МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»

Факультет радіоелектроніки, комп'ютерних систем та інфокомунікацій

Кафедра комп'ютерних систем, мереж і кібербезпеки

Лабораторна робота

з Системного програмування
(назва дисципліни)

на тему: «Вивчення системних викликів Win32 API для роботи з процесами та потоками»

Виконав: студент 3-го курсу групи №525ст2
напряму підготовки (спеціальності)
123-«Комп'ютерна інженерія»
(шифр і назва напряму підготовки (спеціальності))
Золотопуп А.С.
(прізвище й ініціали студента)
Прийняв: асистент каф.503
Мозговий М.В.
(посада, науковий ступінь, прізвище й ініціали)

Національна шкала: _____
Кількість балів: _____
Оцінка: ECTS _____

Харків – 2020

Цель работы:

Изучение системных вызовов Win32 API работы с процессами, создание дочерних процессов.

Изучение системных вызовов по работе с потоками. Использование TLS памяти потока.

Постановка задачи:

Программа 1:

Написать программу, реализующую упаковку и распаковку zip архивов. Программа должна использовать утилиту 7z.exe, которая будет непосредственно выполнять упаковку и распаковку файлов путем запуска в дочернем процессе. Программа должна поддерживать такие операции как:

1. Распаковка архива в папку

2. Упаковка одного файла в новый архив

Для получения максимальной оценки необходимо выполнить обработку ошибок от дочернего процесса путем перенаправления потока вывода. Это позволит родительскому процессу получить содержимое консоли, сформированное программой 7z.exe и по этому тексту определить была ошибка или нет.

Программа 2:

Написать программу, которая может создавать 2 и более потоков (кол-во задается в командной строке). Перед запуском потоков программа заполняет для каждого потока исходный массив целочисленных значений (5-10 элементов) от 10 до 100. Каждый поток должен найти для каждого элемента массива его наибольший делитель, сохраняя полученные значения в TLS память. После нахождения всех значений он должен вывести сумму всех полученных значений и напечатать свой идентификатор. Расчет наибольшего делителя и вычисление конечной суммы должны реализовываться двумя отдельными функциями.

Код программы:

```
#define _CRT_SECURE_NO_WARNINGS
#include <Windows.h>
#include <stdio.h>
#include <locale.h>
using namespace std;

#define PACK 1
#define UNPACK 2
```

```
#define EXIT_APP 0


LPCSTR unpackCommand = "7z.exe e ";
LPCSTR packCommand = "7z.exe a -tzip ";

void UnpackFiles(LPSTR unpackFile, LPSTR resultFile);
void PackFiles(LPSTR unpackFile, LPSTR resultFile);
void ShowError();

int main()
{
	int menu;
	while (true)
	{
		cout << "[" << PACK << "] Pack files" << endl;
		cout << "[" << UNPACK << "] Unpack files" << endl;
		cout << "[" << EXIT_APP << "] Exit" << endl;
		cin >> menu;
		switch (menu)
		{
		case UNPACK:
		{
			LPSTR unpackFile = new CHAR[MAX_PATH];
			LPSTR resultFile = new CHAR[MAX_PATH];

			printf("Input full path to zip\n");
			scanf("%s", unpackFile);
			printf("Input full path to directory\n");
			scanf("%s", resultFile);
			UnpackFiles(unpackFile, resultFile);
			break;
		}
		case PACK:
		{
			LPSTR packFile = new CHAR[MAX_PATH];
			LPSTR resultFile = new CHAR[MAX_PATH];

			printf("Input full path to File/Directory\n");
			scanf("%s", packFile);
			printf("Input full path to archive (.zip)\n");
			scanf("%s", resultFile);
			PackFiles(packFile, resultFile);
			break;
```

```
		}
		case EXIT_APP: {
			return 0;
		}

		default: printf("Invalid input!\n"); break;
		}
		system("pause");
		system("cls");
	}
}

void UnpackFiles(LPSTR unpackFile, LPSTR resultFile)
{
	LPSTR commandLine = new CHAR[MAX_PATH];
	ZeroMemory(commandLine, MAX_PATH);

	strncpy(commandLine, unpackCommand, MAX_PATH - strlen(commandLine));
	strncat(commandLine, unpackFile, MAX_PATH - strlen(commandLine));

	strncat(commandLine, " -o", MAX_PATH - strlen(commandLine));
	strncat(commandLine, resultFile, MAX_PATH - strlen(commandLine));

	strncat(commandLine, " -y", MAX_PATH - strlen(commandLine));

	HANDLE hReadPipe;
	HANDLE hWritePipe;

	SECURITY_ATTRIBUTES saAttr;
	saAttr.nLength = sizeof(SECURITY_ATTRIBUTES);
	saAttr.bInheritHandle = TRUE;
	saAttr.lpSecurityDescriptor = NULL;

	CreatePipe(&hReadPipe, &hWritePipe, &saAttr, 0);
	if (hReadPipe == INVALID_HANDLE_VALUE || hWritePipe ==
INVALID_HANDLE_VALUE)
		exit(1);

	if (!SetHandleInformation(hReadPipe, HANDLE_FLAG_INHERIT, 0))
		exit(1);

	STARTUPINFOA si;
	ZeroMemory(&si, sizeof(STARTUPINFOA));
	si.cb = sizeof(si);
```

```
	si.hStdError = hWritePipe;
	si.dwFlags |= STARTF_USESTDHANDLES;

	PROCESS_INFORMATION pi;
	ZeroMemory(&pi, sizeof(pi));

	if (!CreateProcessA(NULL, commandLine, NULL, NULL, TRUE, 0, NULL,
NULL, &si, &pi))
	{
		ShowError();
	}
	else
	{
		WaitForSingleObject(pi.hProcess, INFINITE);
		DWORD readed = 0;
		LPSTR result = new CHAR[1024];
		ZeroMemory(result, 1024);
		OVERLAPPED overlapped;
		while (ReadFile(hReadPipe, result, 1024, &readed, &overlapped))
		{
			printf("%s", result);
		}
	}

	CloseHandle(hReadPipe);
	CloseHandle(hWritePipe);
	CloseHandle(pi.hProcess);
	CloseHandle(pi.hThread);
}
void PackFiles(LPSTR packFile, LPSTR resultFile)
{

	LPSTR commandLine = new CHAR[MAX_PATH];
	ZeroMemory(commandLine, MAX_PATH);
	strncpy(commandLine, packCommand, MAX_PATH - strlen(commandLine));
	strncat(commandLine, resultFile, MAX_PATH - strlen(commandLine));

	strncat(commandLine, " ", MAX_PATH - strlen(commandLine));
	strncat(commandLine, packFile, MAX_PATH - strlen(commandLine));

	HANDLE hReadPipe;
	HANDLE hWritePipe;

	SECURITY_ATTRIBUTES saAttr;
```

