УДК 004.415.25

ПРОГРАММА РЕАЛИЗАЦИИ АЛГОРИТМА ХАБЕРМАНА (ПО РАСПРЕДЕЛЕНИЮ РЕСУРСОВ)

HABERMAN ALGORITHM IMPLEMENTATION PROGRAM (RESOURCE ALLOCATION)

В. Ю. Глазков, М.А. Букина
Ю. Е. Глазков, к.т.н., доцент,
Тамбовский государственный технический университет, г. Тамбов.

Аннотация. Статья посвящена одному из фундаментальных аспектов современных вычислительных комплексов – мультипрограммированию, обеспечивающему возможность параллельного выполнения множества процессов. Основное внимание уделено разработке специализированного программного комплекса, моделирующего алгоритм Хабермана, предназначенный для оптимизации управления распределением ресурсов. Объектом исследования служит операционная система, в рамках которой планируется интеграция данной модели. Центральным предметом анализа является программная реализация алгоритма Хабермана, основная функция которого состоит в эффективном управлении доступными ресурсами. Разрабатываемое программное решение обеспечивает оптимизацию распределения ресурсов с учётом заданных приоритетов, что способствует повышению общей эффективности функционирования системы.

Ключевые слова: операционная система, ресурсы, алгоритм Хабермана, регулируемое распределение, программа, интерфейс, граф, тупик, процесс, цикл, график.

Abstract. The article is devoted to one of the fundamental aspects of modern computing systems – multiprocessing, which provides the ability to execute multiple processes in parallel. The main focus is on the development of a specialized software package that simulates Haberman's algorithm designed for optimizing resource allocation management. The object of study is an operating system within which this model is planned to be integrated. The central subject of analysis is the software implementation of Haberman's algorithm, whose primary function is efficient resource management. The developed software solution ensures optimization of resource distribution based on given priorities, thereby contributing to improved overall system performance.

Keywords: operating system, resources, Haberman's algorithm, regulated distribution, program, interface, graph, deadlock, process, cycle, schedule.

Одной из ключевых характеристик современных электронных вычислительных машин (ЭВМ) является возможность мультипрограммирования, которое подразумевает «обработку наборов

задач и обслуживание потоков задач» [4, с. 19]. Для выполнения каждой задачи требуется использование определённых ресурсов системы, таких как центральный процессор (CPU), оперативная память (RAM) и периферийные устройства ввода-вывода, включая накопители данных, сетевые интерфейсы и прочие аппаратные модули. Важно отметить, что потребности задач в ресурсах могут варьироваться таким образом, что возникает ситуация, когда ни одна из них не способна завершить своё выполнение. Это состояние называется тупиком. Исследование и разработка практически применимых алгоритмов для разрешения тупиковых ситуаций имеют «большое прикладное значение для обеспечения информационной безопасности вычислительного процесса» [1, с. 184]. Чтобы предотвратить возникновение подобных ситуаций, необходимо грамотно управлять доступными ресурсами, распределяя их между задачами таким образом, чтобы всегда была возможность продолжить выполнение хотя бы одной из них.

Оптимизация использования указанных ресурсов приобретает особую значимость ввиду их ограниченной доступности. В условиях многозадачности, характерной для современных компьютерных систем, множество программных потоков может одновременно претендовать на одни и те же ресурсы, такие как процессорное время, объём доступной оперативной памяти или доступ к внешним устройствам. Операционная система выполняет функцию координатора этих запросов, обеспечивая сбалансированное и рациональное распределение ресурсов с учётом приоритета отдельных процессов.

Важнейшей задачей операционной системы является минимизация потенциальных конфликтов при совместном использовании общих ресурсов. Для этого применяются механизмы регулирования очередности выполнения задач, позволяющие наиболее важным процессам получать первоочередной доступ к необходимым ресурсам. Кроме того, операционная система предотвращает возникновение ситуаций, связанных с попытками нескольких процессов установить исключительный контроль над одним и тем же ресурсом, что потенциально может привести к состоянию взаимоблокировки (*deadlock*).

