Министерство науки и высшего образования РФ

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №1

по дисциплине «Основы операционных систем»

на тему «Порождение и завершение процессов и нитей»

Вариант 13

Выполнили: студенты группы 22ВВП1

Беляев Д. И.

Демин М. С.

Приняли:

Егоров В. Ю.

Федюнин Р. Н.

Пенза 2024

**Цель работы:** Изучение методов и средств порождения процессов и нитей. Изучение способов синхронизации процессов и нитей через ожидание окончания их выполнения.

**Задание:** Породить цепочку из 10 процессов. Предыдущие процессы в цепочке должны завершаться по завершении любого из следующих.

**Описание данных:**

В данном коде реализована программа, которая создает несколько процессов, каждый из которых выполняет определенные задачи. Основные данные и параметры, используемые в программе:

1. Максимальное количество процессов:

- MAX\_PROCESS\_COUNT: Максимальное количество дочерних процессов, которые могут быть созданы. В данном случае — 10.

2. Время сна:

- MIN\_SLEEP\_TIME: Минимальное время (в миллисекундах), в течение которого процесс будет "спать" (10000 мс = 10 секунд).

- MAX\_SLEEP\_TIME: Максимальное время (в миллисекундах), в течение которого процесс будет "спать" (20000 мс = 20 секунд).

3. Аргументы командной строки:

- processNum: Номер текущего процесса, передаваемый через аргументы командной строки.

- prevPid: Идентификатор предыдущего процесса, также передаваемый через аргументы командной строки.

4. Структуры Windows API:

- PROCESS\_INFORMATION: Структура, содержащая информацию о процессе и его потоке.

- STARTUPINFO: Структура, содержащая информацию о том, как должен быть создан новый процесс.

**Описание структуры программы:**

Программа состоит из нескольких функций, каждая из которых отвечает за определенную задачу:

1. startNextProcess:

- Создает новый дочерний процесс с помощью функции CreateProcess.

- Формирует командную строку для запуска нового процесса, передавая ему номер текущего процесса и идентификатор предыдущего процесса.

- Возвращает структуру PROCESS\_INFORMATION, содержащую информацию о созданном процессе.

2. terminatePrevProcess:

- Завершает предыдущий процесс по его идентификатору.

- Использует функцию OpenProcess для получения дескриптора процесса и TerminateProcess для его завершения.

3. fakeRun:

- Выполняет основную работу текущего процесса.

- Вычисляет оставшееся время работы на основе номера процесса и случайного времени сна.

- Если это последний процесс, то он "спит" на рассчитанное время.

- В противном случае ожидает завершения дочернего процесса с помощью WaitForSingleObject.

4. main:

- Основная функция программы.

- Получает идентификатор текущего процесса и устанавливает начальные параметры (время сна и номер процесса).

- Запускает следующий процесс с помощью startNextProcess и выполняет работу текущего процесса с помощью fakeRun.

**Листинг:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <windows.h>

#include <time.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_PROCESS\_COUNT 10

#define MIN\_SLEEP\_TIME 10000

#define MAX\_SLEEP\_TIME 20000

PROCESS\_INFORMATION startNextProcess(int processNum, int prevPid)

{

PROCESS\_INFORMATION pi;

STARTUPINFO si;

ZeroMemory(&si, sizeof(si));

si.cb = sizeof(si);

TCHAR cmd[MAX\_PATH];

GetModuleFileName(NULL, cmd, sizeof(cmd) / sizeof(cmd[0]));

char src[MAX\_PATH];

sprintf(src, "code.exe %d %d", processNum, prevPid);

TCHAR cmdArgs[MAX\_PATH] = { 0, };

MultiByteToWideChar(CP\_ACP, MB\_PRECOMPOSED, src, strlen(src), cmdArgs, MAX\_PATH);

if (!CreateProcess(cmd, cmdArgs, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi)) {

printf("CreateProcess failed (%d).\n", GetLastError());

return;

}

return pi;

}

int terminatePrevProcess(int pid, int num)

{

HANDLE process = OpenProcess(PROCESS\_TERMINATE, FALSE, pid);

TerminateProcess(process, 1);

CloseHandle(process);

printf("Process %d terminate by %d\n", num - 1, num);

}

void fakeRun(int num, int runTime, PROCESS\_INFORMATION subPi)

{

runTime -= 1000 \* num;

char\* resultString = "Process %d end work\n";

printf("Process %d start work for %d\n", num, runTime);

if (num == MAX\_PROCESS\_COUNT - 1) {

Sleep(runTime);

}

else {

DWORD exCode = WaitForSingleObject(subPi.hProcess, runTime);

if (exCode != WAIT\_TIMEOUT) {

resultString = "Process %d terminate\n";

}

}

printf(resultString, num);

}

int main(int argc, char\* argv[])

{

int selfpid = GetCurrentProcessId();

srand(time(NULL));

int runTime = rand() % (MAX\_SLEEP\_TIME - MIN\_SLEEP\_TIME + 1) + MIN\_SLEEP\_TIME;

int processNum = argc > 1 ? atoi(argv[1]) : 0;

int prevPid = argc > 1 ? atoi(argv[2]) : -1;

PROCESS\_INFORMATION subPi;

if (processNum < MAX\_PROCESS\_COUNT) {

subPi = startNextProcess(processNum + 1, selfpid);

fakeRun(processNum, runTime, subPi);

CloseHandle(subPi.hProcess);

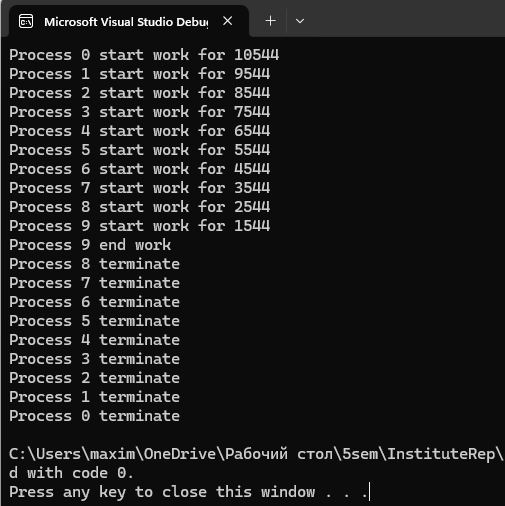
CloseHandle(subPi.hThread);

}

return 0;

}

**Результат выполнения программы:**

****

**Вывод:**

В ходе выполнения лабораторной работы были изучены методы и средства порождения процессов и нитей. Изучены способы синхронизации процессов и нитей через ожидание окончания их выполнения.