Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ: Информатика и системы управления
КАФЕДРА: Программное обеспечение ЭВМ и информационные технологии
ДИСЦИПЛИНА: Типы и структуры данных
ТЕМА: Обработка очередей
ВАРИАНТ: 6

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5

Студент: _____ *Княжев А. В.*
(подпись, дата)

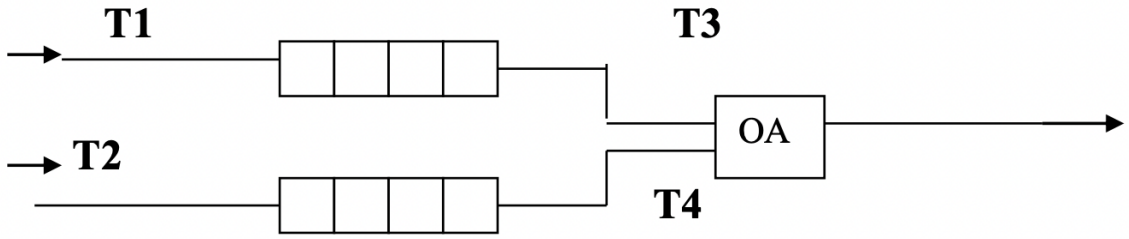
Преподаватель: _____ *Силантьева А. В.*
(подпись, дата)

2021 г.

Содержание

1 Условие задачи

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 T1 и T2, 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 T3 и T4, 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 2-го типа может войти в ОА, если в системе нет заявок 1-го типа. Если в момент обслуживания заявки 2-го типа в пустую очередь входит заявка 1-го типа, то она немедленно поступает на обслуживание; обработка заявки 2-го типа прерывается, и она возвращается в «хвост» своей очереди (система с абсолютным приоритетом и повторным обслуживанием).

Смоделировать процесс обслуживания первых 1000 заявок 1-го типа, выдавая после обслуживания каждых 100 заявок 1-го типа информацию о текущей и средней длине каждой очереди. В конце процесса выдать общее время моделирования и количестве вошедших в систему и вышедших из нее заявок обоих типов, среднем времени пребывания заявок в очереди, количестве «выброшенных» заявок второго типа. Обеспечить по требованию пользователя выдачу на экран адресов элементов очереди при удалении и добавлении элементов. Проследить, возникает ли при этом фрагментация памяти.

2 Техническое задание

Система массового обслуживания состоит из обслуживающих аппаратов (ОА) и очередей заявок двух типов, различающихся временем прихода и обработки. Заявки поступают в очереди по случайному закону с различными интервалами времени (в зависимости от варианта задания), равномерно распределенными от начального значения (иногда от нуля) до максимального количества единиц времени. В ОА заявки поступают из «головы» очереди по одной и обслуживаются за указанные в задании времена, распределенные равновероятно от минимального до максимального значений (все времена — вещественного типа).

Требуется смоделировать процесс обслуживания первых 1000 заявок первого типа, выдавая после обслуживания каждых 100 заявок первого типа информацию о текущей и средней длине каждой очереди и о среднем времени пребывания заявок каждого типа в очереди. В конце процесса необходимо выдать на экран общее время моделирования, время простоя ОА, количество вошедших в систему и вышедших из нее заявок первого и второго типов.

Очередь необходимо представить в виде вектора и списка. Все операции должны быть оформлены подпрограммами. Алгоритм для реализации задачи один, независимо от формы представления очереди. Необходимо сравнить эффективность различного представления очереди по времени выполнения программы и по требуемой памяти. При реализации очереди списком нужно проследить, каким образом происходит выделение и освобождение участков памяти, для чего по запросу пользователя необходимо выдать на экран адреса памяти, содержащие элементы очереди при добавлении или удалении очередного элемента.

Длительности обработки заявок и интервалы между их приходом (единицы времени — е.в.) — случайные равномерно распределенные числа вещественного типа в указанном диапазоне (например, от $t1$ до $t2$).

Рассогласование между средними ожидаемыми временами и временами, полученными в моделирующей программе должно быть ***не больше*** 2–3 %.