В 1971 году Коффман, Элфик и Шошани определили четыре условия, необходимые для возникновения тупиковых ситуаций:

- условие взаимного исключения. Каждый ресурс предоставляется строго одному процессу или свободен. Процессы запрашивают эксклюзивный доступ к ресурсам;

- условие удержания и ожидания. Процессы сохраняют контроль над уже выделенными им ресурсами, одновременно ожидая получения дополнительных ресурсов, занятых другими процессами;

- условие невозможности перераспределения. Ресурсы, однажды предоставленные процессу, не могут быть изъяты принудительно. Они освобождаются только по инициативе самого процесса;

- условие циклического ожидания. Существует цепочка процессов, где каждый процесс удерживает ресурсы, требуемые другим процессам этой цепи.

Все эти четыре условия должны выполняться для того, чтобы возникла ситуация тупика.

Обнаружение тупика представляет собой процесс установления факта того, что произошла тупиковая ситуация, а также идентификацию тех процессов и ресурсов, которые оказались вовлечены в данную ситуацию. В большинстве случаев алгоритмы для обнаружения тупиков используются в системах, где уже выполнены первые три необходимых условия для возникновения такой ситуации. После этого данные алгоритмы анализируют систему, чтобы определить, не возник ли режим кругового ожидания.

Использование алгоритмов для обнаружения тупиков связано с определенными дополнительными затратами процессорного времени. Поэтому здесь вновь возникает необходимость принятия компромиссных решений, характерных для разработки операционных систем. Нужно оценить, оправдывают ли затраты на реализацию этих алгоритмов потенциальную выгоду от выявления и последующего устранения тупиковых ситуаций. Например, метод предотвращения тупиковых ситуаций, основанный на предварительном распределении ресурсов между процессами, характеризуется наличием ряда ограничений. В частности, одним из таких недостатков является потенциальная возможность увеличения времени ожидания до неприемлемо высоких значений. Альтернативный метод предотвращения тупиковых состояний, предусматривающий временную остановку выполнения процесса и освобождение занимаемых им ресурсов, демонстрирует низкую эффективность, особенно в условиях работы с множеством различных типов ресурсов, запрашиваемых процессами в реальном времени [2, 3].

Методы обхода тупиковых ситуаций требуют, чтобы система располагала дополнительной априорной информацией о процессе и его потребностях в ресурсах начиная с момента ввода каждого процесса в систему.

Обход тупика можно рассматривать как предотвращение перехода системы в опасное состояние. Если ни одно из четырёх упоминаемых условий не исключается, то переход в опасное состояние можно предотвратить, при условии, что система обладает сведениями о последовательности запросов, связанных с каждым параллельным процессом. Было доказано, что если текущее состояние системы находится вне зоны опасности, то существует хотя бы одна последовательность

состояний, позволяющая избежать попадания в опасную зону. Таким образом, достаточно проанализировать, приведёт ли немедленное предоставление запрошенного ресурса к попаданию в опасное состояние. В случае положительного ответа запрос следует отклонить; в противном случае его выполнение возможно.

Определение того, является ли текущее состояние системы опасным или безопасным, требует анализа будущих запросов со стороны процессов. Зачастую последовательность таких запросов заранее неизвестна. Однако если известны общие запросы на ресурсы каждого типа, то распределение ресурсов можно контролировать таким образом, чтобы для каждого запроса, предполагая его удовлетворение, проверять наличие среди совокупных запросов от всех процессов такой последовательности, которая могла бы привести к опасной ситуации. Этот метод представляет собой пример контролируемого распределения ресурсов.

Одним из наиболее известных алгоритмов, позволяющим преодолеть данную проблему, является алгоритм Хабермана, называемый алгоритмом регулируемого распределения.

Алгоритм регулируемого распределения Хабермана основан на «представлении распределения ресурсов, запросов и процессов в виде ориентированного графа (орграфа), называемого графом распределения ресурсов (ГРР)» [4, с. 69].

По правилу Хабермана, если граф распределения ресурсов имеет циклы, то такое распределение ресурсов по процессам может привести к образованию тупиковой ситуации [5, 6].

Это правило позволяет построить следующий алгоритм распределения ресурсов:

- при поступлении нового запроса необходимо проверить наличие свободных ресурсов;

- если свободные ресурсы отсутствуют, то формируется отказ, а процесс блокируется. В противном случае выполняется следующее действие;

- если свободный ресурс имеется, производится его предварительное распределение для данного процесса, после чего корректируются данные о доступности ресурсов (например, обновляется графа ресурсов и требований – ГРР);

- проверяется состояние ГРР на предмет наличия циклов;

- если в ГРР обнаружены циклы, то предварительное распределение отменяется, восстанавливается предыдущее состояние ГРР, формируется отказ, и процесс блокируется.

