Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

ОТЧЁТ

к лабораторной работе №1

на тему

**УПРАВЛЕНИЕ ПОТОКАМИ, ПРОЦЕССАМИ, НИТЯМИ**

Студент: Ющук И.А.

Преподаватель: Гриценко Н.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Формулировка задачи 3](#_Toc177507035)

[2 Описание функций программы 4](#_Toc177507036)

[2.1 Парсинг файла 4](#_Toc177507037)

[2.2 Выполнение внешних программ 4](#_Toc177507038)

[2.3 Обработка ошибок 5](#_Toc177507039)

[Заключение 6](#_Toc177507040)

[Список использованных источников 7](#_Toc177507041)

[Приложение А (обязательное) Исходный код программы 8](#_Toc177507042)

1 ФОРМУЛИРОВКА ЗАДАЧИ

Выполнение внешних программ в соответствии с заданным расписанием (конфигурацией), загружаемым из файла/файлов (формат можно заимствовать от cron или использовать свой упрощенный). Расписание содержит имена исполняемых файлов, время (условие) их выполнения, параметры запуска. Необходимо обрабатывать возможные ошибки выполнения.

2 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Согласно формулировке задачи, были спроектированы следующие функции программы:

* Парсинг файла с расписанием запуска, именем и параметрами программ.
* Функцию выполнения внешних программ.
* Функция вывода ошибок исполнения.

## **2.1 Парсинг файла**

При запуске программы открывается заданный текстовый файл и из него берутся данные для создания вектора из структур задач (рисунок 2.1).

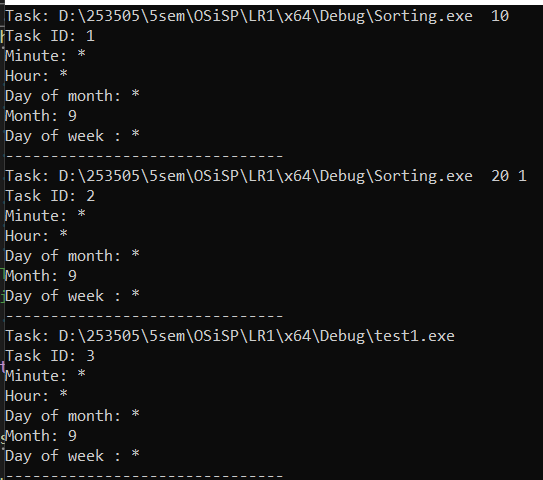
****

Рисунок 2.1 – Результат парсинга файла

## **2.2 Выполнение внешних программ**

После парсинга начинает выполняться бесконечный цикл на каждой итерации выполняется проверка какие внешние программы нужно выполнить и запускает в случае, если совпадает время запуска с текущим временем. В консоль выводится информации о запуске программы. (рисунок 2.2).

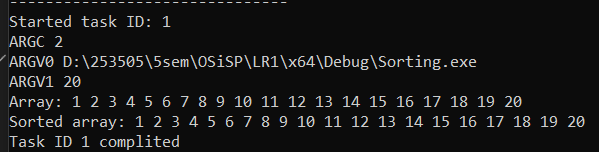


Рисунок 2.2 – Вывод в консоль сообщения р начале, процессе и окончании выполнения

## **2.3 Обработка ошибок**

В случае если путь к исполняемому файлу указан неверно или в процессе выполнения программы возникает ошибка информация об ошибке выводтся в консоль (рисунок 2.3).

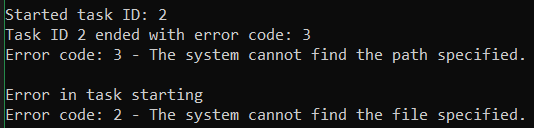


Рисунок 2.3 – Вывод ошибок исполнения

ЗАКЛЮЧЕНИЕ

Лабораторная работа по созданию упрощенного аналога планировщика cron показала, что все заявленные функциональные требования успешно выполнены. Планировщик демонстрирует корректную работу с конфигурационными файлами, правильно выполняет задачи по расписанию, обрабатывает ошибки и ведет логирование. Важно отметить, что проект выполнен с соблюдением основных принципов надежности и безопасности, что обеспечивает стабильную работу системы при различных сценариях выполнения задач.