Сравнить эффективность (по памяти и по времени выполнения) обработки очереди списком и массивом.

Программа должна выводить меню с возможностью выбирать варианты. При вводе нуля на запрос варианта меню, происходит выход из программы.

2.1 Общие входные данные

∗ номер выбранного пункта меню.

2.2 Входные и выходные данные пунктов меню

2.2.1 Моделирование процесса с использованием очереди-массива

Входные данные

* два вещественных числа — максимальное и минимальное время $T1$, с которым заявки приходят в первую очередь;
* два вещественных числа — максимальное и минимальное время $T2$, с которым заявки приходят во вторую очередь;
* два вещественных числа — максимальное и минимальное время $T3$, в течение которого заявки из первой очереди обрабатываются обслуживающим аппаратом;
* два вещественных числа — максимальное и минимальное время $T4$, в течение которого заявки из второй очереди обрабатываются обслуживающим аппаратом.

Выходные данные

* для каждой 100-ой вышедшей заявки первого типа информация о текущей и средней длине каждой очереди;
* результат моделирования:
  - общее время моделирования;
  - теоретическое время моделирования;
  - сравнения этих времен, погрешность моделирования;
  - количество вошедших в систему и вышедших из нее заявок каждого типа;
  - среднее время пребывания заявок в очереди для каждого типа;
  - количество «выброшенных» заявок второго типа.

2.2.2 Моделирование процесса с использованием очереди-списка

Входные данные

* ответ на вопрос, нужно ли выводить адреса добавляемых и удаляемых элементов;
* два вещественных числа — максимальное и минимальное время $T1$, с которым заявки приходят в первую очередь;
* два вещественных числа — максимальное и минимальное время $T2$, с которым заявки приходят во вторую очередь;

* два вещественных числа — максимальное и минимальное время $T3$, в течение которого заявки из первой очереди обрабатываются обслуживающим аппаратом;
* два вещественных числа — максимальное и минимальное время $T4$, в течение которого заявки из второй очереди обрабатываются обслуживающим аппаратом.

Выходные данные

* для каждой 100-ой вышедшей заявки первого типа информация о текущей и средней длине каждой очереди;
* результат моделирования:
  - общее время моделирования;
  - теоретическое время моделирования;
  - сравнения этих времен, погрешность моделирования;
  - количество вошедших в систему и вышедших из нее заявок каждого типа;
  - среднее время пребывания заявок в очереди для каждого типа;
  - количество «выброшенных» заявок второго типа.
* опционально адреса добавляемых и удаляемых элементов.

2.2.3 Сравнение скорости работы различных реализаций очереди

Входные данные

* количество элементов, добавляемых/удаляемых в очереди;

Выходные данные

* время выполнения операций push и pop для очереди-списка и очереди-массива;
* сравнение этих времен.

2.2.4 Сравнение потребления памяти различных реализаций очереди

Входные данные

* количество элементов очереди;

Выходные данные

* объемы памяти, занимаемые очередью-списком и очередью-массивом;
* сравнение этих объемов.

2.3 Действие программы

Программа осуществляет работу со очередями в формате массива и списка.

2.4 Обращение к программе

Программа может быть запущена из командных оболочек `sh/bash/zsh/fish`, а также от IDE, способных работать с языком Си. Программа не принимает никаких аргументов. Название исполняемого файла — app.exe, так что команда может быть вызвана из командной оболочки в корневой папке проекта как:

```
$ ./app.exe
```

2.5 Аварийные ситуации

2.5.1 Общие аварийные ситуации

1. невалидный номер пункта меню (строка или пустая строка);
2. неверный номер пункта меню (такого пунта меню не существует).

2.5.2 Пункт меню №1

1. невалидные времена;
2. времена с нарушениями логики, когда минимальный промежуток больше максимального.

2.5.3 Пункт меню №2

1. невалидный ответ на вопрос о том, нужно ли выводить адреса;
2. некорретный ответ на вопрос о том, нужно ли выводить адреса;
3. невалидные времена;
4. времена с нарушениями логики, когда минимальный промежуток больше максимального.

