РЕАЛИЗАЦИЯ МОДЕЛИ МОДУЛЯ ОЧЕРЕДИ ЗАДАЧ

К.В. Захарченко
Кафедра теоретических основ электротехники,
Белорусский государственный университет информатики и радиоэлектороники
Минск, Республика Беларусь
E-mail: cvzakharchenko@gmail.com

При разработке планировщиков задач часто возникает задача организации обработки очереди задач. Подобная задача возникает в различных устройствах, принимающих и обрабатывающих команды из внешних источников, в промежуточных модулях, задача которых является обработать какой-то набор элементов, сохранив на выходе их порядок. Зачастую, элементы делятся на группы, так, что важен только порядок элементов внутри группы. В данной работе рассмотрены плюсы и минусы подходов к реализации структур данных для описанной ситуации. Описан такой примитив, как мульти-очередь или multi-fifo. Проведены тесты производительности различных структур данных.

Введение

Проблему реализации планировщика задач на аппаратном уровне зачастую решают достаточно тривиально: задачи записываются в циклический буфер, при извлечении задачи, она в отдельной структуре данных помечается, как невалидная. Когда список задач становится достаточно разреженным, наступает процесс уплотнения списка (консолидации), запись задач происходит по указателю на первое свободное место в буфере. Таким образом, циклический буфер представляет из себя простейшую реализацию очереди с помощью массива, индексируемого циклически. Однако, позволив использование более высокоуровневых структур данных, можно добиться более производительной работы очереди и избавиться от этапа консолидации. В данной работе рассмотрена структура данных, которая во-первых позволяет реализовать работу очереди задач более эффективно, чем тривиальная реализация, во-вторых позволяет сохранить некоторые особенности тривиальной версии.

I. Аппаратная реализация очереди задач

Аппаратная реализация механизма планировщика как правило ограниченна реализацией на основе массива. Такая реализация достаточно проста концептуально, но страдает от ряда недостатков. Хранение очереди в массиве позволяет за константное время удалять и добавлять элементы в конец или начало. Но очередь задач зачастую представляет из себя совокупность из нескольких очередей задач, принадлежащих различным классам, исполняющим единицам. Как изображено на рис. 1, очередь Q состоит из задач типа 1, 2 и 3, то есть представляет из себя 3 очереди. В таком случае необходимо удаление не только из начала очереди, но и из середины, что уже невозможно выполнить за константное время, использую реализацию в виде массива. Если удалять элементы, не сдвигая остальные освободившееся место, то со временем очередь заполнится пробелами и в хвосте не останется места для записи новых элементов. Поэтому, чтобы уплотнить расположение элементов очереди, используют процедуру консолидации.

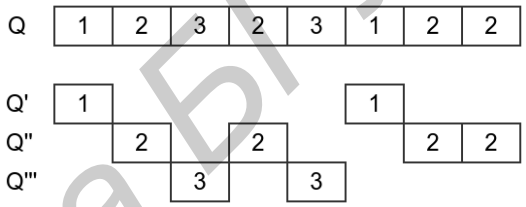

Рис. 1 – Реализация очереди на основе циклического буфера

Консолидация проходит по всей очереди и устанавливает элементы на свободные места. Такой процесс работает за линейное время и, если должным образом подобрать условия запуска консолидации, то амортизированное время вставки и удаления любой команды из очереди не изменится из-за затрат на консолидацию. Другая проблема хранения нескольких очередей в одном буфере – это то, что поиск элемента, лежащего в голове определённой субочереди может тоже занять линейное время при определённых условиях.

II. Модель очереди

При написании кода, реализующего поведение аппаратной очереди задач на языке достаточно высокого уровня, задачу можно решить намного эффективнее, если использовать реализацию очереди на основе списков. Однако, важно сохранить некоторые особенности поведения очереди на основе циклического буфера. Используем схему, представленную на рис. 2.

