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 №05 ПО ДИСЦИПЛИНЕ: ТИПЫ И СТРУКТУРЫ ДАННЫХ

Обработка очередей.

Студент **Батуев А.Г.**

Группа **ИУ7-36Б**

Вариант **1**

Название предприятия **НУК ИУ МГТУ им. Н. Э. Баумана**

Студент _____ **Батуев А.Г.**

Преподаватель _____ **Никульшина Т.А.**

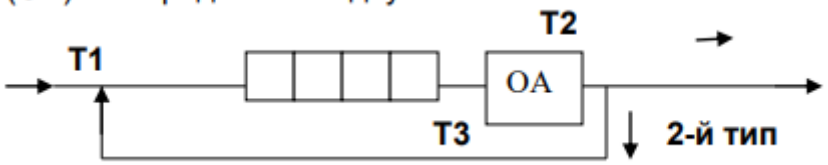
2024 г.

Условие задачи

Цель работы: отработка навыков работы с типом данных «очередь», представленным в виде одномерного статического массива (представление динамическим массивом можно добавить по желанию) 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.

Вариант 1.

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



Заявки 1-го типа поступают в "хвост" очереди по случайному закону с интервалом времени T1, равномерно распределенным от 0 до 5 единиц времени (е.в.). В ОА они поступают из "головы" очереди по одной и обслуживаются также равновероятно за время T2 от 0 до 4 е.в., после чего покидают систему.

Единственная заявка 2-го типа постоянно обращается в системе, обслуживаясь в ОА равновероятно за время T3 от 0 до 4 е.в. и возвращаясь в очередь не далее 4-й позиции от "головы". В начале процесса заявка 2-го типа входит в ОА, оставляя пустую очередь. (Все времена – вещественного типа)

Смоделировать процесс обслуживания первых 1000 заявок 1-го типа. Выдавать после обслуживания каждых 100 заявок 1-го типа информацию о текущей и средней длине очереди, количестве вошедших и вышедших заявок и о среднем времени пребывания заявок в очереди. В конце процесса выдать общее время моделирования, время простоя аппарата, количество вошедших в систему и вышедших из нее заявок первого типа и количество обращений заявок второго типа. По требованию пользователя выдать на экран адреса элементов очереди при удалении и добавлении элементов. Проследить, возникает ли при этом фрагментация памяти.

Техническое задание

На вход **программа получает**:

- опция из меню (цифра от 0 до 10)
- очередь (выполненная списком и массивом)

Программа выводит:

- Состояние очереди (адреса элементов)
    - Для очереди на списке удаленные адреса
- Результаты моделирования очереди
- Результаты измерения скорости добавления / удаления

Программа должна реализовывать создание, хранение, вывод очереди, а также операции добавления, удаления элемента из очереди и моделирование очереди.

Обращение к программе осуществляется по указанию названия программы (./app.exe).

Необходимо внимательно **обработать возможные аварийные ситуации**, которые могут включать:

- неправильный ввод данных
- ошибка выделение памяти
- переполнение очереди
- удаление из пустой очереди

Описание внутренних структур данных

request_type_t - представляет тип заявки.

- FIRST — заявка первого типа.
- SECOND — заявка второго типа.

request_t - описывает заявку, содержащую параметры прихода и обработки.

- type — тип заявки (request_type_t).
- coming — время прихода заявки.
- processing — время обработки заявки.

deleted_node_t - представляет удалённый узел очереди, используемый для управления памятью.

- void *address — адрес удалённого узла.
- struct deleted_node *next — указатель на следующий удалённый узел.

arr_queue_t - представляет очередь фиксированного размера, хранящую элементы в массиве.

- request_t array[MAX_QUEUE_LEN] — массив заявок фиксированного размера.
- unsigned in — индекс для добавления нового элемента.
- unsigned out — индекс для удаления элемента.
- unsigned length — текущая длина очереди.

queue_item_t - описывает узел в связной очереди.

- request_t data — данные заявки.
- struct queue_item *prev — указатель на предыдущий элемент.

list_queue_t - представляет очередь на основе связного списка.

