Санкт-Петербургский Государственный Университет

Математико-механический факультет

Кафедра системного программирования

Обучение информатике в школах и ВУЗах на примере ОСРВ Embox

Курсовая работа студента 445 группы
Дзендзик Дарьи Анатольевны

Научный руководитель: Бондарев А. В.

аспирант кафедры СП СПбГУ

Санкт-Петербург

2012

Оглавление

Введение

Описание проблемы

В вузах, занимающихся подготовкой IT-специалистов, бытует мнение, что самое главное – это научить студента думать и решать задачи, а программировать он и так научится. Для промышленности же важно именно умение программировать, а не способность решать задачи, так как задачи, как правило, довольно типовые. В итоге специалистов, только что защитивших диплом и способных сразу же работать в промышленности, практически нет.

Эту проблему освещает в своей статье создатель языка C++ Бьерн Страуструп[1]. В частности, он пишет о том, что учебные задачи совсем не похожи на промышленные, а глубокое знание теории вовсе не гарантирует способность применить полученные знания на практике.

IT-индустрия как наука сформировалась совсем недавно, только в середине прошлого столетия. Пройдёт, вероятно, ещё не одно десятилетие, прежде чем человечество отыщет универсальный способ обучения программированию и компьютерной науке. Другая проблема, отличающая IT-образование от других областей и фундаментальных наук – наличие огромного количества стремительно развивающихся технологий. В итоге: *“...мы часто можем встретить студентов с высокими оценками по алгоритмам, структурам данных и технологии программирования, которые, тем не менее, разбираются во всех тонкостях на уроке по операционным системам, абсолютно не учитывая структуры данных, алгоритмы и структуру программного обеспечения. В результате получается плохое представление неразборчивого беспорядка.”[1]*

Решение проблемы в мире: практика в западных ВУЗах

В западных вузах уже поняли проблему несоответствия теоретической и практической подготовки специалистов. Например, была принята практика участия студентов в создании достаточно больших и сложных проектов на практике в самых разных областях.

Возьмем курс архитектуры операционных систем. Это одна из классических задач системного программирования. Без сомнения, полностью разобраться в работе алгоритмов на теоретической основе невозможно без практики. Да и сам алгоритм, на сколько бы эффективным он ни был, без своей реализации не представляет из себя никакой ценности.

Для практических занятий по курсу построения операционных систем и системного ПО в ведущих IT-вузах, как правило, используют существующие ОС, так или иначе адаптированные для обучения или изначально разработанные для применения в академических целях . Самая известная ОС для обучения – это, конечно, MINIX Эндрю Таненбаума[2]. Но существуют и другие, например, операционная система Nachos[3], используемая в университете Беркли и предназначенная именно для обучения студентов. В том же университете Беркли для исследований в области ОС и сетевых технологий используют различные версии BSD. В MIT используется специальная упрощенная версия UNIX Xv6[4].

Решение проблемы в России

Статья Страуструпа описывает ситуацию в западном обществе. Очевидно, что там эту проблему осознали и уже есть пути улучшения сложившейся ситуации. Так ли всё обстоит в России? В рамках программы подготовки специалистов в сфере IT студентам читается огромное количество разных курсов, таких как “Компьютерная графика и компьютерное зрение”, “Техника компиляции”, “Базы данных и СУБД”, “Операционные системы”, “Компьютерное моделирование”, “Системы искусственного интеллекта” и т. д. Хорошо, если практика есть хотя бы по половине из этих дисциплин. Если же и есть, то это маленькие учебные задания и лабораторные работы. Таким образом, описанная Страуструпом проблема довольно остро стоит и в наших вузах, и так же как и в западных, есть понимание проблемы и есть попытки найти решение. Например, в СПбГУ и

МФТИ[5] некоторые IT-компании ввели практику проведения студенческих проектов. Эти компании осознают, что подготовка специалиста в рамках компании или поиск и найм требуемого специалиста с рынка, требуют куда больших финансовых затрат, чем его обучение еще в рамках университетской программы[6].

Студенческие проекты

На математико-механическом факультете СПбГУ основоположником студенческих проектов является компания ЗАО Ланит-Терком. В рамках студенческих проектов разработчики и менеджеры этой компании предлагают проекты на различную тематику и с использованием различных технологий, которые применяются в коммерческих разработках. Студент может выбрать наиболее интересный для себя проект и попробовать в нем свои силы, применить полученные теоретические знания на практике, получить опыт программирования и работы в команде. В результате студент компенсирует недостаток практики и учится решать не академические “искусственно-синтетические”, а самые настоящие, реальные задачи. Таким образом, студент погружается в атмосферу, приближенную к промышленной разработке.

Обычно проекты, предложенные отдельной компанией, направлены на изучение конкретной технологии, интересной для неё в данный момент. И длятся они менее года. После участия в таком проекте хорошо зарекомендовавшие себя студенты могут поступить на работу в эту компанию. Предлагать студентам большой и сложный проект с большим количеством технологий на длительный срок для компаний не выгодно, т.к. порог вхождения в такие проекты слишком высок, и студенты могут его не закончить.

При таком подходе студенты, участвующие в этих проектах, получают необходимый для работы в коммерческой компании конкретный навык, но не учитывается связь с остальными дисциплинами для обеспечения качественного университетского образования. Но основная цель высшего образования не получение навыков, достаточных только для конкретной компании и текущей конъюнктуры рынка, а воспитание квалифицированных специалистов, всегда востребованных на рынке.

Одним из выходов могло бы стать применение практики проектов с открытым кодом для студенческих проектов. Это позволило бы:

- продемонстрировать работу студентов не только в одной курирующей компании, но и в других компаниях, заинтересованных в специалистах в данной области;
- участвовать в разработке не только студентам, но и другим

разработчикам, интересующимися данной тематикой (проектом), например, студентам из других вузов, а также зрелым специалистам;

- последнее, в свою очередь, сделало бы возможным организацию сложных составных проектов, в которые были бы вовлечены много разработчиков и компаний. И такие проекты могли бы длится продолжительное время.

