ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Департамент программной инженерии

Отчет по домашнему заданию по дисциплине "Архитектура вычислительных систем"

Пояснительная Записка

Исполнитель:
Студент группы БПИ195(1)
Ни Эдуард
29.11.2020 г.

Москва 2020

Условие задания

ВАРИАНТ23. *Первая военная задача.* Темной-темной ночью прапорщики Иванов, Петров и Нечепорчук занимаются хищением военного имущества со склада родной военной части. Будучи умными людьми и отличниками боевой и строевой подготовки, прапорщики ввели разделение труда: Иванов выносит имущество со склада, Петров грузит его в грузовик, а Нечепорчук подсчитывает рыночную стоимость добычи. Требуется составить многопоточное приложение, моделирующее деятельность прапорщиков. При решении использовать парадигму «производитель-потребитель».

Решение

Для решения задачи будем пользоваться парадигмой конструирования многопоточных приложений **производители и потребители**.

Производители и потребители *– это парадигма взаимодействующих неравноправных потоков. Одни потоки «производят» данные, другие их «потребляют». Часто такие потоки организуются в конвейер, через который проходит информация. Каждый поток конвейера потребляет выход своего предшественника и производит входные данные для своего последователя. Другой распространенный способ организации потоков – древовидная структура или сети слияния, на этом основан, в частности, метод дихотомии.*

По условию, мы имеем трех прапорщиков, которые расхищают склад с военным имуществом. Прапорщик Иванов по нашей парадигме выносит военное имущество (подготавливает для Петрова). Прапорщик Петров загружает машину вынесенным оборудованием (подготавливает для Нечепорчука). Нечепорчук, сидя в машине, подсчитывает рыночную стоимость загруженного Петровым оборудования.

Составление программы

Для работы программы будем использовать библиотеки для языка C++: iostream, unistd.h, omp.h.

Наша программа будет работать бесконечно.

Москва 2020

Текст программы

```
#include <iostream>
#include <unistd.h>
#include <omp.h>

const int munSize = 10;     //Количество военного имущества.
int cntr = 0;    //Количество вынесеного имущества.
int cost = 0;     //Суммманая стоимость вынесенного имущества.
int munition[munSize];  //Военное имущество.
int delCost = 4;     //Делитель Стоимости одного оборудования на рынке.
int front = 0; //Индекс для чтения из буффера.
int rear = 0;    //Индекс для записи в буффер.
bool iDone = false; //Иванов выполнил работу.
bool pDone = false; //Петров выполнил работу.
bool nDone = true;  //Нечепорчук выполнил работу.


/**
 * Метод передается в поток в качестве производителя.
 * @param mun Идентификатор производителя.
 */
void *Producer(void *mun);

/**
 * Метод передается в поток в качестве потребителя.
 * @param mun Идентификатор потребителя.
 */
void *Consumer(void *mun);

/**
 * Метод передается в поток в качестве счетчика срабатываний производителя-
потребителя.
 * Работает аналогично потребителю.
 * @param mun Идентификатор счетчика.
 */
void *Counter(void *mun);

int main() {
#pragma omp parallel num_threads(3)
    {
        int trN = omp_get_thread_num();
        if (trN == 1) {
            std::string pName = "Иванов";
            Producer((void *) &(pName));
        } else if (trN == 2) {
            std::string cName = "Петров";
            Consumer((void *) &(cName));
        } else {
            std::string cntrName = "Нечепорчук";
            Counter((void *) &(cntrName));
        }
    }
    return 0;
}

void *Producer(void *mun) {
    const char *pName = (*((std::string *) mun)).c_str();
    while (true) {
        while (!nDone);
```

