**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

**Кафедра ЕОМ**



**КУРСОВА РОБОТА**

з предмету **“**Системне програмне забезпечення **”**

на тему:

**“Утиліта «Диспетчер задач»”**

Виконав:

Студент групи КІ-35

Морозюк Р.Л.

Прийняв:

Бачинський Р.В.

Львів 2017

**АНОТАЦІЯ**

В курсовій роботі була розроблена програма «Диспетчер задач», вона дозволяє переглядати запущені задачі, запустити нову задачу, переглядати список процесів, список сервісів Windows і список драйверів, а також завершити вибрану задачу чи вибраний процес, переглянути продуктивність системи.

Програма була розроблена за допомогою QtCreator з використанням WinApi функцій і фреймворка Qt 5.5.1.

**ЗМІСТ**

Завдання на курсову роботу …………………………………………………….4

Вступ……………………………………………………………………………….5

**1. Огляд методів управління процесами Windows**.…………..……………...7

**2. Загальний огляд диспетчерів задач**………………………………………..10

2.1. Диспетчер задач Windows…………………………………………………...10

2.2 Process Explorer...…………………………………………………………......11

2.3 Process Lasso………………………………………………………………......12

2.4 RAM Saver Pro………………………………………………………………...13

**3. Розробка диспетчера завдань** ……………………………………………..15

3.1. Розробка вузла перегляду процесів…………………………………………17

*3.2 Розробка граф схеми алгоритму вузла перегляду процесів…*……………..17

3.3. Розробка вузла перегляду сервісів………………………………………….19

*3.4 Розробка граф схеми алгоритму вузла перегляду сервісів*………………...20

3.5. Розробка вузла перегляду задач…………………………………………….21

*3.6 Розробка граф схеми алгоритму перегляду задач....*………………………21

3.7. Розробка вузла відображення швидкодії…………………………………...23

*3.8 Розробка граф схеми алгоритму вузла перегляду швидкодії…*……………23

3.9. Розробка вузла запуску нового процесу…………………………………....24

*3.10 Розробка граф схеми алгоритму вузла запуску нового процесу* ………....24

3.11. Розробка вузла завершеня процесу………………………………………..24

*3.12 Розробка граф схеми алгоритму вузла “Завершення процесу”…*……… 25

**4.Опис інтерфейсу та інструкції користувача.** ……………………………...26

**5. Тестування**……………………………………………………………………..29

5.1 Виявлення помилок відображення процесів, задач, сервісів і драйверів…29

5.2 Виявлення помилок створення нового процесу…………………………….29

5.3. Виявлення помилок видалення процесу…………………………................30

Висновок ………………………………………………………………………...31

Список літератури………………………………………………………………..32

Додатки………………………………………………………………………….....33

**Завдання на курсову роботу**

Розробити програму моніторингу процесів, служб, драйверів Windows з наступними функціями:

* Переглядати список задач
* Переглядати список процесів
* Запускати новий процес
* Зупиняти запущений
* Виводити список сервісів і драйверів

Програма повинна використовувати WinApi функції.

**Вступ**

У сучасній операційній системі одночасно виконуються код ядра (що належить до його різних підсистем) і код програм користувача. При цьому відбуваються різні дії: одні програми і підсистеми виконують інструкції процесора, інші зайняті введенням-виведенням, ще деякі очікують на запити від користувача або інших застосувань. Для спрощення керування цими діями в системі доцільно виділити набір елементарних активних елементів і визначити інтерфейс взаємодії ОС із цими елементами. Коли активний елемент системи зв'язати із програмою, що виконується, ми прийдемо до поняття процесу.

Під процесом розуміють абстракцію ОС, яка об'єднує все необхідне для виконання однієї програми в певний момент часу.

Програма — це деяка послідовність машинних команд, що зберігається на диску, в разі необхідності завантажується у пам'ять і виконується. Можна сказати, що під час виконання програму представляє процес.

Однозначна відповідність між програмою і процесом встановлюється тільки в конкретний момент часу: один процес у різний час може виконувати код декількох програм, код однієї програми можуть виконувати декілька процесів одночасно.

Для успішного виконання програми потрібні певні ресурси. До них належать:

• ресурси, необхідні для послідовного виконання програмного коду (передусім процесорний час);

• ресурси, що дають можливість зберігати інформацію, яка забезпечує виконання програмного коду (регістри процесора, оперативна пам'ять тощо).

