;

– AvaloniaUI. Кроссплатформенный XAML-фреймворк для построения графических интерфейсов пользователя для платформ .NET Framework, .NET Core и Mono.

При этом нужно учитывать, что не все комбинации языков программирования с библиотеками графического интерфейса возможны, удобны в разработке и достаточно стабильны. Для языков Golang и Rust не имеются стабильные версии перечисленных библиотек, для GTK сборка C++-приложений под Windows требует нетривиальной настройки окружения вплоть до эмуляции Linux-окружения в Windows, а AvaloniaUI доступна только под C# и не имеет еще стабильной версии. Среди доступных вариантов остаются:

– C++ и wxWidgets;

– C# и GTK;

– C++ и Qt.

В ходе ознакомления с библиотекой wxWidgets были выявлены следующие недостатки: недостаточно подробная документация, нетривиальная настройка режима Drag’n’Drop (необходим для перемещения элементов ГПИ, представляющих блоки памяти), скудный функционал конструктора форм wxGlade.

У GTK графические элементы выглядят достаточно непривычно в сравнении с родными для Windows приложениями; имеет те же недостатки, что и wxWidgets.

По степени интеграции выигрывает Qt со средой разработки QtCreator.

В качестве системы сборки выбрана qmake – система сборки, специально разработанная для фреймворка Qt.

3.2 Реализация модулей приложения

3.2.1 Модуль обработки заявок

В данном модуле должны быть реализованы операции и алгоритмы обработки заявок, а также определены следующие необходимые типы:

– структура MemoryBlock. Описывает отдельно взятый блок памяти. Основными атрибутами являются: PID (тип int), адрес начала блока (тип int) и размер (тип int). Для доступа к атрибутам реализованы методы pid(), address() и size() соответственно;

– структура MemoryState. Описывает состояние памяти. Содержит два атрибута: список блоков памяти, упорядоченных по начальному адресу (тип std::vector<MemoryBlock>) и список свободных блоков памяти (тип std::vector<MemoryBlock>), упорядоченных в соответствии с выбранной дисциплиной, однако их порядок может меняться при применении операций;

– структуры, описывающие заявки:
  - CreateProcessReq. Основные атрибуты: PID, количество запрашиваемой памяти в байтах, количество запрашиваемой памяти в страницах;
  - TerminateProcessReq. Атрибут: PID;
  - AllocateMemory. Основные атрибуты: PID, количество запрашиваемой памяти в байтах, количество запрашиваемой памяти в страницах;
  - FreeMemory. Основные атрибуты: PID, адрес начала блока памяти.

Для обеспечения возможности хранить заявки различного типа в таких контейнерах, как std::vector, был добавлен тип-сумма Request, являющийся типом std::variant.

В ходе анализа алгоритмов было выяснено, что стратегии отличаются между собой только способом сортировки свободных блоков памяти. Поэтому было принято решение закодировать алгоритмы в виде методов отдельного класса – AbstractStrategy, а особенность каждой дисциплины (способ сортировки блоков памяти) определить через виртуальный метод sortFreeBlocks(), реализованный в классах дисциплин, производных от базового класса AbstractStrategy:

– FirstAppropriateStrategy – дисциплина «Первый подходящий»;
– MostAppropriateStrategy – дисциплина «Наименее подходящий»;
– LeastAppropriateStrategy – дисциплина «Наименее подходящий».

Диаграмма классов данного модуля представлена на рисунке 10.

