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»

Лабораторная Работа №5 «Обработка очередей»

Вариант №1

Студент **Поляков Андрей Игоревич**

Группа **ИУ7-32Б**

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

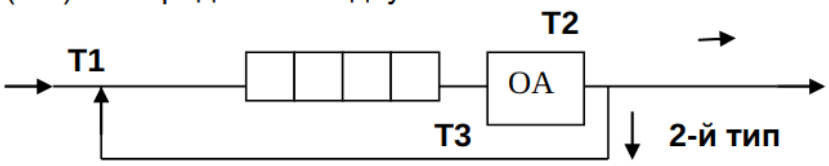
Студент **Поляков А.И.**

Проверяющий

Оценка _____

Описание условия задачи

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)

Техническое задание

Исходные данные:

Пункты меню, выраженные целыми числами 0-4. Внутри некоторых пунктов вводятся числа.

0. Выход

1. Запустить симуляцию для очереди реализованной массивом

2. Запустить симуляцию для очереди реализованной списком

3. Изменить параметры

4. Оценка эффективности программы

Результат:

Результат симуляции, измененные параметры, результат оценки эффективности

Описание задачи:

Смоделировать процесс обслуживания очередей, реализованных массивом и списком, оценить эффективность алгоритма моделирования

Способ обращения к программе:

Запуск с помощью ./app.exe

Аварийные ситуации и ошибки:

1. Ошибка выделения памяти
2. При изменении параметров введено не число
3. Введен некорректный пункт меню

Описание внутренних структур данных

Реализация с помощью статического массива:

```
typedef enum types {T1, T2} types;

struct node_t
{
    types data;
    struct node_t *next;
};

struct list_queue_type
{
    struct node_t *head;
    struct node_t *tail;
    size_t size;
};
```

struct node_t: узел списка

1. data – тип заявки
2. next – указатель на следующий узел

struct list_queue_type : очередь через список

1. head – голова очереди
2. tail – конец очереди
3. size – размер очереди

```
typedef struct
{
    types *array;
    size_t size;
    size_t front;
    size_t back;
    size_t capacity;
} arr_queue_type;
```

arr_queue_type : очередь через массив

1. array – массив
2. size – размер очереди
3. front – голова очереди
4. back – конец очереди
5. capacity – размер выделенной памяти

Описание алгоритма

Исходя из изначальных данных, таких как количество заявок первого типа, времена обработки и прихода для обоих типов заявок, а также позиция в очереди для заявок второго типа, мы производим расчет теоретических параметров. Затем запускаем цикл, на каждом этапе которого выводятся промежуточные результаты. Внутри цикла происходит генерация времени прихода и обработки, их суммирование, а также возможные операции, такие как добавление заявки первого типа, перенаправление заявки второго типа (позиция выбирается случайным образом). Также проводится анализ состояния очереди, среднего времени обработки, количества обработанных заявок первого и второго типов. По завершении цикла подсчитывается количество поступивших и обработанных заявок первого типа, количество обработанных заявок второго типа, практическое время простоя и общее время. Также рассчитывается и выводится погрешность.

Тестовые данные

Негативные тесты

Тест	Входные данные	Результат
Некорректный пункт меню	123	Неверный выбор. Попробуйте снова.
Отрицательное число при изменении параметров	-3	Ошибка ввода
Ноль при изменении параметров	0	Ошибка ввода
Не число при изменении параметров	ab	Ошибка ввода

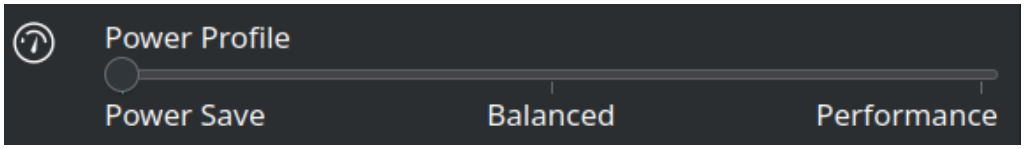
Отслеживание фрагментации

При реализации очереди с помощью списка есть вероятность возникновения фрагментации памяти. Для того чтобы за этим проследить я записывал в файл log.txt адреса выделенной и освобожденной памяти на протяжении симуляции с помощью списка:

