МГТУ им. Н. Э. Баумана

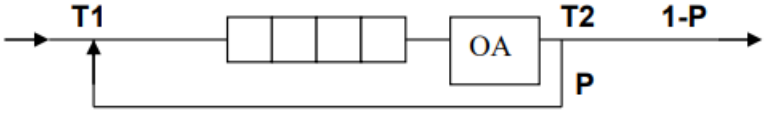
Отчет по лабораторной работе №5 по предмету «Типы и структуры данных»
«Обработка очередей»

Выполнила Параскун София,
ИУ7-34Б
Проверила
Никульшина Т. А.

Москва, 2020 г.

1. Описание условия 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 фрагментации памяти.

2. Описание ТЗ, включающего внешнюю спецификацию

а) описание исходных данных и результатов

Основная программа получает на вход выбранный пользователем пункт меню (целое число). В соответствии с ним производятся операции моделирования процесса обслуживания очереди или вывода состояния.

Программа имеет на входе границы времени поступления заявки в очередь, границы времени ее обработки, необходимое количество обработанных заявок для завершения процесса и промежуточное количество обработанных заявок.

В зависимости от выбранного режима результатом являются промежуточные и итоговые выводы процесса моделирования, информация о занятой/освобожденной памяти, а также сравнительные результаты двух реализаций очереди.

b) Способ обращения к программе

Программа «app.exe» запускается через консоль, после чего можно увидеть меню программы, состоящее из 5 пунктов:
1 — Смоделировать обслуживание 1000 заявок (на основе списка),
2 — Смоделировать обслуживание 1000 заявок (на основе массива),
3 — Показать состояние памяти,
4 — Сравнительная эффективность двух реализаций очередей,
0 — Выход из программы.

Все вводы и выводы также осуществляются в консоли.

c) Описание возможных аварийных ситуаций и ошибок пользователя

№ ситуации	Аварийная ситуация	Реакция программы
1	Ввод некорректного пункта меню	Сообщение «Its end of working the program!», завершение программы
2	Переполнение массива при таковой реализации очереди	Сообщение «Array is full! Simulating stopped.», возврат в главное меню
3	Попытка просмотра пустого списка адресов	Сообщение «Simulating on list didn’t use yet.», возврат в главное меню

3. Описание внутренних СД

```
#define MAX 10
#define TOTAL_NEED 1000
#define INTER_NEED 100
#define COMING_START 0
#define COMING_END 6
#define PROCESSING_START 0
#define PROCESSING_END 1
```

Для начала опишем константы, используемые в программе.
TOTAL_NEED – необходимое количество обработанных заявок, INTER_NEED – промежуточные состояния вывода, COMING_START и COMING_END – минимальное и максимальное время приходи заявки Т1, PROCESSING_START и PROCESSING_END – минимальное и максимальное время обработки заявки в ОА Т2, MAX – расширение, используемое при реализации очереди массивом.

Для реализации основных алгоритмов обработки очередей были использованы следующие структуры:

```
struct queue
{
    struct queue_slot *pin;
    struct queue_slot *pout;
    int len;
    int in_num;
    int state;
    int max;
    double total_stay_time;
};
```

Структура queue содержит общую информацию об очереди: pin – указатель на «хвост» очереди, pout – указатель на «голову» очереди, len – длина очереди, in_num – число вошедших в очередь заявок, state – используется для вычисления средней длины очереди, max — максимальная длина очереди, total_stay_time – время нахождения заявок в очереди.

Структура queue_slot содержит всего 2 поля: arrival_time – время «прихода» в очередь, next – указатель на следующий элемент списка. Данная структура также используется и при реализации очереди массивом, в таком случае поле next просто игнорируется.

```
struct queue_slot
{
    double arrival_time;
    struct queue_slot *next;
};
```

```
struct machine
{
    double time;
    double downtime;
    int triggering;
    int processed_count;
};
```

Структура machine используется для описания состояния обслуживающего аппарата. Она содержит поля time – текущее время состояния очереди, downtime – время простоя ОА, triggering – количество срабатываний автомата, processed_count – количество обработанных и вышедших из очереди заявок.

```
struct mem_slot
{
    struct queue_slot *queue_slot;
    int busy;
    struct mem_slot *next;
};
```

И заключительная структура mem_slot используется для отслеживания используемых/освобожденных участков памяти. Queue_slot является указателем на участок памяти элемента очереди, busy – состояние участка (1 — занят, 0 — освобожден), next – указатель на следующий элемент списка mem.

4. Описание алгоритма

Исходное состояние — пустая очередь, все используемые времена равны 0. Обслуживание очереди продолжается до тех пор, пока не из нее не выйдет 1000 элементов или пока не переполнится массив (в случае реализации очереди массивом).

