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 «Обслуживающий аппарат. Очередь»

Студент Родинков Алексей Глебович

Группа ИУ7 – 31БВ

Преподаватель Силантьева Александра Васильевна

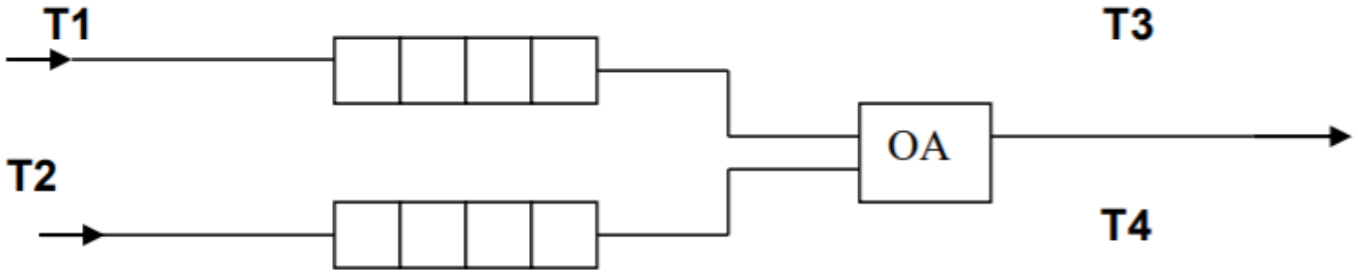
2023г

Оглавление

Описание условия задачи..... 2
Выходные данные.....2
Обращение к программе..... 3
Описание возможных аварийных ситуаций и ошибок пользователя..... 3
Описание внутренних структур данных..... 3
Описание внутренних подпрограмм..... 4
Описание алгоритма..... 5
Тестирование.....6
Временные замеры.....7
Ответы на вопросы.....8
Вывод.....10

Описание условия 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 ждет первого освобождения ОА и далее поступает на обслуживание (система с относительным приоритетом). Смоделировать процесс обслуживания первых 1000 заявок 1-го типа. Выдать на экран после обслуживания каждых 100 заявок 1-го типа информацию о текущей и средней длине каждой очереди, количестве вошедших и вышедших заявок и о среднем времени пребывания заявок в очереди. В конце процесса выдать общее время моделирования и количество вошедших в систему и вышедших из нее заявок обоих типов. По требованию пользователя выдать на экран адреса элементов очереди при удалении и добавлении элементов. Проследить, возникает ли при этом фрагментация памяти.

Выходные данные

При первом или втором пункте программа моделирует очередь при помощи массива или списка. Она будет работать пока не будет проанализировано 1000 запросов. Каждые 100 запросов программа выводит параметры на момент времени:

- Вышедшие заявки(обработанные)
- Вошедшие в систему заявки
- Неудачные заявки – заявки, которые не удалось положить в очередь из-за ее переполнения.
- Текущая длина очереди
- Средняя длина очереди
- Среднее время ожидания в очереди

Эти данные выводятся для каждой из очередей.

В итоге будут выведены итоговые переменные программы: общее время программы, время простоя, время работы. Также будет выведено ожидаемое время моделирования и погрешность.

При выборе оценки эффективности программа выведет информацию о времени (мс) и затраченной памяти (байт) данного алгоритма: полное заполнение очереди и ее полная очистка.

Обращение к программе

Для запуска программы необходимо, находясь в директории с программой в терминале выполнить команду:
./app.exe

Описание возможных аварийных ситуаций и ошибок пользователя

#define OVERFLOW_ERROR 1
#define ALLOC_ERROR 2

Аварийные случаи - ошибка выделение памяти. При переполнении очереди программа уведомит пользователя, но не завершится аварийно. При неверном выборе программа уведомит пользователя и потребует ввести значение заново.

Описание внутренних структур данных

- Структура для описания очереди с использованием массива

```
typedef struct
{
   task_t **data;
   int pin;              //  количество входящих заявок
   int pout;            // количество исходящих заявок
   int amount;       // текущий размер
   int maxamount; // максимальный размер
   int allocated;   // выделенная память

} arr_queue_t;
```

- Структура для описания очереди с использованием списка

```
typedef struct
{
   node_t *pin;    // вышедший элемент списка
   node_t *pout; // обработанный элемент списка

   int amount;       // количество текущих заявок
   int maxamount; // максимальный размер заявок

} list_queue_t;
```

- Структура для описания узла списка

```
typedef struct node node_t;

struct node
{
   task_t *task;   // задача
   node_t *next; // указатель на следующий элемент
};
```

- Структура для описания статистики очереди

```
typedef struct
```

```
{
   int tasks_in; // Вошедшие заявки
   int tasks_out; // Обработанные заявки
   int tasks_failed; // Неудачные заявки
   int calls; // Вызовы ОА
   int overall_len; // Общая длина очереди
} sim_log_t;
```

- Структура для описания заявки

```
typedef struct
{
   int n;
} task_t;
```

Примечание: структура выбрана так лишь для примера, предусмотрена легкая возможность изменить информацию заявки.