2.5.4 Пункт меню №3

1. слишком большое количество элементов;
2. слишком маленькое количество элементов;
3. невалидное количество элементов.

2.5.5 Пункт меню №4

1. слишком большое количество элементов;
2. слишком маленькое количество элементов;
3. невалидное количество элементов.

3 Структуры данных

В данной работе используется статичесике массивы, записи и односвязные списки.

3.1 Модуль для работы с ошибками

3.1.1 Типы данных

```
/**
 * text - текст ошибки;
 * code - код ошибки;
 * func - функция, в которой случилась ошибка.
 */
typedef struct
{
    const char *text;
    int code;
    const char *func;
} error_t;
```

3.1.2 Функции

```
/**
 * Создание новой ошибки.
 * text - текст ошибки;
 * code - код ошибки;
 * func - функция, в которой случилась ошибка.
 */
error_t new_error(const char *text, int code, const char *func);
```

```
/**
 * Создание ошибки-маркера успеха.
 * func - функция, в которой был создан "успех".
 */
error_t new_success(const char *func);
```

```c
/**
 * Проверка, отображает ли ошибка ошибочную ситуацию.
 * err - исходная ошибка.
 */
bool is_failure(error_t err);
```

3.2 Модуль для работы со строками

3.2.1 Основные константы

```c
#define MYSTRING_SIZE 257
```

3.2.2 Типы данных

```c
/**
 * Тип строки длиной 256 символов + терминальный нуль.
 */
typedef char mystring_t[MYSTRING_SIZE];
```

3.2.3 Функции

```c
/**
 * Чтение строки из файла.
 * f - файл;
 * str - строка, в которую считываем.
 */
error_t f_read_line(FILE *f, mystring_t str);
```

```c
/**
 * Чтение целого числа из стандартного потока ввода.
 * n - считываемое число.
 */
error_t read_int(int *n);
```

3.3 Модуль «ядра» программы

3.3.1 Функции

```
/**
 * Запуск основного диалога программы.
 */
error_t eng_work();
```

3.4 Модуль для тестирования скорости работы

3.4.1 Основные константы

```
/**
 * Количество запусков тестов производительности.
 */
#define BENCH_NUM_OF_RUNS 1000
```

3.4.2 Функции

```
/**
 * Замер времени операции "push" для очереди-массива.
 * n - количество элементов очереди;
 * timer - среднее время выполнения операции.
 */
error_t bench_queue_arr_push(size_t n, long long *timer);
```

```
/**
 * Замер времени операции "pop" для очереди-массива.
 * n - количество элементов очереди;
 * timer - среднее время выполнения операции.
 */
error_t bench_queue_arr_pop(size_t n, long long *timer);
```

```
/**
 * Замер времени операции "push" для очереди-списка.
 * n - количество элементов очереди;
 * timer - среднее время выполнения операции.
 */
error_t bench_queue_list_push(size_t n, long long *timer);
```

```
/**
 * Замер времени операции "pop" для очереди-списка.
 * n - количество элементов очереди;
 * timer - среднее время выполнения операции.
 */
error_t bench_queue_list_pop(size_t n, long long *timer);
```

3.5 Модуль для работы с очередью-массивом

3.5.1 Основные константы

```
/**
 * Максимальное количество элементов очереди-массива.
 */
#define QA_QUEUE_SIZE 20000
```

3.5.2 Типы данных

```
/**
 * Очередь-массив.
 * content - содержимое очереди;
 * head - указатель на голову очереди, то есть на адрес
 *        удаляемого элемента;
 * tail - указатель на хвост очереди, то есть на адрес
 *        куда будет вставляться очередной элемент;
 * len - длина очереди.
 */
typedef struct queue_array_t
{
    task_t content[QA_QUEUE_SIZE];
    task_t *head;
    task_t *tail;
    size_t len;
} queue_array_t;
```

