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 №5
«ОБРАБОТКА ОЧЕРЕДЕЙ»

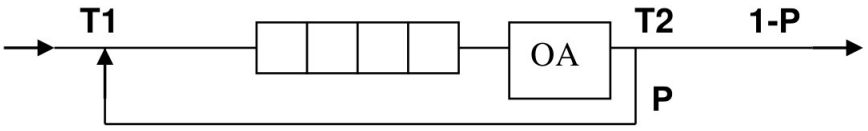
Студент Хамзина Регина Ренатовна
фамилия, имя, отчество

Студент, группа ИУ7-33Б — Хамзина Р.Р.,

2020 г.

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 фрагментация памяти.

Техническое задание

Входные данные:

1. *Целое число* — номер пункта меню, который вызывает описанное в пункте действие.
2. *Целое число* — элемент очереди при добавлении в нее. Максимальное число элементов равно 1001.

Выходные данные:

1. *Целые числа* — текущие элементы очереди.
2. *Адреса* — адреса текущих элементов очереди или адреса свободных областей.
3. *Временная характеристика* — время работы функций.
4. *Объемная характеристика* — объем структуры данных.

Функция программы:

Операции работы с очередью — добавление элемента, удаление элемента, печать текущего состояния и адресов элементов, печать свободных областей, моделирование процесса обслуживания 1000 заявок.

Обращение к программе:

Программа запускается из терминала в директории с проектом при помощи команды «./app.exe».

Аварийные ситуации

1. Некорректный ввод номера пункта меню
Входные данные : не целое число или целое число, большее 10, или целое число, меньшее 0.
Выходные данные : сообщение «Команда введена неверно».

2. Переполнение очереди — при добавлении элемента в очередь, он заполнен 10001 элементом.
Входные данные : целое число — значение элемента.
Выходные данные : сообщение «Очередь переполнена».

3. Удаление элемента из пустой очереди
Входные данные : пустая очередь.
Выходные данные : сообщение «Очередь пуста».

4. Вывод пустой очереди
Входные данные : пустая очередь.
Выходные данные : сообщение «Очередь пуста».

Внутренняя структура данных

Для реализации очереди при помощи массива используется структура:

```
typedef struct
{
    int queue[MAX_CAPACITY];
    int head;
    int tail;
    int size;
} stack_array_t;
```

Её поля:
int queue[MAX_CAPACITY] — очередь, где *MAX_CAPACITY* — её максимальный размер, равный 1001.
int head — индекс начала очереди;
int tail — индекс конца очереди;
int size — текущее число элементов;

Для реализации очереди при помощи односвязного списка используется структуры:

```
typedef struct list queue
{
   int data;
   struct list queue *next;
} list_queue;
```

Её поля:
int data — значение элемента;
*struct list_queue *next* — указатель на следующий элемент;

```
typedef struct list_queue_t
{
   int size;
   list_queue *head;
   list_queue *tail;
} list_queue_t;
```

Её поля:
int size — текущее число элементов;
*list_queue *head* — указатель на начало очереди;
*list_queue *tail* — указатель на конец очереди;

Для массива свободных областей используется структура:

*int *free_elements* — массив адресов;
int count_free_elements — текущее число адресов;

Алгоритм

Работа всей программы основана на работе пользователя с меню. Каждый пункт меню вызывает описанное в пункте действие. 0 — Выход из программы.

Алгоритм заполнения очереди и добавления в нее элементов реализуется при помощи операции *push* — вставка элемента в конец очереди - в соответствии с каждой структурой данных.
Алгоритм удаления элемента и вывода текущего состояния очереди и массива свободных областей реализуется при помощи операции *pop* — получить из очереди первый элемент — в соответствии с каждой структурой данных.
Алгоритм моделирования процесса обслуживания: до тех пор, пока обслуживающий аппарат не покинут 1000 заявок, программа генерирует время прихода новой заявки и время обработки предыдущей заявки. Выполняет действие, занимаемое меньше времени и прибавляет это время к времени моделирования. Обновляется время для более быстрого действия, и снова сравнивается. При выходе из очереди генерируется вероятность возврата ее в очередь. Если очередь пустая во время того, когда должен происходить процесс обработки, то увеличивается время простоя.