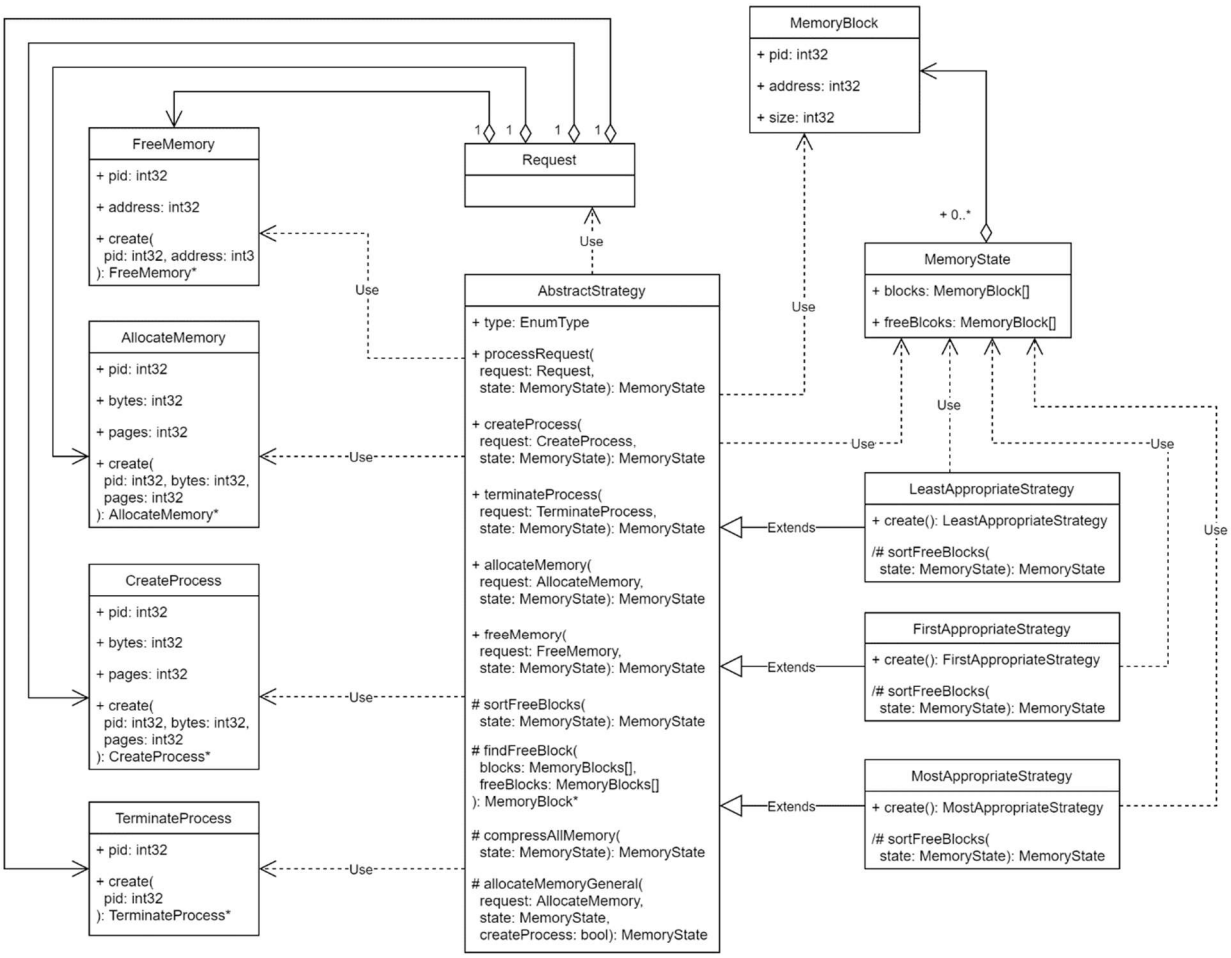


Рисунок 10 – Диаграмма классов модуля обработки заявок

3.2.2 Модуль загрузки и сохранения заданий в файл

Данный модуль реализует функционал сохранения текущего прогресса выполнения задания в файл, загрузку задания из файла с защитой от непредвиденных изменений.

Состояние задания определяется такими параметрами как:

- список всех заявок;
- количество уже выполненных заявок;
- количество допущенных пользователем ошибок;
- тип дисциплины;
- состояние памяти после последней выполненной заявки.

Все перечисленные выше параметры должны быть сохранены в файл. Последний параметр позволяет защитить содержимое файла от изменений, направленных на подделку результатов выполнения задания, потому что для этого атакующему необходимо будет самостоятельно обработать заявки, чтобы получить искомое состояние памяти.

