МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

имени М. В. Ломоносова

Факультет вычислительной математики и кибернетики

Отчет по заданию курса
«Введение в распределенные системы и сети»

Статическое планирование выполнения задач

Выполнила:

студентка 203 группы

Волосникова Полина

Москва

2018

Содержание

Цель работы

Изучить метод статического планирования выполнения задач, применяемый в однопроцессорных вычислительных системах реального времени, предназначенных для обработки больших массивов данных.

Постановка задачи

Требуется написать программу, которая по входным данным, заданным в xml-файле, строит статическое расписание выполнения работ в соответствии с данной схемой построения расписания по правилу выбора работы с наименьшим временем завершения при условии старта в данный момент. Программа должна вывести результат в xml-файл.

Описание алгоритма

Для решения поставленной задачи был использован алгоритм, основанный на последовательном сдвиге вправо по оси времени и размещении работ на ней по мере сдвига точки планирования `t`.

Создается вектор `res`, в котором будут хранится элементы класса `Result` с описанием имени работы `name`, логической переменной `check`, обозначающей возможность выполнения работы, и временем старта `time`. Также используется массив `v` элементов класса `Job`, содержащими полное описание каждой работы, и очередь `q` с приоритетом, содержащая те работы, у которых директивный срок начала работы меньше `t`. Работы с меньшим временем обработки имеют больший приоритет.

Изначально массив `v` сортируется по директивному сроку начала работы, и переменная `t` устанавливается равной нулю.

Далее начинается сам алгоритм. Если очередь пуста, но не все работы были обработаны – это означает, что `t` не попадает в директивный интервал ни одной работы, тогда `t` устанавливается равной времени старта следующей необработанной работы (работы с наименьшим значением срока начала). Далее в очередь добавляются все работы, у которых срок начала работы меньше `t`. Затем извлекается верхний элемент очереди. Если данная работа может быть выполнена, то она добавляется в массив `res` с соответствующими значениями, и к `t` прибавляется значение времени выполнения выбранной работы. Иначе данная работа заносится в массив res с полем `check`, равным `false`.

Затем массив res сортируется в том порядке, в котором изначально стояли работы и выводит данные в файл output_static.xml.

Программа на C++

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <queue>
#include <string>
#include <cstring>
#include "tinyxml.h"

using namespace std;

// Класс, содержащий всю информацию о данной работе
class Job {
public:
    string name = "name";
    int time_start = 0, time_finish = 0, time_dur = 0, num = 0;
    Job() {}
};


// Класс, содержащий всю информацию, нужную для вывода
class Result {
public:
    string name;
    int num, time = 0;
    bool check;
    Result(string s, int n, int t = 0, bool ch = false) :     name(s),
num(n), time(t), check(ch) {}
};

// Компаратор для сортировки работ по времени начала
bool
cmp_job_sort(const Job &a, const Job &b) {
    return a.time_start < b.time_start;
}

// Компаратор для сортировки работ по порядоку их ввода
bool
cmp_res_sort(const Result &a, const Result &b) {
    return a.num < b.num;
}

// Приоритет для очереди работ - с наименьшим временем выполнения
struct cmp_job_q {
    bool operator() (const Job &a, const Job &b) const {
        return (a.time_dur > b.time_dur);
    }
};

int main()
{
    vector <Result> res;
    vector <Job> v;
    priority_queue <Job, vector <Job>, cmp_job_q> q;
    TiXmlDocument input;
```

```
    input.LoadFile("input_static.xml");
    TiXmlElement *p = input.FirstChildElement();
    int cnt = 0;
    for (TiXmlElement *tjob = p->FirstChildElement(); tjob != nullptr;
tjob = tjob->NextSiblingElement()) {
        ++cnt;
        Job j;
        j.num = cnt;
        j.name = tjob->Attribute("name");
        j.time_start = atoi(tjob->Attribute("start_deadline"));
        j.time_finish = atoi(tjob->Attribute("finish_deadline"));
        j.time_dur = atoi(tjob->Attribute("duration"));
        if (j.time_finish - j.time_start < j.time_dur) {
            res.push_back({ j.name, j.num });
        }
        else {
            v.push_back(j);
        }
    }
    int t = 0;

    // Сортировка работ по времени старта
    sort(v.begin(), v.end(), cmp_job_sort);
    unsigned int i = 0;