Москва 2020

```
        nDone = false;
        int data = rand() % 10000; //Генерируем номер оборудования
        munition[rear] = data;
        rear = (rear + 1) % munSize;
        printf("Прапорщик %s выносит имущество № [%d] со склада.\n", pName,
data);
        sleep(1);
        iDone = true;
    }
}

void *Consumer(void *mun) {
    const char *cName = (*((std::string *) mun)).c_str();
    int res;

    while (true) {
        while (!iDone); //Пока Иванов не сделал вынес.
        iDone = false;
        res = munition[front];
        front = (front + 1) % munSize;
        printf("Прапорщик %s грузит оборудование № [%d] в грузовик.\n", cName,
res);
        ++cntr;

        sleep(1);
        pDone = true;
    }
}

void *Counter(void *mun) {
    const char *cntrName = (*((std::string *) mun)).c_str();

    while (true) {
        while (!pDone);
        pDone = false;
        if (cntr > munSize & (cntr - 1) % munSize == 0)
            printf("Оборудования там больше, чем казалось... Берем все!\n");
        cost += munition[cntr % munSize - 1] / delCost;  //Суммируем стоимость
вынесенного оборудования.

        printf("Прапорщик %s подчитал, что было вынесено оборудования (%dшт.) на
сумму %d у.е.\n", cntrName, cntr,
               cost);

        sleep(1);
        nDone = true;
    }
}
```

Описание программы

1. Глобальные переменные.

```
const int munSize = 10;    //Количество военного имущества.
int cntr = 0;   //Количество вынесеного имущества.
int cost = 0;    //Суммманая стоимость вынесенного имущества.
int munition[munSize];  //Военное имущество.
int delCost = 4;    //Делитель Стоимости одного оборудования на рынке.
int front = 0; //Индекс для чтения из буффера.
int rear = 0;   //Индекс для записи в буффер.
bool iDone = false; //Иванов выполнил работу.
bool pDone = false; //Петров выполнил работу.
bool nDone = true;  //Нечепорчук выполнил работу.
```

Рисунок 1 – Глобальные переменные

Задаем значение для количества вынесенного имущества *munSize* и массив, в которое будем вносить *int munition[munSize]*. Инициализируем два индекса, указывающие на чтение буфера и записи – *front* и *rear*. Создаем 3 переменные типа bool. Они будут сообщать друг другу, что какой-то из прапорщиков завершил работу.

2. Метод производителя Producer.

```
void *Producer(void *mun) {
    const char *pName = (*((std::string *) mun)).c_str();
    while (true) {
        while (!nDone);
        nDone = false;
        int data = rand() % 10000; //Генерируем номер оборудования
        munition[rear] = data;
        rear = (rear + 1) % munSize;
        printf( format: "Прапорщик %s выносит имущество № [%d] со склада.\n", pName, data);
        sleep( seconds: 1);
        iDone = true;
    }
}
```

Рисунок 2 – Метод производителя

Метод представляет собой производителя(ей), который подготавливает данные для потребителя(ей). В аргументах метода переменная void *mun нужно для идентификации потребителя. Через разыменование получаем строку – имя. Далее в бесконечном цикле ждем, когда прапорщик Нечепорчук выполнит свое действие, чтобы Иванов начал работу. Генерируем с помощью rand() номер оборудования, в

диапазоне от 1 до 10000. Записываем в украденное оборудование новое: munition[rear] = data. Выводим сообщение о том, что прапорщик выносит имущество под номером data со склада, приостанавливаем поток на 1 секунду. После чего, говорим, что Иванов выполнил свою работу.

3. Метод потребителя Consumer.

```
void *Consumer(void *mun) {
    const char *cName = (*((std::string *) mun)).c_str();
    int res;

    while (true) {
        while (!iDone); //Пока Иванов не сделал вынес.
        iDone = false;
        res = munition[front];
        front = (front + 1) % munSize;
        printf( format: "Прапорщик %s грузит оборудование № [%d] в грузовик.\n", cName, res);
        ++cntr;

        sleep( seconds: 1);
        pDone = true;
    }
}
```

Рисунок 3 – Метод Consumer

Метод представляет собой потребителя(ей), который работает с данными, подготовленными производителем. В аргументах метода переменная void *mun нужно для идентификации потребителя. Через разыменование получаем строку – имя. Далее в бесконечном цикле ждем, когда прапорщик Иванов выполнит свое действие, чтобы Петров начал работу. Получаем добавленное производителем оборудование и выводим сообщение о том, что прапорщик грузит оборудование res в грузовик, приостанавливаем поток на 1 секунду. После чего, говорим, что Петров выполнил свою работу.