- если циклы не выявлены, то ресурс закрепляется за данным процессом окончательно.

Алгоритм Хабермана позволяет обойти тупики, однако он сложен в реализации из-за необходимости работы с графом распределения ресурсов.

Для его реализации была разработана программа на языке *Python* с использованием библиотек: *tkinter* для создания графического интерфейса, *networkx* для работы с графами, *matplotlib* для визуализации данных двумерной и трёхмерной графики и модуля *datetime* для работы с временем. Основные компоненты программы включают в себя поля для ввода процесса, ресурса и времени использования ресурса процессом, кнопки для добавления процесса и ресурса, для выделения ресурса процессу, освобождения ресурса и построения графа распределения (рис. 1).

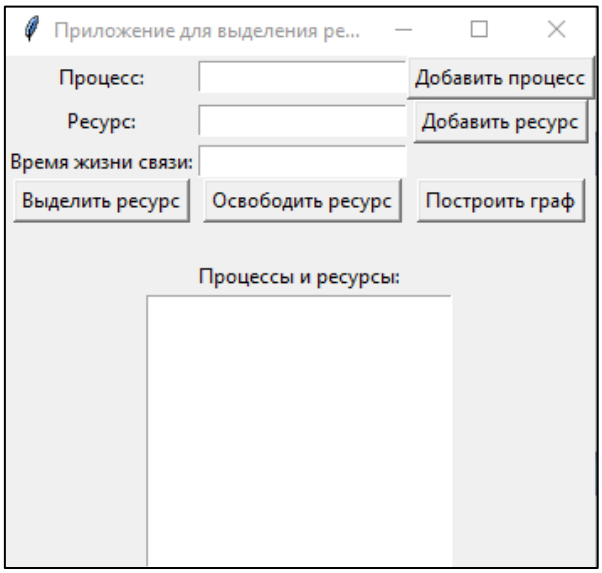

Рис. 1. Демонстрация интерфейса программы

Для добавления нового процесса и ресурса необходимо ввести их имена в поля «Процесс» и «Ресурс» и нажать кнопку «Добавить процесс» (рис. 2). В случае, если процесс или ресурс с таким именем уже существует, то программа выдаст ошибку (рис. 3).

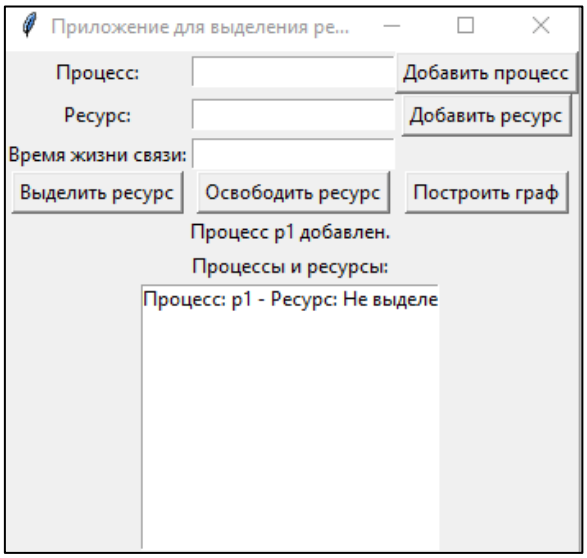

Рис. 2. Демонстрация успешного добавления процесса

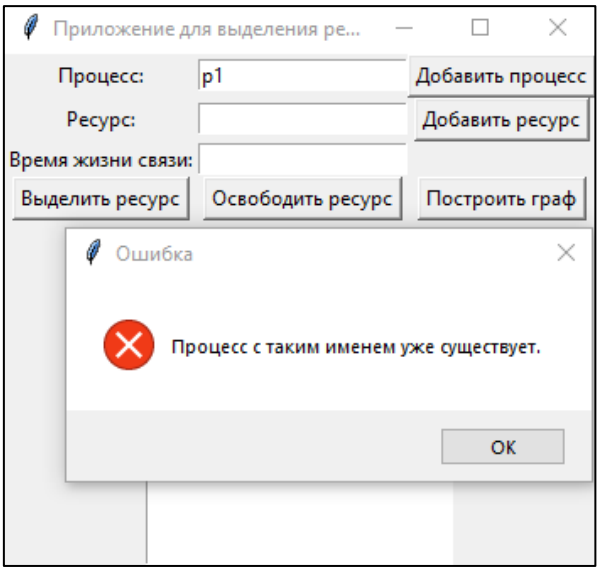

Рис. 3. Демонстрация ошибки добавления процесса

После добавления процесса (p_i) и ресурса (r_i) можно создать граф (рис. 4), для этого необходимо нажать кнопку «Построить граф».