В качестве формата файла задания был выбран JSON, потому что он имеет синтаксис, достаточно удобный как для человека, так и для машины. JSON является текстовым форматом, а не бинарным, поэтому файлы в формате JSON можно просматривать и редактировать в любом текстовом редакторе.

Задания в файле сохранены в виде массива JSON-объектов. Структура объекта задания представлена в таблице 1.

Таблица 1 – Структура объекта задания

Поле	Тип	Описание
type	String	Тип задания. Значение: “MEMORY_TASK”
strategy	String	Название дисциплины. Допустимые значения: “FIRST_APPROPRIATE”, “MOST_APPROPRIATE”, “LEAST_APPROPRIATE”
completed	Number	Количество обработанных заявок
fails	Number	Количество допущенных пользователем ошибок
state	Object	Объект, описывающий состояние памяти
requests	Array	Массив заявок, которые диспетчер должен обработать

В объекте, описывающем состояние памяти, (поле “state”) хранятся списки всех блоков памяти, упорядоченных по адресу. Структуры объектов

состояния памяти, блока памяти и заявки представлены в таблицах 2, 3 и 4 соответственно.

Таблица 2 – Структура объекта состояния памяти

Поле	Тип	Описание
blocks	Array	Массив из дескрипторов всех доступных блоков памяти

Таблица 3 – Структура объекта блока памяти

Поле	Тип	Описание
pid	Number	PID процесса, которому выделен данный блок или -1, если блок свободный
address	Number	Адрес начала блока памяти в страницах
size	Number	Размер блока памяти в страницах

Таблица 4 – Структура объекта заявки

Заявка на создание процесса		
Поле	Тип	Описание
type	String	Тип заявки. Значение: “CREATE_PROCESS”
pid	Number	PID процесса
bytes	Number	Запрашиваемый объем памяти в байтах
Заявка на завершение процесса		
Поле	Тип	Описание
type	String	Тип заявки. Значение: “TERMINATE_PROCESS”
pid	Number	PID процесса
Заявка на выделение блока памяти процессу		
Поле	Тип	Описание
type	String	Тип заявки. Значение: “ALLOCATE_MEMORY”
pid	Number	PID процесса
bytes	Number	Запрашиваемый объем памяти в байтах
Заявка на освобождение блока памяти		
Поле	Тип	Описание
type	String	Тип заявки. Значение: “FREE_MEMORY”
pid	Number	PID процесса
address	Number	Адрес начала блока памяти в страницах

Пример содержимого файла задания представлен на рисунке 11.

```
[
  {
    "completed": 0,
    "requests": [
      { "bytes": 376831, "pid": 1, "type": "CREATE_PROCESS" },
      { "bytes": 102399, "pid": 1, "type": "ALLOCATE_MEMORY" },
      { "bytes": 180223, "pid": 3, "type": "CREATE_PROCESS" },
      { "pid": 5, "type": "TERMINATE_PROCESS" },
      { "address": 147, "pid": 1, "type": "FREE_MEMORY" }
    ],
    "state": {
      "blocks": [
        { "address": 0, "pid": -1, "size": 256 }
      ]
    },
    "strategy": "FIRST_APPROPRIATE",
    "type": "MEMORY_TASK"
  }
]
```


Рисунок 11 – Пример содержимого файла задания

Загрузка заданий из файла реализована в функции loadTasks(), сохранение – saveTasks().

Диаграмма классов данного модуля представлена на рисунке 12.