Процесом називають сукупність одного або декількох потоків і захищеного адресного простору, у якому ці потоки виконуються.

На відміну від процесів потоки розпоряджаються загальною пам'яттю. Дані потоку не захищені від доступу до них інших потоків за умови, що всі вони виконуються в адресному просторі одного процесу. Це надає додаткові можливості для розробки застосувань, але ускладнює програмування.

Захищений адресний простір процесу задає абстракцію виконання коду на окремій машині, а потік забезпечує абстракцію послідовного виконання команд на одному виділеному процесорі.

Таку технологію називають відображенням, у пам'ять (memory mapping).

1. **Огляд методів управління процесами Windows.**

Протягом існування **процесу** його виконання може бути багаторазово перерване і продовжене. Для того, щоб відновити виконання **процесу,** необхідно відновити стан його операційного середовища. Стан операційного середовища відображається станом регістрів і програмного лічильника, режимом роботи процесора, покажчиками на відкриті файли, інформацією про незавершені операції введення-виведення, кодами помилок виконуваних даним процесом системних викликів і т.д. Ця інформація називається **контекстом процесу.**

Для того щоб ОС могла керувати процесами, вона повинна мати всю необхідну для цього інформацією. З цією метою на кожний процес заводиться **дескриптор процесу.**

**Дескриптор -** спеціальна інформаційна структура, яка заводиться на кожний процес (описувач завдання, блок управління завданням).

У загальному випадку дескриптор містить наступну інформацію:

1. Ідентифікатор процесу.
2. Тип (або клас) процесу, який визначає для супервізора деякі правила надання ресурсів.
3. Пріоритет процесу.
4. Зміну стану, яка визначає, в якому стані знаходиться процес (готовий до роботи, в стані виконання, очікування пристрої введення-виведення і т.д.)
5. Захищену область пам'яті (або адресу такої зони), в якій зберігаються поточні значення регістрів процесора, якщо процес переривається, не закінчивши роботи. Ця інформація називається **контекстом завдання.**
6. Інформацію про ресурси, якими володіє процес і / або має право користуватися (покажчики на відкриті файли, інформація про незавершені операції введення / виводу і т.п.).
7. Місце (або його адреса) для організації спілкування з іншими процесами.
8. Параметри часу запуску (момент часу, коли процес має активізуватися, і періодичність цієї процедури).
9. У разі відсутності системи управління файлами - адреса завдання на диску в її початковому стані і адреса на диску, куди вона вивантажується з оперативної пам'яті, якщо її витісняє інша.

**Дескриптор процесу** в порівнянні з контекстом містить більш оперативну інформацію, яка повинна бути легко доступна підсистемі планування процесів. Контекст процесу містить менш актуальну інформацію і використовується операційною системою тільки після того, як прийнято рішення про відновлення перерваного процесу.

**Дескриптори,** як правило, постійно розташовуються в оперативній пам'яті з метою прискорити роботу супервізора, який організовує їх у списки (черги) і відображає зміну стану процесу переміщенням відповідного описувача з одного списку в інший.

Для кожного стану (за винятком стану виконання для однопроцесорній системи) ОС веде відповідний список завдань, що знаходиться в цьому стані. Однак для стану очікування може бути не один список, а стільки, скільки різних видів ресурсів можуть викликати стан очікування.

Наприклад, станів очікування завершення операції введення / виведення може бути стільки, скільки пристроїв введення / виводу міститься в системі.

## Процеси і потоки

Щоб підтримувати багтозадачність, ОС повинна визначити і оформити для себе ті внутрішні одиниці роботи, між якими буде поділятися процесор і інші ресурси комп'ютера. В даний час у більшості ОС визначені два типи одиниць роботи:

* **Процес** (більш велика одиниця роботи).
* **Потік** (нитка або тред) - більш дрібна одиниця роботи, яку вимагає для свого виконання процес.

**Коли говорять про процеси,** то тим самим хочуть відзначити, що ОС підтримує їх відособленість: у кожного процесу є свій віртуальний адресний простір, кожному процесу призначаються свої ресурси - файли, вікна та ін. Така відособленість потрібна для того, щоб захистити один процес від іншого , оскільки вони, спільно використовують всі ресурси обчислювальної системи, конкурують один з одним.