3.5.3 Функции

```
/**
 * Создание пустой очереди.
 */
queue_array_t qa_create();
```

```
/**
 * Добавление элемента в очередь.
 * q - очередь;
 * el - добавляемый элемент.
 */
error_t qa_push(void *q, task_t el);
```

```
/**
 * Удаление элемента из очереди.
 * q - очередь;
 * el - удаленный элемент.
 */
error_t qa_pop(void *q, task_t *el);
```

```
/**
 * Проверка очереди на пустоту.
 * q - очередь.
 */
bool qa_empty(void *q);
```

```
/**
 * Получение длины очереди.
 * q - очередь;
 * len - длина очереи.
 */
error_t qa_len(void *q, size_t *len);
```

3.6 Модуль для работы с очередью-списком

3.6.1 Типы данных

```
/**
 * Элемент очереди-списка.
 * content - содержимое элемента очереди;
 * next - указатель на следующий элемент очереди.
 */
typedef struct queue_node_t queue_node_t;
struct queue_node_t
{
    task_t content;
    queue_node_t *next;
};
```

```
/**
 * Очередь-список.
 * head - указатель на голову очереди (откуда удаляем);
 * tail - указатель на хвост очереди (после которого добавляем);
 * len - длина очереди.
 */
typedef struct queue_list_t
{
    queue_node_t *head;
    queue_node_t *tail;
    size_t len;
} queue_list_t;
```

3.6.2 Функции

```
/**
 * Создание пустой очереди.
 */
queue_list_t ql_create();
```

```c
/**
 * Добавление элемента в очередь.
 * q - очередь;
 * el - добавляемый элемент.
 */
error_t ql_push(void *q, task_t el);
```

```c
/**
 * Удаление элемента из очереди.
 * q - очередь;
 * el - удаленный элемент.
 */
error_t ql_pop(void *q, task_t *el);
```

```c
/**
 * Проверка очереди на пустоту.
 * q - очередь.
 */
bool ql_empty(void *q);
```

```c
/**
 * Освобождение очереди-списка.
 * q - очередь.
 */
void ql_free(void *q);
```

3.7 Модуль для работы с обрабатывающий автоматом

3.7.1 Основные константы

```c

```

3.7.2 Типы данных

```
/**
 * Тип заявки.
 * primary - первичная, из первой очереди;
 * secondary - вторичная, из второй очереди.
 */
typedef enum task_type_t
{
    primary,
    secondary,
} task_type_t;
```

```
/**
 * Заявка.
 * content - содержимое заявки;
 * type - типа заявки;
 * enter_time - время приходя заявки в очередь.
 */
typedef struct task_t
{
    int content;
    task_type_t type;
    double enter_time;
} task_t;
```

```c
/**
 * Обрабатывающий автомат.
 * current_task - текущая заявка;
 * is_working - работает ли автомат?
 * start_time - время начала обработки текущей заявки;
 * process_min_time - минимальное время обработки заявок каждого типа;
 * process_min_time - максимальное время обработки заявок каждого типа.
 */
typedef struct oa_t
{
    task_t current_task;
    bool is_working;
    double start_time;
    double process_time;
    double process_min_time[2], process_max_time[2];
} oa_t;
```

3.7.3 Функции

```c
/**
 * Добавление заявки в автомат.
 * oa - обрабатывающий автомат;
 * task - заявка;
 * timer - текущее время.
 */
error_t oa_start(oa_t *oa, task_t task, double timer);
```

```c
/**
 * Окончание работы автомата над текущей заявкой.
 * oa - обрабатывающий автомат;
 * exited - вышедшая заявка.
 */
error_t oa_finish(oa_t *oa, task_t *exited);
```

```
/**
 * Пришло ли время аппарату оканчивать работу над заявкой?
 * oa - обрабатывающий автомат;
 * timer - текущее время.
 */
bool oa_done(oa_t *oa, double timer);
```

```
/**
 * Время, когда автомат закончит работать над текущей заявкой.
 * oa - обрабатывающий автомат.
 */
double oa_end_time(oa_t *oa);
```

```
/**
 * Новая заявка.
 * type - тип заявки;
 * timer - текущее время.
 */
task_t task_new(task_type_t type, double timer);
```

3.8 Модуль для моделирования процесса

3.8.1 Основные константы

```
/**
 * Количество обработанных заявок первого типа для выхода.
 */
#define PROCESSED_STOP 1000
```

```
/**
 * Количество обработанных заявок первого типа для вывода статистики.
 */
#define PROCESSED_STATS 100
```

