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)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5

По дисциплине «Типы и структуре данных»

Название: Обработка очередей

Студент Дремин Кирилл

Группа ИУ7 – 36Б

Тип лабораторной работы: Учебная

Вариант №7

Преподаватель Барышникова Марина Юрьевна

2022 г.

Содержание

Описание условия задачи

Реализовать операции работы с очередью, представленной в виде массива и односвязного линейного списка, оценив преимущества и недостатки каждой реализации.

Сравнить эффективность двух реализаций при моделировании бизнес-процесса (обработка заявок в очереди).

Система состоит из обработчика ОА и очереди заявок. Новые заявки поступают по случайному закону с интервалом времени T1 (от 0 до 6), обработка каждой занимает время в интервале T2 (от 0 до 1), и после обработки заявка поступает обратно в очередь, подвергаясь обработке 5 раз.

После 5 обработок, заявка покидает очередь.

Описание ТЗ

Описание исходных данных и результатов:

Программа реализует коллекцию однотипных объектов, работающую по принципу FIFO, или очередь.

Предоставляются операции добавления элемента в очередь, удаления элемента из очереди, а также расчёта её длины, выделения и освобождения памяти.

Способ обращения к программе

Обращение происходит посредством вызова заранее скомпилированного файла и ввода через консоль.

Описание входных данных

Программа позволяет произвести настройку моделирования и его запуск.

```
1. Задать интервал T1 (интервал времени между приходом заявок)
2. Задать интервал T2 (интервал времени обработки заявок)
3. Изменить число заявок при моделировании
4. Изменить статус вывода адресов памяти при работе с очередью на основе списка
```

```
5. Запустить моделирование
0. Выход из программы
```

Корректный ввод - цифра от 0 до 5.

Описание возможных аварийных ситуаций и ошибок пользователя:

1. Ввод некорректных данных.
2. Ошибки выделения динамической памяти.
3. Указание не имеющих смысла диапазонов (левая граница больше правой и наоборот).

Описание внутренних СД:

Программа оперирует АТД, описывающим операции над очередями:

```
#define queue_def(subname, type) \
typedef struct queue_name(subname, type) queue_name(subname, type);
\
// Добавить элемент в очередь \
err_t queue_enqueue(subname, type) (queue_name(subname, type) *queu
e, type element); \
// Удалить элемент из очереди \
err_t queue_dequeue(subname, type) (queue_name(subname, type) *queu
e, type *result); \
// Выделить память под очередь \
queue_name(subname, type) *queue_alloc(subname, type)(); \
// Освободить память, выделенную под очередь \
void queue_free(subname, type)(queue_name(subname, type) *queue); \
// Посчитать длину очереди
size_t queue_len(subname, type) (queue_name(subname, type) *queue);
\
// Посчитать размер занимаемой очередью области памяти \
size_t queue_sizeof(subname, type)(queue_name(subname, type) *queue
);
```

Данный АТД является полиморфным по названию (используется для определения механизма работы) и по типу хранимых данных. Также описанная очередь подразумевается, как не владеющая - она не производит освобождения памяти, выделенной под свои элементы, при их освобождении.

Очередь на основе массива реализована следующим образом:

```
#define arr_queue_def(type) queue_def(arr, type) \
struct queue_name(arr, type) \
{ \
    type *arr;                                   \
    type *pin;                                   \
    type *pout;                                  \
    type *end;                                   \
};
```

Здесь arr и end - указатели на начало и конец выделенного массива памяти, pin и pout - указатели на место вставки и исключения элемента из очереди.

Реализована не кольцевая, динамическая очередь, расширяющаяся по необходимости. Также реализован эффективный механизм оптимизации данных в очереди - в отличие от классической модели не происходит сдвига очереди после каждого удаления элемента, сдвиг происходит только при достижении рабочим участком очереди конца выделенного участка памяти.

Очередь на основе списка реализована следующим образом:

```
#define list_queue_def(type) queue_def(list, type) \
typedef struct node_name(type)                     \
{                                                  \
    type value;                                    \
    struct node_name(type) *next;\
} node_name(type);                                          \
struct queue_name(list, type) \
{ \
    node_name(type) *pin;                          \
    node_name(type) *pout;\
};
```

Узел списка хранит в себе данные и указатель на следующий узел, в то время как сама очередь хранит указатели на первый и последний узел для вставки и удаления соответственно.