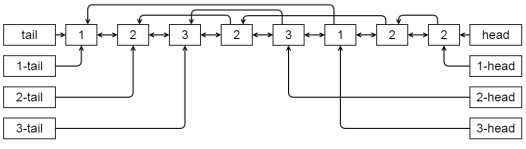

Рис. 2 – Реализация очереди на основе multi-fifo

Данная структура данных представляет из себя двусвязный список с одним дополнительным указателем у каждого элемента, указывающим на следующий элемент в списке, принадлежащий тому же классу, что и текущий элемент. Назовём такую структуру данных мульти-очередь или multi-fifo. Добавим к каждому элементу счётчик, изначально инициализирующийся нулём, который будет обозначать количество удалённых элементов, следующих за текущим элементом. Вставка нового элемента в очередь состоит в замене указателей в голове очереди. Удаление элемента во-первых обновляет указатели в следующем и предыдущем элементе, а во-вторых прибавляет значение счётчика к предыдущему элементу очереди, а к глобальному счётчику удалённых элементов прибавляет единицу. Если удаляемый элемент самый крайний в очереди и не имеет перед собой другого элемента кроме указателя на голову очереди, то значение счётчика вычитается из глобального счётчика удалённых элементов. Таким образом, во время работы структуры данных постоянно будет поддерживаться переменная, имеющая значение равное количеству команд, удалённых и вышедших за границы очереди. Процесс консолидации заключается в обнулении глобального счётчика и обнулении счётчиков у всех элементов. Простейшим образом это решается за линейное время. Данная структура данных позволяет симулировать поведение очереди задач на базе циклического буффера, однако делает это гораздо быстрее и прозрачнее, в то же время сохраняя все особенности аппаратной реализации. Интерфейс класса на языке c++, реализующего мульти-очередь представлен ниже:

```
template<class TYPE>
class multi_fifo
{
public:
    multi_fifo(uint32_t fifos_amount_);

    void push_back(uint32_t n, TYPE val);
    TYPE pop_front(uint32_t n);
    uint32_t size();
    uint32_t size(uint32_t n);
    uint32_t deleted();
    void consolidation();
};
```

Реализация может интегрироваться в модели аппаратных устройств как с помощью агрегации, так и наследованием от класса $multi_fifo$. Сравнительные тесты показывают выйгрыш по сравнению с другими методами реализации данного функционала.

III. Сравнение реализаций

Рассмотрим таблицу асимптотической оценки скорости выполнения запросов в различные реализации очереди.

Таблица 1 – Скорость выполнения основных запросов в различных реализациях

Скорость	Буфер. Худший случай.	Буфер. Среднее.	Multi-fifo. Худший случай.	Multi-fifo. Среднее.
Вставка в очередь.	O(N)	O(1)	O(1)	O(1)
Вставка в субочередь	O(N)	O(1)	O(1)	O(1)
Удаление из очереди	O(1)	O(1)	O(1)	O(1)
Удаление из субочереди	O(N)	O(1)	O(1)	O(1)
Консолидация	O(N)	O(N)	O(N)	O(N)

По итоговой таблице становится видно, что, хоть амортизированная оценка скорости выполнения основных запросов и совпадает, в худшем случае реализация на основе двусвязного списка выигрывает перед реализацией на базе кольцевого буфера. Процесс консолидации теряет необходимость, однако симулируется для поддержания соответствия поведению аппаратной реализации.

IV. Выводы

Предложенная структура данных позволяет удобно симулировать поведение циклического буфера, однако предлагает более простую и быструю реализацию поставленной задачи. Асимптотические оценки показывают выигрыш в скорости работы, хотя амортизированное время на выполнение запроса иногда одинаковое. Данная структура данных может применяться при построении моделей устройств памяти, а так же при необходимости в более специфических ситуациях.

1. Дональд Кнут Искусство программирования, том 2. Получисленные алгоритмы = The Art of Computer Programming, vol.2. Seminumerical Algorithms. – 3-е изд. – Москва: «Вильямс», 2007. – 832 с.
2. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ = Introduction to algorithms. – 2-е изд. – М.: Издательский дом «Вильямс», 2011.
3. Роберт Седжвик Фундаментальные алгоритмы на C. Анализ/Структуры данных/Сортировка/Поиск = Algorithms in C. Fundamentals/Data Structures/Sorting/Searching. – СПб.: ДиаСофтЮП, 2003. – 672 с.
4. Никлаус Вирт. «Алгоритмы и структуры данных». Prentice-Hall, Inc., 1985.