СПб НИУ ИТМО

кафедра ИПМ

Системное программное обеспечение

Лабораторная работа № 4

Управление процессами

Работу выполнил:

Студент II курса

Группы № 2120

Журавлев Виталий

Санкт-Петербург

2014 г.

**Цель работы:**

Расширьте возможности программы 6.1 “Параллельный поиск указанного текстового шаблона” (grepMP) таким образом, чтобы она принимала также параметры командной строки, а не только текстовый шаблон.

**Теоритические основы:**

*Процесс* представляет собой объект, обладающий собственным независимым виртуальным адресным пространством, в котором могут размещаться код и данные, защищенные от других процессов. В свою очередь, внутри каждого процесса могут независимо выполняться одна или несколько *потоков* (threads). Поток, выполняющийся внутри процесса, может сама создавать новые потоки и новые независимые процессы, а также управлять взаимодействием объектов между собой и их синхронизацией.

Создавая процессы и управляя ими, приложения могут организовывать параллельное выполнение нескольких задач, обеспечивающих обработку файлов, проведение вычислений или связь с другими системами в сети. Допускается даже использование нескольких процессоров с целью ускорения обработки данных.

Программа grepMP создает процессы для поиска указанного текстового шаблона (или параметра командной строки) в файлах, по одному процессу на каждый файл. Эта программа моделирует UNIX-утилиту grep, хотя используемая методика применима к любой программе, которая полагается на стандартный вывод. Рассматривается программа поиска как "черный ящик" и считается, что она является просто исполняемой программой, выполнение которой должно контролироваться родительским процессом.

Командная строка программы имеет следующий вид:

grepMP *шаблон/параметр* F1 F2 … FN

Программа выполняет следующие виды обработки:

• Для поиска указанного шаблона в каждом из входных файлов, от F1 до FN, используется отдельный процесс, запускающий один и тот же исполняемый модуль. Для каждого процесса программа создает командную строку такого вида: grep *шаблон* FK.

• Полю hStdOut структуры STARTUPINFO нового процесса присваивается значение дескриптора временного файла, который определяется как наследуемый.

• Программа организует ожидание завершения всех процессов поиска, используя для этого функцию WaitForMultipleObjects.

• По завершении всех процессов поиска осуществляется поочередный вывод результатов. Вывод временного файла осуществляет процесс, выполняющий утилиту cat.

• Возможности функции WaitForMultipleObjects ограничиваются лишь максимально допустимым количеством дескрипторов, которое устанавливается значением MAXIMUM\_WAIT\_OBJECTS (64), поэтому она вызывается многократно.

• Для определения успешности попытки нахождения данным процессом заданного шаблона программа использует код завершения процесса grep.

**Код программы:**

#include "stdafx.h"

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#include <iostream>

using namespace std;

int \_tmain(DWORD argc, LPTSTR argv[])

{

HANDLE hTempFile;

SECURITY\_ATTRIBUTES StdOutSA = { sizeof(SECURITY\_ATTRIBUTES), NULL, TRUE };

TCHAR CommandLine[MAX\_PATH + 100];

STARTUPINFO StartUpSearch, StartUp;

PROCESS\_INFORMATION ProcessInfo;

DWORD iProc, ExCode;

HANDLE \*hProc = new HANDLE[argc - 2];

typedef struct { TCHAR TempFile[MAX\_PATH]; } PROCFILE;

PROCFILE \*ProcFile = new PROCFILE[argc - 2];

GetStartupInfo(&StartUpSearch);

GetStartupInfo(&StartUp);

for (iProc = 0; iProc < argc - 2; iProc++) {

\_stprintf(CommandLine, \_T("%s%s %s"), \_T("findstr "), argv[1], argv[iProc + 2]);

GetTempFileName(\_T("."), \_T("gtm"), 0, ProcFile[iProc].TempFile);

hTempFile = CreateFile(ProcFile[iProc].TempFile, GENERIC\_WRITE | GENERIC\_READ, FILE\_SHARE\_READ | FILE\_SHARE\_WRITE, &StdOutSA, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

StartUpSearch.dwFlags = STARTF\_USESTDHANDLES;

StartUpSearch.hStdOutput = hTempFile;

StartUpSearch.hStdError = hTempFile;

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

CreateProcess(NULL, CommandLine, NULL, NULL, TRUE, 0, NULL, NULL, &StartUpSearch, &ProcessInfo);

hProc[iProc] = ProcessInfo.hProcess;

}

for (iProc = 0; iProc < argc - 2; iProc += MAXIMUM\_WAIT\_OBJECTS) WaitForMultipleObjects(

min(MAXIMUM\_WAIT\_OBJECTS, argc - 2 - iProc), &hProc[iProc], TRUE, INFINITE);

for (iProc = 0; iProc < argc - 2; iProc++) {

if (GetExitCodeProcess(hProc[iProc], &ExCode) && ExCode == 0) {

if (argc > 3) \_tprintf(\_T("%s:\n"), argv[iProc + 2]);

HANDLE FileHandle = CreateFile(ProcFile[iProc].TempFile, GENERIC\_READ | GENERIC\_WRITE, FILE\_SHARE\_READ | FILE\_SHARE\_WRITE, NULL, OPEN\_EXISTING, 0, NULL);

HANDLE MapHandle = CreateFileMapping(FileHandle, NULL, PAGE\_READWRITE, 0, 0, NULL);

DWORD FileSize = GetFileSize(FileHandle, NULL);

char \*memory = (char\*)MapViewOfFile(MapHandle, FILE\_MAP\_WRITE, 0, 0, FileSize);

printf("%s \n", memory);

}

CloseHandle(hProc[iProc]);

DeleteFile(ProcFile[iProc].TempFile);

}

free(ProcFile);

free(hProc);

return 0;

}

**Вывод:**

В ходе выполнения лабораторной работы были изучены основы управления процессами и реализация определенного алгоритма несколькими процессами для работы одновременно с несколькими файлами (по одному процессу на файл).

Так же были рассмотрены Unix-утилиты grep и cat и их аналоги для windows.