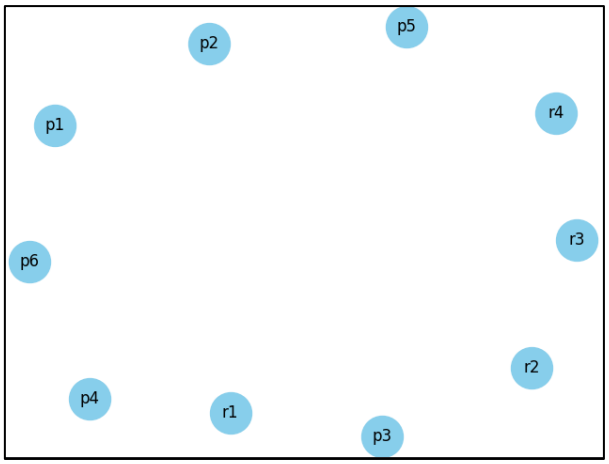

Рис. 4. Демонстрация добавленных процесса и ресурса
в поле построения графа

Далее задаём время использования ресурса процессом в поле «Время жизни связи», выделяем ресурс процессу, нажав кнопку «Выделить ресурс» (рис. 5) и строим граф. В начале будет построен предварительный граф распределения ресурсов, где связь между процессом и ресурсом помечена пунктирной линией (рис.6) и в случае отсутствия циклов пунктирная линия смениться сплошной, что означает закрепления ресурса за процессом (рис. 7).