В случае пустой очереди или текущего времени ОА большего времени поступления последней заявки создается новая заявка. В таком случае

учитывается время простоя автомата и происходит синхронизация текущего времени ОА с временем поступления в очередь новой заявки.

Далее производится работа самого обслуживающего автомата, который с вероятностью P = 0.8 отправляет обрабатываемую заявку в «хвост» очереди и с вероятностью 1 – P покидает очередь.

Каждые 100 обработанных записей выводится промежуточная информация о текущей длине очереди и средней длине очереди. По достижении 1000 вышедших из очереди заявок, моделирование прекращается и выводится итоговая информация о процессе, а именно: ожидаемое время моделирования, общее время моделирования, погрешность общего относительно ожидаемого, количество вошедших и вышедших из системы заявок, среднее время нахождения в очереди, время простоя ОА, количество срабатываний ОА.

5. Описание используемых функций

В используемых ниже функциях queue – общая структура стека, mem – список, хранящий адреса используемых/освобожденных элементов, mem_used – объем используемой внутри функции памяти для очереди.

*unsigned long simulate_service(struct mem_slot **mem, int *mem_used)* – моделирует процесс обслуживания очереди на основе списка.

void new_req(double *total_time, struct queue *queue, struct machine *machine, struct mem_slot **mem) — создает новую заявку в очереди, реализуемой списком, резервируя при этом участок памяти в списке mem.

*int processing(struct machine *machine, struct queue *queue, struct mem_slot **mem)* - реализует алгоритм обработки заявки в очереди, реализуемой списком.

unsigned long simulate_service_arr(struct mem_slot **mem, int *mem_used) – моделирует процесс обслуживания очереди на основе массива.

void new_req_arr(double *total_time, struct queue *queue, struct machine *machine, stuct queue_slot *array, int mass_len) — создает новую заявку в очереди, реализуемой массивом.

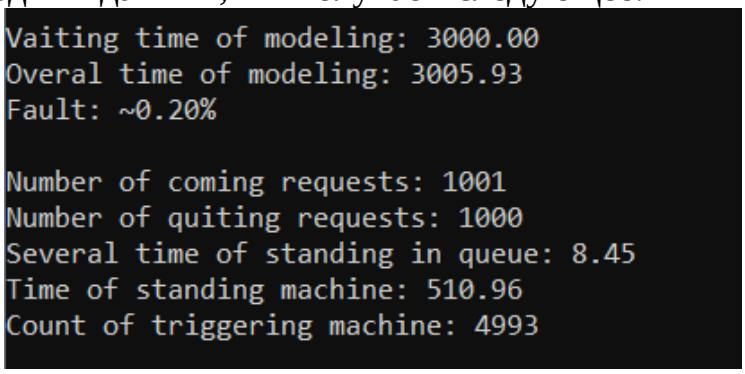
*int processing(struct machine *machine, struct queue *queue, struct queue_slot *arr, int arr_len)* -реализует алгоритм обработки заявки в очереди, реализуемой массивом.

void show_mem(struct mem_slot **mem) – выводит список используемых участков памяти при реализации очереди списком. Если процесс на основе списка еще не был запущен, выводит сообщение. Также освобождает список mem при его просмотре.

*void queue_compare(struct mem_slot **mem)* – производит сравненине двух реализаций очереди по времени и памяти, также выдает результаты о фрагментации.

6. Результаты эффективности

Одним из результатов работы программы является работа обслуживающего автомата. Для исходных данных, мы получаем следующее.



Как видим, неточность времени моделирования находится в пределах погрешности (не более 2-3%).
Теперь рассмотрим сравнительную эффективность двух реализаций на разных размерах.

Для времени прихода от 0 до 6:

Кол-во вышедших из ОА заявок	Время (список)	Время (массив)	Память (список)	Память (массив)
100	520094	101928	160	800
500	2662338	812352	208	4000
1000	10704175	1980433	368	8000
2000	16773293	5198432	384	16000

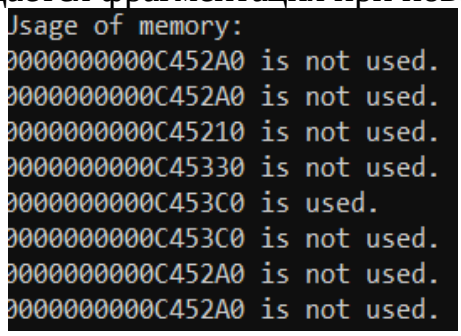
Для времени прихода от 0 до 4:

Кол-во вышедших из ОА заявок	Время (список)	Время (массив)	Память (список)	Память (массив)
100	623524	130418	464	800
500	4746116	895874	2016	4000
1000	8002748	1785166	2864	8000
2000	28570593	7943910	8224	16000

Для времени прихода от 0 до 2(на 500 элементах массива в 1000 элементов уже не хватает, поэтому приходится расширить его до 10000):

Кол-во вышедших из ОА заявок	Время (список)	Время (массив)	Память (список)	Память (массив)
100	1067894	161694	2336	800
500	11218939	981110	10624	40000
1000	27699935	2058690	24832	80000
2000	77891106	4208196	52352	160000

Также заметим, что наблюдается фрагментация при повторном выделении памяти.