У загальному випадку **процеси** просто ніяк не пов'язані між собою і можуть належати навіть різним користувачам, що розділяють одну обчислювальну систему. Іншими словами, у разі процесів ОС вважає їх абсолютно незв'язаними і незалежними. При цьому саме ОС відповідає за конкуренцію між процесами щодо ресурсів.

Для підвищення швидкодії процесів є можливість задіяти внутрішній паралелізм у самих **процесах.**

Наприклад, деякі операції, що виконуються додатком, можуть вимагати для свого виконання досить тривалого використання ЦП. У цьому випадку при інтерактивній роботі з додатком користувач змушений довго чекати завершення замовленої операції і не може управляти додатком до тих пір, поки операція не виконається до самого кінця. Такі ситуації зустрічаються досить часто, наприклад, при обробці великих зображень в графічних редакторах. Якщо ж програмні модулі, які виконують такі тривалі операції, оформляти у вигляді самостійних «підпроцесів» **(потоків),** які будуть виконуватися паралельно з іншими «підпроцесами», то у користувача з'являється можливість паралельно виконувати декілька операцій в рамках однієї програми (процесу).

Можна виділити наступні відмінності **потоків від процесів:**

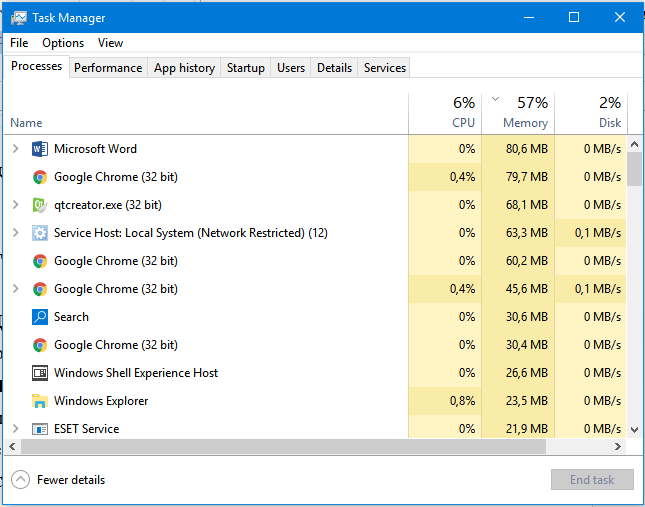
* ОС для потоків не повинна організовувати повноцінну віртуальну машину.
* Потоки не мають своїх власних ресурсів, вони розвиваються в тому ж віртуальному адресному просторі, можуть користуватися тими ж файлами, віртуальними пристроями та іншими ресурсами, що і даний процес.
* Єдине, що потокам необхідно мати, - це процесорний ресурс. У однопроцесорній системі потоки поділяють між собою процесорний час так само, як це роблять звичайні процеси, а в багатопроцесорної системі можуть виконуватися одночасно, якщо не зустрічають конкуренції через звернення до інших ресурсів.

**2. Загальний огляд диспетчерів задач**

**2.1. Диспетчер задач Windows**

**Диспетчер завдань** в операційних системах сімейства Microsoft Windows - утиліта для виводу на екран списку запущених процесів і споживаних ними ресурсів (зокрема статус, процесорний час і споживана оперативна пам'ять). Також є можливість деякої маніпуляції процесами.

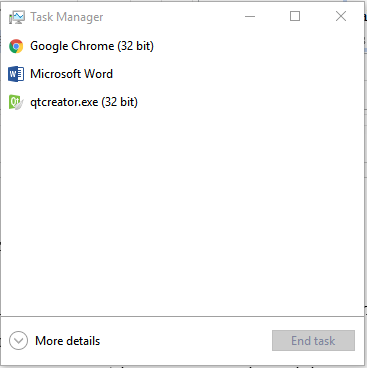
Windows Task Manager в Windows XPможна викликати, одночасно натиснувши клавіші Ctrl+Alt+Del. Диспетчер завдань можна також запустити в командному рядку, ввівши ім'я його виконуваного файлу (taskmgr.exe) або вибравши відповідний пункт в контекстному меню панелі завдань.