- queue_item_t *head — указатель на первый элемент в очереди.
- queue_item_t *tail — указатель на последний элемент в очереди.
- deleted_node_t *deleted — указатель на список удалённых узлов для повторного использования.
- unsigned length — текущая длина очереди.

```
#define MAX_QUEUE_LEN 2000

typedef enum order_type
{
    FIRST = 1,
    SECOND
} request_type_t;

typedef struct
{
    request_type_t type;
    double coming;
    double processing;
} request_t;

typedef struct deleted_node
{
    void *address;
    struct deleted_node *next;
} deleted_node_t;
```

```
typedef struct
{
    request_t array[MAX_QUEUE_LEN];
    unsigned in;
    unsigned out;
    unsigned length;
} arr_queue_t;

typedef struct queue_item
{
    request_t data;
    struct queue_item *prev;
}queue_item_t;

typedef struct list_queue
{
    queue_item_t *head;
    queue_item_t *tail;
    deleted_node_t *deleted;
    unsigned length;
} list_queue_t;
```

Основные функции программы

Программа реализует следующие основные функции:

- void action(int option, arr_queue_t *arr_queue, list_queue_t *list_queue); Выполняет действие в зависимости от переданного параметра option, представляет меню в программе:
    1. Создание очереди (массивом)
    2. Создание очереди (списком)
    3. Удаление элемента (из массива)
    4. Удаление элемента (из очереди)
    5. Показать состояние очереди (массива)
    6. Показать состояние очереди (списка)
    7. Моделирование (массивом)
    8. Моделирование (списком)
    9. Обновить время
    10. Замерить скорость
    11. Выход
- void show_arr_queue(arr_queue_t queue); Отображает текущее состояние очереди, реализованной как массив.
- void show_list_queue(list_queue_t queue); Отображает текущее состояние очереди, реализованной как связный список.
- int arr_push(arr_queue_t *queue, request_t request); Добавляет заявку request в очередь на основе массива. Возвращает 0 при успешном добавлении, -1 при переполнении.
- int arr_insert_at(arr_queue_t *queue, request_t request, unsigned position); Вставляет в заданную позицию очереди, выполненную массивом.
- request_t *arr_pop(arr_queue_t *queue); Удаляет и возвращает указатель на заявку из очереди, реализованной как массив. Возвращает NULL, если очередь пуста.
- int list_push(list_queue_t *queue, request_t request); Добавляет заявку request в очередь на основе связного списка.
- void list_insert_at(list_queue_t *queue, request_t new_request, unsigned position); Вставляет в заданную позицию очереди, выполненную списком.
- queue_item_t *list_pop(list_queue_t *queue); Удаляет и возвращает указатель на заявку из очереди, реализованной как связный список. Возвращает NULL, если очередь пуста.
- void simulate_service_system_array(); Выполняет моделирование системы массового обслуживания с использованием очереди на основе массива.

- void simulate_service_system_list(); Выполняет моделирование системы массового обслуживания с использованием очереди на основе связного списка.

Описание алгоритма добавления и удаления

Для очереди-массива:

- Добавление (arr_push и arr_pushs)
    1. Проверяется, заполнена ли очередь
        - Если заполнена, возвращается ошибка переполнения
    2. Заявка записывается в позицию queue->in в массиве.
    3. Индекс queue->in увеличивается по модулю для циклического использования массива.
    4. Увеличивается длина очереди.
    5. В arr_pushs выполняется добавление нескольких элементов
- Удаление (arr_pop и arr_pops)
    1. Проверяется, пуста ли очередь.
        - Если пуста, возвращается NULL.
    2. Извлекается заявка из позиции queue->out массива.
    3. Индекс queue->out увеличивается по модулю для циклического использования массива.
    4. Уменьшается длина очереди
    5. В arr_pops выполняется удаление нескольких элементов

Для очереди-связного списка:

- Добавление (list_push и list_pushs)
    1. Проверяется, заполнена ли очередь.
        - Если заполнена, возвращается ошибка переполнения
    2. Выделяется память для нового узла.
        - Если память не выделена, возвращается ошибка выделения.
    3. В новый узел записываются данные заявки, и он присоединяется к концу очереди (queue->tail).
    4. Если очередь была пуста, новый узел становится началом очереди (queue->head).
    5. Увеличивается длина очереди (queue->length).
    6. В list_pushs выполняется добавление нескольких элементов.
- Удаление (list_pop и list_pops)
    1. Проверяется, пуста ли очередь.
        - Если пуста, возвращается NULL.
    2. Узел из начала очереди (queue->head) удаляется.
    3. Указатель queue->head перемещается на следующий узел.
        - Если очередь становится пустой, обнуляется queue->tail.
    4. Уменьшается длина очереди (queue->length).
    5. В list_pops удалённые узлы добавляются в список deleted для повторного использования.

