|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство образования и науки 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 (ИУ)

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии (ИУ7)

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5  
«ОБРАБОТКА ОЧЕРЕДЕЙ»**

Название предмета: Типы и структуры данных

Студент: Малышев Иван Алексеевич

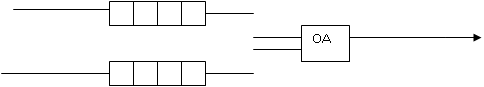
Группа: ИУ7-31Б

*2020 г.*

**Цель работы**: 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 объёму памяти.

**Описание условия задачи**

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

 Т1

Т3

Т2 Т4

Заявки 1-го и 2-го типов поступают в "хвосты" своих очередей по случайному закону с интервалами времени Т1 и Т2, равномерно распределёнными 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, распределённые 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:

1. она вошла в пустую систему;
2. перед ней обслуживалась заявка её же типа;
3. 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 неё 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**

**Входные данные**

1. **Целое число, представляющее собой номер команды:** целое число от 0 до 4.
2. **Действительный числа, означающие интервалы входа или обработки заявок:** действительные неотрицательные числа.

**Выходные данные**

1. Результат выполнения индивидуального задания – промежуточная и конечная информация о состояниях очередей и времени моделирования.
2. Оценка погрешности.

**Функция программы:** программа выполняет ряд функций, указанных при её первом запуске. Она позволяет:

1. Смоделировать процесс с очередями в виде массива
2. Смоделировать процесс с очередями в виде списка
3. Изменить интервалы времени
4. Сравнить время работы и занимаемую память реализаций очереди

При выборе второго пункта программа перед запуском моделирования спрашивает, хочет ли пользователь выводить освобождённые участки памяти.

**Способ обращения к программе:**

К программе можно обратиться через терминал ОС по её имени «app.exe». Дальнейшие инструкции будут выведены после запуска.

**Аварийные ситуации:**

* Некорректный номер операции, т. е. число меньше нуля или больше 4
* Ввод некорректного интервала времени входа или обработки заявки

Во всех указанных случаях программа завершится корректно или сообщит об ошибке.

**Структуры данных**

Реализация очереди на основе динамического массива

typedef struct

{

size\_t head, tail; - указатели на начало и конец очереди

size\_t queue\_size, elem\_size, capacity; - текущий размер очереди, размер элемента (в байтах), вместимость очереди

void \*data; - данные

} a\_queue;

Реализация очереди на основе линейного односвязного списка:

typedef struct node

{

void \*data; - данные

struct node \*next; - указатель на следующий узел

} node\_t; - узел

typedef struct

{

node\_t \*head, \*tail; - указатели на начало и конец очереди

size\_t queue\_size, elem\_size; - текущий размер очереди, размер элемента (в байтах)

} l\_queue;

**Описание функций**

node\_t \*create\_node(size\_t elem\_size) – создать узел списка

void free\_node(node\_t \*node) – очистить узел списка

l\_queue \*l\_create(size\_t queue\_size, size\_t elem\_size) – создать очередь в виде списка

void l\_destroy(l\_queue \*queue) – очистить очередь в виде списка

int l\_push(void \*queue, void \*src) – добавить элемент в очередь в виде списка

int l\_pop(void \*queue, void \*dst) – вытащить элемент из очереди в виде списка

bool l\_is\_empty(void \*queue) – проверка пустоты очереди в виде списка

bool l\_is\_full(void \*queue) – проверка полноты очереди в виде списка (по факту – возвращает ноль: нужна лишь для единообразия с очередью в виде массива)

node\_t \*push\_addr\_list(node\_t \*head, void \*addr) – добавить адрес узла в список удалённых узлов

void free\_addr\_list(node\_t \*head) – очистить список удалённых узлов

void print\_addr\_list(void) – печать списка удалённых узлов

a\_queue \*a\_create(size\_t queue\_size, size\_t elem\_size) – создать очередь в виде массива

void a\_destroy(a\_queue \*queue) - очистить очередь в виде массива

int a\_push(void \*queue, void \*src) - добавить элемент в очередь в виде массива

int a\_pop(void \*queue, void \*dst) – вытащить элемент из очереди в виде массива

bool a\_is\_empty(void \*queue) – проверка пустоты очереди в виде массива

bool a\_is\_full(void \*queue) – проверка полноты очереди в виде массива

int unit\_sim(void \*q1, void \*q2, push\_fp push, pop\_fp pop, empt\_fp is\_empty, full\_fp is\_full, size\_t count, size\_t step, bool verbose, preset\_t t, stats\_t \*stats) – функция моделирования

int int\_sscan(FILE \*f, int \*dst) – ввод числа

int bool\_sscan(FILE \*f, bool \*dst) – ввод булевого значения

**Алгоритм**

1. Пользователь вводит номер операции из меню.
2. Пока пользователь не введёт 0 (выход из программы), ему будет предложено выполнять набор действий.
3. При настройке интервалов пользователь вводит нужные ему интервалы с клавиатуры.