    // Составление расписания
    while (i < v.size()) {
        // Если на данный момент нет работ, которые могут выполняться,
то нужно передвинуть время
        if (!q.size()) {
            if (t <= v[i].time_start) {
                t = v[i].time_start;
            }
        }
        // Добавляем работу в очередь, если она может выполняться
        if (v[i].time_start <= t) {
            q.push(v[i]);
            ++i;
            continue;
        }
        // Выбор следующей работы
        if (q.top().time_finish >= t + q.top().time_dur) {
            res.push_back({ q.top().name, q.top().num, t, true });
            t += q.top().time_dur;
        }
        else {
            res.push_back({ q.top().name, q.top().num });
        }
        // Убираем из очереди использованную работу
        q.pop();
    }

    // Дообрабатываем очередь
    while (q.size()) {
        if (q.top().time_finish > t + q.top().time_dur) {
            res.push_back({ q.top().name, q.top().num, t, true });
            t += q.top().time_dur;
        }
```

```
        else {
            res.push_back({ q.top().name, q.top().num });
        }
        q.pop();
    }

    // Выставляем работы в исходный порядок
    sort(res.begin(), res.end(), cmp_res_sort);

    // Создаем файл и записываем в него результат
    TiXmlDocument output;
    TiXmlElement *tagtrace = new TiXmlElement("trace");
    output.LinkEndChild(tagtrace);
    for (auto j : res) {
        if (j.check) {
            TiXmlElement *tstart = new TiXmlElement("start");
            tstart->SetAttribute("name", j.name.c_str());
            tstart->SetAttribute("time", to_string(j.time).c_str());
            tagtrace->LinkEndChild(tstart);
        }
        else {
            TiXmlElement *tunsched = new TiXmlElement("unsched");
            tunsched->SetAttribute("name", j.name.c_str());
            tagtrace->LinkEndChild(tunsched);
        }
    }
    output.SaveFile("output_static.xml");
    return 0;
}
```

Тестирование

1. **Входной файл:**

<system>
<job name=”job1” start_deadline="0" finish_deadline="55" duration="5"/>
<job name=”job2” start_deadline="0" finish_deadline="40" duration="10"/>
<job name=”job3” start_deadline="45" finish_deadline="70" duration="20"/>
<job name=”job4” start_deadline="45" finish_deadline="70" duration="20"/>
</system>

Выходной файл:

<trace>
  <start name=””job1”” time="0" />
  <start name=””job2”” time="5" />
  <start name=””job3”” time="45" />
  <unsched name=””job4”” />
</trace>

2. **Входной файл:**

<system>
<job name=”job1” start_deadline="0" finish_deadline="25" duration="5"/>
<job name=”job2” start_deadline="0" finish_deadline="20" duration="10"/>
<job name=”job3” start_deadline="0" finish_deadline="25" duration="5"/>
<job name=”job4” start_deadline="0" finish_deadline="25" duration="5"/>
<job name=”job5” start_deadline="0" finish_deadline="25" duration="5"/>
</system>

Выходной файл:

<trace>
  <start name=””job1”” time="0" />
  <unsched name=””job2”” />
  <start name=””job3”” time="5" />
  <start name=””job4”” time="15" />
  <start name=””job5”” time="10" />
</trace>

3. Данный тест

Входной файл:

<system>
<job name=”job1” start_deadline="0" finish_deadline="55" duration="5"/>
<job name=”job2” start_deadline="0" finish_deadline="40" duration="10"/>
<job name=”job3” start_deadline="0" finish_deadline="45" duration="20"/>
<job name=”job4” start_deadline="85" finish_deadline="105" duration="15"/>
<job name=”job5” start_deadline="90" finish_deadline="110" duration="15"/>
<job name=”job6” start_deadline="100" finish_deadline="155" duration="5"/>
<job name=”job7” start_deadline="100" finish_deadline="140" duration="10"/>
<job name=”job8” start_deadline="100" finish_deadline="145" duration="20"/>
</system>

Выходной файл:

```
<trace>
   <start name=""job1”" time="0" />
   <start name=""job2”" time="5" />
   <start name=""job3”" time="15" />
   <start name=""job4”" time="85" />
   <unsched name=""job5”" />
   <start name=""job6”" time="100" />
   <start name=""job7”" time="105" />
   <start name=""job8”" time="115" />
</trace>
```

Выводы

Из тестов видно, что происходит дискриминация тех работ, у которых время выполнения больше. Это означает, что при таком методе вычислительная система реального времени будет обрабатывать мелкие работы, а работы, требующие большего времени, будут поступать на обработку в конце или вовсе не обрабатываться.