Примером подобного подхода может являться программа “Google Summer of Code” [7]. В этой программе компания “Google” платит деньги за участие студентов в open-source проектах. Целью программы является вовлечение студентов в open-source сообщество. Причем учатстниками программы являются реальные проекты с открытым кодом, и именно разработчики этих проектов ставят задачи и контролируют их выполнение. Таким образом гарантируется, что студент получит реальную технологическую практику в рамках выбранной им самим темы.

Основная часть

Студенческий проект Embox

В рамках одного из студенческих проектов на математико-механическом факультете разрабатывается операционная система реального времени Embox. Основным её преимуществом является модульность и легковесность, что делает ее удобной для использования в академических целях (обучение, исследования).

Embox как студенческий проект успешно развивается более двух лет. В нем уже приняли участие более 10 студентов, по материалам проекта защищено 2 диплома, и порядка 10 курсовых работ. Успеху проекта способствовало и то, что изначально он был задуман как открытый распределенный проект, вследствие чего использовались соответствующие практики:

1. Весь код располагается в открытом репозитории, доступном в Интернете. В проекте используются некоторые технологии промышленного программирования (проводится код-ревью, использование issue трекера);
2. Документация по проекту также доступна через Интернет. Причем документируются не только конкретные модули и алгоритмы, но и процессы установки и развертывания средств разработки и другие материалы, позволяющие легко войти в проект. На wiki публикуются теоретические материалы по теме операционных систем, а также описания практических примеров;
3. Кроме студентов в проект вовлечены и сторонние разработчики, заинтересованные данной тематикой, что позволяет улучшить качество кода;
4. В проекте приняты средства коммуникации, используемые в распределенных проектах. Существует несколько списков рассылки, обеспечивающих коммуникации участников проекта. Один список освещает организационные вопросы, второй посвящён техническим вопросам и проблемам. Если у студента или участника проекта возникает какая-то проблема или вопрос, то он может его отправить в группу и оперативно получить ответ от других участников проекта.

Обучение на базе Embox

Небольшое количество строчек кода, которе содердит ОС Embox, способствует легкому пониманию архитектуры. Модульность позволяет разобраться в отдельных частях и алгоритмах, а также отследить взаимодействие между ними. Embox вполне может стать российским аналогом обучающей операционной системы с учётом специфики российского образования.

Студенты, принимающие участие в проекте имеют возможность понять теоретические основы на практике, “прощупать” их изнутри, провести ряд эксперементов, “поиграть” с системой.. Однако, на данный момент таких ребят не так много, по сути дела это участники проекта. Построение на базе проекта набора лабораторных работ, помогло бы и другим студентам легко погрузиться в данную тему, и ощутить архитектуру ОС изнутри.

Ведь как уже упоминалось, все алгоритмы и механизмы, применяемы в ОС РВ Embox подробно описаны на wiki-страницах, доступных on-line в Интернете. С ними может ознакомится любой желающий.

В проекте появились рад алгоритмов-примеров, которые хорошо прокомментированы и имеют подробное описание своей работы. К ним имеется набор тестов демонстрирующий их работу в различных ситуациях. На их основе можно построить лабораторные работы по курсу “Опреационные системы”. В рамках данной курсовой работы в системе появился алгоритм выделения памяти Malloc. Подробное его описание можно найти в приложении.

Выполняя лабораторные работы, у студентов появляется возможность хорошо изучить и понять принципы построения операционных и сетевых систем. К тому же студенты заинтерисовавшиеся данной тематикой могут продолжить свои научные исследования, а также улучшить свои навыки в промышленном программировании. Ведь данный проект применяется ЗАО Ланит-Терком в собственных коммерческих разработках. и от студентов также требуют применения различных практик промышленных и распределенных проектов: Code-review, комментирование и докумментирование разработанных модулей, unit тестирование, общение в группе на английском языке. Все это позволяет добится существенно лучших результатов в подготовке специалистов, ведь за свою работу они имеют куда большую практику программирования. Вот пример статистики с открытого сайта “http://ohoh.net” для нескольких студентов проектах[10] (табл.1-3):

Таблица 1. Участник 1.

Language	Experience	Median Monthly Commits	Median Monthly Lines Changed	Total Commits	Total Lines Changed	Comment Ratio
C	11m	5	389	85	13,344	16.1%
Make	8m	1	4	16	42	-
XML	1m	1	8	1	8	-
All Languages	11m	8	406	112	13,394	16.0%

Таблица 2. Участник 2:

Language	Experience	Median Monthly Commits	Median Monthly Lines Changed	Total Commits	Total Lines Changed	Comment Ratio
C	10m	7	634	115	14,667	17.5%
Make	10m	4	20	47	411	30.5%
All Languages	10m	7	645	146	15,078	17.9%

Таблица 3. Участник 3:

Language	Experience	Median Monthly Commits	Median Monthly Lines Changed	Total Commits	Total Lines Changed	Comment Ratio
C	1y 10m	9	664	354	30,047	15.0%
Make	1y 8m	6	28	200	7,972	19.9%
Assembly	1y 0m	2	58	37	1,047	21.0%
C++	4m	1	6	5	23	-
XML	3m	2	14	16	324	0.0%
AWK	1m	2	21	2	21	-
Lisp	1m	1	5,819	1	5,819	-
HTML	1m	1	3	1	3	-
All Languages	1y 10m	15	1,188	603	45,256	16.2%

Обучение школьников

Чтобы вырастить первоклассного специалиста в области IT, необходимо заинтересовать его ещё в школе. Если ребёнок не заканчил специализированную школу или интернат, то при выборе высшего учебного заведения и будущей профессии он плохо представляет, что именно его ожидает в университете. С другой стороны, преподаватели, объясняя некоторый материал, предполагает, что студент обладает теми или иными знаниями в определённой области. Однако, студент первого курса вовсе не обязан знать материал, который выходит за рамки школьной программы, но необходим для понимания основ изучаемого предмета. (Отмечу, что вступительные испытания также не подразумевают знание подобных вещей.)