Сравнение различных реализаций очереди

В ходе сравнения эффективности реализации очередей было использовано моделирование обработки заявок некой системой.

Измеряемые параметры: время (в условных единицах моделируемой системы), ошибка относительно теоретических расчётов, затраты динамической памяти (максимальные) и реального времени на выполнение моделирования.

Было произведено моделирование обработки заявок в кол-ве 1000, результат:

Тип	Время (ед. вр.)	Ошибка	Память	Время (мс)	Заявок вошло	Заявок вышло	Сработал ОА
Теория	3002.500	+0.000%	0 bytes	0 ms	--------	---------	----------
Массив	3002.939	+0.015%	36 bytes	15 ms	4000	1000	5000
Список	3002.337	-0.005%	32 bytes	12 ms	4000	1000	5000

Было произведено несколько замеров при моделировании обработки заявок в кол-ве 10000, результат:

1.

Тип	Время (ед. вр.)	Ошибка	Память	Время (мс)	Заявок вошло	Заявок вышло	Сработал ОА
Теория	30002.500	+0.000%	0 bytes	0 ms	--------	--------	----------
Массив	30000.810	-0.006%	36 bytes	62 ms	40000	10000	50000
Список	30004.371	+0.006%	32 bytes	118 ms	40000	10000	50000

2.

Тип	Время (ед. вр.)	Ошибка	Память	Время (мс)	Заявок вошло	Заявок вышло	Сработал ОА
Теория	30002.500	+0.000%	0 bytes	0 ms	---------	---------	----------
Массив	30001.046	-0.005%	36 bytes	61 ms	40000	10000	50000
Список	30000.966	-0.005%	32 bytes	104 ms	40000	10000	50000

3.

Тип	Время (ед. вр.)	Ошибка	Память	Время (мс)	Заявок вошло	Заявок вышло	Сработал ОА
Теория	30002.500	+0.000%	0 bytes	0 ms	---------	---------	----------
Массив	30001.957	-0.002%	36 bytes	75 ms	40000	10000	50000
Список	30005.866	+0.011%	32 bytes	69 ms	40000	10000	50000

Также для тестирования работы алгоритма было проведено моделирование ситуации с иными параметрами системы: среднее время обработки одного запроса составляет 1 единицу времени, а интервал между поступлениями запросов - 3 ед. времени.

Для 1000 заявок:

Тип	Время (ед. вр.)	Ошибка	Память	Время (мс)	Заявок вошло	Заявок вышло	Сработал ОА
Теория	7331.000	+0.000%	0 bytes	0 ms	--------	--------	----------
Массив	7446.248	+1.572%	5964 bytes	10 ms	8925	1000	7444
Список	7420.730	+1.224%	17700 bytes	94 ms	8890	1000	7418

Для 5000 заявок:

Тип	Время (ед. вр.)	Ошибка	Память	Время (мс)	Заявок вошло	Заявок вышло	Сработал ОА
Теория	37720.000	+0.000%	0 bytes	0 ms	--------	--------	----------
Массив	37143.175	-1.529%	29560 bytes	51 ms	44519	5000	37139
Список	37109.568	-1.618%	88464 bytes	2182 ms	44475	5000	37106

Также для 3, 5 и 10 заявок было осуществлено моделирование с выводом адресов обрабатываемых узлов списка (результат не приводится в силу большого объёма).

Рассмотрев полученные сообщения о выделенных/освобожденных участках памяти, можно сделать вывод, что значительной дефрагментации памяти не происходит, адреса памяти часто переиспользуются при освобождении-выделении.

Сравнение реализаций

Реализация на основе массива является гораздо более эффективной, чем реализация на основе списка за счёт линейности расположения в памяти и быстрого доступа к своим элементам. Однако, в отличие от реализации на основе списка, данная реализация требует дополнительных усилий для её создания, таких как, например, разработка функции сжимания памяти обратно к началу выделенного массива и функции увеличения доступного объёма памяти. Последняя функция может быть эффективно реализована (с минимальными накладными расходами) только в случае, если заранее известна специфика требуемой работы, так как зависит от выбора множителя увеличения.