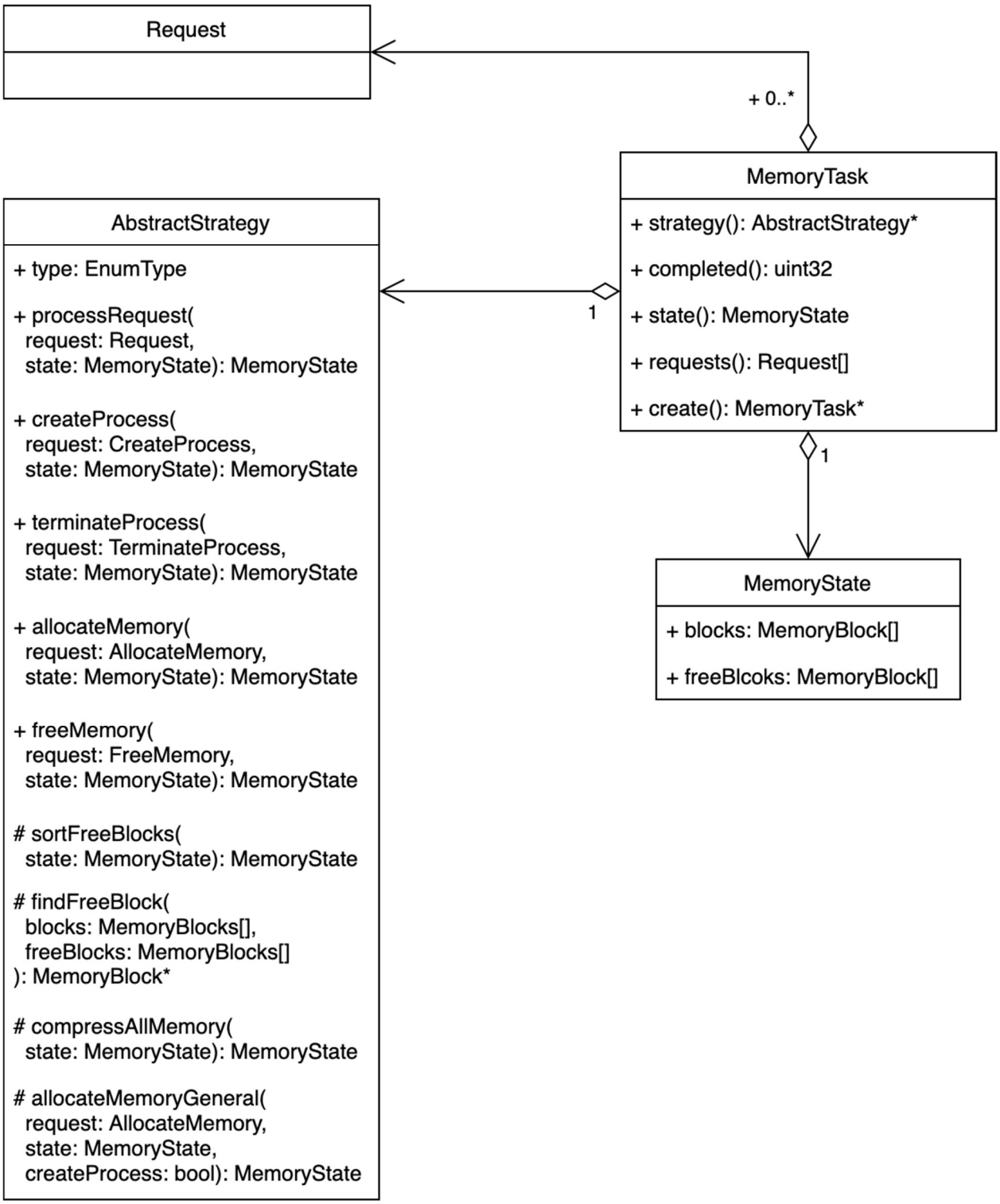


Рисунок 12 – Диаграмма классов модуля загрузки и сохранения задания в файл

3.2.3 Модуль-генератор заданий

Данный модуль реализован в виде набора функций, выполняющих следующие задачи:

– функции генерации заявок на основе данного состояния памяти: genCreateProcess(), genTerminateProcess(), genAllocateMemory(), genFreeMemory();

– функция выбора случайным образом дисциплины;

– функция generate(), использующая описанные выше функции для построения задания. Количество заявок по умолчанию – 40.

Опытным путем было выбрано соотношение между количеством некорректных и корректных заявок – 1 к 7.

Также в генераторе установлено ограничение на максимальное количество процессов – 16.

