|  |  |
| --- | --- |
|  | **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

Богатырев И.С.

Работу проверил:

Барышникова М.Ю.

Никульшина Т.А.

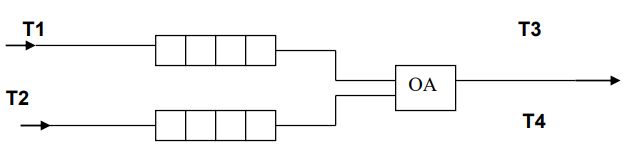
Москва, 2021 г.

**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**

Время обработки в очередях, пункт выбора меню.

**Выходные данные**

Текущее состояние очереди, временные замеры, количество занимаемой памяти.

**Возможные аварийные случаи**

Некорректный ввод пункта меню

**Структуры данных**

**Структура очереди, реализованной массивом:**

*typedef struct queue\_arr*

*{*

*float upcoming;*

*int begin, end;*

*float queue[QUEUE\_MAX\_SIZE];*

*int got;*

*} queue\_arr\_t;*

*upcoming – время до генерации следующего элемента очереди;*

*begin, end – индексы начала и конца очереди в массиве;*

*queue[QUEUE\_MAX\_SIZE] – статический не кольцевой массив для хранения очереди;*

*got – количество элементов, пришедших в очередь.*

**Структура очереди, реализованная связанным списком:**

*typedef struct queue\_list*

*{*

*float upcoming;*

*int begin, end;*

*node\_t \*\*head;*

*int got;*

*} queue\_list\_t;*

*upcoming – время до генерации следующего элемента очереди;*

*end – фактически – количество элементов в списке;*

*\*\*head – связанный список;*

*got – количество элементов, пришедших в очередь.*

**Структура обслуживающего аппарата:**

*typedef struct serviceApparatus*

*{*

*bool isBusy;*

*int passed;*

*float timer;*

*float process\_time;*

*int queue\_1\_passed;*

*int queue\_2\_passed;*

*int q1\_len\_sum;*

*int q2\_len\_sum;*

*int measurements;*

*} service\_apparatus\_t;*

*isBusy – статус занятости аппарата*

*passes – общее количество обработанных элементов;*

*timer – общее время работы;*

*queue\_1\_passed – количество обработанных элементов из первой очереди;*

*queue\_2\_passed – количество обработанных элементов из второй очереди;*

*q1\_len\_sum – сумма длин первой очереди на протяжении выполнения симуляции;*

*q2\_len\_sum – сумма длин второй очереди на протяжении выполнения симуляции;*

*measurements – количество выполненных замеров длин.*

**Алгоритм**

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.

**Предварительный 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 \* ((4 + 1) / 2) = 3000

Оценка эффективности

На очереди в 1000 элементов

|  |  |  |
| --- | --- | --- |
|  | Массив | Связанный список |
| Время моделирования (е.в.) | 3023.234131 | 3012.053955 |
| Добавление в очередь (такты) | 5 | 3756 |
| Удаление из очереди (такты) | 1969 | 61 |
| Размер элемента очереди (байт) | 6 | 47 |

**Контрольные вопросы**

**Что такое очередь?**

Очередь — структура данных, для которой выполняется правило FIFO, то есть первым зашёл — первым вышел. Вход находится с одной стороны очереди, выход — с другой.

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

При хранении массивом: кол-во элементов \* размер одного элемента очереди. Память выделяется на стеке при компиляции, если массив статический. Либо память выделяется в куче, если массив динамический.

При хранении списком: кол-во элементов \* (размер одного элемента очереди + указатель на следующий элемент). Память выделяется в куче для каждого элемента отдельно.

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

При хранении обычным статическим массивом память не освобождается – элементы, следующие за удаляемым, смещаются влево на один. При хранении списком, ссылка на головной элемент изменяется, а память, занимаемая удаляемым элементов, освобождается.

**Что происходит с элементами очереди при ее просмотре?**

Эти элементы удаляются из очереди.

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

Зная максимальный размер очереди, лучше всего использовать кольцевой статический массив. Не зная максимальный размер, стоит использовать связанный список, так как такую очередь можно будет переполнить только если закончится оперативная память.

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

При реализации статическим массивом, очередь всегда ограничена по размеру.

При реализации связанным списком, используется больше памяти.

Скорость добавления нового элемента выше у массива, удаления – у списка.

**Что такое фрагментация памяти?**

Фрагментация памяти — разбиение памяти на куски, которые лежат не рядом друг с другом. Можно сказать, что это чередование свободных и занятых кусков памяти.

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

Корректное освобождение памяти.

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

При запросе памяти, ОС находит подходящий блок памяти и записывает его в «таблицу» занятой памяти. При освобождении, ОС удаляет этот блок памяти из «таблицы» занятой пользователями памяти.

**Выводы**

Использование связанных списков невыгодно при реализации очереди. Списки проигрывают как по памяти, так и по времени обработки. Но, когда заранее неизвестен максимальный размер очереди, то можно использовать связанные списки, так как в отличии от статического массива, списки ограничены в размерах только размером оперативной памяти. Стоит отметить, что при проведении тестов ни разу не наблюдалась фрагментация памяти, но даже при этом, список проигрывает во времени обработки очереди.