Выберу параметры так, чтобы было проще отследить фрагментацию

```
Выберите действие: 3
T1:
От: 5
До: 6
T2:
От: 1
До: 2
T3:
От: 1
До: 2
Количество заявок: 10
```

Запущу программу, когда машина находится в режиме энергосбережения

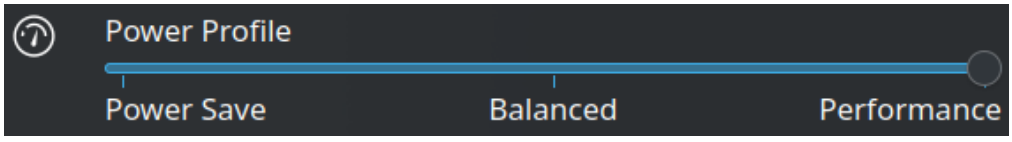


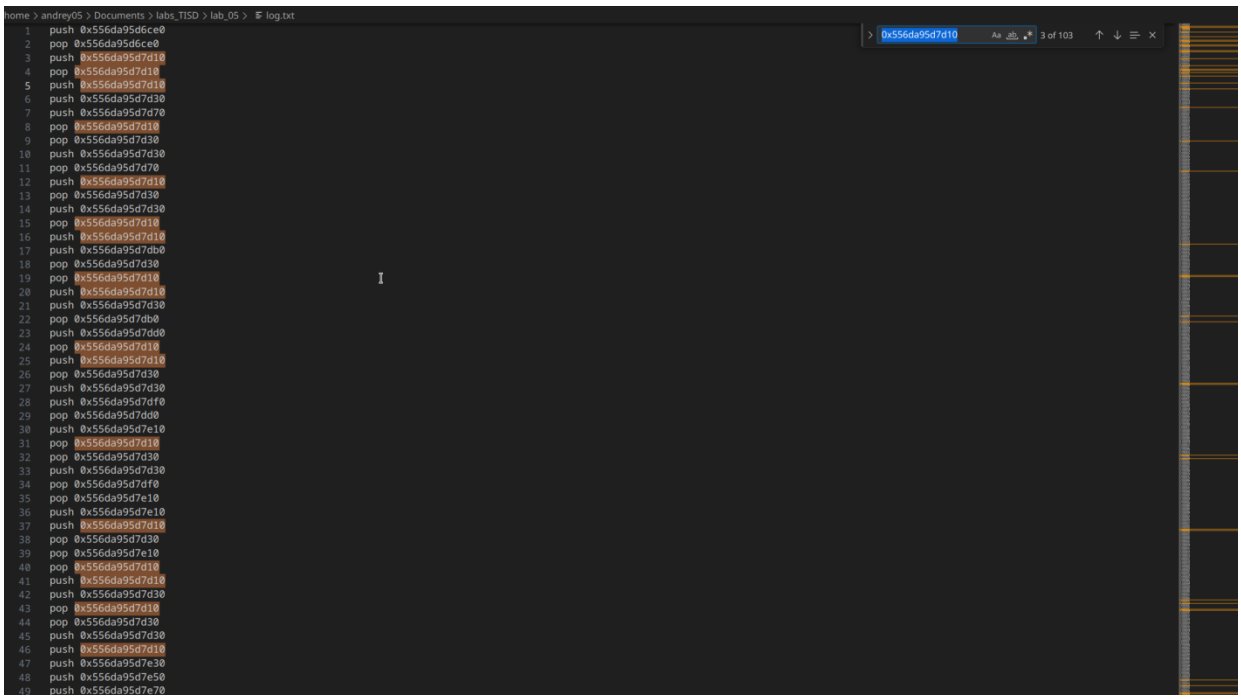
Можно отследить, что по ходу работы программы некоторые адреса, используются по несколько раз, однако программа вскоре переходит к адресам, которые находятся дальше в памяти и не использует прошлые освобожденные участки

```
push 0x561530968a30
pop 0x561530968a30
push 0x5615309680d0
pop 0x5615309680d0
push 0x561530968570
pop 0x561530968570
push 0x561530968570
push 0x5615309687d0
pop 0x561530968570
pop 0x5615309687d0
push 0x561530968570
pop 0x561530968570
push 0x561530966ca0
pop 0x561530966ca0
push 0x561530966ca0
push 0x5615309680b0
pop 0x561530966ca0
pop 0x5615309680b0
push 0x561530966ca0
pop 0x561530966ca0
push 0x561530966ca0
push 0x561530967d10
pop 0x561530966ca0
pop 0x561530967d10
push 0x561530966ca0
pop 0x561530966ca0
push 0x5615309687f0
pop 0x5615309687f0
push 0x5615309687f0
push 0x561530968830
pop 0x5615309687f0
pop 0x561530968830
push 0x5615309687f0
pop 0x5615309687f0
push 0x5615309680f0
pop 0x5615309680f0
push 0x5615309680f0
push 0x561530968130
pop 0x5615309680f0
pop 0x561530968130
push 0x5615309680f0
pop 0x5615309680f0
push 0x561530968150
```