3.2.4 Модуль графического интерфейса

Проанализировав интерфейс предыдущей установки, было принято решение внести в него несколько небольших изменений:

– кнопку «Отклонить» заменить на кнопку «Сбросить». При нажатии на нее все изменения, сделанные в ходе обработки текущей заявки, будут сброшены;

– вместо пиктограмм на кнопках будут содержаться соответствующие их смыслу надписи;

– для удобства пользования для кнопок «Подтвердить» и «Отклонить» добавлены клавиатурные сочетания «Alt+Enter» и «Ctrl+Z» соответственно;

– для свободных и занятых блоков памяти сделаны более заметные пиктограммы в виде кружков разного цвета;

– при попытке закрыть основное окно программы будет выводиться диалог подтверждения, чтобы исключить потерю результатов из-за случайного закрытия программы;

– на случай поиска и устранения непредвиденных неполадок сделано автоматическое сохранение сгенерированного задания в файл во временной папке пользователя. Для доступа к ней в меню приложения добавлен отдельный пункт (рисунок 15).

Экранная форма основного окна программы представлена на рисунке 13.

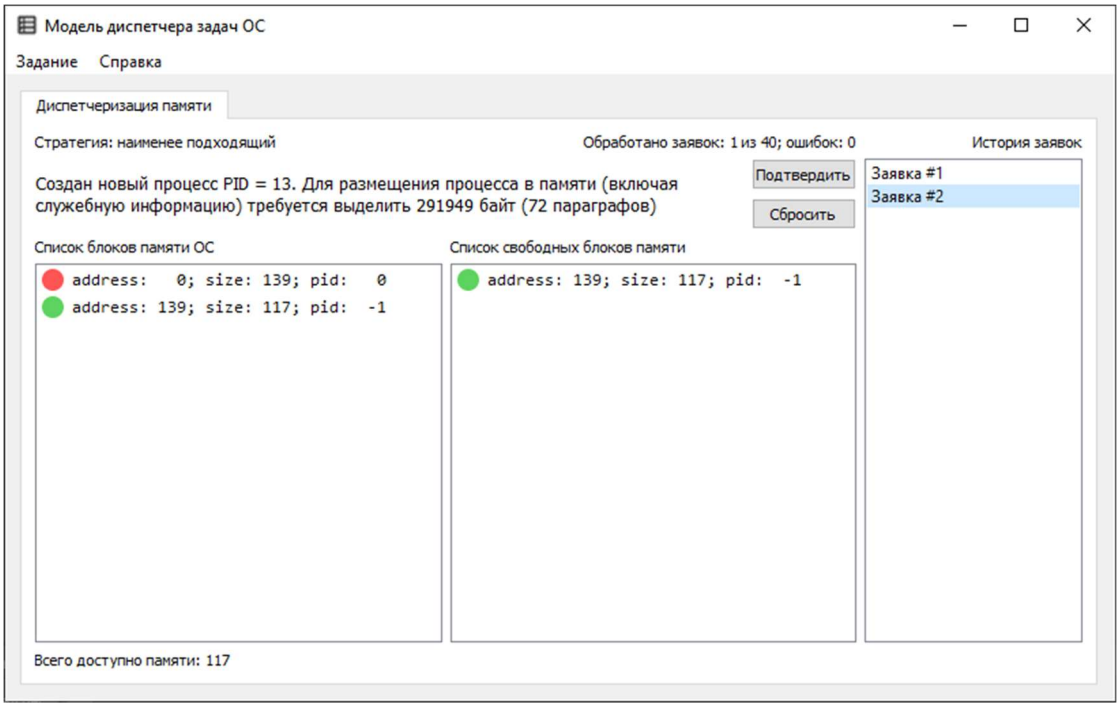


Рисунок 13 – Экранная форма основного окна программы

Диаграмма классов данного модуля представлена на рисунке 14.

