|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение высшего образования**  **«Московский государственный технический университет имени Н.Э. Баумана**  **(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)** |

Отчет по лабораторной работе № 5

«Обработка очередей»

Работу выполнил:

студент группы ИУ7-32Б

Лемешев А.П.  
Работу проверил:  
Силантьева Т.А.

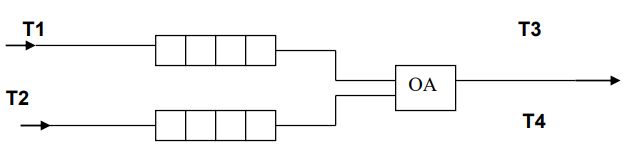
Москва, 2021 г.

# Условие задачи

Отработка навыков работы с типом данных «очередь», представленным в виде одномерного массива и односвязного линейного списка. Сравнительный анализ реализации алгоритмов включения и исключения элементов из очереди при использовании двух указанных структур данных. Оценка эффективности программы (при различной реализации) по времени и по используемому объему памяти.

# Техническое задание

Система массового обслуживания состоит из обслуживающего аппарата (ОА) и двух очередей заявок двух типов.



Заявки 1-го и 2-го типов поступают в "хвосты" своих очередей по случайному закону с интервалами времени Т1 и Т2, равномерно распределенными от 1 до 5 и от 0 до 3 единиц времени (е.в.) соответственно. В ОА они поступают из "головы" очереди по одной и обслуживаются также равновероятно за времена Т3 и Т4, распределенные от 0 до 4 е.в. и от 0 до 1 е.в. соответственно, после чего покидают систему. (Все времена – вещественного типа). В начале процесса в системе заявок нет.

Заявка любого типа может войти в ОА, если:

а) она вошла в пустую систему;

б) перед ней обслуживалась заявка ее же типа;

в) перед ней из ОА вышла заявка другого типа, оставив за собой пустую очередь (система с чередующимся приоритетом).

Смоделировать процесс обслуживания первых 1000 заявок 1-го типа, выдавая после обслуживания каждых 100 заявок информацию о текущей и средней длине каждой очереди, а в конце процесса - общее время моделирования и количество вошедших в систему и вышедших из нее заявок обоих типов. По требованию пользователя выдать на экран адреса элементов очереди при удалении и добавлении элементов. Проследить, возникает ли при этом фрагментация памяти.

**Входные данные**

Целое число: либо 1, либо 2, где 2 означает, что будет происходить ввод новых времен, 1 - не будет.

Пункт меню: от 1 до 3.  
 1. Моделирование обслуживания очереди в виде статического массива

2. Моделирование обслуживания очереди в виде списка

3. Сравнение эффективности двух типов организации

В пунктах 1 и 2 целое число: либо 1, либо 2, где 2 означает, что будет происходить вывод массива свободных областей, 1 - не будет.

**Выходные данные**

Для пункта 1 и 2:

Каждые 100 обработанных заявок выводится:

- Количество обработанных заявок

- Количество обработанных заявок 1го типа

- Текущая очередь заявок 1го типа

- Текущая очередь заявок 2го типа

- Средняя длина очереди 1го типа

- Средняя длина очереди 2го типа

После 1000 обработанных заявок 1го типа:

- Время симуляции

- Высчитанное время симуляции

- Погрешность по симуляции

- Время простоя

- Высчитанное время простоя *(опционально)*

- Погрешность по простою *(опционально)*  
 - Количество заявок 1го типа поступивших в ОА

- Количество обработанных заявок 1го типа

- Количество заявок 2го типа поступивших в ОА

- Количество обработанных заявок 2го типа

- Время выполнения в Секундах и Наносекундах

Для пункта 3:

- Размер массива

- Размер списка

- Средняя время добавления заявки из массива (в тиках)

- Средняя время добавления заявки из списка (в тиках)

- Средняя время удаления заявки из массива (в тиках)

- Средняя время удаления заявки из списка (в тиках)

**Действие программы**

Моделирование обслуживания 1000 заявок в очереди, сравнение эффективности обработки очереди как статического массива и очереди как списка.

**Обращение к программе**

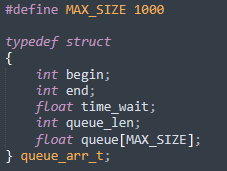
Запуск программы производится через командную оболочку MSYS2 для Windows, терминал sh/bash для Linux. Запуск программы происходит с помощью команды: ./app.exe.

**Возможные аварийные случаи**

1. Некорректный ввод пункта меню: пустая строка
2. Некорректный ввод пункта меню: буква
3. Некорректный ввод пункта меню: отрицательное число
4. Некорректный ввод пункта меню: число, не подходящее ни к одному пункту
5. Пункт 1, 2: ввод некорректного пункта при выборе вывода свободных адресов
6. Переполнение очереди в виде статического массива.