4. Метод счетчика Counter.

```
void *Counter(void *mun) {
    const char *cntrName = (*((std::string *) mun)).c_str();
    while (true) {
        while (!pDone);
        pDone = false;
        if (cntr > munSize & (cntr - 1) % munSize == 0)
            printf( format: "Оборудования там больше, чем казалось... Берем все!\n");
        cost += munition[cntr % munSize - 1] / delCost;  //Суммируем стоимость вынесенного оборудования.

        printf( format: "Прапорщик %s подчитал, что было вынесено оборудования (%dшт.) на сумму %d у.е.\n", cntrName, cntr,
                cost);

        sleep( seconds: 1);
        nDone = true;
    }
}
```

Рисунок 5 – Метод Counter

Работает почти также, как и Consumer. Представляет собой потребителя(ей), который работает с данными, который подготовил прошлый потребитель (производитель для Consumer). В аргументах метода переменная void *mun нужно для идентификации потребителя. Через разыменование получаем строку – имя. Далее в бесконечном цикле ждем, когда прапорщик Петров выполнит свое действие, чтобы Нечепорчук начал работу. Прибавляем к переменной cost стоимость оборудования, которое занес в машину прошлый прапорщик (munition[cntr % munSize - 1] = = munition[front - 1]). Стоимость военного имущества расчитываем как номер имущества делить на delCost, delCost = 4. Выводим сообщение о том, что прапорщик посчитал стоимость всего оборудования, которое было загружено в машину, останавливаем поток на 1 секунду. После чего, говорим, что Нечепорчук выполнил свою работу.