Из этого можно сделать вывод, что при работе программы возникает фрагментация

Теперь запущу ту же симуляцию в режиме максимальной производительности





Теперь можно отследить, что пробелы в памяти не возникают, т.е. отсутствует фрагментация.

Работа программы

По условию:

T1 – Время поступления заявок 1 типа в очередь – от 0 до 5 е.в., <T1>=2.5

T2 – Время обработки заявок 1 типа – от 0 до 4 е.в., <T2>=2

T3 – Время обработки заявок 2 типа – от 0 до 4 е.в., <T3>=2

Количество обработанных заявок 1-ого типа – N=1000

Так как <T1> больше чем <T2>, время симуляции определяется временем поступления заявок:

Общее время работы и количество вошедших заявокЖ

```
e_now_time = e_arr_time * num_req;
e_in_tasks = e_now_time / e_arr_time < num_req ? num_req : e_now_time / e_arr_time;
```

Если <T1> меньше чем <T2>

```
e_now_time = (e_serv_time + e_serv2_time / 3) * num_req;
e_in_tasks = e_now_time / e_arr_time < num_req ? num_req : e_now_time / e_arr_time;
```

Результат работы программы:

```
Время моделирования: 2710.764064 (ожидаемое: 2750.000000, погрешность: 1.426761%)
Время простоя автомата: 0.000000 (ожидаемое: 0.000000, погрешность: 0.000000%)
Число вошедших заявок: 1093 (ожидаемое: 1100, погрешность: 0.636364%)
Число вышедших заявок: 1000 (ожидаемое: 1000, погрешность: 0.000000%)
Количество обращений заявок второго типа: 336
```

Как видно, все погрешности находятся в пределе 2%

Замеры

Программа сравнивает время симуляции системы обслуживания. Для каждого из тестов программа выполняет многочисленные замеры, пока их RSE не становится <5. Затем все тестовые случаи сравниваются и оценивается их эффективность.

Для 10 заявок:

Реализация с помощью массива:

```
|  Время, нс  | Кол-во итераций | RSE
|    11237.60 |              10 | 3.94
```

Занимаемая память - 64 байт

Реализация с помощью списка:

```
|  Время, нс  | Кол-во итераций | RSE
|    10657.70 |              10 | 2.65
```

Занимаемая память - 72 байт

Разность производительности = 579.90 нс

Реализация с помощью массива дольше на 5.441136 %

Разность занимаемой памяти = 8 байт

Реализация с помощью списка больше на 12 %

Для 100 заявок:

Реализация с помощью массива:

Время, нс	Кол-во итераций	RSE
19380.80	10	4.01

Занимаемая память - 136 байт

Реализация с помощью списка:

Время, нс	Кол-во итераций	RSE
14387.20	10	4.11

Занимаемая память - 216 байт

Разность производительности = 4993.60 нс

Реализация с помощью массива дольше на 34.708630 %

Разность занимаемой памяти = 80 байт

Реализация с помощью списка больше на 58 %

Для 300 заявок:

Реализация с помощью массива:

