**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Образовательная программа бакалавриата «Программная инженерия»

|  |  |  |  |
| --- | --- | --- | --- |
|  | **РАЗРАБОТАТЬ МНОГОПОТОЧНОЕ ПРИЛОЖЕНИЕ, РЕШАЮЩЕЕ ЗАДАЧУ О НАСЛЕДСТВЕ**  **Пояснительная записка** | | |
|  |  | |
| **Исполнитель**  студент группы БПИ197  \_\_\_\_\_\_/И. Ракичевич /  «17» ноября 2020 г. | |
|  | | |
|  | |  |

**Москва 2020**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  | | |
|  | **Пояснительная записка**  **Листов 14** | | | |
|  |  | | |
|  | | |
|  | | | |
|  | | |  |

**Москва 2020**

СОДЕРЖАНИЕ

[1. НАИМЕНОВАНИЕ ПРОГРАММЫ 4](#_Toc56541504)

[2. УСЛОВИЕ 5](#_Toc56541505)

[3. АЛГОРИТМ 6](#_Toc56541506)

[3.1 Описание алгоритма и функционирования программы 6](#_Toc56541507)

[3.1.1 Общий алгоритм 6](#_Toc56541508)

[3.1.2 Структуры 6](#_Toc56541509)

[3.1.2 Функция main 6](#_Toc56541510)

[3.1.3 Функция check\_record 7](#_Toc56541511)

[3.1.4 Функция \*task\_proc 7](#_Toc56541512)

[3.2 Код программы 9](#_Toc56541513)

[3.3 Текст работы 13](#_Toc56541514)

[3.4 Выполнение работы 13](#_Toc56541515)

[ПРИЛОЖЕНИЕ 1 СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 14](#_Toc56541516)

# НАИМЕНОВАНИЕ ПРОГРАММЫ

Наименование программы: «Многопоточное приложение, решающее задачу о наследстве».

# УСЛОВИЕ

Разработать многопоточное приложение, решающее задачу о наследстве (Вариант 18).

Условие задачи:

«18. Задача о наследстве. У старого дона Энрике было два сына, у каждого из сыновей – еще по два сына, каждый из которых имел еще по два сына. Умирая, дон Энрике завещал все свое богатство правнукам в разных 6 долях. Адвокат дон Хосе выполнил задачу дележа наследства в меру своих способностей. Правнуки заподозрили адвоката в укрывательстве части наследства. Требуется создать многопоточное приложение, которое при известных сумме завещания дона Энрике и доле каждого наследника, проверяет честность адвоката. При решении использовать принцип дихотомии.»

# АЛГОРИТМ

## Описание алгоритма и функционирования программы

### Общий алгоритм

Используется принцип дихотомии и модель построения многопоточных приложений - Рекурсивный параллелизм, с использованием функций библиотеки POSIX Threads или стандартной библиотеки C++.

### Структуры

struct inh\_share {

double ratio; // Соотношение.

int sum; // Значение наследства для одного человека.};

struct inh\_task {

int total\_sum; // Сумма наследства.

int n; // Размер куска массива.

inh\_share \*part; // Адрес начала куска.};

### 3.1.2 Функция main

int main() {

// Чтение из файла.

std::ifstream f("input.txt");

inh\_task t;

void \*rv;

int res, i;

// Количество наследников и сумма наследства, считанных из файла.

f>>t.n>>t.total\_sum;

// Доля.

t.part = new inh\_share[t.n\*sizeof(inh\_share)];

// Заполняем массив наследств и структуру для первого вызова. Вызываем.

// Если результат совпал с суммой наследства, то все довольны, и адвокат ничего не присвоил себе.

for (i=0; i<t.n; i++)

f>>t.part[i].ratio>>t.part[i].sum;

rv = task\_proc(&t);

res = (int)(intptr\_t)rv;

delete [] t.part;

// Проверка итога работы. Если все правильно, то программа возвращает 0, иначе 1.

if (res == t.total\_sum) {

std::cout<<"The inheritance shared correctly\n";

return 0;

} else {

std::cout<<"The inheritance shared incorrectly\n";

return 1; }}

### Функция check\_record

int check\_record(int total, const inh\_share \*ps)

{

// Происходит проверка на "честность" адвоката, то есть правильно ли произошло деление наследства.

// Если деление наследства произошла правильно, то вернется значение наследства для одного человека, иначе -1.

return (fabs(ps->ratio\*total-ps->sum) < 1)?ps->sum:-1;}

### Функция \*task\_proc

void \*task\_proc(void \*arg)

{

inh\_task \*ptask = (inh\_task \*)arg;

// Если размер куска - одна запись, то он честно проверяет ее корректность и возвращает результат,

// приведенный к типу void\*.

if (ptask->n == 1)

return (void \*)(intptr\_t)check\_record(ptask->total\_sum, ptask->part);

else {

// Если же в куске больше элементов, то создаем 2 куска примерно одной длины: левый и правый. И запускаем нити на обработку.

int nleft = ptask->n>>1;

inh\_task task\_left = { ptask->total\_sum, nleft, ptask->part };

inh\_task task\_right = { ptask->total\_sum, ptask->n - nleft, ptask->part+nleft };

pthread\_t thread\_left, thread\_right;

void \*rv\_left, \*rv\_right;

int res\_left, res\_right;

// Содаем два новых потока: левый и правый.

pthread\_create(&thread\_left, NULL, task\_proc, &task\_left);

pthread\_create(&thread\_right, NULL, task\_proc, &task\_right);

// Откладывание выполнение вызывающего данную функцию потока до тех пор, пока не будет выполнен поток thread.

pthread\_join(thread\_left, &rv\_left);

pthread\_join(thread\_right, &rv\_right);

// Результаты работы потоков в целочисленном значении.

res\_left = (int)(intptr\_t)rv\_left;

res\_right = (int)(intptr\_t)rv\_right;

// Возвращает сумму работы двух потоков.