Функции программы

Функции для работы с очередью, реализованной при помощи массива:

*int array_push_queue(array_queue_t *queue, int element)* — добавление элемента в конец очереди;

*int array_pop_queue(array_queue_t *queue)* — извлечение элемента из начала очереди;

int print_array_queue(array_queue_t queue) — печать текущего состояния очереди;

void simulate_array_queue(void) — моделирование процесса обслуживания 1000 заявок;

Функции для работы с очередью, реализованной при помощи односвязного списка, и с массивом свободных областей:

*int list_push_queue(list_queue_t *queue, int element)* — добавление элемента в конец очереди;

*int list_pop_queue(list_queue_t *queue)* — извлечение элемента из начала очереди;

int print_list_queue(list_queue_t queue) — печать текущего состояния очереди;

void simulate_list_queue(void) — моделирование процесса обслуживания 1000 заявок;

*list_queue * node_create(const int element)* — создание узла списка*;*

*void free_list_queue(list_queue_t *queue)* — освобождение памяти списка*;*

void print_list_address(list_queue_t queue) — печать массива адресов элементов;

*void print_free_address(int *free_elements, int count)* — печать массива свободных областей;

Тесты

	Тест	Ввод	Вывод
1	Неверный пункт меню: больше 10	12	Команда введена неверно
2	Неверный пункт меню: меньше 0	-1	Команда введена неверно
3	Неверный пункт меню: не целое число	a	Команда введена неверно
4	Добавление в очередь, в которой уже 1001 элемент	1/5, 2	Очередь переполнена
5	Удаление элемента из пустой очереди	2/6	Очередь пуста
6	Печать пустой очереди	3/7	Очередь пуста
7	Валидное добавление элемента в очередь	1/5, 2	Временная характеристика
8	Валидное удаление элемента	2/6	Удаленный элемент: 2
15	Печать очереди, как массива	3, в очереди: 652	Элемент: 2 Элемент: 5 Элемент: 6
16	Печать очереди, как односвязного списка, и массива свободных областей	7, в очереди: 765	Очередь в виде списка: Элемент: 5 Элемент: 6 Элемент: 7

			Адреса элементов очереди в виде списка: Адрес: 0x55fb4d96fab0 Адрес: 0x55fb4d96fad0 Адрес: 0x55fb4d96faf0 Адреса свободных областей: Адрес: 0x55fb4d96e2e0

Теоретический расчет времени для процесса моделирования

По условию задачи вероятность возврата заявки в конец очереди 0.8, а вероятность выхода 0.2. Тогда ОА будет работать с простоем. Время прихода — от 0 до 6 е.в., время обработки от 0 до 1 е.в.

Ожидаемое *время моделирования* будет определяться как произведение среднего интервала между приходом заявок и количеством вошедших заявок.

model_time = ((0 е.в. + 6 е.в.) / 2) * 1000 = 3000 е.в.

Ожидаемое *время обработки* будет определяться как произведение среднего времени обработки заявки, умноженного на количество обработанных заявок. Так как заявка выходит с вероятностью 0.2, чтобы получить 1000 заявок на выходе необходимо 5 проходов.

process_time =((0 е.в. + 1 е.в.) / 2) * 1000 * 5 = 2500 е.в.

Время простоя ОА определяется, как разность времени моделирования и времени обработки.

downtime = model_time — process_time = 500 е.в.

На практике при выполнении программы получены следующие результаты:

```
Ожидаемое время моделирования: 3000 е.в.
Общее время моделирования: 3012.337878 е.в.

Количество вошедших заявок: 5105, из них повторно возвращенные заявки: 4104
Количество вышедших заявок: 5104, из них не вернувшихся обратно: 1000

Ожидаемое время простоя: 500 е.в
Время простоя аппарата: 447.427308 е.в.

Время среднего пребывания заявки в очереди: 355.547371 е.в.
Количество срабатываний ОА: 5104

Проверка работы:
Погрешность по вошедшим заявкам: 0.309988%
Погрешность по вышедшим заявкам: 0.411263%
```

Сравниваем практические результаты с теоретическими расчетами.
Для проверки правильности работы системы по входу общее время моделирования делим на время прихода одной заявки:
3012.337878 / 3 = 1004.112626 заявок

Определяем предполагаемое количество вошедших заявок, фактически их пришло 5105 — 4104 = 1001, т.е. *погрешность по вошедшим заявкам* составляет

|100%(1001 -1004.11)/1004.11|= 0.309%

Определим *погрешность по вышедшим заявкам*. Расчетное время работы равно 3000е.в., а фактическое – 3012.337878. То есть, погрешность составит

|100%(3012.337878-3000)/3000| = 0,411%.