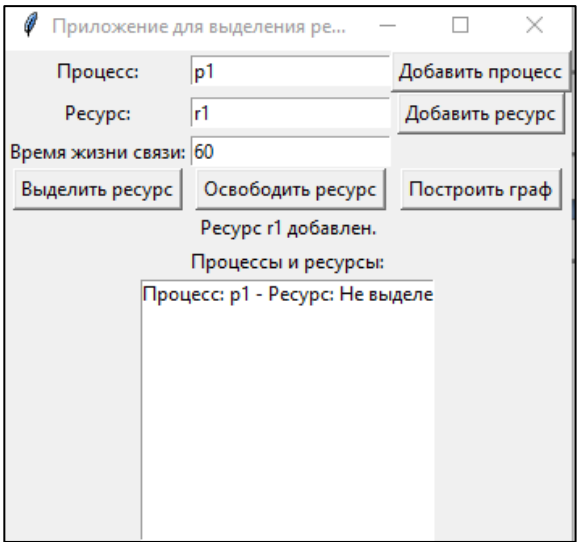

Рис. 5. Демонстрация выделения ресурса процессу и задания времени использования

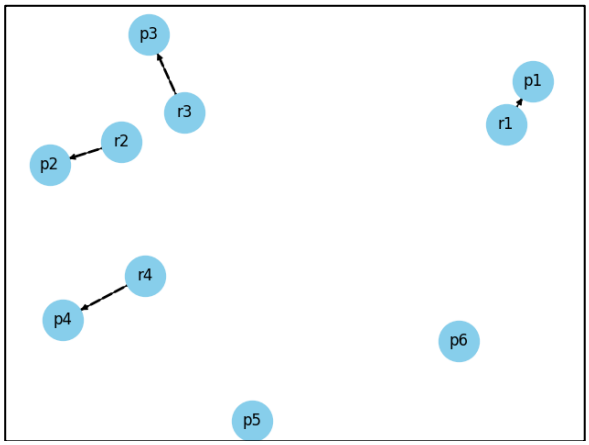

Рис. 6. Демонстрация предварительного графа распределения Ресурсов

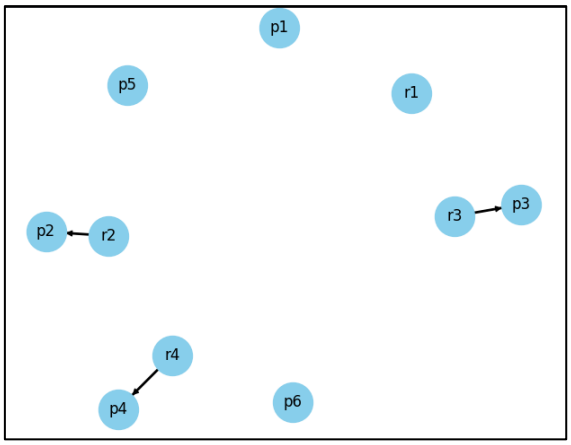

Рис. 7. Демонстрация закрепления ресурса за процессом на графе распределения ресурсов

В случае наличия цикла выдаётся ошибка (рис. 8), ресурс не закрепится за процессом и будет восстановлен граф распределения ресурсов (рис. 9).

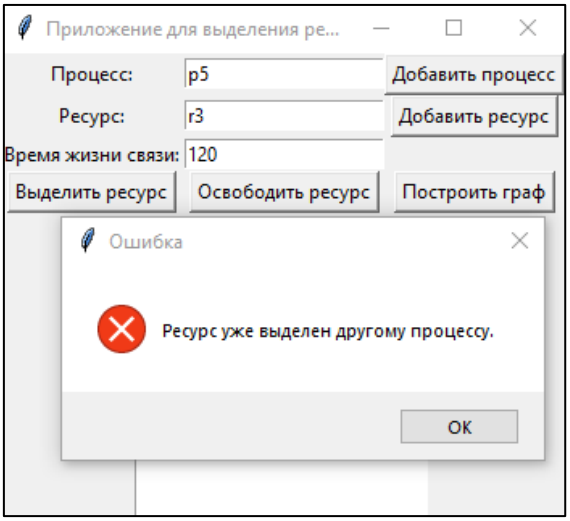

Рис. 8. Демонстрация ошибки выделения ресурса

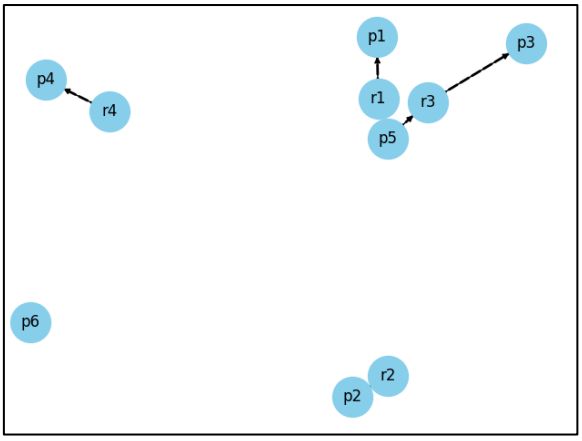

Рис. 9. Демонстрация предварительного графа распределения ресурсов

В ходе реализации процедуры тестирования разработанного программного обеспечения был осуществлён всесторонний анализ и глубокое исследование всех аспектов его функциональности. Основное внимание было сосредоточено на различных сценариях интеракции пользователя с данным программным продуктом, охватывающих обширный диапазон потенциальных ситуаций, возникающих при его использовании. В результате тестирования было установлено, что программное обеспечение работоспособно и эффективно выполняет возложенные на него функции.

Список литературы

1. Ногин В.Д. Принятие решений при многих критериях. СПб.: Изд-во «ЮТАС», 2007. 104 с.

2. Можаров Г. П. Разрешение конфликтов в компьютерных системах // Наука и Образование. МГТУ им. Н.Э. Баумана. 2015. № 8. С. 184-194.

3. Андреев А.М., Можаров Г.П., Сюзев В.В. Многопроцессорные вычислительные системы: теоретический анализ, математические модели и применение. М.: Изд-во МГТУ им. Н.Э. Баумана, 2011. 334 с.

4. Громов Ю. Ю., Мартемьянов Ю. Ф., Яковлев А. В. Операционные системы. Концепции построения и обеспечения безопасности. Тамбов: Изд-во ФГБОУ ВПО «ТГТУ», 2015. 288 с.

5. Habermann A. N. Prevention of system deadlocks // Communs ACM. 1969. № 7, P. 373-377, 385.

6. Havender J. W. Avoiding deadlock in Multitasking systems //. IBM System Journal. 1968, 2, № 7, P.74-84.