У компании Google существует аналог программы “Google Summer of code” для школьников программа “Google Code-in“[12].Эта программна расчинана на участников предуниверситетских образовательных учреждений, то есть для школьников от 13 до 18 лет.

В отличие от “Google Summer of Code”, задачи направлены не только на программирование: бывают задачи по локализации ПО, написанию документации, разработке интерфейсов пользователя и продвижению проектов (например, посредством статей в Википедии, видеоуроков) и др.

На базе Embox также можно построить обучение информатики и алгоритмизации для старших школьников. На примере данного проекта можно наглядно объяснить работу алгоритмов и продемонстрировать пример работы реальной программы. Школьники, как и студенты, могли бы попробовать свои силы в разработке несложного программного кода. Таким образом они не только познакомятся с реальным проектом и в будущем будут иметь представление об этом, не будут бояться принять участие в проектах с открытым кодом, но и получат опыт программирования и лучше усвоят преподаваемый материал.

ОСРВ Embox перенесена на плаформу LEGO Mindstorm. Это позволяет обучать школьников в более интересной для них форме. Обучение с помощью роботов более эффективно развивает алгоритмическое мышление и более наглядно демонстрирует некоторые фундоментальные понятия информатики и программированя. Кроме этого существуют различные конкурсы и соревнования, участие в которых в значительной степени стимулирует желание заниматься программированием и информатикой в целом.

В рамках данной курсовой работы было создано руководство пользователя по сборке, настройки и использованию ОСРВ Embox для роботов Lego, ориентированное на

школьников средних и старших классов.

Заключение

Выводы

Образование в любой научной сфере должно производиться с учётом её специфики. В IT-индустрии необходимо найти тонкую грань между теорией и практикой, которые так сильно отличаются друг от друга. Возможно, этой гранью являются студенческие проекты. Безусловно, студенты должны наравне с теорией достаточное количество времени и сил уделять практическим занятиям. И очень важно, чтобы у них был под рукой доступный и удобный инструмент для изучения и исследования. Операционная система реального времени Embox вполне может стать таким инструментом.

Приложение 1

Алгоритм выделения памяти Malloc

Для нормальной работы программы необходимо выделять и освобождать память непосредственно в процессе её выполнения.

Устройство памяти:

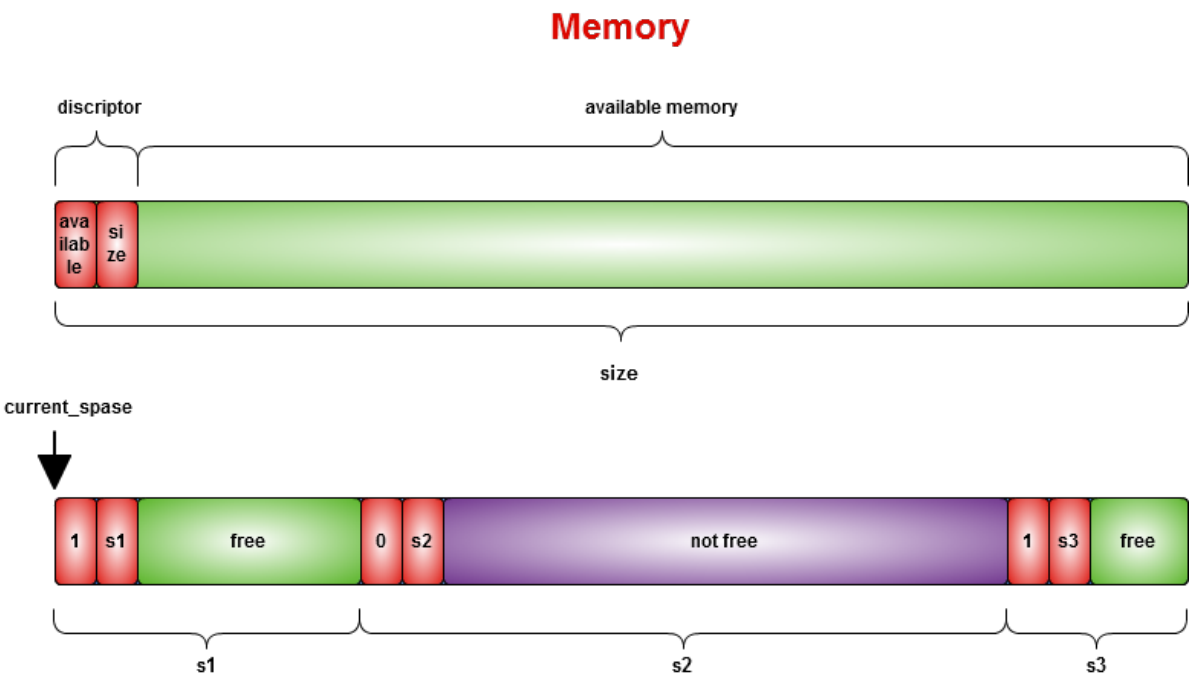

Рисунок 1.

Отведём непрерывный буфер и будем использовать его как для выделения памяти, так и для хранения служебной информации.

Весь буфер можно разбить на блоки произвольного размера со следующей структурой: в начале каждого блока поместить дескриптор - две ячейки (8 байт). В первой хранится информация о том, свободен блок или нет. Во второй хранится размер блока. Изначально имеется один свободный блок, его размер равен размеру всей памяти.