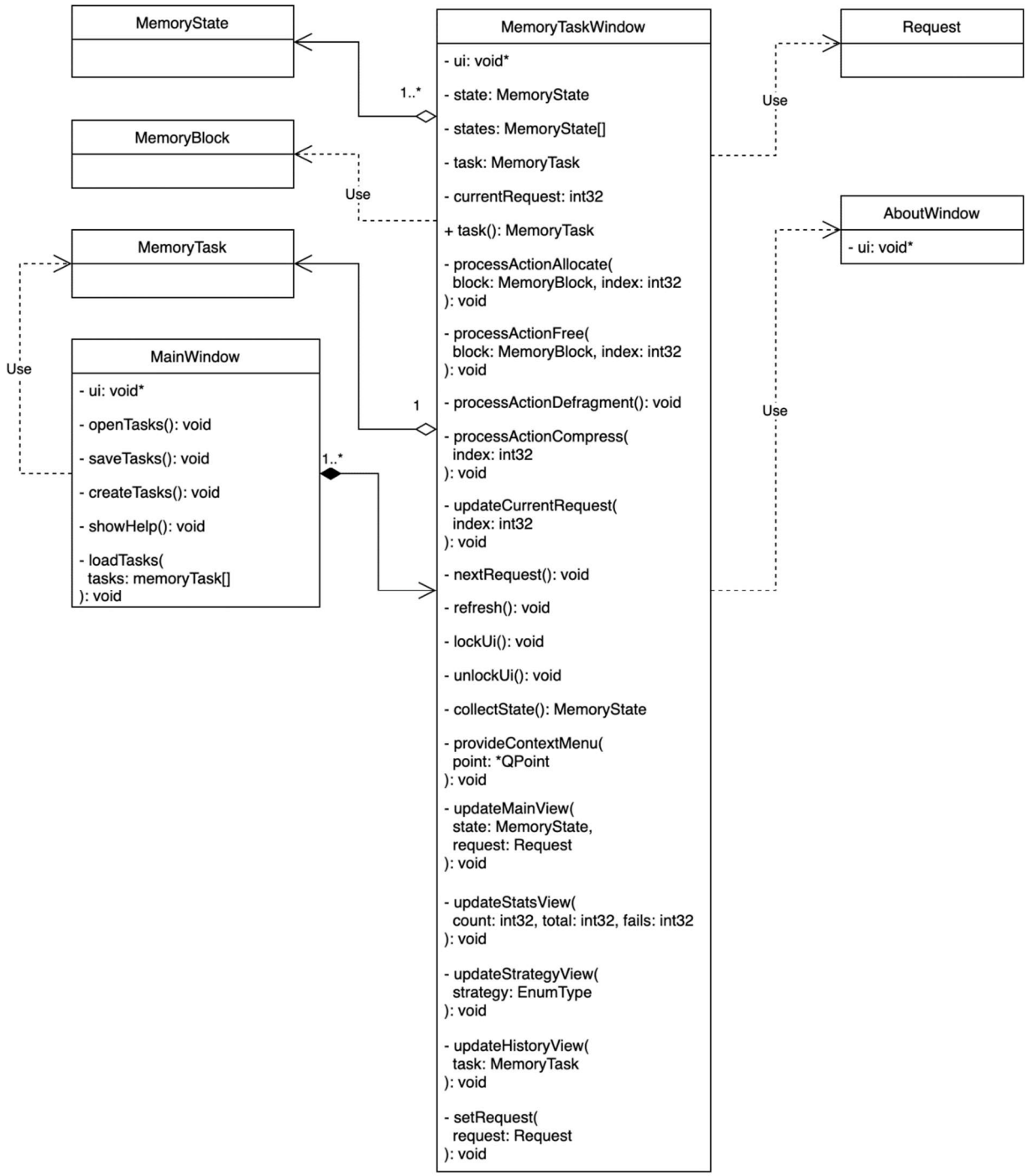


Рисунок 14 – Диаграмма классов модуля графического интерфейса

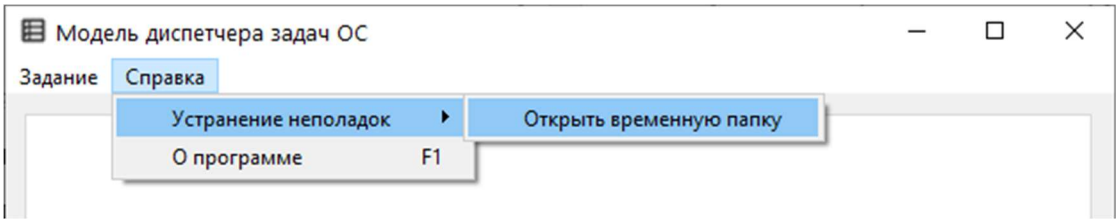


Рисунок 15 – Пункт меню для доступа к временной папке

На основе разработанной структуры приложения и алгоритмов функционирования была выполнена программная реализация приложения. Модули приложения были реализованы в виде классов и функций; в графический интерфейс пользователя были внесены небольшие улучшения, а также был разработан формат файлов заданий. Диаграмма разработанных классов представлена в приложении А.

Заключение

В ходе выполнения курсового проекта было разработано программное обеспечение – лабораторная установка «Модель диспетчера памяти операционной системы». Изначально предполагалось исправление и доработка существующей установки, однако такая задача оказалось крайне сложной в сравнении с разработкой нового приложения.

В качестве направления дальнейшего развития можно выбрать разработку отдельного конструктора заданий для преподавателя и обеспечение шифрования файлов заданий. Также планируется разработка лабораторной установки для планировщика процессов операционной системы.

