Міністерство Освіти І НАУКИ України

Національний університет "Львівська політехніка"

Інститут **ІКНІ**

Кафедра **ПЗ**

ЗВІТ

До лабораторної роботи № 3

На тему: *“Створення та керування процесами засобами API в операційній системі WINDOWS”*

З дисципліни: *“Операційні системи”*

Лектор:

доцентка каф. ПЗ

Грицай О.Д.

Виконав:

ст. гр. ПЗ-22

Климок Н.І.

Прийняла:

доцентка каф. ПЗ

Грицай О.Д.

« ____ » _____ 2022 р.

∑= ____

Львів – 2022

Тема роботи: створення та керування процесами засобами API в операційній системі WINDOWS.

Мета роботи: ознайомитися з багатопоточністю в ОС Windows. Навчитися працювати з процесами, використовуючи WinAPI-функції.

Теоретичні відомості

Процес містить деяку стартову інформацію для потоків, які в ньому створюватимуться. Процес має містити хоча б один потік, який система скеровує на виконання.

Для створення нового процесу та його головного потоку використовується функція CreateProcess().

```
BOOL CreateProcessA(
    LPCSTR lpApplicationName,
    LPSTR lpCommandLine,
    LPSECURITY_ATTRIBUTES lpProcessAttributes,
    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    BOOL bInheritHandles,
    DWORD dwCreationFlags,
    LPVOID lpEnvironment,
    LPCSTR lpCurrentDirectory,
    LPSTARTUPINFOA lpStartupInfo,
    LPPROCESS_INFORMATION lpProcessInformation);
```

Якщо функція виконалась успішно, то повертається ненульове значення. Якщо сталась помилка, то повернеться нуль.

Параметри:

1. lpApplicationName

Назва модуля, який потрібно виконати. Це може бути як повний шлях та ім’я, так і скорочене ім’я. У випадку скороченого іменні, розглядається поточний каталог. Обов'язково вказувати розширення. Параметр може мати значення NULL. У такому випадку ім'ям програми важатиметься перше ім'я, що стоїть в стрічці lpCommandLine.

2. lpCommandLine

Командний рядок, який потрібно виконати. Unicode-версія цієї функції CreateProcessW може змінювати вміст цього рядка. Цей параметр не може бути вказівником на пам'ять - читання (наприклад, змінну const). Якщо цей параметр є константним рядком, функція може викликати порушення доступу. Параметр lpCommandLine може бути NULL. У цьому випадку функція використовує рядок, на який вказує lpApplicationName, як командний рядок.

3. lpProcessAttributes

Вказівник на структуру SECURITY_ATTRIBUTES, що визначає чи повернений дескриптор об’єкту нового процесу може бути успадкований дочірнім процесом. Якщо NULL, то дескриптор не успадковується.

4. lpThreadAttributes

Вказівник на структуру SECURITY_ATTRIBUTES, що визначає чи повернений дескриптор об’єкту нового процесу може бути успадкований дочірнім процесом. Якщо NULL, то дескриптор не успадковується.

5. bInheritHandles

Відповідає за успадкування дескрипторів. Якщо TRUE, то кожен дескриптор, що можна успадкувати у процесі що викликається, успадковується новим процесом. Успадковані дескриптори мають теж значення і права доступу як і оригінальні дескриптори.

6. dwCreationFlags

Прапорці, які керують класом пріоритетності та створенням процесу. Цей параметр dwCreationFlags також керує новим класом пріоритетності процесу, який використовується для визначення пріоритетів планування потоків процесу. Якщо жоден із прапорів класу пріоритетів не вказаний, клас пріоритету за замовчуванням до NORMAL_PRIORITY_CLASS, якщо класом пріоритету процесу створення не є IDLE_PRIORITY_CLASS або BELOW_NORMAL_PRIORITY_CLASS. У цьому випадку дочірній процес отримує типовий клас пріоритетності процесу виклику.

7. lpEnvironment

Вказівник на блок оточення для нового процесу. Якщо цей параметр NULL, новий процес використовує середовище процесу виклику.

8. lpCurrentDirectory

Повний шлях до поточного каталогу для процесу. Якщо цей параметр NULL, новий процес матиме той самий поточний диск та каталог, що і процес виклику.