Таким образом, проверка работы системы по входным и выходным данным показала, что погрешность работы системы составляет не более 0.4%, что удовлетворяет поставленному условию.

Оценка эффективности

Время добавления элемента в стек (тики):

Число элементов	Массив	Список
10	1520	41600
50	9100	95100
100	18200	166400

Время удаления элемента из стека (тики):

Число элементов	Массив	Список
10	1300	3400
50	7800	16800
100	15600	35700

Объем памяти (байты):

Число элементов	Массив	Список
10	4016	240
50	4016	12000
1000	4016	24000

Ответы на контрольные вопросы

Что такое очередь?

Очередь - структура данных, позволяющая добавлять элементы только в конец, а удалять только из начала. Принцип работы очереди: первым пришел – первым вышел, т.е. First In – First Out (FIFO).

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

Для кольцевого статического массива: память для всей очереди выделяется на стеке. Если массив динамический память выделяется на куче постепенно. Объём выделяемой памяти равен n * sizeof (элемента очереди), где n – количество элементов в очереди.

Для связного списка: память выделяется постепенно на куче при добавлении нового элемента. Объём выделяемой памяти равен n * (sizeof(элемент очереди) + sizeof(указатель на следующий элемент)), где n – количество элементов в очереди

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

При хранении кольцевым массивом память не освобождается, а просто меняется указатель на конец очереди.

При хранении списком, память под удаляемый кусок освобождается.

Что происходит с элементами очереди при ее просмотре?

Элементы удаляются из очереди.

Каким образом эффективнее реализовывать очередь. От чего это зависит?

По времени обработки выигрышной является реализация очереди в виде кольцевого массива. Зная максимальный размер очереди, лучше всего использовать кольцевой статический массив. Не зная максимальный размер, стоит использовать связный список, так как такую очередь можно будет переполнить только, если закончится оперативная память.

В каком случае лучше реализовать очередь посредством указателей, а в каком – массивом?

Ответ в вопросе № 5.

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

Связный список:

1) недостатки: может возникнуть фрагментация памяти, а также такой способ реализации является проигрышным по времени;

2) достоинства: размер очереди ограничен лишь объёмом оперативной памяти.

Статический кольцевой массив:

1) недостатки: очередь всегда будет ограниченна определенным размером

2) достоинства: операции при такой реализации выполняются гораздо быстрее.

Что такое фрагментация памяти?

Фрагментация памяти - разбиение памяти на фрагменты, которые лежат не рядом друг с другом. Так появляются фрагменты памяти, которые нельзя использовать.

На что необходимо обратить внимание при тестировании программы?

На корректное освобождение памяти и аварийные ситуации: переполнение очереди и пустая очередь.

Каким образом физически выделяется и освобождается память при динамических запросах?

ОС выбирает какой-то блок памяти, куда будет происходить запись данных (причем иногда в выбранном блоке может быть недостаточно памяти для наших нужд, что приводит к возникновению ошибок) При освобождении памяти ОС удаляет данные из данного блока памяти, тем самым делая его снова свободным.

Выводы

Очередь, реализованная связным списком, проигрывает по времени обработки статическому массиву, так как при работе со статическим массивом необходимо работать только с указателем, а при работе со списком с указателем, а также с памятью. Массив в большинстве случаев выигрывает по памяти. Но, когда заранее неизвестен максимальный размер очереди, то можно использовать связанный списки, так как в отличии от статического массива, списки ограничены в размерах только размером оперативной памяти. При реализации очереди кольцевым списком переполнение очереди возможно только в случае, когда очередь заполнена вся полностью. Фрагментация памяти при работе с очередью, реализованной при помощи односвязного списка не происходила.