3.8.2 Типы данных

```
/**
 * Параметр моделирования.
 * push - функция "push" для очереди;
 * pop - функция "pop" для очереди;
 * len - функция "len" для очереди;
 * empty - проверка очереди на пустоту;
 * q1 - первая очередь;
 * q2 - вторая очередь;
 * t1_min, t1_max - диапазон времени на T1;
 * t2_min, t2_max - диапазон времени на T2;
 * t3_min, t3_max - диапазон времени на T3;
 * t4_min, t4_max - диапазон времени на T4.
 */
typedef struct process_config_t
{
    error_t (*push) (void *, task_t);
    error_t (*pop) (void *, task_t *);
    error_t (*len) (void *, size_t *);
    bool (*empty) (void *);

    void *q1;
    void *q2;

    double t1_min, t1_max;
    double t2_min, t2_max;
    double t3_min, t3_max;
    double t4_min, t4_max;
} process_config_t;
```

3.8.3 Функции

```
/**
 * Основной цикл моделирования процесса.
 * config - настройки моделирования.
 */
error_t process(process_config_t config);
```

4 Описания алгоритмов

1. вывод главного меню;
2. получение у пользователя номера пункта меню:

4.1 Пункт меню №1

1. получить у пользователя времена;
2. добавить их в параметры запуска моделирования;
3. для текущего времени проверяется, какое событие может сейчас произойти: добавление в одну из очередей или окончание работы автомата;
4. в зависимости от загруженности автомата после обработки добавлений происходит проверка занятости автомата;
5. если он пуст, в него кладутся заявки по приоритетности;
6. если не пуст, но там есть заявка второго типа, а в очереди есть заявки первого типа, то заявка первого типа вымещает ту, что в автомате, которая возвращается во вторую очередь;
7. вывод подробной статистики моделирования.

4.2 Пункт меню №2

1. см. предыдущий пункт.

4.3 Пункт меню №3

1. получить у пользователя количество элементов;
2. для каждого вида очереди посчитать среднюю скорость добавления и удаления введенного количества элементов и вывести на экран;
3. вывести процентное превосходство скорости работы очереди-массива над очередью-списоком.

4.4 Пункт меню №4

1. получить у пользователя количество элементов;
2. для каждого вида очереди посчитать количество памяти, затрачиваемое для хранения введенного количества элементов;

3. вывести процентное превосходство объема занимаемой памяти очереди-списка над очередью-массивом.

5 Тестирование

Для проверок корректности работы программы было проведено функциональное тестирование. Таблица с тестовыми данными для «позитивных» и «негативных» случаев приведена ниже.

Так как входных/выходных данных может быть очень много, то в соответствующих полях могут быть указаны наиболее значимые части.

5.1 «Негативные» тесты

5.1.1 Ввод невалидного пункта меню

Входные данные: abc
T1: —
T2: —
T3: —
T4: —
Результат: 104: Строка не является корректным числом (to_integer)

5.1.2 Ввод некорректного пункта меню

Входные данные: 12
T1: —
T2: —
T3: —
T4: —
Результат: 161: введен неверный пункт меню (main_menu_dialog)

5.1.3 Невалидное время

Входные данные: 1 или 2
T1: abc 3
T2: —
T3: —
T4: —
Результат: 161: неверный диапазон T1 (read_config_times_dialog)

5.1.4 Меньшая граница времени больше большей

Входные данные: 1 или 2
T1: 1 3
T2: 3 1

T3: —

T4: —

Результат: 161: неверный диапазон T2 (read_config_times_dialog)

5.1.5 Ввод невалидного количества элементов

Входные данные: 3 → abc или 4 → a

T1: —

T2: —

T3: —

T4: —

Результат: 104: строка не является корректным числом (to$_i$*nteger*)

5.1.6 Ввод неверного количества элементов

Входные данные: 3 → 1000000 или 4 → 1000000

T1: —

T2: —

T3: —

T4: —

Результат: 100: слишком большой размер (eng_work)