9. lpStartupInfo

Вказівник на структуру STARTUPINFO або STARTUPINFOEX. Дескриптори в STARTUPINFO або STARTUPINFOEX повинні закриватися CloseHandle, коли вони більше не потрібні.

10. lpProcessInformation

Вказівник на структуру PROCESS_INFORMATION, яка отримує ідентифікаційну інформацію про новий процес. Дескриптори в PROCESS_INFORMATION повинні бути закриті за допомогою CloseHandle, коли вони більше не потрібні.

Перед викликом функції створення процесу необхідно задати структури STARTUPINFO та PROCESS_INFORMATION.

Структура **PROCESS_INF0RMATION** містить чотири поля:

1. hProcess — дескриптор створеного процесу;
2. hThread — дескриптор його головного потоку;
3. dwProcessId — ідентифікатор процесу (process id, pid);
4. dwThreadId — ідентифікатор головного потоку (thread id, tid).

ЗАВДАННЯ

1. Створити окремий процес, і здійснити в ньому розв'язок задачі згідно варіанту у відповідності до порядкового номера у журнальному списку (підгрупи).

2. Реалізувати розв'язок задачі у 2-ох, 4-ох, 8-ох процесах. Виміряти час роботи процесів за допомогою функцій WinAPI. Порівняти результати роботи в одному і в багатьох процесах.

3. Для кожного процесу реалізувати можливість його запуску, зупинення, завершення та примусове завершення («вбиття»).

4. Реалізувати можливість зміни пріоритету виконання процесу.

5. Продемонструвати результати виконання роботи, а також кількість створених процесів у "Диспетчері задач", або подібних утилітах (н-д, ProcessExplorer)

Варіант 7:
Табулювати функцію ln x, задану розкладом в ряд Тейлора, в області її визначення на відрізку від A до B (кількість кроків не менше 100 000 –задається користувачем).

Хід виконання роботи

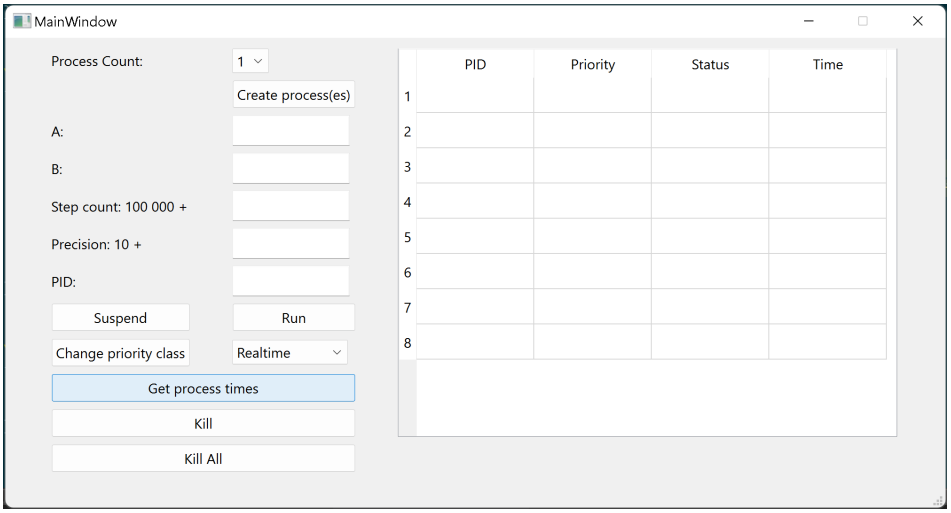
Для початку, створю програму program_7.c, яка прийматиме параметри для розкладу ряду Тейлора на певному проміжку з певним кроком і точністю.

Код програми program_7.c:

```
#include <string>
#include <iostream>
#include <cmath>

void tabulate_lnx(double a, double b, double step, double
iter_count)
{
	for (double n = a; n <= b; n += step)
	{
		double num, mul, cal, sum = 0;
		num = (n - 1) / (n + 1);
```

```
            // terminating value of the loop
            // can be increased to improve the precision
            for (int i = 1; i <= iter_count; i++) {
                  mul = (2 * i) - 1;
                  cal = pow(num, mul);
                  cal = cal / mul;
                  sum = sum + cal;
            }
            sum = 2 * sum;
            std::cout << "ln(" << n << ")=" << sum << std::endl;
      }
}

int main(int argc, char** argv)
{
      if (argc != 5) return 1;
      double a = atof(argv[1]);
      double b = atof(argv[2]);
      double step = atof(argv[3]);
      int iter_count = atoi(argv[4]);
    std::cout << "a: " << a << " b " << b << " step " << step << "
" << " iter count" << iter_count << std::endl;
      tabulate_lnx(a, b, step, iter_count);
}
```