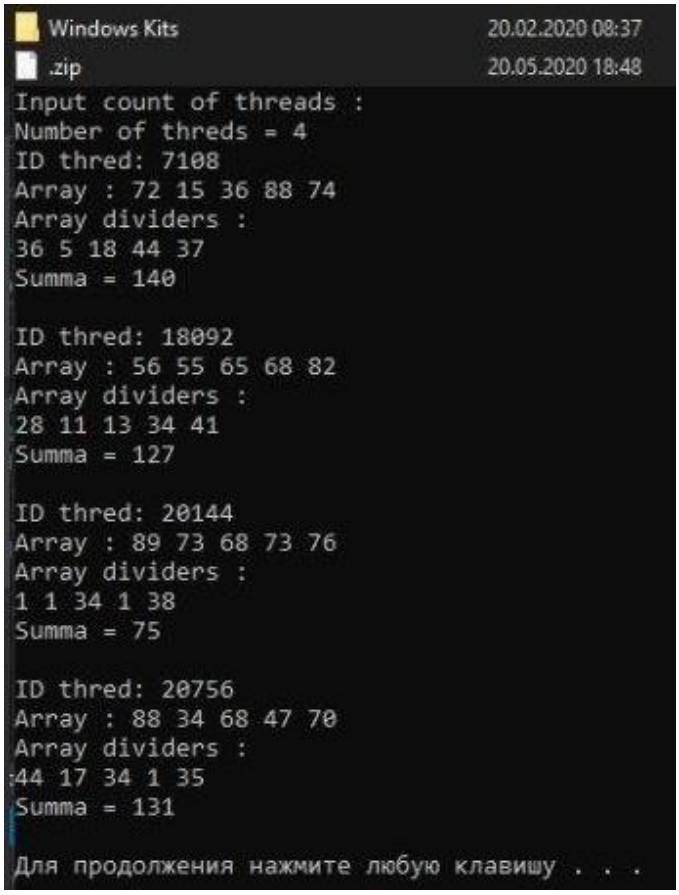
```
	saAttr.nLength = sizeof(SECURITY_ATTRIBUTES);
	saAttr.bInheritHandle = TRUE;
	saAttr.lpSecurityDescriptor = NULL;

	CreatePipe(&hReadPipe, &hWritePipe, &saAttr, 0);
	if (hReadPipe == INVALID_HANDLE_VALUE || hWritePipe ==
INVALID_HANDLE_VALUE)
		exit(1);

	if (!SetHandleInformation(hReadPipe, HANDLE_FLAG_INHERIT, 0))
		exit(1);

	STARTUPINFOA si;
	ZeroMemory(&si, sizeof(STARTUPINFOA));
	si.cb = sizeof(si);
	si.hStdError = hWritePipe;
	si.dwFlags |= STARTF_USESTDHANDLES;

	PROCESS_INFORMATION pi;
	ZeroMemory(&pi, sizeof(pi));


	if (!CreateProcessA(NULL, commandLine, NULL, NULL, TRUE,
NORMAL_PRIORITY_CLASS, NULL, NULL, &si, &pi)) {
		ShowError();
	}
	else
	{
		WaitForSingleObject(pi.hProcess, INFINITE);
		DWORD readed = 0;
		LPSTR result = new CHAR[1024];
		ZeroMemory(result, 1024);
		OVERLAPPED overlapped;
		while (ReadFile(hReadPipe, result, 1024, &readed, &overlapped))
		{
			printf("%s", result);
		}
	}
	CloseHandle(hReadPipe);
	CloseHandle(hWritePipe);
	CloseHandle(pi.hProcess);
	CloseHandle(pi.hThread);
}
```

```
void ShowError()
{
	LPVOID msg;
	DWORD e_code = GetLastError();
	FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER |
FORMAT_MESSAGE_FROM_SYSTEM,
		NULL,
		e_code,
		MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
		(LPTSTR)&msg,
		0,
		NULL
	);
	wprintf(L"\nERROR : %s\n", (char*)msg);
}
#define _CRT_SECURE_NO_WARNINGS
#include <Windows.h>
#include <stdio.h>
#include <locale.h>
using namespace std;

#define ARRAY_MAX 5

CRITICAL_SECTION criticalSection;
HANDLE* threads;
int tlsIndex;


void startThreds(int count);
DWORD WINAPI threadAction(LPVOID arr);
void FindDivider(int* mainArr, int* resultArr);
int GreatestCommonFactor(int num);
int ArraySum(int* arr);
void ShowArray(int* arr);



int main()
{
	//srand(time(0));
	int countThreds;
	cout << "Input count of threads :" << endl;
	cout << "Number of threds = ";