Алгоритм:

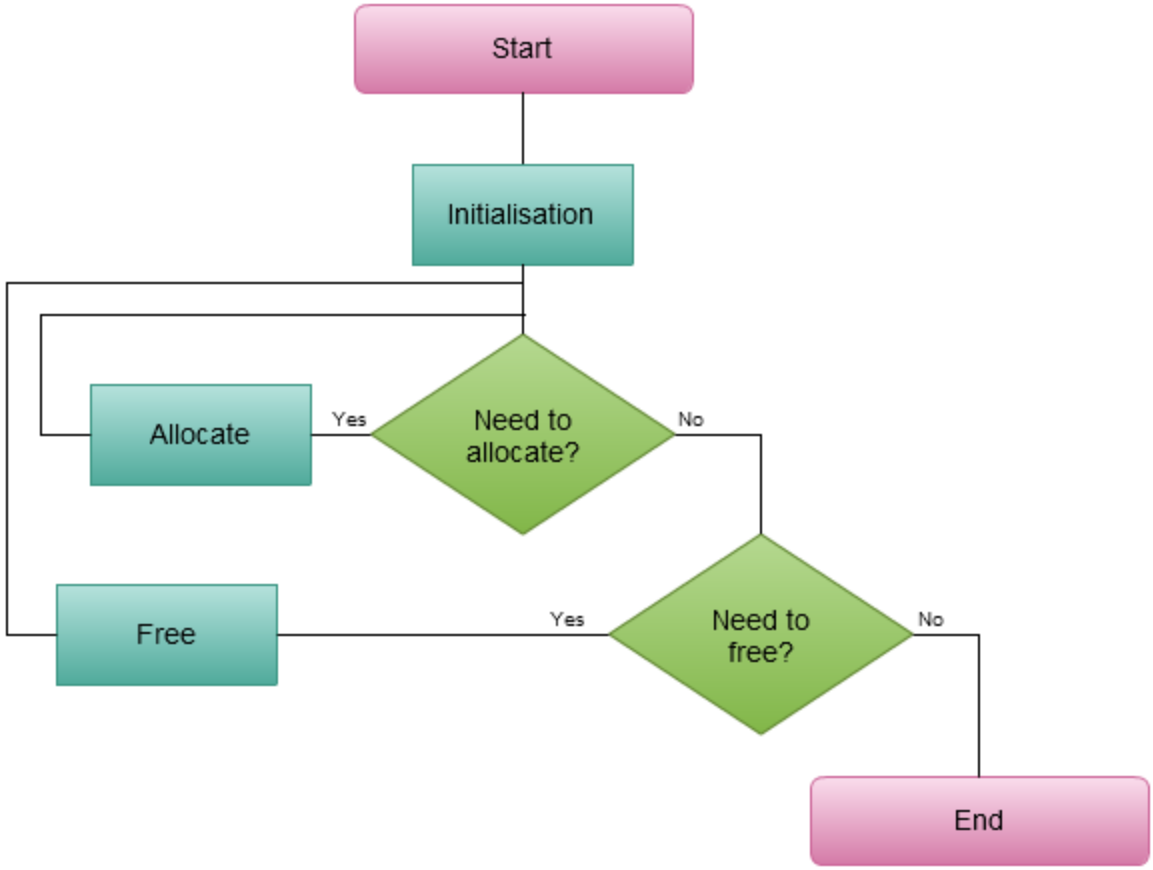

Рисунок 2.

Инициализация

Прежде всего проходит инициализация. Вся имеющаяся память представляет собой один свободный блок. Указатель current_space установлен на начало этого блока, т.е. на начало памяти.