Тепер, створю графічний інтерфейс для програми, що створюватиме процеси, які запускатимуть розклад рядів Тейлора:



Реалізую задання діапазону від A до B, задання кроку (від 100 000), задання точності (від 10 ітерацій). Також, реалізую створення процесу, призупинення процесу, запуск простоюючого процесу, вивід процесорного часу процеса, а також термінацію процесу та зміну пріоритету по ідентифікатору процеса.

Код програми:

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

#include <tchar.h>

std::array priorities = {REALTIME_PRIORITY_CLASS,
HIGH_PRIORITY_CLASS, ABOVE_NORMAL_PRIORITY_CLASS,
                         NORMAL_PRIORITY_CLASS,
BELOW_NORMAL_PRIORITY_CLASS, IDLE_PRIORITY_CLASS};

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    QTableWidget *table = ui->table;
    for (int i = 0; i < 8; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            QTableWidgetItem *newItem = new QTableWidgetItem;
            table->setItem(i, j, newItem);
            QString curr_text = table->item(i, j)->text();
        }
    }
}

MainWindow::~MainWindow()
{
    delete ui;
}

BOOL MainWindow::terminateProcess(const DWORD dwProcessId, const
UINT uExitCode)
{
```

```
    int vectorIndex = -1;
    for (unsigned long int i = 0; i < procInfos.size(); i++)
        vectorIndex = procInfos.at(i).dwProcessId == dwProcessId ?
procInfos.at(i).dwProcessId : vectorIndex;
    if (vectorIndex == -1)
    {
        QMessageBox::warning(this, "No such child PID found", "Make
sure you entered correct PID");
        return false;
    }
    DWORD dwDesiredAccess = PROCESS_TERMINATE;
        BOOL  bInheritHandle  = FALSE;
        HANDLE hProcess = OpenProcess(dwDesiredAccess,
bInheritHandle, dwProcessId);
        if (hProcess == NULL)
            return FALSE;

        BOOL result = TerminateProcess(hProcess, uExitCode);

        CloseHandle(hProcess);

        return result;
}

void MainWindow::terminateAll()
{
    while (!procInfos.empty())
    {
        PROCESS_INFORMATION curr = procInfos.at(procInfos.size() -
1);
        terminateProcess(curr.dwProcessId, 0);
        procInfos.pop_back();
    }
}