Перечень сокращений

ГПИ – графический пользовательский интерфейс

ОС – операционная система

ОП – оперативная память

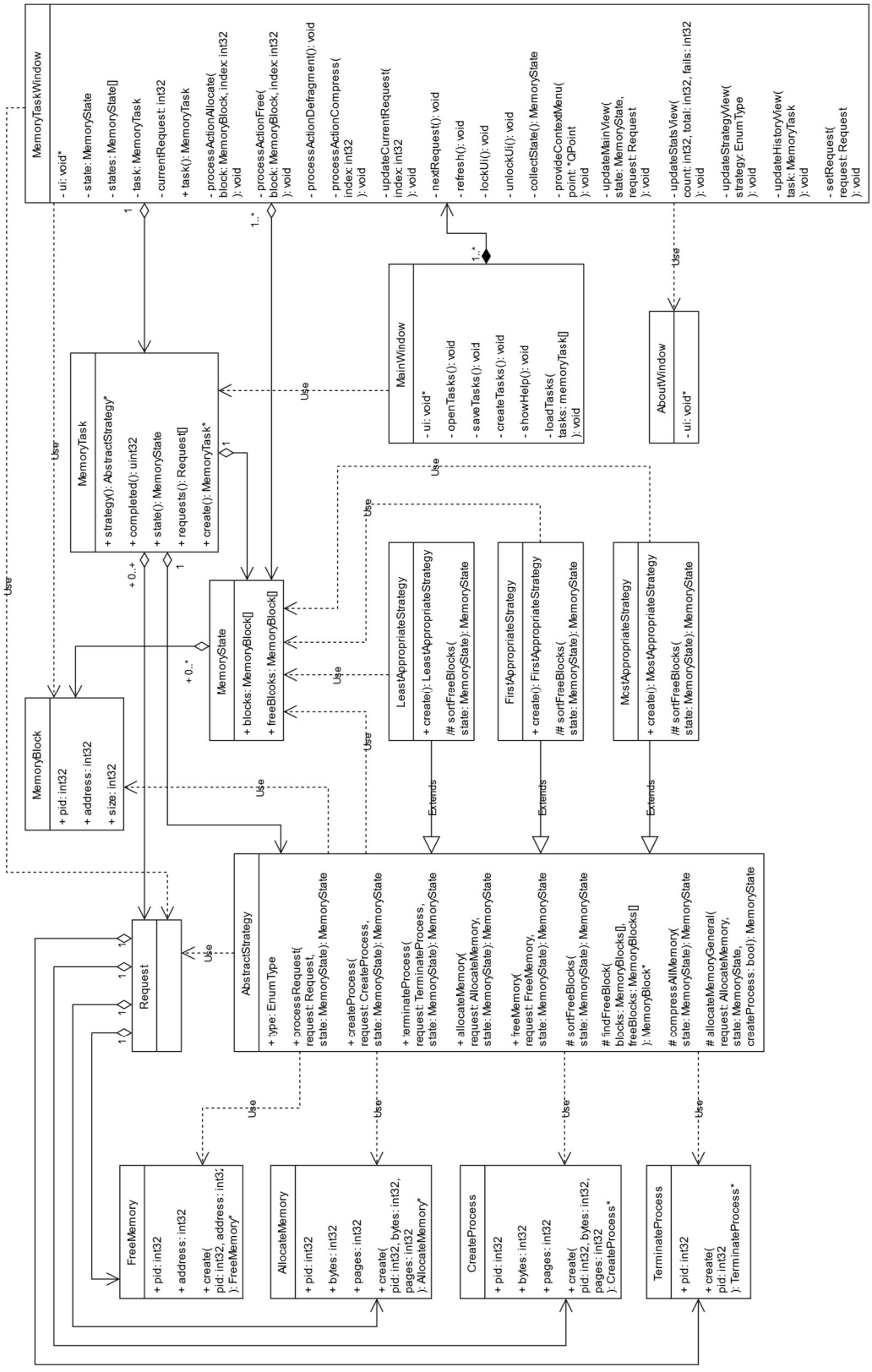
ПК – персональный компьютер

PID – process identifier, идентификатор процесса

Приложение А

(обязательное)

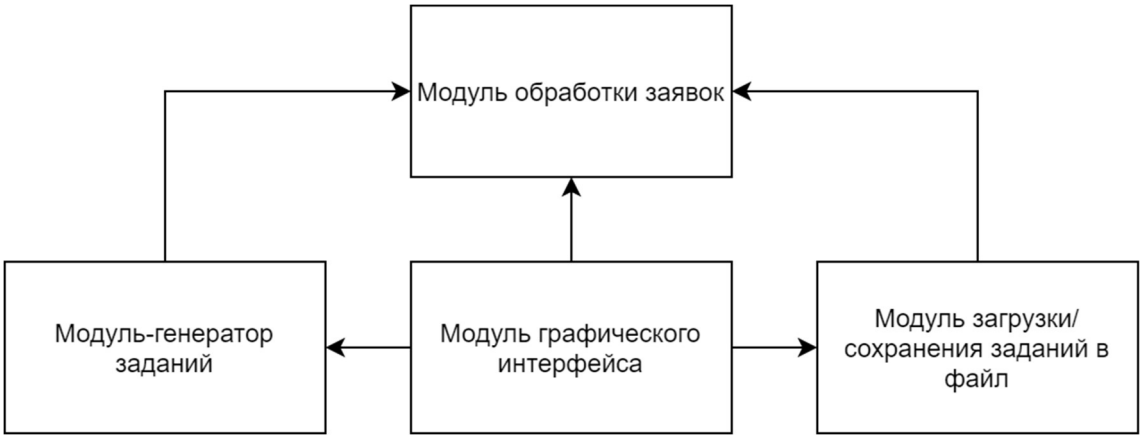
Диаграмма классов



Приложение Б

(обязательное)

Модульная структура приложения



Приложение В

(справочное)

Библиографический список

1 Караваева О. В. Операционные системы. Управление памятью и процессами: учебно-методическое пособие / О. В. Караваева. – Киров: ФГБОУ ВО «ВятГУ», 2016. – 39 с.

2 Avalonia мои за и против / Хабр [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/471728/ – Загл. с экрана.

3 Qt Reference Pages | Qt 5.14 [Электронный ресурс]. – Режим доступа: https://doc.qt.io/qt-5/reference-overview.html. – Загл. с экрана. – Англ.

4 AvaloniaUI/Avalonia: A multi-platform .NET UI framework [Электронный ресурс]. – Режим доступа: https://github.com/AvaloniaUI/Avalonia. – Загл. с экрана. – Англ.

5 wxWidgets – Wikipedia [Электронный ресурс]. – Режим доступа: https://en.wikipedia.org/wiki/WxWidgets. – Загл. с экрана. – Англ.

6 GTK – Wikipedia [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/GTK. – Загл. с экрана. – Англ.

7 Qt – Wikipedia [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Qt. – Загл. с экрана. – Англ.