Выделение памяти

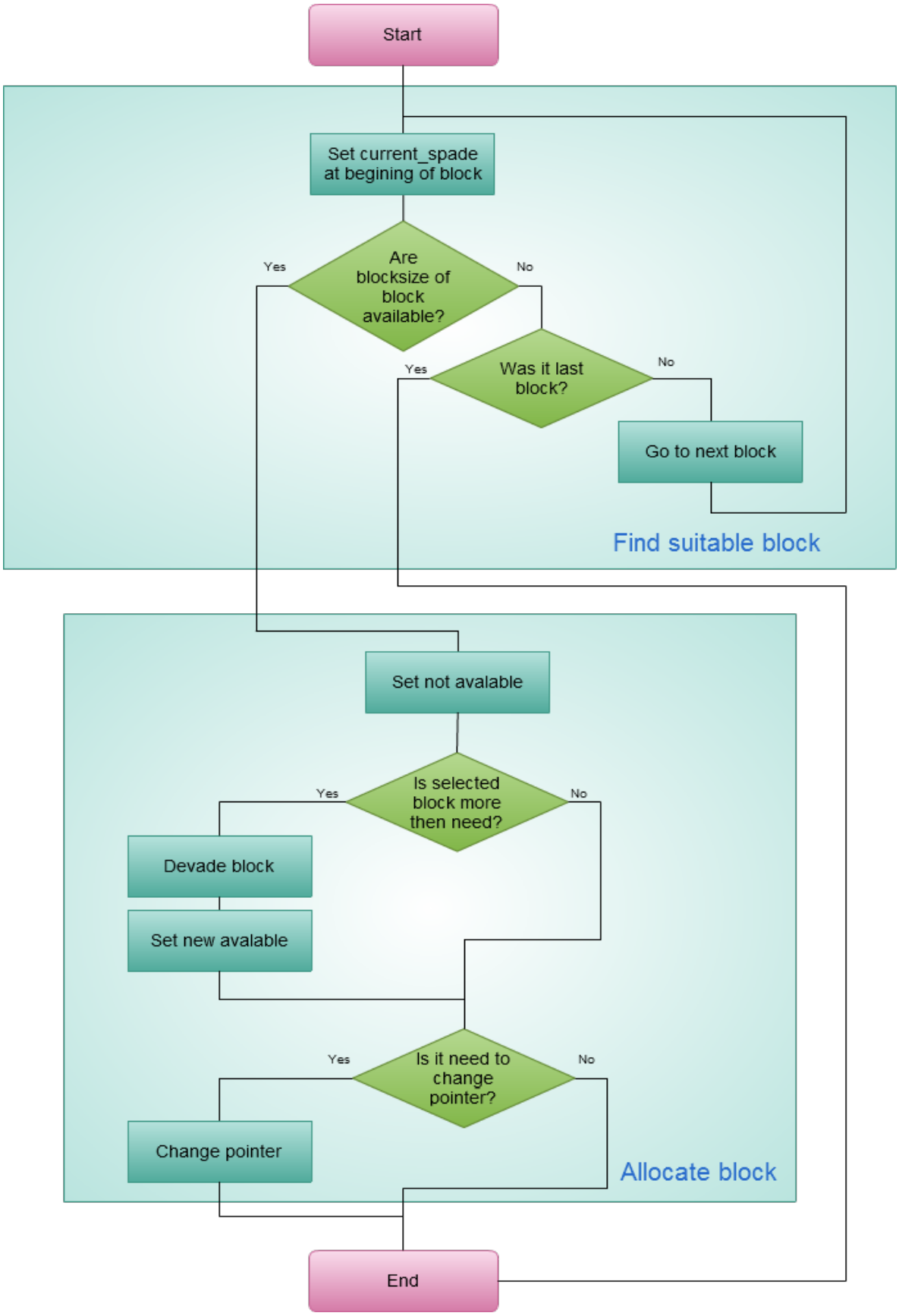

Рисунок 3.

Выделение памяти происходит в два этапа:

I. Сначала необходимо найти подходящий по размеру свободный блок. Поиск начинается с адреса, на который указывает current_space. До этого указателя вся память гарантированно занята.

Рассмотрим позицию указателя current_space. Функция find_suit_block заводит новую переменную структуры, и прежде всего нужно удостовериться, что адрес указывает на доступное пространство. Далее проверяет:

а) является ли блок свободным;

б) вмещает ли он в себя блок, который необходимо выделить.

Если оба условия верны, то блок является подходящим и функция возвращает адрес этого блока. Если одно из условий ложно, то рассматривается следующий блок, адрес которого формируется по средствам прибавлением к началу блока его размера. Новый блок рассматривается аналогичным образом. На каждом шаге производится проверка: если переменная дошла до конца доступной памяти, то функция возвращает NULL.

II. Непосредственное выделение памяти функцией memory_allocate. Прежде всего проверяем, удалось ли найти подходящий блок. Если нет, то выдаётся соответствующее сообщение, иначе от подходящего кусочка нужно разбить на два блока. Первый помечается как занятый, его размер - размер запрашиваемой памяти + размер дескриптора. Второй блок - это то, что осталось. Необходимо определить размер этой памяти и пометить её как свободную.

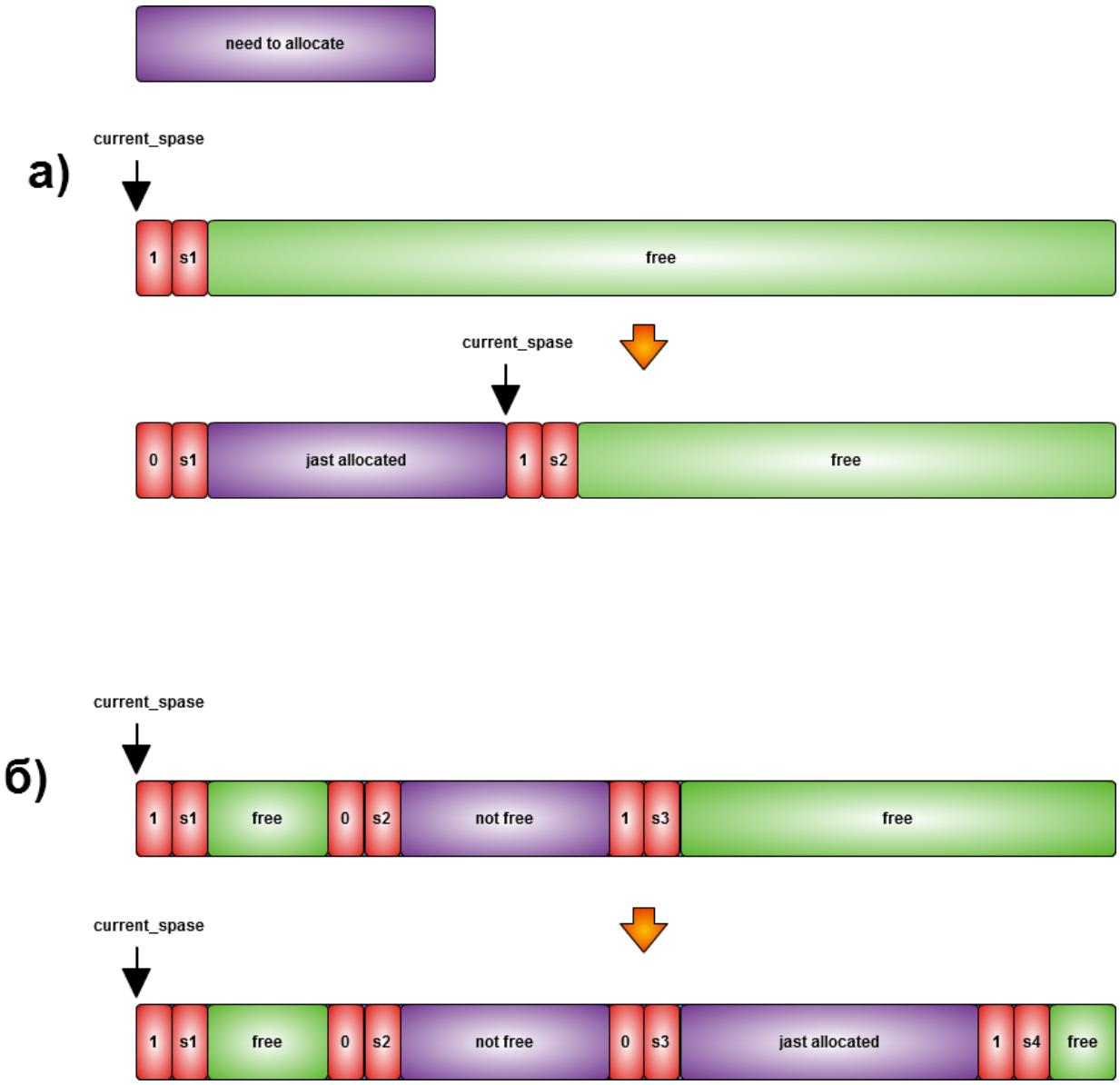

Рисунок 4.