Описание алгоритма моделирования

Алгоритм моделирует работу системы массового обслуживания, которая включает поток заявок двух типов и обслуживающий аппарат (ОА). Симуляция продолжается до тех пор, пока не будет обработано 1000 заявок первого типа. Работа алгоритма описывается следующим образом:

Инициализация системы
В начале работы создаётся пустая очередь для хранения заявок. Задаются начальные значения времени для поступления заявок первого и второго типов, а также параметры обработки. Инициализируются переменные для сбора статистики, включая количество заявок, среднее время ожидания, общее время простоя ОА и кумулятивную длину очереди.

Основной цикл симуляции
Алгоритм работает в рамках цикла, который завершается, когда количество обработанных заявок первого типа достигает 1000.

Поступление заявок.

- Для заявок первого типа проверяется, достигло ли текущее время момента их следующего прихода. Если да, создаётся новая заявка, которая добавляется в очередь. Обновляется статистика, связанная с заявками, и рассчитывается момент следующего прихода.
- Заявки второго типа обрабатываются по-своему. В самом начале в программу поступает заявка второго типа. Как только она выходит из ОА она сразу же помещается либо в конец очереди (если длина очереди меньше 4) или на 4 позицию

Обслуживание заявок.
Если ОА свободен и очередь не пуста, извлекается заявка из очереди для обслуживания. Рассчитывается её время ожидания в очереди, которое добавляется к общей статистике. Определяется момент окончания обработки заявки. Если заявка принадлежит к первому типу, увеличивается счётчик обработанных заявок этого типа.

Завершение обслуживания.
Когда время обслуживания текущей заявки истекает, ОА становится свободным, что позволяет обрабатывать следующую заявку из очереди.

Простои обслуживающего аппарата.
Если очередь пуста, а ОА свободен, фиксируется время простоя аппарата.

Сбор статистики.

- На каждом шаге вычисляется длина очереди для последующего расчёта её среднего значения.

- Каждые 100 обработанных заявок первого типа производится промежуточный расчёт статистики, включающий текущую длину очереди, среднее время ожидания, общее количество обработанных заявок и другую информацию.

Завершение симуляции
Когда обработка 1000 заявок первого типа завершается, рассчитывается общее время моделирования. Сравниваются ожидаемое время (на основе среднего интервала между заявками) и фактическое, после чего вычисляется процент расхождения между ними.

Итоговая статистика
По завершении работы алгоритма выводится сводка, содержащая:

- Общее время моделирования;
- Время простоя обслуживающего аппарата;
- Количество заявок первого и второго типов, поступивших и обработанных системой;
- Средние значения времени ожидания и длины очереди;
- Процентное расхождение между ожидаемым и фактическим временем моделирования.

Набор тестов

Функция arr_push

Тестовый сценарий	**Входные данные**	**Ожидаемый результат**
Позитивный: добавление в пустую очередь	Очередь пустая, request = {FIRST, 1.0, 2.0}	Элемент добавлен, очередь содержит один элемент, длина = 1
Позитивный: добавление в заполненную очередь с местом	Очередь содержит 2 элемента, request = {SECOND, 3.0, 4.0} (не полный размер)	Элемент добавлен, очередь увеличена на 1, длина увеличена
Негативный: переполнение очереди	Очередь содержит MAX_QUEUE_LEN элементов, request = {FIRST, 5.0, 6.0}	Возвращён код ошибки QUEUE_OVERFLOW_ERROR, очередь не изменена

Функция arr_pop

Тестовый сценарий	**Входные данные**	**Ожидаемый результат**
Позитивный: извлечение из непустой очереди	Очередь содержит элементы {FIRST, 1.0, 2.0}	Вернётся указатель на первый элемент {FIRST, 1.0, 2.0}, длина уменьшится
Негативный: извлечение из пустой очереди	Очередь пустая	Возвращён NULL, состояние очереди не изменилось

Функция list_push

Тестовый сценарий	**Входные данные**	**Ожидаемый результат**
Позитивный: добавление в пустую очередь	Очередь пустая, request = {SECOND, 2.0, 3.0}	Элемент добавлен, очередь содержит один элемент, длина = 1
Позитивный: добавление в очередь с элементами	Очередь содержит 3 элемента, request = {FIRST, 4.0, 5.0}	Новый элемент добавлен в хвост, длина увеличена
Негативный: переполнение очереди	Очередь содержит MAX_QUEUE_LEN элементов, request = {FIRST, 6.0, 7.0}	Возвращён код ошибки QUEUE_OVERFLOW_ERROR, очередь не изменена
Негативный: ошибка выделения памяти	Симулируется ошибка malloc	Возвращён код ошибки ALLOCATE_ERROR

