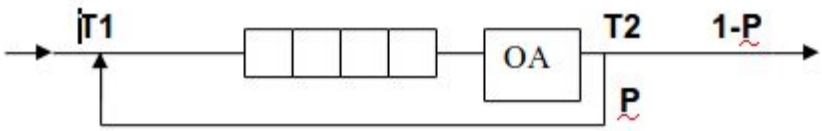
Лабораторная работа №4
по дисциплине «Типы и структуры данных»
Тема: Обработка очередей.

Горохова Ирина
ИУ7-31

Условие задачи:

Система массового обслуживания состоит из обслуживающего аппарата (ОА) и очереди заявок:



Заявки поступают в "хвост" очереди по случайному закону с интервалом времени T1, равномерно распределенным от 0 до 6 единиц времени (е.в.). В ОА они поступают из "головы" очереди по одной и обслуживаются также равновероятно за время T2 от 0 до 1 е.в., Каждая заявка после ОА с вероятностью P=0.8 вновь поступает в "хвост" очереди, совершая новый цикл обслуживания, а с вероятностью 1-P покидает систему. (Все времена – вещественного типа). В начале процесса в системе заявок нет.

Смоделировать процесс обслуживания до ухода из системы первых 1000 заявок. Выдавать после обслуживания каждых 100 заявок информацию о текущей и средней длине очереди. В конце процесса выдать общее время моделирования и количество вошедших в систему и вышедших из нее заявок, среднее время пребывания заявки в очереди, время простоя аппарата, количество срабатываний ОА. Обеспечить по требованию пользователя выдачу на экран адресов элементов очереди при удалении и добавлении элементов. Проследить, возникает ли при этом фрагментация памяти.

Входные данные:

Целое число - номер выбранного пункта меню работы с программой. Вводится через консоль при запросе “Выберите пункт меню: ”.

Выходные данные:

Программа выводит на экран:

- Таблицу с графами “Обработано заявок”, “Текущая длина очереди”, “Средняя длина очереди” после обслуживания каждых 100 заявок (до 1000 включительно).
- Общее время моделирования, количество вошедших в систему и вышедших из нее заявок, количество срабатываний обслуживающего аппарата, время простоя аппарата.

- Процент расхождения расчетного времени и получившегося при работе программы для входа и для выхода.
- При проверке на фрагментацию памяти - таблицу с графами “Заявка” и “Адрес в памяти” (адрес расположения данной заявки в памяти).
- Время, затраченное на приход N заявок в очередь и на их уход из очереди при реализации списком и массивом
- Память, используемая N заявками в очереди при реализации списком и массивом.

Работа с программой:

Работа с программой осуществляется с помощью меню:

```
МЕНЮ ПРОГРАММЫ:
(1) Очередь в виде массива
(2) Очередь в виде списка
(3) Проверка фрагментации при работе со списком
(4) Сравнение времени работы и используемой памяти
(0) Завершение работы
Выберите пункт меню:
```

(1) Очередь в виде массива. Моделирует процесс обслуживания до ухода из системы первых 1000 заявок. **Очередь реализована массивом.** Выводит таблицу после выхода каждых 100 заявок, общее время моделирования, количество вошедших и вышедших заявок, количество срабатываний ОА.

(2) Очередь в виде списка. Моделирует процесс обслуживания до ухода из системы первых 1000 заявок. **Очередь реализована линейным односвязным списком.** Выводит таблицу после выхода каждых 100 заявок, общее время моделирования, количество вошедших и вышедших заявок, количество срабатываний ОА.

(3) Проверка фрагментации при работе со списком. Работа с подменю:

```
(1) Добавление заявки в очередь
(2) Исключение заявки из очереди
(0) Выход
```

(1) Создает заявку и добавляет ее в очередь. Выводит на экран адреса памяти, по которому заявки располагаются.

(2) Исключает заявку из очереди. Выводит на экран таблицу оставшихся в очереди заявок.

(3) Завершение работы с подменю.

(4) Сравнение времени работы и используемой памяти. Выводит на экран информацию о времени прихода N заявок, времени ухода N заявок при реализации очереди списком и при реализации очереди массивом. А также выводит на экран память, используемую для реализации очереди списком и массивом.