*Рис. 1 Вікно диспетчера задач Windows*

Диспетчер завдань у Windows 10 - вбудована в операційну систему утиліта. Вона містить вкладки:

* **Процеси.** Різноманітні дані про усі запущені в системі процеси, можна завершувати, міняти пріоритет, задавати відповідність процесорам (у багатопроцесорних системах)
* **Швидкодія.** Графіки завантаження процесора (процесорів), використання оперативної пам'яті.
* **Історія додатків.** Історія використання процесорного часу додатком.
* **Автозагрузка.** Перелік програм які завантажуються разом з завантаженням ОС.
* **Користувачі** (тільки у режимі адміністратора). Маніпулювання активними користувачами.
* **Деталі.** Детальна інформація про запущені процеси.
* **Служби** (починаючи з Vista). Відомості про усі служби Windows.
* **Додатки.** Дозволяє перемкнутися в потрібне застосування, або завершити його. Можна переглянути нажавши кнопку “Fewer details”.



*Рис. 2 Запущені програми Windows*

**2.2 Process Explorer**

Розробник: [Sysinternals](http://translate.googleusercontent.com/translate_c?hl=uk&ie=UTF-8&sl=ru&tl=uk&u=http://www.sysinternals.com/&prev=_t&rurl=translate.google.com.ua&twu=1&usg=ALkJrhjtGwi0c1bepDPmkpBsbqpo6u1OPQ)

Один з найбільш відомих аналогів Диспетчера завдань. Process Explorer в режимі реального часу проводить постійний моніторинг всіх запущених в системі процесів. Для кожного процесу програма відображає докладну інформацію:

- Використання пам'яті;

- Відсоток завантаження процесора;

- Власник процесу;

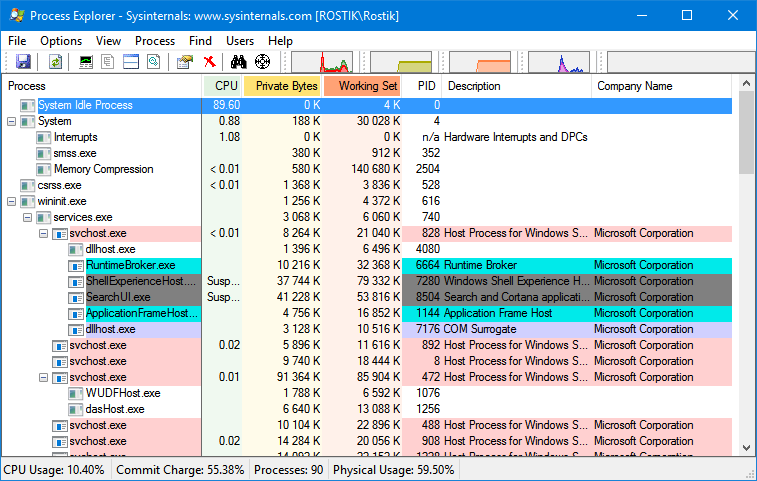
- Родинні відносини між процесами у вигляді дерева;

- Список бібліотек, використовуваних обраним процесом;

- Використовувані протоколи TCP / IP;

- Розташування виконуваного файлу.

Крім того, Process Explorer дозволяє закривати процеси, встановлювати їх пріоритет, а також шукати DLL, використовувані процесами.



*Рис. 3 Вікно програми Process Explorer*

**2.3 Process Lasso**

Розробник: [Bitsum Technologies](http://translate.googleusercontent.com/translate_c?hl=uk&ie=UTF-8&sl=ru&tl=uk&u=http://www.bitsum.com/prolasso.php&prev=_t&rurl=translate.google.com.ua&twu=1&usg=ALkJrhh9MSU6LwJ1iY1Ez56163eylZq4-w)

Основним завданням цієї програми є автоматичний розподіл циклів центрального процесора. Завдяки цьому поліпшується реактивність операційної системи при сильному завантаженні CPU. Process Lasso може звільняти ресурси для процесів за рахунок зниження пріоритетів процесів, яким потрібно багато циклів.