Время, нс	Кол-во итераций	RSE
55303.75	20	4.01

Занимаемая память - 124 байт

Реализация с помощью списка:

Время, нс	Кол-во итераций	RSE
44433.00	10	3.03

Занимаемая память - 120 байт

Разность производительности = 10870.75 нс

Реализация с помощью массива дольше на 24.465487 %

Разность занимаемой памяти = 4 байт

Реализация с помощью массива больше на 3 %

Для 1000 заявок:

```
Реализация с помощью массива:
|  Время, нс  | Кол-во итераций | RSE
|   136271.56 |              50 | 4.23
```

Занимаемая память - 520 байт

Реализация с помощью списка:

```
|  Время, нс  | Кол-во итераций | RSE
|    50621.00 |              10 | 2.05
```

Занимаемая память - 184 байт

Разность производительности = 85650.56 нс

Реализация с помощью массива дольше на 169.199660 %

Разность занимаемой памяти = 336 байт

Реализация с помощью массива больше на 182 %

Можно отследить, что по времени реализация с помощью списка всегда выигрывает, но по памяти массив выигрывает примерно до 300 заявок, это может быть связано с тем, что массив реализован динамически и при выделении дополнительной памяти, она выделяется с большим запасом.

Ответы на вопросы

1. FIFO и LIFO:

- FIFO (First-In-First-Out): Это метод управления данными, при котором элемент, добавленный первым, обрабатывается первым, а последний добавленный элемент обрабатывается последним.

- LIFO (Last-In-First-Out): В этом методе последний добавленный элемент обрабатывается первым, а первый добавленный элемент обрабатывается последним.

2. Выделение памяти для очереди:
- FIFO: В реализации FIFO используется структура данных "очередь". Обычно используется динамическое выделение памяти для хранения элементов очереди. Объем памяти зависит от количества элементов в очереди.
- LIFO: LIFO часто реализуется с использованием стека. Память также выделяется динамически, и объем зависит от количества элементов в стеке.

3. Освобождение памяти при удалении элемента:
- FIFO и LIFO: Память освобождается при удалении элемента путем освобождения выделенного под него блока памяти.

4. Просмотр элементов очереди:
- FIFO: Элементы просматриваются в порядке их добавления, начиная с первого.
- LIFO: Элементы просматриваются в порядке обратном их добавлению, начиная с последнего.

5. Эффективность физической реализации очереди зависит от:
- Типа операций: Например, операции вставки и удаления.
- Структуры данных: Выбор подходящей структуры данных для конкретной задачи.
- Алгоритмов: Эффективность используемых алгоритмов в реализации.

6. Достоинства и недостатки различных реализаций:
- FIFO: Подходит для моделирования реальных очередей, но неэффективен при частых удалениях элементов из середины.
- LIFO: Эффективен при частых добавлениях и удалениях с конца, но не подходит для моделирования реальных очередей.

7. Фрагментация памяти:
- Фрагментация памяти: Это явление, когда доступная память разбивается на мелкие фрагменты, которые не могут быть эффективно использованы.

8. Алгоритм "близнецов":
- Алгоритм "близнецов": Используется для уменьшения фрагментации памяти путем слияния смежных свободных блоков.

9. Дисциплины выделения памяти:
- First Fit: Выделяет первый подходящий блок памяти.

- Best Fit: Выделяет наименьший подходящий блок памяти.
- Worst Fit: Выделяет наибольший подходящий блок памяти.

10. Тестирование программы:
- Корректность: Проверка правильности выполнения программы.
- Эффективность: Оценка производительности и использования ресурсов.
- Устойчивость: Проверка на устойчивость к ошибкам и непредвиденным ситуациям.

11. Выделение и освобождение памяти при динамических запросах:
- Выделение памяти: Обычно выполняется с помощью функций, таких как malloc
- Освобождение памяти: С использованием функций типа free

Вывод

Итак, использование очереди в программировании значительно упрощает управление данными и обеспечивает последовательную и организованную обработку задач. Для повышения производительности стоит выбрать представление с помощью массива. Если важны удобство и неограниченный размер, то более предпочтительным будет использование реализации в виде списка.