void MainWindow::createProcess(const double a, const double b,
const double step, const int iterCount)
{
    std::string command = "D:\\program_7.exe " + std::to_string(a)
+ " " + std::to_string(b) + " " + std::to_string(step) + " " +
std::to_string(iterCount);
    STARTUPINFO si;
```

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

    TCHAR tszCmdLine[1024] = {0};
    mbstowcs(tszCmdLine, command.c_str(), 1024);
    _tprintf(tszCmdLine);
    if (!CreateProcess(NULL, tszCmdLine, NULL, NULL, true,
CREATE_NEW_CONSOLE, NULL, NULL, &si, &pi)) {
        QMessageBox::warning(this, "Warning", "Could not create
child process." + QString::number(GetLastError()));
        throw 1;
    }
    procInfos.push_back(pi);
}

void MainWindow::createProcesses(const int count, const double a,
const double b, const int stepCount, const double iterCount)
{
    double step = (b - a) / stepCount;
    double partitionSize = (b - a) / count;
    for (int i = 0; i < count; i++)
    {
        createProcess(a + partitionSize * i, a + partitionSize * (i
+ 1), step, iterCount);
    }
}

QString MainWindow::getPID(PROCESS_INFORMATION pi)
{
    return QString::number(pi.dwProcessId);
}

QString MainWindow::getPriority(PROCESS_INFORMATION pi)
{
    HANDLE processHandle = OpenProcess(PROCESS_ALL_ACCESS, FALSE,
pi.dwProcessId);
    DWORD priority = GetPriorityClass(processHandle);
    CloseHandle(processHandle);
    switch (priority)
    {
          case    ABOVE_NORMAL_PRIORITY_CLASS:
```

```
            return QString("Above normal");
        case    BELOW_NORMAL_PRIORITY_CLASS:
            return QString("Below normal");
        case    HIGH_PRIORITY_CLASS:
            return QString("High");
        case    IDLE_PRIORITY_CLASS:
            return QString("Idle");
        case    NORMAL_PRIORITY_CLASS:
            return QString("Normal");
        case    REALTIME_PRIORITY_CLASS:
            return QString("Realtime");
        default:
            return QString("WTF");
    }
}

QString MainWindow::getStatus(PROCESS_INFORMATION pi)
{
    HANDLE processHandle = OpenProcess(PROCESS_ALL_ACCESS, FALSE,
pi.dwProcessId);
    DWORD status = WaitForSingleObject(processHandle, 0);
    QString result = "";
    switch(status) {
        case WAIT_OBJECT_0:
             result = "Terminated";
             break;
        default:
            int value = SuspendThread(processHandle);
             ResumeThread(processHandle);
             result = value == 0 ? "Running" : "Alive";
    }
    CloseHandle(processHandle);
    return result;
}

QString MainWindow::getExecutionTime(PROCESS_INFORMATION pi)
{
    long C_TIME = 0, E_TIME = 0, K_TIME = 0, U_TIME = 0;
    GetProcessTimes(pi.hProcess, (FILETIME *)&C_TIME, (FILETIME
*)&E_TIME, (FILETIME *)&K_TIME, (FILETIME *)&U_TIME);
    return QString::number(U_TIME * pow(10.0,-3), 'g', 10) + "ms";
}
```

```
void MainWindow::displayProcessInTable(const PROCESS_INFORMATION
pi, const int index)
{
   int row = index;
   ui->table->item(row, 0)->setText(getPID(pi));
   ui->table->item(row, 1)->setText(getPriority(pi));
   ui->table->item(row, 2)->setText(getStatus(pi));
   ui->table->item(row, 3)->setText(getExecutionTime(pi));
}

void MainWindow::updateTable()
{
   for (int i = 0; i < 8; i++)
   {
       for (int j = 0; j < 4; j++)
       {
           QTableWidgetItem *curr = ui->table->item(i, j);
           ui->table->item(i, j)->setText(QString(""));
       }
   }
   for (unsigned long int i = 0; i < procInfos.size(); i++)
   {
       displayProcessInTable(procInfos.at(i), i);
   }
}

void MainWindow::on_btnCreateProcess_clicked()
{
   QTableWidgetItem *any = ui->table->item(0, 0);
   terminateAll();
   double a = ui->txtA->toPlainText().toDouble();
   double b = ui->txtB->toPlainText().toDouble();
   int stepCount = 100000 + ui->txtStep->toPlainText().toInt();
   int iterCount = 10 + ui->txtIterCount->toPlainText().toInt();
   int procCount = ui->cbProcessCount->currentText().toInt();
   createProcesses(procCount, a, b, stepCount, iterCount);
   updateTable();
}


void MainWindow::on_btnGetTime_clicked()
```