(0) Завершение работы. Завершение работы с программой.

Структуры данных:

Структура очереди при реализации массивом:

```
struct queue_arr
{
    struct request queue[MAX_SIZE];
    int first;
    int last;
};
```

queue[MAX_SIZE] - массив
first - расположение первой заявки в очереди
last - расположение последней заявки в очереди

Структура очереди при реализации списком:

```
struct queue_list
{
    int length;
    struct request *in, *out;
};
```

length - длина очереди
in - последняя заявка в очереди
out - первая заявка в очереди

Структура заявки:

```
struct request
{
    float arrival_time;
    float process_time;
    float down_time;
    struct request *next;
};
```

arrival_time - время прихода заявки
process_time - время обработки заявки
down_time - время ожидания
next - следующая заявка в списке (при реализации массивом не используется)

Функции:
Инициализация очереди:
*void init_queue_list(struct queue_list *que);*
*void init_queue_arr(struct queue_arr *que);*
Добавление заявки в очередь:
*void qin_list(struct queue_list *que, struct request *new);*
*int qin_arr(struct queue_arr *que, struct request new);*
Удаление заявки из очереди:
*struct request qout_list(struct queue_list *que);*

*struct request qout_arr(struct queue_arr *que);*
Проверка на пустоту очереди:
*int empty_queue_list(struct queue_list *que);*
*int empty_queue_arr(struct queue_arr *que);*
Создание новой заявки:
*void create_new_request_list(struct request *new, float t1, float t2, float t3, float t4);*
struct request create_new_request_arr(float t1, float t2, float t3, float t4);

Фрагментация памяти:
Добавим 3 заявки в очередь. Их адреса 0x1c82850 0x1c82870 0x1c82890.

Заявка		Адрес
поступление	обработка	в памяти
2.16	0.11	0x1c82850
1.69	0.10	0x1c82870
5.71	0.73	0x1c82890

Удалим одну заявку из очереди. Ее адрес 0x1c82850.

Заявка		Адрес
поступление	обработка	в памяти
1.69	0.10	0x1c82870
5.71	0.73	0x1c82890

Добавим новую заявку в очередь. Она расположилась в памяти по адресу 0x1c82850.

Заявка		Адрес
поступление	обработка	в памяти
1.69	0.10	0x1c82870
5.71	0.73	0x1c82890
0.39	0.91	0x1c82850

Таким образом, можно сделать вывод, что фрагментации памяти при реализации списком нет.

Сравнение времени выполнения программы:

	Массив		**Список**	
Количество заявок	**Время входа**	**Время выхода**	**Время входа**	**Время выхода**
10	**2 472**	**3 924**	**4 012**	**1 164**
100	**17 988**	**77 920**	**28 936**	**12 272**
500	**87 080**	**126 244**	**133 976**	**52 532**

Можно сделать вывод, что использование **массива** дает выигрыш по времени при **добавлении заявок** примерно в **1,5 раза.**
Но при **удалении заявок** из очереди эффективнее использование **списка.** Выигрыш по времени **в 3-4 раза**, так как при реализации массивом тратится время на сдвиг элементов к началу очереди.

Сравнение используемой памяти:

Количество заявок	Реализация массивом	Реализация списком
100	**2408 байт**	**2424 байт**

При реализации очереди списком и реализации очереди массивом размер используемой памяти отличается несущественно.

При реализации массивом размер очереди ограничен до MAX_SIZE (размера массива). Следовательно, возможно переполнение очереди. При реализации очереди списком размер очереди ограничен лишь объемом оперативной памяти выделенной программе.

Также при реализации очереди списком возможна фрагментация памяти, но в результате проверки на компьютере с ОС Linux Ubuntu 16.04 ее выявлено не было.

Вопросы к лабораторной работе:

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 ее 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 ее 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 ее 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 фрагментации не возникает, однако может возникнуть переполнение памяти, а добавление и удаление элементов

сложнее. Способ реализации зависит от того, в чем мы больше 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 нужные данные.

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.