Перестановка указателя

Следующим шагом следует переместить указатель current_space. Это делается с целью оптимизации последующего поиска свободного блока.

Проверяем: совпал ли указатель на текущее положение в памяти с началом выделенного блока. Возможно два варианта:

а) Да. Значит до конца только что выделенного блока вся память занята, и указатель можно передвинуть. Тогда новый поиск начнётся с нового места.

Б) Нет. Значит до этого блока в памяти есть свободный блок, который не подошёл по размеру (он мог образоваться в результате удаления). Следовательно, в данный момент current_space указывает на него. Следующий поиск следует начинать с того же блока, поэтому указатель current_space не меняет своего

положения.

Замечание: current_space не всегда указывает на свободное место. Рассмотрим ситуацию, когда есть «дырка» и нам нужно выделить блок, совпадающий с размером свободной области, и есть место для дескриптора. В этом случае после выделения памяти current_space начнёт указывать на следующий блок, который занят. Именно поэтому поиск следующего свободного блока начинается с проверки «Занят ли этот блок?» Эта ситуация не самая оптимальная, т. к. в ходе последующего поиска сразу же встречается занятый блок. Но данное продвижение всё равно сужает область поиска для следующего блока. Если будет освобождаться область, предшествующая указателю current_space, то он переместится и снова будет указывать на свободный блок.

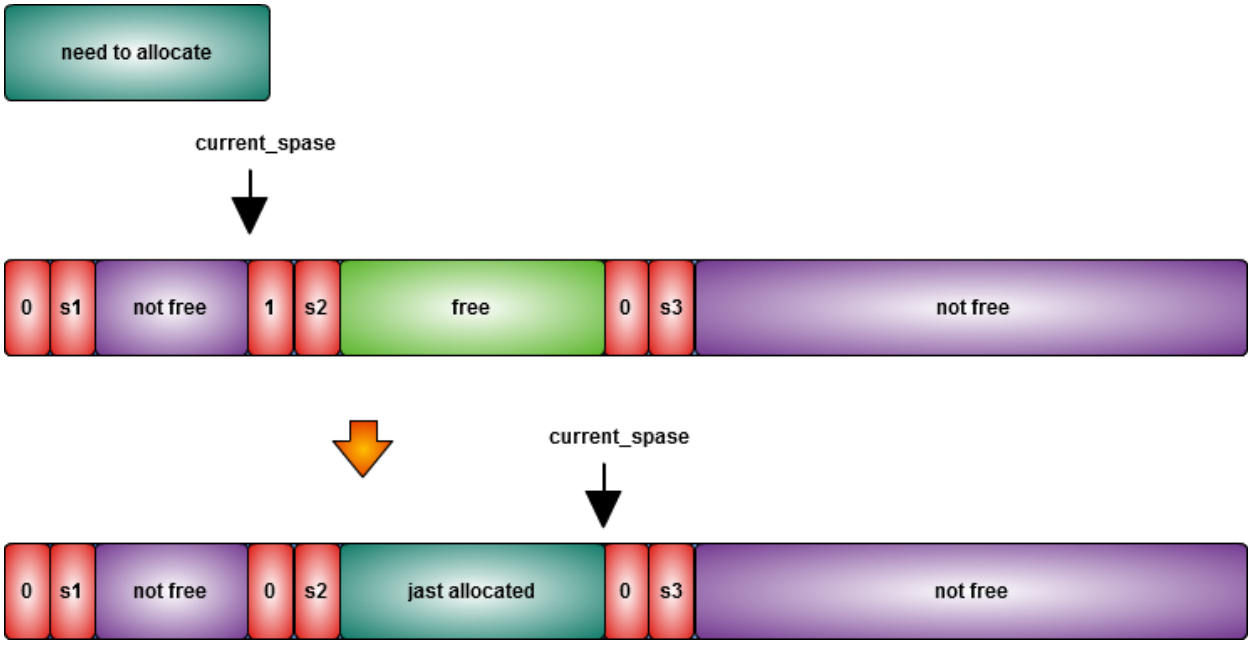

Рисунок 5.

Следует помнить, что оставшаяся память должна быть больше размера дескриптора блока. Если оставшийся блок меньше, то его следует просто добавить к выделяемой памяти, т.к. если попытаться организовать эту память в отдельный блок и следующий за ним блок окажется не пустым, то произойдёт затирание этого блока.