5.2 «Позитивные» тесты

5.2.1 Моделирование с параметрами из задания для массива

Входные данные: 1

T1: 1 5

T2: 0 3

T3: 0 4

T4: 0 1

Результат:

```
Работа системы завершена.
Общее время моделирования = 3064.6
Расчетное время моделирования = 3000.0
Расхождение = 2.1 %

Информация об очередях
ОЧЕРЕДЬ 1:
  Количество вошедших заявок = 1000
  Количество вышедших заявок = 1000
  Среднее время пребывания заявки в очереди = 0.0
ОЧЕРЕДЬ 2:
  Количество вошедших заявок = 2006
  Количество вышедших заявок = 1958
  Среднее время пребывания заявки в очереди = 0.0
  Количество "выброшенных" заявок = 635
```

5.2.2 Моделирование с параметрами из задания для списка

Входные данные: $2 \to 2$

T1: 1 5

T2: 0 3

T3: 0 4

T4: 0 1

Результат:

```
Работа системы завершена.
Общее время моделирования = 2959.8
Расчетное время моделирования = 3000.0
Расхождение = 1.3 %

Информация об очередях
ОЧЕРЕДЬ 1:
  Количество вошедших заявок = 1000
  Количество вышедших заявок = 1000
  Среднее время пребывания заявки в очереди = 0.0
ОЧЕРЕДЬ 2:
  Количество вошедших заявок = 1976
  Количество вышедших заявок = 1756
  Среднее время пребывания заявки в очереди = 0.1
  Количество "выброшенных" заявок = 590
```

5.2.3 Заявки в первую очередь поступают очень часто

Входные данные: 1

T1: 0 1

T2: 1 6

T3: 0 3

T4: 0 3

Результат:

```
Работа системы завершена.
Общее время моделирования = 1513.6
Расчетное время моделирования = 1500.0
Расхождение = 0.9 %

Информация об очередях
ОЧЕРЕДЬ 1:
  Количество вошедших заявок = 3016
  Количество вышедших заявок = 1000
  Среднее время пребывания заявки в очереди = 1.0
ОЧЕРЕДЬ 2:
  Количество вошедших заявок = 427
  Количество вышедших заявок = 0
  Среднее время пребывания заявки в очереди = 0.0
  Количество "выброшенных" заявок = 0
```

5.2.4 Автомат очень быстро обрабатывает заявки

Входные данные: 1

T1: 0 10

T2: 0 10

T3: 0 1

T4: 0 1

Результат:

```
Работа системы завершена.
Общее время моделирования = 5015.7
Расчетное время моделирования = 5000.0
Расхождение = 0.3 %

Информация об очередях
ОЧЕРЕДЬ 1:
  Количество вошедших заявок = 1000
  Количество вышедших заявок = 1000
  Среднее время пребывания заявки в очереди = 0.0
ОЧЕРЕДЬ 2:
  Количество вошедших заявок = 992
  Количество вышедших заявок = 992
  Среднее время пребывания заявки в очереди = 0.0
  Количество "выброшенных" заявок = 101
```

5.2.5 Все времена одинаковы

Входные данные: 1

T1: 0 4

T2: 0 4

T3: 0 4

T4: 0 4

Результат:

```
Работа системы завершена.
Общее время моделирования = 2049.1
Расчетное время моделирования = 2000.0
Расхождение = 2.4 %

Информация об очередях
ОЧЕРЕДЬ 1:
  Количество вошедших заявок = 1019
  Количество вышедших заявок = 1000
  Среднее время пребывания заявки в очереди = 0.0
ОЧЕРЕДЬ 2:
  Количество вошедших заявок = 1049
  Количество вышедших заявок = 5
  Среднее время пребывания заявки в очереди = 20.1
  Количество "выброшенных" заявок = 23
```