	cin >> countThreds;
```

```
	threads = new HANDLE[countThreds];
	InitializeCriticalSection(&criticalSection);
	startThreds(countThreds);
	WaitForMultipleObjects(countThreds, threads, TRUE, INFINITE);
	DeleteCriticalSection(&criticalSection);
	system("pause");
	return 0;
}


int GreatestCommonFactor(int num)
{
	int j = num / 2;
	for (int i = j; i >= 2; i--)
	{
		if (num % i == 0)
			return i;
	}
}

int ArraySum(int* arr)
{
	int result = 0;
	for (int i = 0; i < ARRAY_MAX; i++)
	{
		result += arr[i];
	}
	return result;
}

void ShowArray(int* arr)
{
	for (int i = 0; i < ARRAY_MAX; i++) {
		printf("%i ", arr[i]);
	}
}


void startThreds(int count)
{
	InitializeCriticalSection(&criticalSection);
	int** arrays = new int*[count];
	tlsIndex = TlsAlloc();
	for (int i = 0; i < count; i++)
```

```
	{
		arrays[i] = new int[ARRAY_MAX];
		for (int j = 0; j < ARRAY_MAX; j++)
			arrays[i][j] = rand() % 90 + 10;

		threads[i] = CreateThread(NULL, 0, threadAction, arrays[i], NULL,
NULL);
	}
}

DWORD WINAPI threadAction(LPVOID param)
{
	int sum = 0;

	EnterCriticalSection(&criticalSection);
	printf("ID thred: %u\n", GetCurrentThreadId());

	int* array = (int*)param;
	TlsSetValue(tlsIndex, (LPVOID)(new int[ARRAY_MAX]));
	printf("Array : ");
	ShowArray(array);
	FindDivider(array, (int*)TlsGetValue(tlsIndex));
	printf("\nArray dividers :\n");

	ShowArray((int*)TlsGetValue(tlsIndex));
	sum = ArraySum((int*)TlsGetValue(tlsIndex));
	printf("\nSumma = %d\n\n", sum);

	LeaveCriticalSection(&criticalSection);
	return 0;
}

void FindDivider(int* inArr, int* outArr)
{
	for (int i = 0; i < ARRAY_MAX; i++)
	{
		outArr[i] = GreatestCommonFactor(inArr[i]);
	}
}
```

Результат работы:



Выводы:

В результате выполнения данной лабораторной работы были изучены системные вызовы Win32 API работы с процессами; создание дочерних процессов, а также системных вызовов по работе с потоками. и использование
TLS памяти потока.