Тестирование программы

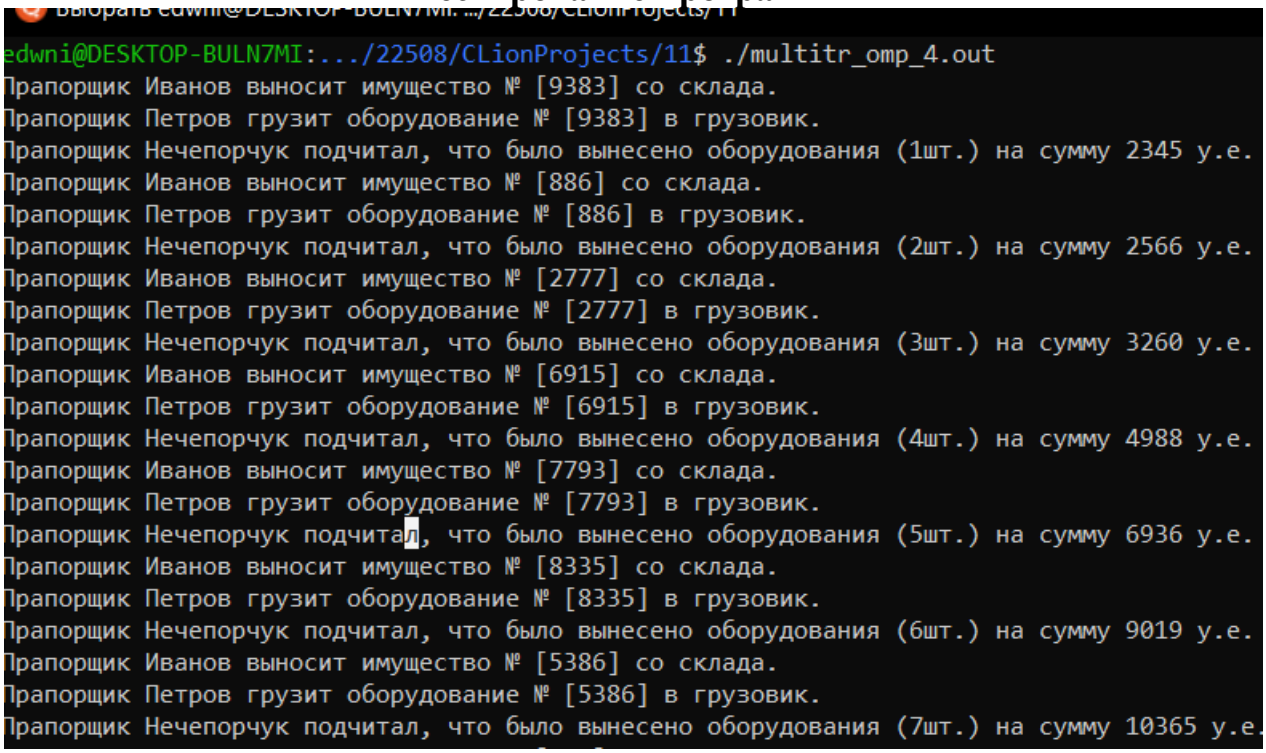

Рисунок 6- результат выполнения 1

Как видно из (рис. 6), программа работает корректно. Прапорщик Иванов выносит имущество под номером N со склада, а Иванов кладет его в грузовик. Нечепорчук подсчитывает количество вынесенных объектов и сумму в условных единицах. Нужно заметить, что Нечепорчук фиксирует результаты только действия прапорщика Иванова.

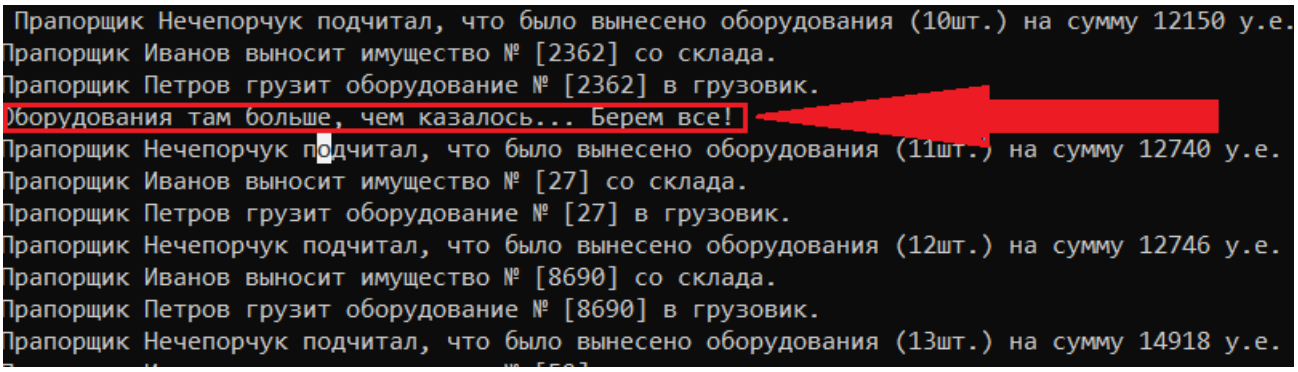

Рисунок 7 - результат выполнения 2

По (рис. 7) видно: так как мы ограничили количество грузов до 10, программа должна выполняться бесконечно. Поэтому каждый кратный 10ти раз мы будем выводить сообщение о том, что оборудования больше, чем, было заявлено.

Москва 2020

Используемые источники

1. SoftCraft, Архитектура вычислительных систем. Многопоточность [Электронный ресурс] //URL: http://www.softcraft.ru/edu/comparch/lect/07-parthread/multitreading.pdf, свободный. (дата обращения 12.11.2020, режим доступа: свободный).
2. Киберфорум, Архитектура вычислительных систем. Многопоточное программирование.Синхронизация [Электронный ресурс] //URL: https://www.cyberforum.ru/blogs/18334/blog2965.html, свободный. (дата обращения 27.11.2020, режим доступа: свободный).
3. ИНТУИТ. Операционные системы – аспекты параллелизма. [Электронный ресурс] //URL: https://intuit.ru/studies/courses/4447/983/lecture/14923?page=4, свободный. (дата обращения 25.11.2020, режим доступа: свободный).
4. SoftCraft, Архитектура вычислительных систем. Многопоточное программирование. OpenMP [Электронный ресурс] //URL: http://www.softcraft.ru/edu/comparch/practice/thread/03-openmp/, свободный. (дата обращения 27.11.2020, режим доступа: свободный).

Москва 2020