# Структуры данных

**Структура очереди, реализованной массивом (сдвижной):**



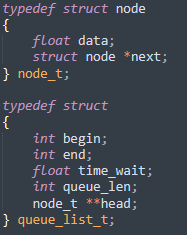
*begin* и *end* – индексы начала и конца очереди

*time\_wait* – время до следующей заявки

*queue\_len* – длина очереди

*queue[MAX\_SIZE]* – массив для хранения очереди

**Структура очереди, реализованная связанным списком:**



*node\_t* – тип данных для элемента списка

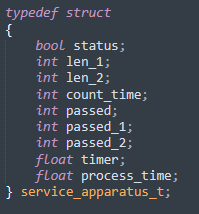
*begin* и *end* – количество элементов в списке

*time\_wait* – время до следующей заявки

*queue\_len* – длина очереди

*\*\*head* – связанный список

**Структура обслуживающего аппарата:**



*status* – статус аппарата

*len\_1* – длины первой очереди

*len\_2* – длины второй очереди

*count\_time* – количество замеров длин

*passed* – количество обработанных заявок

*passed\_1* – количество обработанных заявок 1го типа

*passed\_2* – количество обработанных заявок 2го типа

*timer* – время работы аппарата

*process\_time* – время обработки заявки

**Описание алгоритма**

1. Выводится меню программы

2. Пользователь вводит номер любой команды, которой соответствует свое назначение

3. Ввод осуществляется, пока не будет совершена ошибка при вводе (аварийная ситуация) или пока не будет введен 0 (означает выход из программы)

**Алгоритм аппарата**

Сначала генерируется время появления первых элементов в обоих очередях. Затем, пока из очереди не выйдет 1000 элементов из первой очереди, из «время-прибытия» 1, 2 и «время-обработки-в-очереди» вычитаем самую меньшую величину. Когда «время обработки» становится равно 0, статус ОА с «занято» меняется на «не занято», начиная обрабатывать следующий элемент, согласно установленными задачей правилам.

**Расчет моделирования**

Так как среднее время прихода больше среднего времени обслуживания, то время моделирования будет определяться временем прихода заявок первого типа.

Так нам требуется продолжать моделирование, пока из ОА не выйдут 1000 элементов из первой очереди, мы будем ориентироваться на время прихода 1000 элементов из первой очереди:

Ожидаемое время моделирования — 1000 \* ((5 + 1) / 2) = 3000

# Оценка эффективности

Размер массива 3000.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Заполненность | Добавление | | Удаление | | Память | |
| список | массив | список | массив | список | массив |
| 750 (25 %) | 3841 | 10 | 83 | 2248 | 12008 | 12016 |
| 1500 (50%) | 14041 | 18 | 212 | 8936 | 24008 | 12016 |
| 2250 (75%) | 30605 | 23 | 277 | 26615 | 36008 | 12016 |
| 3000 (100%) | 55564 | 32 | 564 | 43711 | 48008 | 12016 |

Очередь, как список, выгоднее по памяти, чем очередь, как массив, примерно до заполненности массива в 25%.

Массив проигрывает списку значительно по скорости удаления элемента, т.к. массив является сдвижным и при каждом удалении массив переписывает все свои элементы.

# Вывод

В случае моделировании обработки, очередь как массив всегда эффективнее в добавлении очереди как список, но по скорости удаления массив проигрывает, т.к. является сдвижным и после каждого удаления все элементы массива переписываются. По памяти список выгоднее массива до заполненности 25%. В ситуации, когда очередь заполняется быстрее, чем освобождается, лучше использовать список, в ином случае лучше использовать массив. Списки так же выгодно использовать, когда заранее неизвестен максимальный размер очереди

При проведении тестов ни разу не наблюдалась фрагментация памяти.

# Ответы на контрольные вопросы

**1. Что такое очередь?**

Очередь – это последовательный список переменной длины, включение в который идет с одной стороны (с хвоста), а исключение – с другой стороны (с головы). Принцип работы очереди: последний пришел – последний вышел, то есть, Last In – Last Out.