7. Выводы по проделанной работе

В ходе этой лабораторной удалось поработать с таким типом данных как очередь. Также удалось реализовать операции взаимодействия с ним: добавление-удаление элементов, реализация механизма обработки очереди обслуживающим автоматом .

Очередь была реализован двумя способами — массивом и односвязным списком. После проведения оценки эффективности, стало понятно, что массив намного

эффективнее по времени (в несколько раз), при этом выигрыш списка по памяти независимо от времени прихода заявок и размерности очевиден. НО не стоит забывать, что это касается реализации на динамическом массиве. Так как при использовании статического мы не можем расширять его размер, что может привести к переполнению и прекращению работы функции обслуживания. Однако если в случае динамического массива нам придется перевыделять память, на переписывание значений уйдет много времени. Из этого можно сделать вывод, что в случае когда очередь не предполагает больших размеров, целесообразнее использовать массив, но когда идет речь об обработке огромного количества заявок, которые могут привести к переполнению, в целях экономии времени стоит использовать односвязный список.

Также стоит сказать, что все вышеописанное применимо к кольцевой очереди, недостатки которой очевидны — более сложная реализация защиты наслоения указателей Pout и Pin, что влечет неопределенное состояние заполненности массива, в случае перевыделения и переписывания данных в другую динамическую область время выполнения приблизится к списку.

Ответы на контрольные вопросы

1. Что такое очередь?

Очередь — последовательный список переменной длины, включение элементов в который идет с одной стороны «с хвоста», а исключение - с другой стороны «с головы». Принцип работы очереди: первым пришел — первым вышел, т. е. First In – First Out (FIFO).

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

При реализации массивом единоразово (в случае статики) выделяется память для хранения только самих элементов. При реализации списком для каждого элемента дополнительно выделяется память для хранения адреса следующего элемента.

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

При реализации очереди списком указателю Pout присваивается значение следующего элемента списка, а память из-под удаляемого элемента освобождается.
При реализации очереди массивом память из-под удаляемого элемента не освобождается, так как изначально она была выделена под весь массив, а значит и освобождаться будет весь массив после его использования.

4. Что происходит с элементами очереди при ее просмотре?

При просмотре очереди элементы удаляются и происходит очистка очереди, влекущая за собой освобождение памяти в случае реализации ее списком

5. Каким образом эффективнее реализовать очередь? От чего это зависит?

Выяснилось, что эффективнее и по памяти, и по времени реализовывать очередь массивом. Это зависит от самой структуры элемента в случае памяти, и от запросов в оперативную память в случае времени.

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

Если размер очереди неизвестен и ресурсы, затрачиваемые на перевыделение памяти при переполнении массива критичны, то лучше использовать список. Иначе быстрее и менее затратно по памяти использовать массив.

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

В случае массива недостатком является фиксированный размер очереди.
В случае односвязного списка недостатками являются бОльшие затраты по памяти, замедление процесса, а также фрагментация.

8. Что такое фрагментация памяти?

Фрагментация — возникновение участков памяти, которые не могут быть использованы. Фрагментация может быть внутренней — при выделении памяти блоками остается не задействованная часть, может быть внешней — свободный блок, слишком малый для удовлетворения запроса. Необходимо минимизировать суммарный объем образующихся дыр.

9. На что необходимо обратить внимание при тестировании программы?

При тестировании программы необходимо обратить внимание на корректность выводимых данных, проследить за выделением и освобождением выделяемой динамически памяти, предотвратить возможные аварийные ситуации.

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

В оперативную память поступает запрос, содержащий необходимый размер выделяемой памяти. Выше нижней границы свободной кучи HeapPtr осуществляется поиск блока памяти подходящего размера. В случае если такой найден, в вызываемую функцию возвращается указатель на эту область и внутри кучи она помечается как занятая. Если же найдена область, большая необходимого размера, то блок делится на две части, указатель на одну возвращается в вызываемую функцию и помечается как занятый, указатель на другую остается в списке свободных областей. В случае если области памяти необходимого размера не было найдено, в функцию возвращается NULL.

При освобождении памяти происходит обратный процесс. Указатель на освобождаемую область поступает в оперативную память, если это возможно объединяется с соседними свободными блоками, и помечается свободным.