Функция list_pop

Тестовый сценарий	**Входные данные**	**Ожидаемый результат**
Позитивный: извлечение из непустой очереди	Очередь содержит элементы {FIRST, 1.0, 2.0}, {SECOND, 3.0, 4.0}	Вернётся указатель на {FIRST, 1.0, 2.0}, длина уменьшится

Негативный: извлечение из пустой очереди	Очередь пустая	Возвращён NULL, состояние очереди не изменилось

Аналитическая часть

Все замеры происходили не ноутбуке с подключенным питанием и в режиме высокой производительности.
Характеристики ноутбука:

- ОС: Windows 11
- Оперативная память: 16 Гб DDR4
- Процессор: Intel core i5-12500H

Анализ памяти.
Предположим, что на этапе инициализации очередь рассчитан на 1000 элементов. Без учёта хранения указателя на такую структуру и структуры deleted_note_t, т.к. она нужна только для отслеживания фрагментации.
Размер структуры request_t = 24 байта
Размер queue_item_t = 32 байта

Метод хранения очереди	Формула для расчета занимаемой памяти	Количество занимаемой памяти (байт)
Массив	Sizeof(int) * 3 + sizeof(request_t) * len	24024
Список	Sizeof(int) + sizeof(queue_item_t *) * (len + 2)	32

Размер очереди, выполненного в виде массива, остается статичным, а вот размер списка динамичный и зависит от кол-ва элементов. Так что при полной заполненности ситуация становится такой:

Метод хранения	Количество занимаемой памяти (байт)
Массив	24024
Список	32068

Список становится невыгоден уже на 751 элементах

Метод хранения	Количество занимаемой памяти (байт)
Массив	24024
Список	24036

Таким образом, массив предпочтителен в системах с фиксированным объемом данных, тогда как список обеспечивает гибкость, но с увеличением накладных расходов.

Анализ времени

Представление очереди	Вид тестирования	Фактическое количество элементов	Время усреднённое (с)
Массив	Добавление	30000	0.000060
		50000	0.000080
		70000	0.000090
		90000	0.000110
		100000	0.000120
	Удаление	30000	0.000020
		50000	0.000040
		70000	0.000060
		90000	0.000070
		100000	0.000080
Список	Добавление	30000	0.001090
		50000	0.002890
		70000	0.003410
		90000	0.003780
		100000	0.003830
	Удаление	30000	0.000310
		50000	0.000560
		70000	0.000720
		90000	0.000920
		100000	0.001020

Моделирование очереди (массив)

Этап обслуживания	Текущая длина очереди	Средняя длина очереди	Кол-во вошедших заявок	Кол-во вышедших заявок	Среднее время пребывания в очереди (е.в)
100 заявок	2	9.39	101	100	23.549
200 заявок	3	6.23	202	200	16.606
300 заявок	5	6.56	304	300	17.293
400 заявок	10	7.47	409	400	19.271
500 заявок	11	8.31	510	500	21.469
600 заявок	18	9.12	617	600	23.215
700 заявок	24	10.98	723	700	27.421
800 заявок	25	12.41	824	800	30.805
900 заявок	23	13.90	922	900	34.805
1000 заявок	20	14.77	1019	1000	37.075

ИТОГОВАЯ СТАТИСТИКА

Значения временных промежутков: t1b=0.0, t1e=5.0, t2b=0.0, t2e=4.0, t3b=0.0, t3e=4.0
Общее время моделирования: 2543.408
Ожидаемое время моделирования: 2500.000
Время простоя ОА: 0.000000
Количество вошедших заявок первого типа: 1019
Количество вышедших заявок первого типа: 1000
Количество обращений заявок второго типа: 279

ПРОЦЕНТ РАССОГЛАСОВАНИЯ

Расхождение между расчетным и фактическим временем моделирования: 1.74%

Моделирование очереди (список)