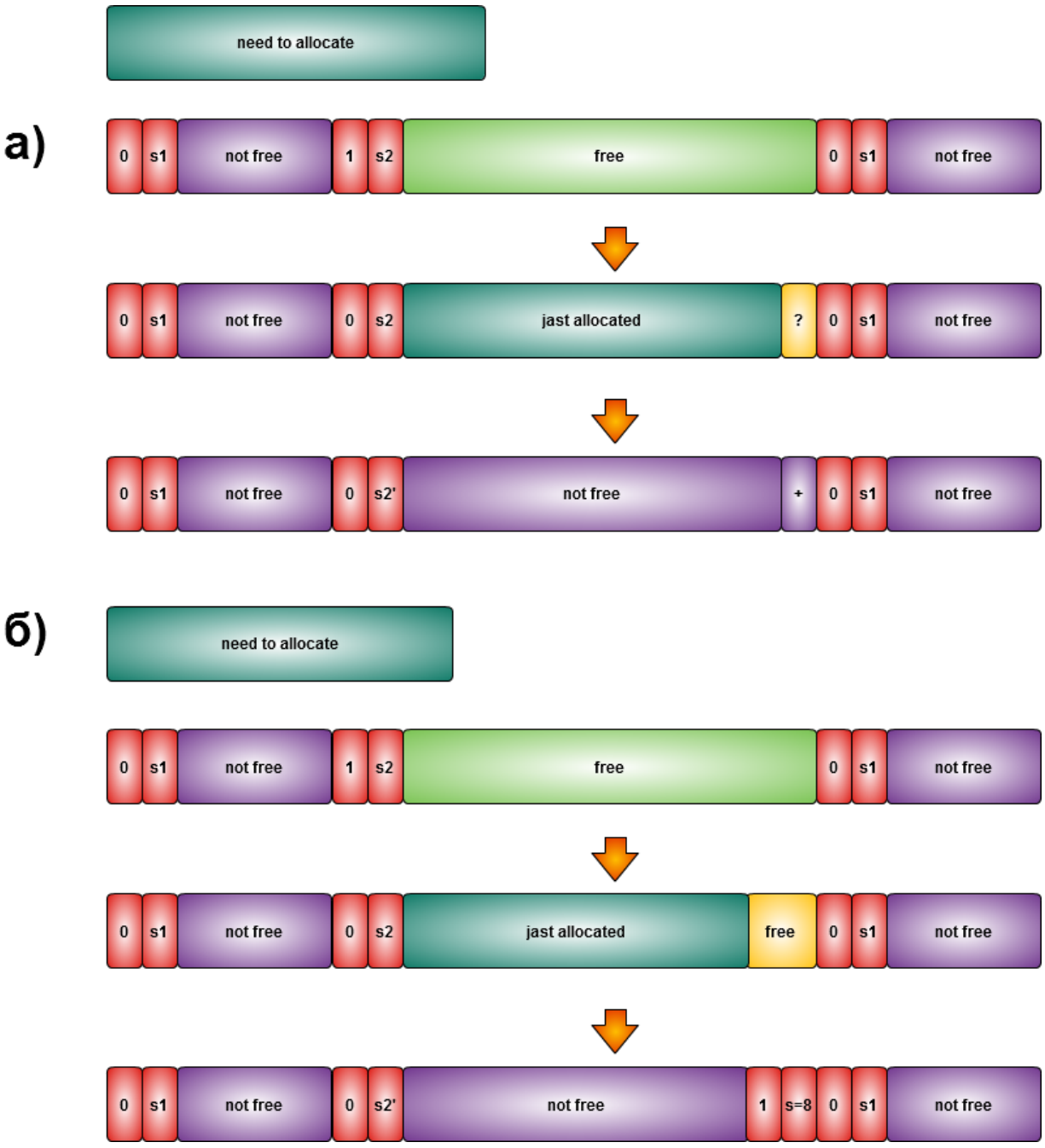

Рисунок 6.

Результатом, возвращаемым после выделения памяти, является адрес внутри выделяемого блока, куда пользователь или программа, запрашивающая память, сможет записать данные.

Освобождение памяти.

На вход функции memory_free подаётся адрес, по которому располагаются данные, которые следует удалить - внутри блока (не адрес блока). Соответственно перед этим адресом находиться дескриптор блока, т.е. если отнять размер дескриптора, получится указатель на информацию о блоке. Данный блок нужно освободить, т.е. в его дескрипторе указать "свободен".

После нескольких удалений может возникнуть проблема фрагментации: два или более свободных блока могут оказаться рядом и их следует склеить, чтобы можно было вместить блок большего размера, чем каждый из них, (но не больше, чем их сумма без размера дескриптора). Поэтому после каждого удаления следует провести дефрагментацию - процесс устранения фрагментации.

Дефрагментация

Освобождение может быть проведено в четырех случаях:

а) Внутри занятой памяти. Слева и справа занятые блоки. Дефрагментация не требуется;

б) До свободного блока. Справа свободный, слева занят. Нужно склеить два кусочка;

в) После свободного блока. Слева свободный, справа занят. Нужно склеить два кусочка;

г) Между свободными блоками. Слева и справа свободные блоки. Нужно склеить три кусочка.