В дальнейшем рекомендуется рассмотреть возможность расширения функциональности. Например, можно добавить поддержку более сложных форматов конфигураций, таких как YAML или JSON, что позволит пользователю гибче настраивать расписания. Также можно разработать графический интерфейс для упрощения процесса конфигурации и визуализации выполнения задач. Дополнительно стоит рассмотреть реализацию функций мониторинга и уведомлений о статусе задач, таких как отправка сообщений на электронную почту или через мессенджеры, что повысит информированность пользователей.

Одним из направлений улучшения может стать поддержка многопоточности и распределенного выполнения задач для оптимизации работы в многопроцессорных системах и сетевых кластерах. Это обеспечит выполнение задач на нескольких устройствах с централизованным управлением и синхронизацией. Кроме того, стоит обратить внимание на улучшение механизма обработки ошибок, включая автоматическое восстановление прерванных задач и создание системы приоритетов для выполнения задач в зависимости от их важности.

Наконец, оптимизация использования системных ресурсов, таких как оперативная память и процессорное время, позволит повысить производительность планировщика, особенно при увеличении количества задач и объема данных. Это позволит использовать разработанный планировщик в более сложных и нагруженных системах, что расширит его применимость и возможности в реальных проектах.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] CreateProcess function [Электронный ресурс].

– Электронный ресурс. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/procthread/creating-processes

[2] GetExitCodeProcess function [Электронный ресурс].

– Электронный ресурс. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-getexitcodeprocess

[3] GetLastError function [Электронный ресурс].

– Электронный ресурс. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/errhandlingapi/nf-errhandlingapi-getlasterror

ПРИЛОЖЕНИЕ А  
(обязательное)  
Исходный код программы

#include <windows.h>

#include <commdlg.h>

#include <iostream>

#include <fstream>

#include <sstream>

#include <string>

#include <vector>

#include <ctime>

#include <algorithm>

#include "Parsing.h"

using namespace std;

void logError(DWORD errorCode) {

LPVOID errorMessageBuffer = nullptr;

// Получаем сообщение об ошибке, соответствующее коду ошибки

FormatMessage(

FORMAT\_MESSAGE\_ALLOCATE\_BUFFER | FORMAT\_MESSAGE\_FROM\_SYSTEM | FORMAT\_MESSAGE\_IGNORE\_INSERTS,

NULL,

errorCode,

MAKELANGID(LANG\_NEUTRAL, SUBLANG\_DEFAULT), // Язык по умолчанию

(LPWSTR)&errorMessageBuffer, // Указатель на буфер, который будет заполнен строкой

0, // Размер буфера (0 для автоматического определения)

NULL

);

std::wcout << L"Error code: " << errorCode << L" - " << (LPWSTR)errorMessageBuffer << std::endl;

}

bool runExternalProgram(Task& task) {

STARTUPINFO si = { sizeof(si) };

PROCESS\_INFORMATION pi;

/\* si.dwFlags = STARTF\_USESTDHANDLES;

si.hStdInput = GetStdHandle(STD\_INPUT\_HANDLE);

si.hStdOutput = GetStdHandle(STD\_OUTPUT\_HANDLE);

si.hStdError = GetStdHandle(STD\_ERROR\_HANDLE);\*/

string command = task.program + " "+ task.args;

wstring commandW(command.begin(), command.end());

if (!CreateProcess(

NULL, // Имя исполняемого файла

&commandW[0], // Командная строка

NULL, // Атрибуты защиты процесса

NULL, // Атрибуты защиты потока

TRUE, // Наследование дескрипторов

0, // Флаги создания

NULL, // Указатель на блок переменных окружения

NULL, // Текущий каталог

&si, // Структура STARTUPINFO

&pi)) { // Структура PROCESS\_INFORMATION

DWORD errorCode = GetLastError();

cerr << "Error in task starting\n";

logError(errorCode);

//logError("Error in task start. Error code: ");

return false;

}

cout << "Started task ID: " << task.taskId << std::endl;

WaitForSingleObject(pi.hProcess, INFINITE);

DWORD exitCode;

if (GetExitCodeProcess(pi.hProcess, &exitCode)) {

if (exitCode == 0) {

cout << "Task ID " << task.taskId << " complited" << endl<<endl;

}

else {

cerr << "Task ID " << task.taskId << " ended with error code: " << exitCode << endl;

try {

logError(exitCode);

}

catch (...)