// Если кто-то из них вернул -1, то это означает, что где-то доля наследства была неверной. И вернется -1 приведенный к void \*

// В противном случае мы вернем сумму результатов, приведенную к void\*

return (void \*)(intptr\_t)((res\_left != -1 && res\_right !=-1)?res\_left+res\_right:-1);}}

## Код программы

#include<cmath>

#include<pthread.h>

#include<cstdint>

#include<fstream>

#include<iostream>

struct inh\_share {

double ratio; // Соотношение.

int sum; // Значение наследства для одного человека.

};

// Структура куска массива на обработку.

struct inh\_task {

int total\_sum; // Сумма наследства.

int n; // Размер куска массива.

inh\_share \*part; // Адрес начала куска.

};

int check\_record(int total, const inh\_share \*ps)

{

// Происходит проверка на "честность" адвоката, то есть правильно ли произошло деление наследства.

// Если деление наследства произошла правильно, то вернется значение наследства для одного человека, иначе -1.

return (fabs(ps->ratio\*total-ps->sum) < 1)?ps->sum:-1;

}

void \*task\_proc(void \*arg)

{

inh\_task \*ptask = (inh\_task \*)arg;

// Если размер куска - одна запись, то он честно проверяет ее корректность и возвращает результат,

// приведенный к типу void\*.

if (ptask->n == 1)

return (void \*)(intptr\_t)check\_record(ptask->total\_sum, ptask->part);

else {

// Если же в куске больше элементов, то создаем 2 куска примерно одной длины: левый и правый. И запускаем нити на обработку.

int nleft = ptask->n>>1;

inh\_task task\_left = { ptask->total\_sum, nleft, ptask->part };

inh\_task task\_right = { ptask->total\_sum, ptask->n - nleft, ptask->part+nleft };

pthread\_t thread\_left, thread\_right;

void \*rv\_left, \*rv\_right;

int res\_left, res\_right;

// Содаем два новых потока: левый и правый.

pthread\_create(&thread\_left, NULL, task\_proc, &task\_left);

pthread\_create(&thread\_right, NULL, task\_proc, &task\_right);

// Откладывание выполнение вызывающего данную функцию потока до тех пор, пока не будет выполнен поток thread.

pthread\_join(thread\_left, &rv\_left);

pthread\_join(thread\_right, &rv\_right);

// Результаты работы потоков в целочисленном значении.

res\_left = (int)(intptr\_t)rv\_left;

res\_right = (int)(intptr\_t)rv\_right;

// Возвращает сумму работы двух потоков.

// Если кто-то из них вернул -1, то это означает, что где-то доля наследства была неверной. И вернется -1 приведенный к void \*

// В противном случае мы вернем сумму результатов, приведенную к void\*

return (void \*)(intptr\_t)((res\_left != -1 && res\_right !=-1)?res\_left+res\_right:-1);

}

}

int main() {

// Чтение из файла.

std::ifstream f("input.txt");

inh\_task t;

void \*rv;

int res, i;

// Количество наследников и сумма наследства, считанных из файла.

f>>t.n>>t.total\_sum;

// Доля.

t.part = new inh\_share[t.n\*sizeof(inh\_share)];

// Заполняем массив наследств и структуру для первого вызова. Вызываем.

// Если результат совпал с суммой наследства, то все довольны, и адвокат ничего не присвоил себе.

for (i=0; i<t.n; i++)

f>>t.part[i].ratio>>t.part[i].sum;

rv = task\_proc(&t);

res = (int)(intptr\_t)rv;

delete [] t.part;

// Проверка итога работы. Если все правильно, то программа возвращает 0, иначе 1.

if (res == t.total\_sum) {

std::cout<<"The inheritance shared correctly\n";

return 0;

} else {

std::cout<<"The inheritance shared incorrectly\n";

return 1;

}

}

// Комментарий 1: Есть проблема с ловлей ошибки округления. Когда не удается поделить сумму поровну так,

// чтобы при сложении всех кусков всегда получалась изначальная сумма.

// Комментарий 2: Предполагается, что у дона Энрике сумма наследства меньше, чем 2^31.

## Текст работы

Репозиторий на git: https://github.com/mikamurasaki2/ABC\_HW3

### Выполнение работы

Для проверки работоспособности программы рекомендуется использовать файл input.txt, в котором находятся входные данные.

По завершению работы программы на экран выведется одно из двух, в зависимости от входных данных (т.е был ли честным адвокат или нет):

1. The inheritance shared correctly
2. The inheritance shared incorrectly

# ПРИЛОЖЕНИЕ 1 СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. SoftCraft (разноликое программирование), Архитектура вычислительных систем: <http://www.softcraft.ru/edu/comparch/>
2. Онлайн ресурс: https://encyclopediaofmath.org/wiki/Dichotomy\_method
3. Требования к оформлению. 2020-2021 уч.г.: http://www.softcraft.ru/edu/comparch/tasks/t03/