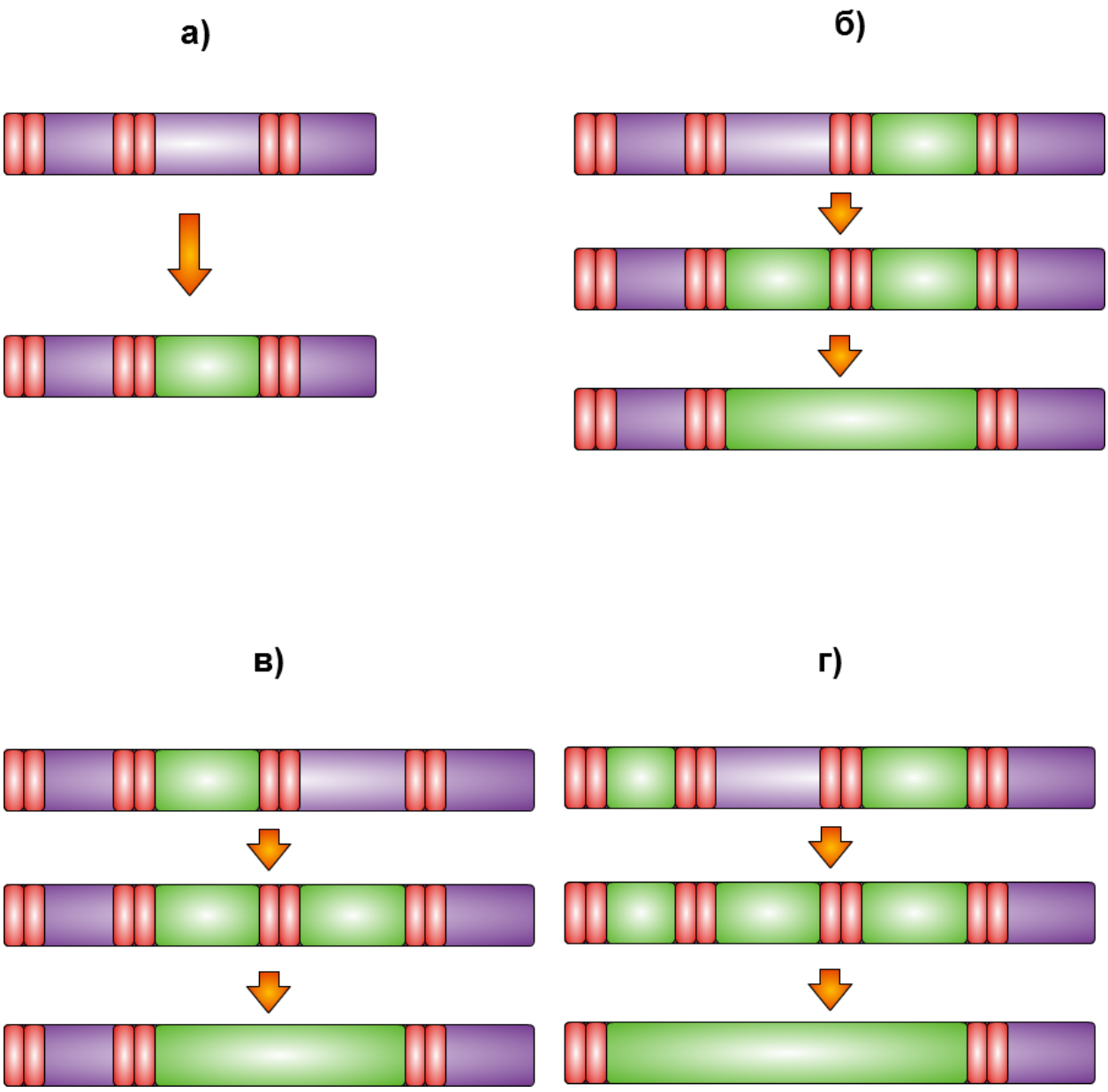

Рисунок 7.

Идея дефрагментации в данной реализации состоит в том, что обрабатывается вся свободная память в порядке встречи. Таким образом, второй и третий пункт будут обрабатываться одинаково. А четвёртый аналогичен второму и третьему: дефрагментация применяется дважды (три свободных блока -> два свободных блока -> один свободный блок).

Есть указатель current_space. С него можно начать, т.к. до него гарантированно нет свободных кусков памяти. Рассматриваем текущий блок и следующий за ним. Если текущий свободен и следующий тоже, то склеиваем их и снова проверяем с того же адреса (но блок уже другой - объединённый). Если следующий блок занят - проходим и ищем следующий свободный и повторяем проверку для него.

Также в данном алгоритме присутствует внешняя дефрагментация - накладные расходы памяти на дескрипторы блоков (на схемах помечены красным цветом).

Тест

Написала ручками тест, с разными ситуациями. Опишу кратко: Проходит 5 фаз:

1. Выделение. Память по порядочку заполняется. Размер выделяемой памяти - 0x100 с некоторым коэффициентом, который записан в массив koef_alloc[]. Ссылка на адрес, который является результатом выделения, записывается в массив succes_alloc[number_add_block].

2. Некоторые блоки освобождаются. Номера блоков, которые освобождаются по порядку, записаны в массив number_of_block[]

3. Снова выделение - заполняются "дырки". При этом некоторые блоки имеют длину меньше чем дескриптор. Последний блок слишком большой.

4. Опять освобождение в более глобальных масштабах. Проверка дефрагментации на всех видах.

5. Опять выделение.

Результат программы совпал с ожидаемым результатом.

Функции:

Основные:

- static void *memory_allocate(size_t req_size) { ... } - выделяет память

- static void memory_free(void *address) { ... } - освобождает память

Вспомогательные:

- static void defrag(struct block_desc *md){ ... } - проводит дефрагментацию

- struct block_desc *find_suit_block(size_t req_size) { ... } - находит подходящий блок

- static int correct_address(char *address){ ... } - проверяет корректность адреса

- static int get_available(struct block_desc *md){ ... } - возвращает информацию о доступности блока - 1, если блок свободен и 0, если занят

- static void set_available(struct block_desc *block){ ... } - помечает блок, как свободный

- static void set_not_available(struct block_desc *block){ ... } - помечает блок, как занятый

- static int correct_size(struct block_desc *md, size_t req_size){ ... } - проверяет, подходи ли рассматриваемый блок по размеру

Тест - пример:

- static void example(void){ ... } - сам тест

- static void memory_init(void){ ... } - выводит состояние памяти - все блоки, их размер и состояние.

Библиография

1. http://habrahabr.ru/blogs/htranslations/105544/
2. http://www.minix3.org/
3. http://www.cs.washington.edu/homes/tom/nachos/
4. http://pdos.csail.mit.edu/6.828/2011/xv6.html
5. http://tormasov.com/blog/vuz-innovatsionnaya-kompaniya-chast-3-sotrudniki-kompaniy-i-rabota-so-studentami/
6. Andrey Terekhov, Karina Terekhova The economics of hiring and staff retention for an IT company in Russia, Proceedings of 4th International Conference SEAFOOD, Springer, 2010
7. http://code.google.com/intl/ru-RU/soc/
8. http://www.sysprog.info/2008/sysprog-2008.pdf
9. http://cacm.acm.org/magazines/2010/1/55760-what-should-we-teach-new-software-developers-why/fulltext
10. http://www.ohloh.net/p/embox/contributors/