**Тестирование**

Положительные тесты

|  |  |  |  |
| --- | --- | --- | --- |
|  | Тест | Пользовательский ввод | Результат |
| 1 | Настройка времени прихода и обработки заявок | 3  0 1.5  2 3  4 5  6 7 | Настройка интервалов успешно завершена |
| 2 | Сравнение времени работы и занимаемую память реализаций очереди | 4 | \*Вывод результатов сравнения |
| 3 | Моделирование процесса обслуживания заявок с очередью в виде массива | 1 | \*Вывод информации о результатах моделирования |
| 4 | Моделирование процесса обслуживания заявок с очередью в виде списка | 2 | \*Вывод информации о результатах моделирования |

Негативные тесты

|  |  |  |  |
| --- | --- | --- | --- |
|  | Тест | Пользовательский ввод | Результат |
| 1 | Неверный интервал времени | 1  S -1 | Неверный ввод |
| 2 | Введено неизвестное действие | 5 | Неизвестная команда |
| 3 | Неверно введена команда | s | Неверная команда |

**Оценка эффективности**

Результаты со стандартными настройками.

Теоретические расчёты:

1000 \* (1 + 5) / 2 = 3000 е. в. – время, за которое придёт 1000 заявок первого типа.

За это же время придёт 3000 / ((0 + 3) / 2) = 2000 заявок второго типа.

Все заявки обработаются за 1000 \* (0 + 4) / 2 + 2000 \* (0 + 1) / 2 = 3000 е. в.

Время, за которое придёт 1000 заявок первого типа, и время обработки всех заявок совпадают – значит автомат работает без простоев и без задержек очереди.

Очередь в виде массива:

ВЫЧИСЛЕНИЕ

Ожидаемое время моделирования: 3000.0

МОДЕЛИРОВАНИЕ

Время моделирования: 2975.7

Время простоя: 9.8

Среднее время обслуживания заявки: 1.0

ПЕРВАЯ ВТОРАЯ

Вошло: 1000 1956

Вышло: 1000 1849

ПОГРЕШНОСТЬ: 0.810%

Очередь в виде списка:

ВЫЧИСЛЕНИЕ

Ожидаемое время моделирования: 3000.0

МОДЕЛИРОВАНИЕ

Время моделирования: 2970.5

Время простоя: 97.7

Среднее время обслуживания заявки: 1.0

ПЕРВАЯ ВТОРАЯ

Вошло: 1000 1971

Вышло: 1000 1953

ПОГРЕШНОСТЬ: 0.985%

Среднее время обработки заявок первого типа больше среднего времени прихода заявки:

Теоретические расчёты:

1000 \* (1 + 5) / 2 = 3000 е. в. – время, за которое придёт 1000 заявок первого типа.

За это же время придёт 3000 / ((0 + 3) / 2) = 2000 заявок второго типа.

Все заявки обработаются за 1000 \* (0 + 10) / 2 + 2000 \* (0 + 1) / 2 = 6000 е. в.

Время, за которое придёт 1000 заявок первого типа, и время обработки всех заявок не совпадают – значит автомат работает либо с простоями, либо с задержками очереди.

В таком случае ожидаемое время моделирования равно максимуму из времени прихода 1000 заявок первого типа или времени их обработки. В данном случае 1000 \* (1 + 5) / 2 < 1000 \* (0 + 10) / 2, поэтому ожидаемое время моделирования будет равно времени обработки 1000 заявок первого типа, т. е. 5000 е. в.

Результат моделирования:

Введите интервал входа заявок в первую очередь(T1): 1 5

Введите интервал входа заявок во вторую очередь(T2): 0 3

Введите интервал обработки заявок первого типа(T3): 0 10

Введите интервал обработки заявок второго типа(T4): 0 1

ВЫЧИСЛЕНИЕ

Ожидаемое время моделирования: 5000.0

МОДЕЛИРОВАНИЕ

Время моделирования: 4926.7

Время простоя: 1.0

Среднее время обслуживания заявки: 1.2

ПЕРВАЯ ВТОРАЯ

Вошло: 1000 3259

Вышло: 1000 15

ПОГРЕШНОСТЬ: 1.471%

Среднее время обработки заявок первого типа меньше среднего времени прихода заявки:

Теоретические расчёты:

1000 \* (4 + 10) / 2 = 7000 е. в. – время, за которое придёт 1000 заявок первого типа.

За это же время придёт 7000 / ((0 + 3) / 2) = 4667 заявок второго типа.

Все заявки обработаются за 1000 \* (0 + 3) / 2 + 4667 \* (0 + 1) / 2 = 3833.5 е. в.

Время, за которое придёт 1000 заявок первого типа, и время обработки всех заявок не совпадают – значит автомат работает либо с простоями, либо с задержками очереди.

В таком случае ожидаемое время моделирования равно максимуму из времени прихода 1000 заявок первого типа или времени их обработки. В данном случае 1000 \* (4 + 10) / 2 > 1000 \* (0 + 3) / 2, поэтому ожидаемое время моделирования будет равно времени прихода 1000 заявок первого типа, т. е. 7000 е. в.