Описание внутренних подпрограмм

Подпрограммы для работы с очередью-массивом

arr_queue_t ***create_arr_queue**(); // Создание
void **delete_arr_queue**(arr_queue_t *queue); // Удаление
int **push_arr_queue**(arr_queue_t *queue, task_t *task); // Добавление элемента в очередь
task_t ***pop_arr_queue**(arr_queue_t *queue); // Удаление элемента из очереди
void **print_arr_queue**(arr_queue_t *queue); // Вывод очереди

Подпрограммы для работы с очередью-списком

list_queue_t ***create_list_queue**(); // Создание
void **delete_list_queue**(list_queue_t *queue); // Удаление
int **push_list_queue**(list_queue_t *queue, task_t *task); // Добавление элемента в очередь
task_t ***pop_list_queue**(list_queue_t *queue); // Удаление элемента из очереди
void **print_list_queue**(list_queue_t *queue); // Печать очереди

Создание узла
node_t ***create_node**(task_t *task);

Создание заявки
task_t ***create_task**();

Общие подпрограммы

void **simulate_arr**(); // Запуск симуляции для очереди-массива
void **simulate_list**(); // Запуск симуляции для очереди-списка
void **efficiency**(); // Запуск оценки эффективности алгоритмов
void **hand_mode**(); // Запуск ручного режима

Вспомогательные подпрограммы

void **make_and_del_arr**(); // Создание, заполнение и удаление очереди-массива

void **make_and_del_list**(); // Создание, заполнение и удаление очереди-списка
float **random_float**(float min, float max); // Получение случайного вещественного числа в интервале
float **max**(float a, float b, float c); // Получение максимума из 3
float **min**(float a, float b, float c); // Получение минимума из 3

Описание алгоритма

Программа получает на вход команду от пользователя. Меню программы:

0 - выход из меню

1 - запустить симуляцию для очереди массивом

2 - запустить симуляцию для очереди списком

3 - оценка эффективности

4 - перейти в ручной режим

5 - поменять параметры симуляции

При выборе пункта 0 происходит выход из программы.

При выборе пункта 1 и 2 выполняется симуляция процесса обслуживания 1000 заявок из первой очереди. 1 – очередь в форме массива; 2 – очередь в форме односвязного линейного списка.

При выборе пункта 3 происходит запуск оценки эффективности алгоритма добавления – удаления заявок для каждого типа очередей.

При выборе пункта 4 происходит переход в подменю.

Меню ручного режима:

1 - добавить заявку в массив-очередь

2 - добавить заявку в список-очередь

3 - удалить заявку из массива-очереди

4 - удалить заявку из списка-очереди

5 - текущее состояние очередей

0 - выйти из меню

При добавлении/удалении заявки выводится ее адрес.

При выборе пункта 5 появляется возможность изменить параметры программы – интервалы доставки, обработки.

Программа генерирует случайное время доставки в каждую очередь, сохраняя это время в «таймерах» для каждой из очередей. Затем если в первой очереди есть заявка, обрабатывается она, время также генерируется и сохраняется в «таймер» ОА. После чего происходит сравнение всех таймер и выбирается действие, которое нужно выполнить в первую очередь.

Тестирование

Вход	Выход
Попытка добавить элемент в заполненную очередь	Очередь переполнена
Попытка удалить элемент из пустой очереди	Очередь пуста
Попытка ввести несуществующий пункт меню	Неверный пункт меню
Попытка ввести пункт меню не целым числом	Неверный пункт меню
Попытка ввести параметры симуляции – интервалы неверно	Некорректный ввод

Моделирование

Промежутки по умолчанию, размеры очередей – 1024

```
Время ожидания: 72.099579
Время работы: 2921.825928
Общее время моделирования : 2993.925537
Ожидаемое теоритическое время: 3000.000000
Погрешность: 0.20%
```

```
Вышедшие заявки 1: 1000
Вошедшие завяки 1: 1000
Неудачные заявки 1: 0
Текущая длина очереди 1: 0
Средняя длина очереди 1: 0
Среднее время ожидания в очереди 1: 0.000000

Вышедшие заявки 2: 1828
Вошедшие завяки 2: 1944
Неудачные заявки 2: 0
Текущая длина очереди 2: 116
Средняя длина очереди 2: 36
Среднее время ожидания в очереди 2: 54.613274
```

Теоретический расчет:

Время:
(1+5)/2 * 1000 = 3000(е.в.)

Выбираем 1+5, так как среднее время доставки в первую очередь (1, 5) больше среднего времени обработки заявок из первой очереди (0, 4).