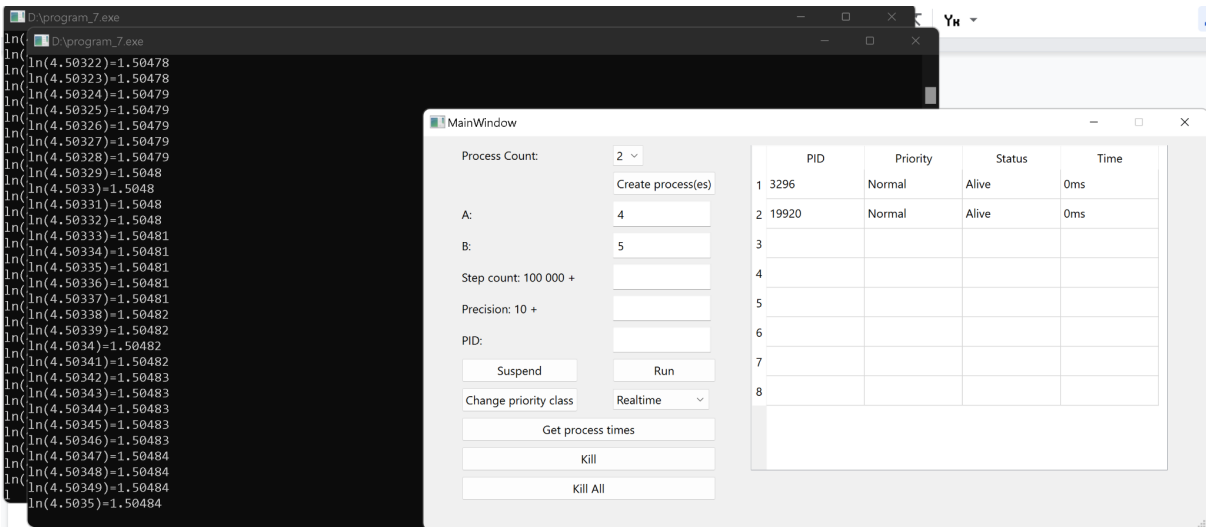
```
{
    updateTable();
}


void MainWindow::on_btnRun_clicked()
{
    int pid = ui->txtPID->toPlainText().toInt();
    for(long unsigned int i = 0; i < procInfos.size(); i++) {
        if (procInfos[i].dwProcessId == pid) {
            ResumeThread(procInfos[i].hThread);
        }
    }
    updateTable();
}


void MainWindow::on_btnSuspend_clicked()
{
    int pid = ui->txtPID->toPlainText().toInt();
    for(int i = 0; i < procInfos.size(); i++) {
        if (pid == procInfos[i].dwProcessId) {
            SuspendThread(procInfos[i].hThread);
        }
    }
    updateTable();
}

void MainWindow::on_btnKill_clicked()
{
    int pid = ui->txtPID->toPlainText().toInt();
    terminateProcess(pid, 1);
    updateTable();
}


void MainWindow::on_btnChangePriority_clicked()
{
    int pid = ui->txtPID->toPlainText().toInt();
    std::string priority =
ui->cbPriorityClass->currentText().toStdString();
```

```
    for(int i = 0; i < procInfos.size(); i++) {
        if(pid == procInfos[i].dwProcessId) {
            SetPriorityClass(procInfos[i].hProcess,
priorities[ui->cbPriorityClass->currentIndex()]);
        }
    }
    updateTable();
}
```

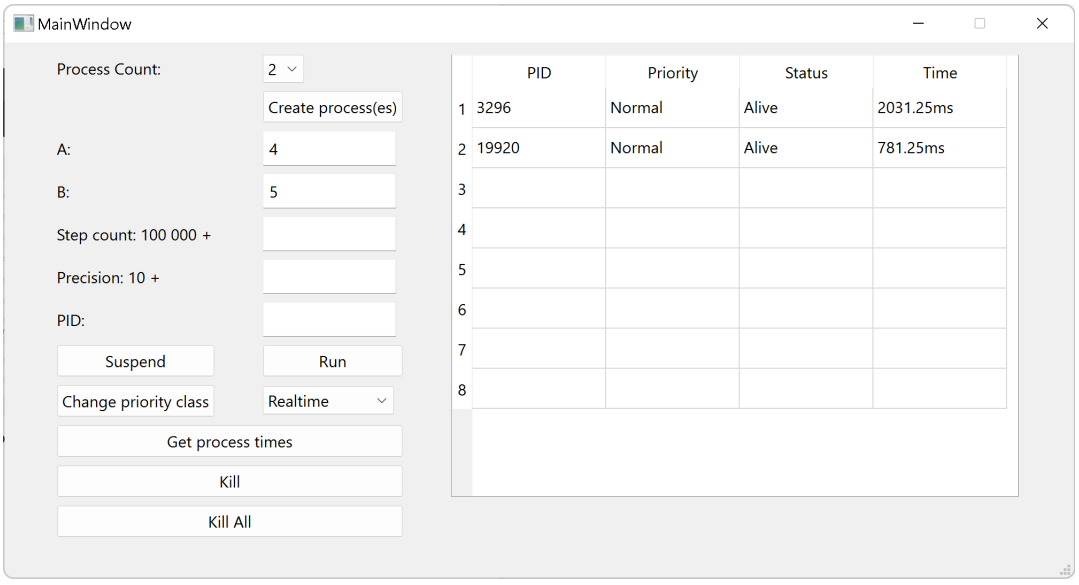
Протокол роботи програми

Запустимо програму.