Результат моделирования:

Введите интервал входа заявок в первую очередь(T1): 4 10

Введите интервал входа заявок во вторую очередь(T2): 0 3

Введите интервал обработки заявок первого типа(T3): 0 3

Введите интервал обработки заявок второго типа(T4): 0 1

ВЫЧИСЛЕНИЕ

Ожидаемое время моделирования: 7000.0

МОДЕЛИРОВАНИЕ

Время моделирования: 7039.8

Время простоя: 3248.0

Среднее время обслуживания заявки: 0.7

ПЕРВАЯ ВТОРАЯ

Вошло: 1000 4552

Вышло: 1000 4552

ПОГРЕШНОСТЬ: 0.568%

Результаты сравнения реализаций при стандартных интервалах:

Введите кол-во повторений: 1000

Итоговое время работы очереди в виде списка: 762

Итоговое время работы очереди в виде массива: 277

Максимальная занимаемая память очередью в виде списка: 104 б

Максимальная занимаемая память очередью в виде массива: 8096 б

**Контрольные вопросы**

**1. Что такое очередь?**

Очередь – последовательной список переменной длины. Включение элементов идёт с «хвоста» списка, исключение – с «головы» списка. Принцип работы: первым пришёл – первым вышел, First In First Out.

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

При реализации очереди списком, под каждый новый элемент выделяется память из кучи, элементы связываются указателями.

При реализации очереди массивом, выделяется блок памяти из N\*sizeof(element) байт, где N – максимальное количество элементов в очереди, элементы следуют друг за другом последовательно.

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

При реализации очереди списком, головной элемент считывается, указатель на «голову» очереди переходит на следующий элемент, считанный элемент удаляется.

При реализации очереди массивом, головной элемент считывается, остальные элементы массива сдвигаются на 1 – длина очереди уменьшается на 1, элемент [1] массива «затирает» головной элемент [0].

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

При просмотре очереди, головной элемент из неё удаляется. Остальные элементы сдвигаются (массив), либо указатель на начало передвигается на следующий элемент (список).

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

При реализации очереди списком, проще всего добавлять и удалять из неё элементы, однако может возникнуть фрагментация памяти. При реализации очереди массивом дефрагментации не возникает, однако может возникнуть переполнение памяти, а добавление и удаление элементов сложнее. Способ реализации зависит от того, в чем мы сильнее ограничены – в памяти или во времени выполнения операций.

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

Очередь лучше реализовывать с помощью указателей, если новые элементы в среднем появляются реже, чем происходит полное очищение очереди – в общем случае фрагментация не возникает. Реализация с помощью указателей применима если требуется строгий контроль фрагментации.

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

При реализации очереди массивом не возникает фрагментации памяти, однако может произойти переполнение очереди, а также затрачивается дополнительное время на сдвиг элементов. Сдвига можно избежать, если использовать кольцевой массив, однако при этом усложняются операции добавления и удаления элементов. Наконец, при реализации списком проще всего реализуются алгоритмы добавления и удаления элементов, но может возникнуть фрагментация.

**8. Что такое фрагментация памяти?**

Фрагментация – чередование участков памяти при последовательных запросах на выделение и освобождение памяти. «Занятые» участки чередуются со «свободными» - однако последние могут быть недостаточно большими для того, чтобы сохранить в них нужное данное.

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

При реализации очереди списком необходимо следить за освобождением памяти при удалении элемента из очереди. Если новые элементы приходят чаще, чем удаляются старые, очередь растёт и может происходить фрагментации памяти.

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

Программа даёт запрос ОС на выделение блока памяти необходимого размера. ОС находит подходящий блок, записывает его адрес и размер в таблицу адресов, а затем возвращает данный адрес в программу.

При запросе на освобождение указанного блока программы, ОС убирает его из таблицы адресов, однако указатель на этот блок может остаться в программе. Попытка считать данные из этого блока может привести к непредвиденным последствиям, поскольку они могут быть уже изменены.

**Вывод**

Эффективность реализации работы со списком линейно зависит от кол-ва операций добавления и извлечения элемента из очереди. Недостатком очереди-списка является то, что при совершении этих операций он обязательно выделяет или освобождает память, поэтому проигрываем по быстродействию по сравнению с очередью-массивом. В тоже время, очередь-массив производит операции выделения/освобождения память один раз, но при этом требует больше памяти для хранения всех заявок, когда как список требует памяти только для действительных заявок.

К недостаткам очереди-списка также можно отнести неравномерность распределения памяти — возникновение внутренних и внешних дыр (фрагментация памяти).

К недостаткам очереди-массива также можно отнести необходимость сдвига элементов при удалении и проблему переполнения.

Также имеет значение, известно ли заранее максимальное кол-во элементов в очереди. Очередь-список позволяет воспользоваться памятью, ограниченной лишь объёмом оперативной памяти компьютера. При переполнении очереди-массива, необходимо совершать дополнительно копирование и расширять размер массива.