5.2.6 Заявки в первой очереди приходят и обрабатываются очень быстро

Входные данные: 1

T1: 0 1

T2: 0 10

T3: 0 1

T4: 0 10

Результат:

```
Работа системы завершена.
Общее время моделирования = 511.6
Расчетное время моделирования = 500.0
Расхождение = 2.3 %

Информация об очередях
ОЧЕРЕДЬ 1:
  Количество вошедших заявок = 1023
  Количество вышедших заявок = 1000
  Среднее время пребывания заявки в очереди = 0.0
ОЧЕРЕДЬ 2:
  Количество вошедших заявок = 103
  Количество вышедших заявок = 0
  Среднее время пребывания заявки в очереди = 0.2
  Количество "выброшенных" заявок = 1
```

5.2.7 Заявки в первой очереди приходят и обрабатываются очень медленно

Входные данные: 1

T1: 0 10

T2: 0 1

T3: 0 10

T4: 0 1

Результат:

```
Работа системы завершена.
Общее время моделирования = 5010.5
Расчетное время моделирования = 5000.0
Расхождение = 0.2 %

Информация об очередях
ОЧЕРЕДЬ 1:
  Количество вошедших заявок = 1006
  Количество вышедших заявок = 1000
  Среднее время пребывания заявки в очереди = 0.0
ОЧЕРЕДЬ 2:
  Количество вошедших заявок = 9990
  Количество вышедших заявок = 25
  Среднее время пребывания заявки в очереди = 58.7
  Количество "выброшенных" заявок = 8
```

5.2.8 Сравнение скорости работы очереди-массива и очереди-списка

Входные данные: $3 \to 6000$

T1: —

T2: —

T3: —

T4: —

Результат:

```
Для 1000 запусков и 6000 элементов.
Очередь-массив
  PUSH: 97 us
  POP: 79 us
Очередь-список
  PUSH: 145 us
  POP: 146 us

Преимущество очереди-массива над очередью-списком
  PUSH: 49 %
  POP: 85 %
```

5.2.9 Сравнение потребления памяти очереди-массива и очереди-списка

Входные данные: $4 \to 6000$

T1: —

T2: —

T3: —

T4: —

Результат:

```
Для 6000 элементов.
Очередь-массив
  Объем занимаемой памяти: 320024 B

Очередь-список
  Объем занимаемой памяти: 144024 B

Преимущество очереди-списка над очередью-массивом
  Объем памяти: 122 %
```

6 Оценка эффективности

6.1 Объем занимаемой памяти

Размер	**Список, Б**	**Массив, Б**	**Преимущество списка, %**
200	4824	320024	6534
400	9624	320024	3225
600	14424	320024	2119
800	19224	320024	1565
1000	24024	320024	1232
2000	48024	320024	566
4000	96024	320024	233
8000	192024	320024	67
13300	**319224**	**320023**	**0**
20000	480024	320024	-33

6.2 Скорость операций

6.2.1 Push

Размер	Массив, мкс	Список, мкс	Преимущество массива, %
200	7	7	0
400	13	11	-15
600	18	15	-17
800	23	18	-22
1000	27	23	-15
2000	43	47	9
4000	70	94	34
8000	122	192	57
20000	280	487	74

6.2.2 Pop

Размер	Массив, мкс	Список, мкс	Преимущество массива, %
200	5	5	0
400	9	9	0
600	11	14	27
800	13	18	38
1000	15	23	53
2000	27	61	126
4000	38	96	153
8000	106	195	84
20000	269	492	83

7 Контрольные вопросы

7.1 Что такое FIFO и LIFO?

FIFO (first in, first out) — принцип работы, при котором элементы, помещенные в некоторую структуру первыми, первыми и будут оттуда извлечены. Этот принцип лежит в основе очереди.

LIFO (last in, first out) — принцип работы, при котором элементы, помещенные в некоторую структуру последними, будут извлечены оттуда первыми. Этот принцип лежит в основе стека.

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

7.2.1 Реализация в виде массива

При реализации в виде статического массива, элементы очереди располагаются в статическом сегменте памяти. Под хранение такой очереди выделяется $2 \times sizeof(type*) + sizeof(size\_t) + N \times sizeof(type)$ памяти, где N — фиксированное максимальное количество элементов очереди.