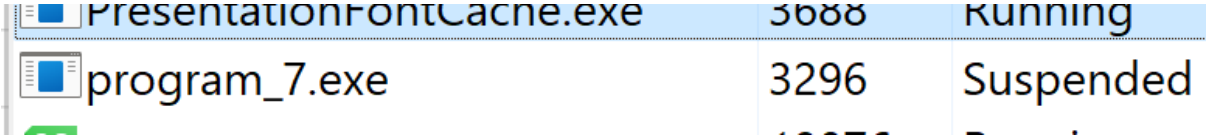
Спробуємо створити 2 процеси, які шукатимуть ряд Тейлора на діапазоні від 4 до 5 з кроком 100 000 та точністю 10:



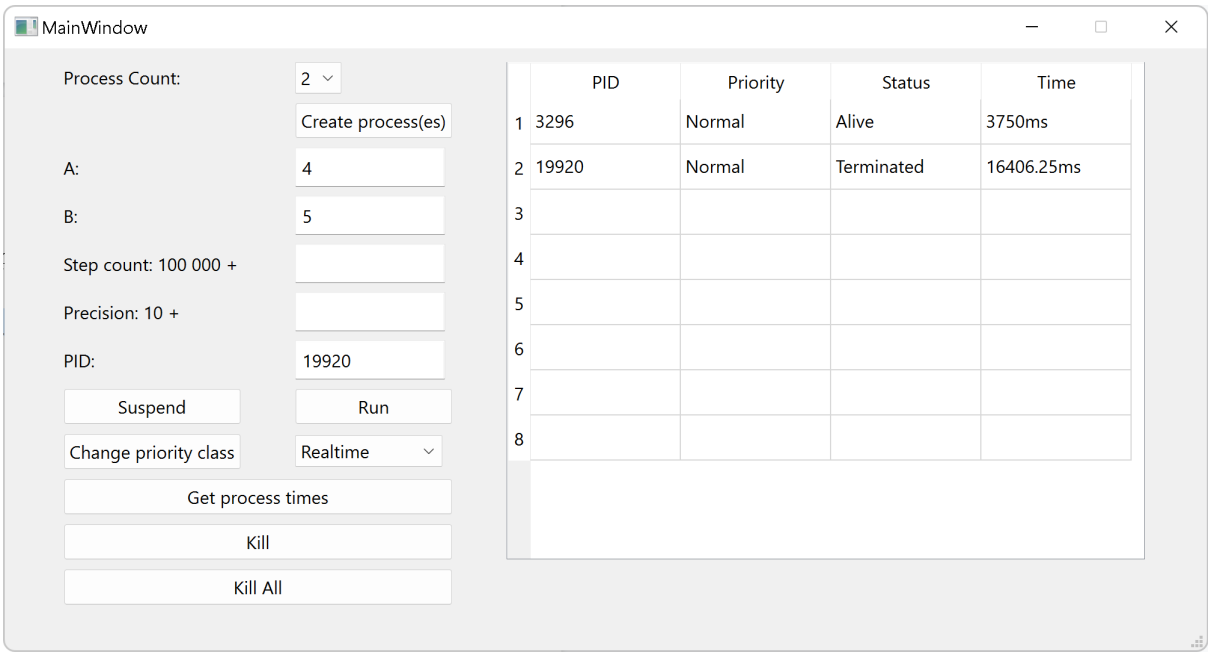
Дістанемо процесорний час, що використали процеси:



Призупинимо процес із PID 3296. Бачимо, що процес призупинився, і диспетчер задач показує статус “Suspended”:

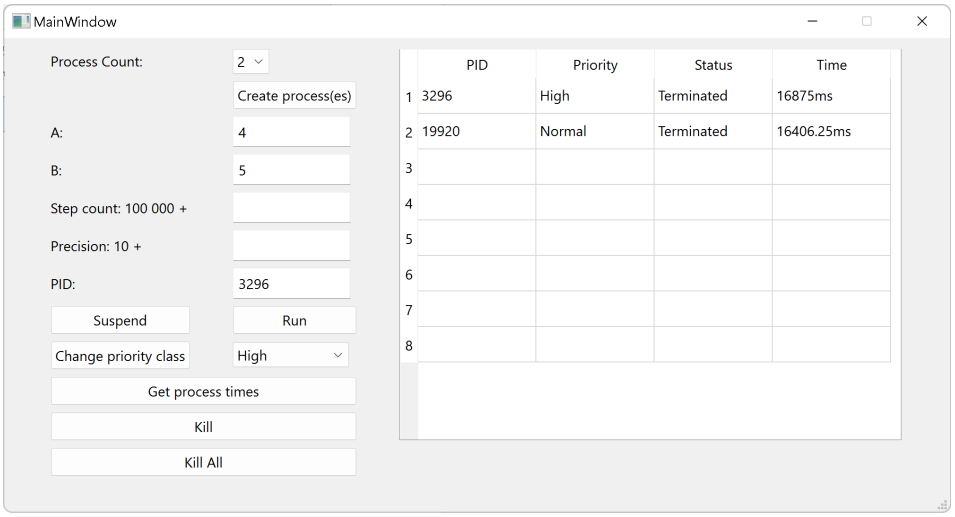


Вб’ємо процес №19920:

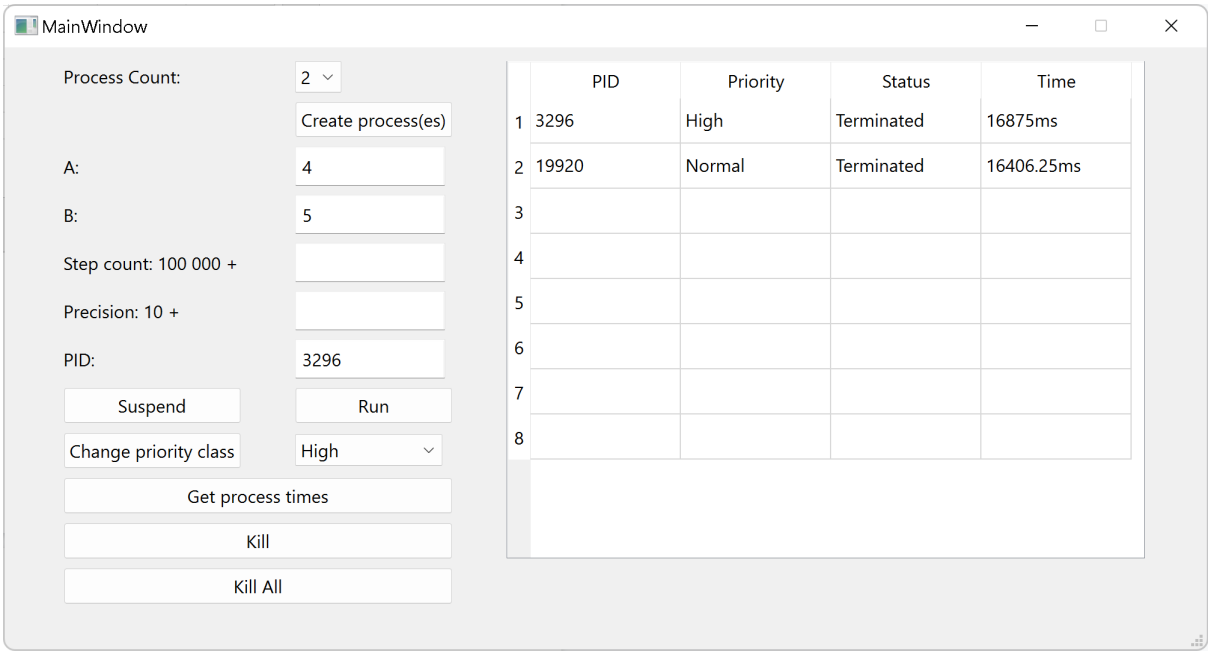


Запустимо процес 3296. Можемо бачити, що ряди Тейлора знову знаходяться, процесорний час збільшується, а диспетчер задач показує статус “Running”:

Змінимо пріоритет на High:



Вб’ємо усі процеси:



Висновок

Виконуючи цю лабораторну роботу, я навчивс працювати з WINAPI та почав краще розуміти, як відбувається взаємодія із процесами на рівні програмного забезпечення. Я зрозумів, як саме програмне забезпечення може запускати, призупиняти та вбивати процеси, змінювати їхній пріоритет та діставати статистику щодо їхньої роботи, як-от процесорний час.