Количество заявок во второй очереди:

Так как среднее время доставки второй очереди – (0 + 3)/2 = 1.5 е.в., то за 3000 е.в должно прийти 2000 заявок. При этом каждая заявка обрабатывается (0 + 1)/2 = 0.5 е.в., что займет 1000 е.в.

Рассчитаем теперь время простоя ОА для первой очереди: обработается 1000 заявок по (0 + 4)/2 = 2 е.в. каждая, что займет 2000 е.в., время простоя при этом 3000 – 2000 = 1000 е.в.

Значит должно обработаться 2000 заявок из 2 очереди.

Промежутки по умолчанию, кроме времени доставки в первую очередь – (1, 7)

```
Время ожидания: 1965.347534
Время работы: 2107.230225
Общее время моделирования : 4072.577637
Ожидаемое теоритическое время: 4000.000000
Погрешность: 1.81%
```

Промежутки по умолчанию, кроме времени обработки заявок из первой очереди – (0, 10)

```
Время ожидания: 0.000000
Время работы: 4994.692383
Общее время моделирования : 4994.692383
Ожидаемое теоритическое время: 5000.000000
Погрешность: 0.11%
```

Временные замеры

	Количество заявок	**Время, мс**
Список	**1**	**43**
Массив	**1**	**40**
Список	**5**	**500**
Массив	**5**	**500**
Список	**10**	**2300**
Массив	**10**	**1500**

	Количество заявок	**Память, байт**
Список	**1**	**1800**

Массив	**1**	**500**
Список	**5**	**8000**
Массив	**5**	**2000**
Список	**10**	**16000**
Массив	**10**	**4000**

Ответы на вопросы

1. Что такое FIFO и LIFO

Метод очереди – FIFO – первый вошел, первый вышел, метод, при котором первым удаляется элемент, поступивший в очередь первым.

Метод стека – LIFO – последний вошел, первый вышел, метод, при котором первым удаляется элемент, поступивший в стек последним.

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

При моделировании линейной очереди на основе одномерного массива выделяется последовательная область памяти из x мест по L байт, где L – размер поля данных для одного элемента размещаемого типа.

Реализация очереди в виде линейного списка. Большинство проблем, возникающих при реализации очереди в виде массива, устраняется при реализации очереди на основе односвязного линейного списка, каждый элемент которого содержит информационное поле и поле с указателем «вперед» (на следующий элемент). В этом случае в статической памяти можно либо хранить адрес начала и конца очереди, либо – адрес начала очереди и количество элементов.

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

При реализации очереди списком память освобождается по элементам: при удалении 1го элемента память освобождается только из-под этого элемента.

При реализации очереди массивом память не освобождается.

4. Что происходит с элементами очереди при ее просмотре?

При классической реализации — он удаляется из очереди.

5. От чего зависит эффективность физической реализации очереди?

От поставленных задач и требований. Массив быстрее и экономичней. Но его размер ограничен. Тогда как список удобен в использовании.

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

При реализации очереди списком недостатком является время работы и затраты памяти подобного алгоритма. Преимуществом является возможность выделять любое количество элементов, при этом вся выделенная память будет задействована.

При реализации очереди массивом недостатком является ограниченность количества вводимых в очередь данных и бесполезность выделенной памяти, которая не используется в данный момент. Преимуществом является быстродействие и небольшое, пусть и бесполезное, количество выделенной памяти.

7. Что такое фрагментация памяти?

Такое положение объектов в памяти, при котором общий объем свободной памяти позволяет вписать некий объект, однако это становится невозможно вследствие того, что непрерывной области свободной памяти в ОП нет. Фрагментация памяти возникает в активно используемой части ОП.

8. Для чего нужен алгоритм «близнецов».

Для ускорения процесса поиска свободного блока памяти.

9. Какие дисциплины выделения памяти вы знаете?

Дисциплина «самый подходящий» – выделяется участок памяти, равный требуемому или превышающий его на минимальную величину.

Дисциплина «первый подходящий» – выделяется участок памяти, больший или равный требуемому, при этом найденный первым.

Алгоритм «близнецов». Идея этого алгоритма состоит в том, что организуются списки свободных блоков отдельно для каждого размера 2^k.

10. На что необходимо обратить внимание при тестировании программы?

При тестировании программы необходимо:

-проверить правильность работы программы при различном заполнении очередей, т.е., когда время моделирования определяется временем обработки заявок и когда определяется временем прихода заявок;

-отследить переполнение очереди, если очередь в программе ограничена.

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

Память представляет собой бинарную кучу с признаком — занятость - незанятость ячейки. Динамический запрос меняет признак ячейки.

Вывод

Исходя из полученных результатов реализация очереди на массивах выигрывает как по времени, так и по по памяти. Также видно, что практическая погрешность не превышает 2% что находится в пределах нормы.