7.2.2 Реализация в виде списка

При реализации в виде списка, элементы очереди располагаются в куче, и размер не должен быть известен заранее. Под хранение такой очереди выделяется $2 \times sizeof(node*) + sizeof(size\_t) + n \times sizeof(node)$ памяти, где n — количество элементов очереди, $node$ — узел списка. Под один узел выделяется $sizeof(node*) + sizeof(int)$ памяти.

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

7.3.1 Реализация в виде массива

При реализации в виде статического массива, элементы очереди при удалении не освобождаются, происходит лишь сдвиг указателя с удаляемого элемента на следующий элемент очереди.

7.3.2 Реализация в виде списка

При реализации в виде списка, элементы очереди при удалении освобождаются.

7.4 Что происходит с элементами очереди при ее просмотре?

Они удаляются.

7.5 От чего зависит эффективность физической реализации очереди?

Эффективность физической реализации очереди зависит от частоты добавления/удаления элементов и изменения размеров очереди.

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

7.6.1 Реализация в виде массива

+ высокая скорость;
+ простота управления памятью;
− относительная сложность реализации;
− так как массив неразрывен, то размер блока очереди физически более ограничен;
− неэффективное потребление памяти, когда выделено сильно больше, чем используется.

7.6.2 Реализация в виде списка

+ относительно высокая скорость;
+ более эффективное потребление памяти;
+ простота и удобство обработки;
− необходимость ручного освобождения;
− медленнее, чем реализация массивом;
− если размер близок к граничным размерам массива, то становится неэффективной по памяти.

7.7 Что такое фрагментация памяти?

Фрагментация — дробление памяти на несмежные свободные фрагменты.

7.8 Для чего нужен алгоритм «близнецов»?

За счет распределения свободных блоков по степеням двойки, поиск области памяти нужного размера осуществляется гораздо быстрее. Если найден блок в два и более раз большего размера, то он просто делится надвое, то есть блоки всегда остаются распределенными по степеням двойки. При возникновении подходящих блоков, они могут быть слиты в один большой блок размером степени двойки.

Таким образом, алгоритм «близнецов» позволяет быстрее искать нужные области памяти, а также оптимально объединять разделенные блоки памяти.

7.9 Какие дисциплины выделения памяти вы знаете?

* «первый подходящий» — выбирается первый подходящий по размеру блок. Соответственно, он может быть сильно больше необходимого размера;

* «самый подходящий» — выбирается либо блок такого же размера, какого было запрошено, либо превышающий его на минимальную величину.

7.10 На что необходимо обратить внимание при тестировании программы?

На работу с динамическими ресурсами, переполнение очереди и фрагментацию памяти.

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

При динамическом распределении памяти объекты размещаются в «куче».

При формировании объекта указывается размер запрашиваемой под объект памяти, и, в случае успеха, выделенная область памяти, становится недоступной при последующих операциях выделения памяти.

Освобождаемая память возвращается в «кучу» и становится доступной при дальнейших операциях выделения памяти.

8 Вывод

Реализация очереди в виде списка эффективнее по памяти, когда заполненность меньше того размеры, который выделен под статический массив. В то же время, при большой заполненности очереди, массив оказывается эффективнее. При выделенном объеме для статического массива в 20000 элементов, преимущество по памяти для списка упало с 6534% при размере очереди 200 элементов, до нуля при 13300 элементах в очереди.

Стоит отметить, что выделение памяти под список более гарантированно, так как проще найти много маленьких блоков памяти, чем один большой.

При тестировании не было обнаружено фрагментации памяти, и это хорошо, так как у фрагментации нет преимуществ, дефрагментация может позволить быстрее получать доступ к данным. Однако, для пользователя разница в работе с фрагментированными и дефрагментированными данными вряд ли будет наблюдаться, так как операционная система предоставляет необходимый интерфейс.

В плане скорости, реализация очереди массивом показала преимущество относительно реализации списком до 74% по операции «push», и до 153% по операции «pop». Из-за накладных расходов и особенностей архитектуры компьютера, на котором производилось тестирование, данные немного смазаны: именно поэтому на небольших данных список оказывается ощутимо быстрее массива на операции «push».

Список литературы

[1] Методические рекомендации по лабораторной работе №5 (*http://wwwcdl.bmstu.ru/*)