В данной программе высокая эффективность по занимаемой памяти достигнута как раз за счёт разработки вышеописанных дополнительных функций. Однако может случиться так, что при слишком большом объёме выделенной под изначальный массив памяти, большая часть не будет востребована, что приведёт к

потере эффективности использования памяти. Данная проблема решается за счёт выбора малого изначального размера массива и стратегии его увеличения по мере необходимости.

Вывод

Полученные относительно эффективности двух реализаций выводы совпадают с таковыми для структуры данных, работающей по принципу LIFO - стека (рассмотрено в ЛР4), так что можно утверждать, что наиболее эффективной реализацией данных структур является использование массива.

Ответы на вопросы

1. Что такое FIFO и LIFO?

FIFO - способ организации доступа к данным в коллекции по принципу “первый вошёл - первый вышел”, LIFO - “последний вошёл - первый вышел”.

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

В случае реализации очереди на основе массива, объём выделяемой памяти под изначальный массив определяется заранее, а в дальнейшем, при необходимости увеличения массива, он расширяется согласно заданной стратегии, в текущей реализации - увеличение размера в 4/3 раза.

В случае реализации очереди на основе массива память выделяется отдельно под каждый элемент, однако, помимо этого, присутствуют затраты памяти на указатели на узлы.

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

В случае реализации очереди на основе массива, память при удалении элементов не освобождается, в случае реализации очереди на основе списка, при удалении элемента, освобождается память, выделенная под узел, его содержащий.

4. Что происходит с элементами очереди при ее просмотре?

Единственный способ просмотреть элементы очереди - удалить их, так что для полного просмотра очереди без её изменения, потребуется циклически пройти по всем элементам, пока не вернемся к изначальному, что, однако, невозможно будет легко отследить при не уникальности элементов очереди.

5. От чего зависит эффективность физической реализации очереди?

Эффективность физической реализации очереди зависит от эффективности её расположения в памяти (большая степень дефрагментации присуща очереди на основе списка), а также эффективности операций с её элементами (операции с адресной арифметикой довольно быстры, в то время как необходимость проходить весь список - медленная операция).

6. Каковы достоинства и недостатки различных реализаций очереди в зависимости от выполненных над ней операций?

В очереди на основе списка гораздо проще происходит процесс вставки и удаления элементов, т.к. нет необходимости обрабатывать случаи переполнения или “уползания” указателей pout и pin к концу массива, как в очереди на его основе в то время, как очередь на основе массива эффективнее в операциях, требующих информации обо всей очереди, например - получение размера.

7. Что такое фрагментация данных, и в какой части ОП она возникает?

Фрагментация данных возникает при таком выделении динамической памяти после освобождения, при котором новая выделенная память располагается не там же, где была освобождена память до этого, а в другом участке памяти. Она может возникать даже несмотря на то, что участок освободившейся памяти, может быть использован для выделения памяти и приводит к замедлению доступа к памяти и повышенному её расходу (т.к. оставшиеся небольшие участки памяти не всегда могут быть использованы).

8. Для чего нужен алгоритм “Близнецов”?

Алгоритм “близнецов” позволяет дробить участок памяти на более мелкие (а также объединять их) для того, чтобы эффективно распределять память между клиентами, в то же время позволяя легко отслеживать свободную и занятую память, разбивая её на блоки.

9. Какие дисциплины выделения памяти вы знаете?

Две основные дисциплины памяти - выделение “самого подходящего” блока (наиболее приближенного к запрашиваемому по размеру) и “первого подходящего” - первого блока, достаточного для использования клиентом.

10. На что необходимо обратить внимание при тестировании программы?

На различные сценарии работы программы - при большем времени обработки и большем времени поступления данных, а также на наличие фрагментации памяти при реализации очереди списком (в случае ограниченности очередей - на переполнение).

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

После вызова функции выделения памяти сначала происходит поиск достаточного участка среди уже выделенной свободной памяти, в случае неудачи происходит системный вызов (с соответствующим переключением контекста в пространство ядра), после чего происходит поиск в куче свободной области памяти (зачастую находится большая область памяти, с расчётом на дальнейшее выделение динамическое памяти), клиент же получает указатель на выделенную область памяти, которая теперь помечена как занятая, в случае неудачи выделения - возвращается NULL.

При вызове функции освобождения памяти происходит поиск указанной области среди всех занятых областей памяти, дальше область помечается как свободная и может быть использована в дальнейшем.