Этап обслуживания	Текущая длина очереди	Средняя длина очереди	Кол-во вошедших заявок	Кол-во вышедших заявок	Среднее время пребывания в очереди (е.в.)
100 заявок	5	4.77	104	100	12.396
200 заявок	10	5.34	209	200	13.946
300 заявок	11	6.94	310	300	17.768
400 заявок	13	8.20	412	400	20.660
500 заявок	16	9.39	515	500	23.729
600 заявок	18	11.3	617	600	27.833
700 заявок	24	12.93	723	700	32.248
800 заявок	18	14.15	817	800	35.576
900 заявок	13	14.92	912	900	37.725
1000 заявок	19	15.18	1018	1000	38.586

ИТОГОВАЯ СТАТИСТИКА

Общее время моделирования: 2558.847
Ожидаемое время моделирования: 2500.000
Время простоя ОА: 0.000000
Количество вошедших заявок первого типа: 1018
Количество вышедших заявок первого типа: 1000
Количество обращений заявок второго типа: 260

ПРОЦЕНТ РАССОГЛАСОВАНИЯ

Расхождение между расчетным и фактическим временем моделирования: 2.35%

Тест моделирования очереди массивом
Время моделирования: 0.005590 с

Тест моделирования очереди списком
Время моделирования: 0.006240 с

Контрольные вопросы

1. *Что такое FIFO и LIFO?*

FIFO (First In, First Out) — это принцип организации данных, где элементы обрабатываются в том порядке, в котором они добавлены.
LIFO (Last In, First Out) — это принцип, где элементы обрабатываются в обратном порядке их добавления.

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

Массивная реализация: фиксированный размер массива выделяется заранее.
На основе связного списка: память выделяется динамически при добавлении каждого элемента.

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

Массивная реализация физически память не освобождается. Указатель смещается, освобождая место логически.
Связный список - узел удаляется с помощью функций free(). Указатель переходит на следующий элемент.

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

При просмотре очереди элементы удаляются.

5. *От чего зависит эффективность физической реализации очереди?*

Очередь на список работает медленнее и под каждый элемент требуется больше памяти, чем при хранении очереди на статическом массиве. Однако при хранении списком количество элементов в очереди ограничено только размером оперативной памяти.

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

Массивная реализация:

- Достоинства - быстрый доступ к элементам
- Недостатки - фиксированный размер

Связный список:

- Достоинства - динамическое изменение размера
- Недостатки - накладные расходы на хранение указателей, фрагментация памяти, сравнительно низкая скорость доступа к элементам.

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

Фрагментация памяти — это дробление памяти на мелкие не смежные свободные области маленького размера. Возникает после выполнения системой большого числа запросов на память, таких, что размеры

подходящих свободных участков памяти оказываются немного больше, чем требуемые.
Возникает в куче, где размещаются динамически выделенные данные.

8. Для чего нужен алгоритм «близнецов»

Алгоритм близнецов нужен, чтобы быстро выделять и освобождать память блоками размера 2^N. Если в системе нет блока подходящего размера, то берется блок большего размера и делится на 2 равные части, получая тем самым два блока “близнеца”.

9. Какие дисциплины выделения памяти вы знаете?

Дисциплина «самый подходящий». Выделяется тот свободный участок, размер которого равен запрошенному или превышает его на минимальную величину.
Дисциплина «первый подходящий». Выделяется первый же найденный свободный участок, размер которого не меньше запрошенного.

10.На что необходимо обратить внимание при тестировании программы?

При реализации кольцевого массива важно обращать внимание на правильность определения in и out значений, удалении из пустого или добавлении в заполненный массив, ошибки выделения памяти.
При реализации списка важно обращаться внимание на правильность использования указателей, не путать указатель на “голову” и “хвост”.

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

ОС хранит определенные структуры данных, которые содержат список адресов свободных и занятых адресов памяти. При поступлении запроса на выделение памяти от программы, выделенный блок памяти помечается как занятый, сохраняются метаданные, указывающие на размер блока, статус и т.д. При освобождении памяти, ранее занятый блок помечается как свободный, два рядом стоящих свободных блока могут объединится.

Заключение

Сравнение показало, что массив лучше подходит для систем с фиксированным числом элементов, требующих высокой производительности. Его статическая структура обеспечивает минимальные накладные расходы на управление данными. Список, напротив, удобен для сценариев с изменяющимся количеством элементов, но его эффективность снижается при большом объеме данных из-за дополнительных затрат на память и управление указателями.

Тем не менее моделирование обработки 1000 заявок в очереди показало, что отличие между представлением списком и массивом составляет только ~10%, несмотря на значительную разницу между скоростью выполнения базовых дейсвтий.