**2. Каким образом, и какой объем памяти выделяется под хранение очереди при различной ее реализации?**

Если очередь хранится в виде статического массива, то объём памяти равен 𝑠𝑖𝑧𝑒 \* 𝑠𝑖𝑧𝑒𝑜𝑓(𝑡𝑦𝑝𝑒) + 𝑠𝑖𝑧𝑒𝑜𝑓(𝑖𝑛𝑡) + 𝑠𝑖𝑧𝑒𝑜𝑓(𝑖𝑛𝑡) + 𝑠𝑖𝑧𝑒𝑜𝑓(s𝑠𝑖𝑧𝑒\_𝑡), и выделяется в стеке на этапе трансляции. Отличие динамического массива только в том, что размер массива 𝑠𝑖𝑧𝑒 известен на этапе запуска программы и объём памяти под такую структуру выделяется в куче. Если же очередь хранится в виде списка, то объём памяти равен 𝑠𝑖𝑧𝑒 \* 𝑠𝑖𝑧𝑒𝑜𝑓(𝑛𝑜𝑑𝑒\_𝑡) + 𝑠𝑖𝑧𝑒𝑜𝑓(𝑛𝑜𝑑𝑒\_𝑡 \*) + 𝑠𝑖𝑧𝑒𝑜𝑓(𝑛𝑜𝑑𝑒\_𝑡 \*) + 𝑠𝑖𝑧𝑒𝑜𝑓(s𝑠𝑖𝑧𝑒\_𝑡), при этом размер 𝑛𝑜𝑑𝑒\_𝑡 равен 𝑠𝑖𝑧𝑒𝑜𝑓(𝑖𝑛𝑡) + 𝑠𝑖𝑧𝑒𝑜𝑓(node\_t \*), память для очереди выделяется во время runtime, при необходимости выделения.

**3. Каким образом освобождается память при удалении элемента из очереди при ее различной реализации?**

Если очередь хранится в виде статического массива, то память не освобождается, ведь кольцевая очередь в данном случае работает по процессу циклического приращения, то есть когда мы пытаемся увеличить любую переменную и достигаем конца очереди, мы начинаем с начала очереди по модулю деления с размером очереди. Если очередь хранится в виде динамического массива или списка, то память, выделенная под удаляемый элемент очереди освобождается.

**4. Что происходит с элементами очереди при её просмотре?**

Элементы очереди удаляются.

**5. Каким образом эффективнее реализовывать очередь. От чего это зависит?**

Обычную очередь, конечно же, лучше организовывать списком, так как в нём не тратится время на сдвиг элементов, который происходит в массиве, при удалении элемента. При реализации кольцевой очереди нет необходимости сдвигать элементы массива, затраты по времени значительно уменьшаются. Но всё же при подборе таких интервалов T1 и T2 и T3, что в очереди как списке почти не приходится создавать новые узлы, список становится эффективнее по времени и по памяти. Но в случае, когда узлы нужно создавать достаточно часто, когда текущий размер очереди достаточно большой, эффективнее и по памяти, и по времени, использовать массив. Таким образом эффективность того или иного способа хранения зависит от подбора интервалов T1 и T2 и T3.

**6. В каком случае лучше реализовать очередь посредством указателей, а в каком – массивом?**

Реализовывать очередь массивом лучше, когда есть уверенность в том, что очередь не будет пустой. Когда же подобраны такие интервалы T1 и T2 и T3, что в очередь достаточно редко добавляются новые элементы, то лучше реализовывать очередь списком.

**7. Каковы достоинства и недостатки различных реализаций очереди в зависимости от выполняемых над ней операций?**

При реализации очереди в виде массива тратится значительное время на удаление элемента, так как необходимо сдвинуть все элементы очереди к голове. Также в случае статического массива есть ограничение по памяти, вследствие которого в большинстве ситуаций расходуется огромное количество лишнего объёма памяти. Преимуществом использования массива является линейность расположения в памяти, что облегчает доступ к элементам очереди. При реализации очереди в виде списка расходуется дополнительная память на указатели, поэтому для одинакового количества элементов в очереди список проиграет массиву. Но, например, в статическом массиве память ограничена, а в динамическом память может перевыделяться неудобно и к тому же при достижении максимального количества (максимальная во время всего моделирования) элементов очереди динамический массив будет перевыделен и после не будет перевыделяться, получается “лишняя” память “останется”, в списке же она обладает своего рода “универсальностью”, иными словами, мы можем контролировать каждый байт этой памяти. К недостаткам списка относится возможная фрагментация памяти при добавлении/удалении элементов.

**8. Что такое фрагментация памяти?**

Фрагментация – это дробление памяти на несмежные свободные области.

**9. На что необходимо обратить внимание при тестировании программы?**

На работу с динамическими ресурсами и на переполнение статического кольцевого массива.

**10. Каким образом физически выделяется и освобождается память при динамических запросах?**

При динамическом распределении памяти объекты размещаются в так называемой куче: при конструировании объекта указывается размер запрашиваемой под объект памяти, и, в случае успеха, выделенная область памяти изымается из кучи, становясь недоступной при последующих операциях выделения памяти. Противоположная по смыслу операция — освобождение занятой ранее под какой-либо объект памяти: освобождаемая память, также условно говоря, возвращается в кучу и становится доступной при дальнейших операциях выделения памяти.