МГТУ им. Н. Э. Баумана

**Дисциплина типы и структуры данных**

**Лабораторный практикум №5**

**по теме: «Обработка очередей»**

**Вариант №4**

Работу выполнил:

студент группы ИУ7-35

Платонова Ольга

Работу проверил:

Москва, 2019 г.

***Цель работы***: приобрести навыки работы с типом данных «очередь», представленным в виде одномерного массива и односвязного линейного списка, провести сравнительный анализ реализации алгоритмов включения и исключения элементов из очереди при использовании указанных структур данных, оценить эффективности программы по времени и по используемому объему памяти.

***Условие задачи***

Заявки 1-го и 2-го типов поступают в "хвосты" своих очередей по случайному закону с интервалами времени Т1 = {1; 5} и Т2 = {0; 3}. В ОА они поступают из "головы" очереди по одной и обслуживаются также равновероятно за времена Т3 = {0; 4} и Т4 = {0; 1}, после чего покидают систему. 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.

***Техническое задание***

1. Организовать очередь в виде массива/списка.

2. Смоделировать 100 заявок, каждая создана по случайному закону (зависит от времени).

3. Каждую заявку поместить в очередь в соответствии с типом.

4. Обработать (достать) заявку соответствующего типа.

5. В случае списка поместить адрес заявки в массив адресов.

6. Измерить время нахождения заявки в каждой очереди.

7. Измерить количество вошедших и вышедших заявок каждой очереди.

8. Произвести анализ по времени, памяти.

*Входные данные:*

Пользователь может самостоятельно выбрать реализацию очередей

Array queue-1 — массив

List queue-2 — список

Если очереди реализованы на списках, то пользователь может дополнительно запросить просмотр адресов освобожденных элементов.

Element addresses - 1

Программист должен рассмотреть ошибки, связанные с некорректным вводом, переполнением и обращением к пустой очереди.

*Выходные данные*

Текущая и средняя длина каждой из очередей для каждой сотой итерации.

Количество вошедших и вышедших запросов для каждой очереди.

Время моделирования и время простоя.

*Структура данных*

*1. Дескриптор очереди*

void init\_discriptor(discriptor \*d)

{

d->p\_in = NULL;

d->p\_out = NULL;

d->count\_request = 0;

d->curr\_size = 0;

d->out\_request = 0;

d->in\_request = 0;

}

*2. Массив*

void array\_discriptor(discriptor \*d, void \*q)

{

init\_discriptor(d);

d->low = q;

d->up = q + (d->max\_num);

d->p\_in = q;

d->p\_out = q;

}

***Описание алгоритма***

* Для массива и списка алгоритм будет одинаковым.

1. Проверить очередь на переполнение ((d1.p\_in == d1.p\_out && \*((char\*)d1.p\_in) != 0)).

2. Смоделировать время входа каждой из заявки.

3. Обработать заявку:

если очередь заявки текущего типа не пуста, а другая очередь пустая, то добавить заявку в очередь, посчитать время. Иначе — обработка заявки другого типа.

array\_push() - смещаем указатель к следующей ячейке памяти, записываем в массив текущий символ, увеличиваем количество элементов массива. Выполняем проверку на переполнение (d->p\_in != d→up).

list\_push() - аналогично.

array\_pop() - считываем данные, перемещаем указатель к предыдущему элементу, уменьшаем количество элементов массива. Проверяем очередь на пустоту (d->p\_in == d->p\_out && \*((char\*)d->p\_in) == '\0').

list\_pop() - аналогично.

***Тесты***

1. Увеличить(уменьшить) время появления и время обработки заявки каждого из типов.

***Проверка***

1. Время моделирования

Рассогласование между средними ожидаемыми и полученными временами, то есть величина, не должна быть больше 2-3%.

При этом

При тестировании получаем

время моделирования: 3075.507815 мс

время простоя составило: 3.213045 мс

вошедших заявок было: 1003 (первая очередь)

2037 (вторая очередь)

вышедших заявок: 1000 (первая очередь)

1923 (вторая очередь)

рассогласование времен: 2.516927%

Сравним полученные данные с теоретическими расчетами

Общее время моделирования / время прихода одной заявки

3075.5 / 3 = 1025.2 заявок

Погрешность состовляет

100 \* (1025.2 - 1003) / 1025.2 = 2.165%

Проверка по входным и выходным данным показала, что погрешность работы системы составляет не более 2.6%, что принадлежит интервалу от 2 до 3.

***Вопросы***

*1. Что такое очередь?*

Очередь — последовательный список переменной длины, включение элементов в который идет с одной стороны («хвоста»), а исключение — с другой («головы»). Принцип работы очереди основан на принципе FIFO (First In — First Out).

|  |  |
| --- | --- |
| Массив | Список |
| *2. Каким образом, и какой объем памяти выделяется под хранении очереди при различной ее реализации?* | |
| Память выделяется последовательно одной областью размером m \* L, где L — размер поля данных для одного элемента. В каждый текущий момент времени выделенная память может быть вся свободна, занята частично или занята полностью. | Память может выделяться как последовательно, так и произвольно, т.е. может возникать фрагментация. |
| *3. Каким образом освобождается память при удалении элемента из очереди при ее различной реализации?* | |
| При удалении элемента указатель смещается к следующему элементу.  Выделенная память остается, а освобождение происходит лишь при завершении программы, целой областью. | При удалении элемента исключается элемент, расположенный по указателю, указатель перемещается к предыдущему элементу, а выделенная память под удаленный элемент освобождается.  Происходит проверка на опустошение очереди. |

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

В классической реализации очереди просмотр содержимого без извлечения ее элементов невозможен.

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

Реализация очереди на массиве эффективна в том случае, когда количество элементов заранее известно и не слишком велико. Поскольку не тратится время на выделение памяти под каждый элемент. Недостатком является частое удаление/добавление элементов, что приводит к необходимости сдвигать все элементы, избегая мнимого переполнения.

В случае, когда очередь реализована на списке, добавление/удаление элементов происходит медленнее, но не возникает проблем, связаннах с фрагментацией памяти, как в массиве.

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

Очередь лучше реализовывать посредством массива, когда количество элементов заранее известно, а указателями — в случае, когда не известно количество, поскольку память ограничена лишь объемом доступной ОП.

*7. Что такое фрагментация памяти?*

Каким образом эффективнее реализовывать очередь

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

Необходимо выполнить проверка на количество вошедших и вышедших элементов, чтобы не было проблемы с переполнением очереди, а также выполнить проверку на время моделирования очередей.

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

Программа запрашивает блок памяти у ОС. ОС находит блок, требуемого размера, записывает адрес этого блока и его размер в таблицу адресов и возвращает в программу полученный адрес.

При попытке освободить блок, ОС удаляет адрес и таблицы адресов и блок считается освобожденным.

***Вывод***

На примере нашей задачи мы убедились, что реализация очереди эффективней на массиве в том случае, когда количество элементов велико.

Что касается времени, в случае массива мы получили 2292 мс, а на списках чуть дольше: 3712 мс, то есть реализация на массиве работает быстрее.