{

}

}

}

CloseHandle(pi.hProcess);

CloseHandle(pi.hThread);

return true;

}

//int compareTasks(const SYSTEMTIME& currentTime, const Task& task1, const Task& task2) {

//

// int monthDiff1 = task1.month == -1 ? 0 : (currentTime.wMonth <= task1.month ? task1.month - currentTime.wMonth : 12 - currentTime.wMonth + task1.month);

// int monthDiff2 = task2.month == -1 ? 0 : (currentTime.wMonth <= task2.month ? task2.month - currentTime.wMonth : 12 - currentTime.wMonth + task2.month);

// if (monthDiff1 != monthDiff2)

// {

// return monthDiff1 < monthDiff2 ? 1 : -1;

// }

//

// int dayDiff1 = task1.dayOfMonth== -1 ? 0: (abs(currentTime.wDay - task1.dayOfMonth));

// int dayDiff2 = task2.dayOfMonth == -1 ? 0 : (abs(currentTime.wDay - task2.dayOfMonth));

// if (dayDiff1 != dayDiff2)

// {

// return dayDiff1 < dayDiff2 ? 1 : -1;

// }

//

// int hourDiff1 = task1.hour == -1 ? 0 : (hourDiff1 = abs(currentTime.wHour - task1.hour));

// int hourDiff2 = task2.hour == -1 ? 0 : (abs(currentTime.wHour - task2.hour));

// if (hourDiff1 != hourDiff2)

// {

// return hourDiff1 < hourDiff2 ? 1 : -1;

// }

//

// int minuteDiff1 = task1.minute == -1 ? 0 : (abs(currentTime.wMinute - task1.minute));

// int minuteDiff2 = task2.minute == -1 ? 0 : (abs(currentTime.wMinute - task2.minute));

// if (minuteDiff1 != minuteDiff2)

// {

// return minuteDiff1 < minuteDiff2 ? 1 : -1;

// }

// return 0;

//}

bool shouldRunTask(const SYSTEMTIME& currentTime, const Task& task)

{

if (task.minute != -1 && task.minute != currentTime.wMinute)

return false;

if (task.hour != -1 && task.hour != currentTime.wHour)

return false;

if (task.dayOfMonth != -1 && task.dayOfMonth != currentTime.wDay)

return false;

if (task.month != -1 && task.month != currentTime.wMonth)

return false;

//TODO dayOfWeek not work yet

/\*if (task.dayOfWeek != -1 && task.dayOfWeek != currentTime.wDayOfWeek)

return false;\*/

return true;

}

bool isOneTimeTask(const Task& task)

{

if (task.minute == -1)

return false;

if (task.hour == -1)

return false;

if (task.dayOfMonth == -1)

return false;

if (task.month == -1)

return false;

//TODO

//if (task.dayOfWeek == -1) return false;

return true;

}

void checkAndRunTasks(vector<Task>& tasks)

{

SYSTEMTIME currentTime;

GetLocalTime(&currentTime);

for (auto& task : tasks)

{

if (shouldRunTask(currentTime, task))

{

/\* thread thread(runExternalProgram, task);

thread.join();\*/

runExternalProgram(task);

if(isOneTimeTask(task))

{

task.shouldRemove = true;

}

}

}

tasks.erase(remove\_if(tasks.begin(), tasks.end(), [](const Task& task)

{

return task.shouldRemove;

}), tasks.end());

}

void infiniteCicle(vector<Task>& tasks)

{

SYSTEMTIME t1, t2;

while (true)

{

checkAndRunTasks(tasks);

Sleep(1000 \* 5);

}

}

int main()

{

string configFilePath = "schedule.txt";

vector<Task> tasks = parseConfigFile(configFilePath);

printTasks(tasks);

infiniteCicle(tasks);

//runExternalProgram(tasks[0]);

return 0;

}