Програма дозволяє:

- Встановлюватися пріоритети;

- Відображати активні і взагалі всі процеси;

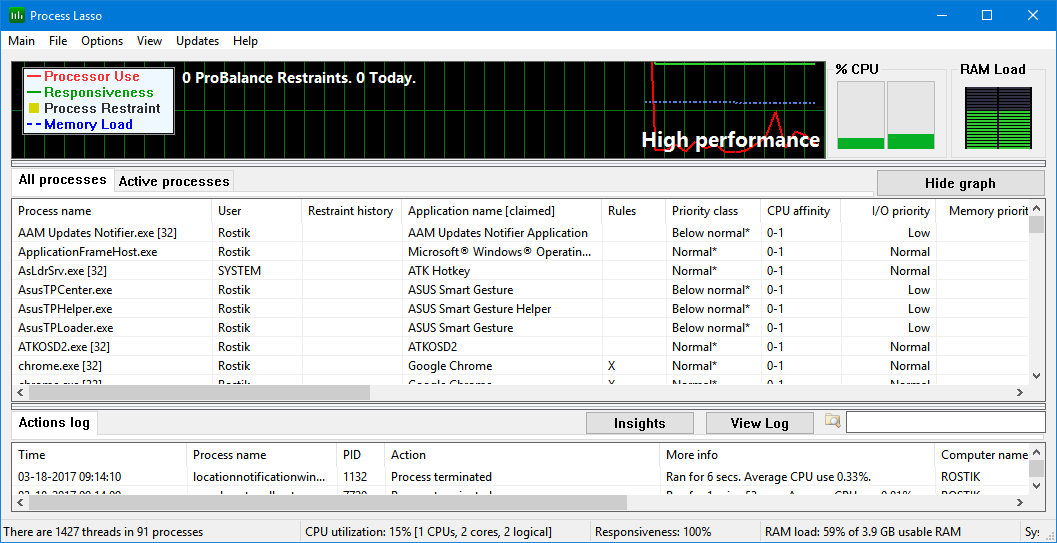
- Показувати завантаження процесора і кількість потоків;

- Встановлювати максимальну кількість копій процесу в системі;

- Завершувати процеси;

- Створювати правила персонально для кожної програми;

- Вести балки протоколювання всіх виконуваних процесів.



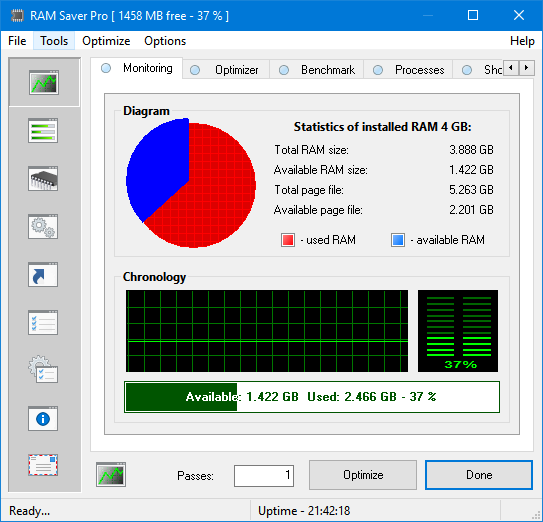
*Рис. 4 Вікно програми Process Lasso*

**2.4 RAM Saver Pro**

Розробник: [WinTools Software Engineering](http://translate.googleusercontent.com/translate_c?hl=uk&ie=UTF-8&sl=ru&tl=uk&u=http://www.wintools.net/&prev=_t&rurl=translate.google.com.ua&twu=1&usg=ALkJrhiSeFwbfnJYTgq6FfI-00c4mBHqtA)

Інструмент для оптимізації оперативної пам'яті, а також її очищення та моніторингу. RAM Saver Pro може звільняти оперативну пам'ять для додатків. Програма веде статистику використання пам'яті, причому може відображати її як в числовому, так і в графічному вигляді. Відзначимо і функцію тестування пам'яті, запобігання витоку пам'яті в погано оптимізованих додатках, дефрагментацію системної пам'яті для швидкого доступу до неї, переміщення невикористовуваних бібліотек і драйверів в swap-файл для звільнення пам'яті, примусове очищення буфера обміну і багато інших корисних функцій.

У плані роботи з процесами RAM Saver Pro не представляє великих можливостей - лише відображення імен процесів, шляхів до виконуваних файлів і розміру використовуваної пам'яті. І ще можливість завершення процесу.



*Рис. 5 Вікно програми RAM Saver Pro*

**3. Розробка диспетчера завдань**

Дана програма має такі основні функціональні вузли:

- Перегляд усіх процесів.

- Перегляд задач.

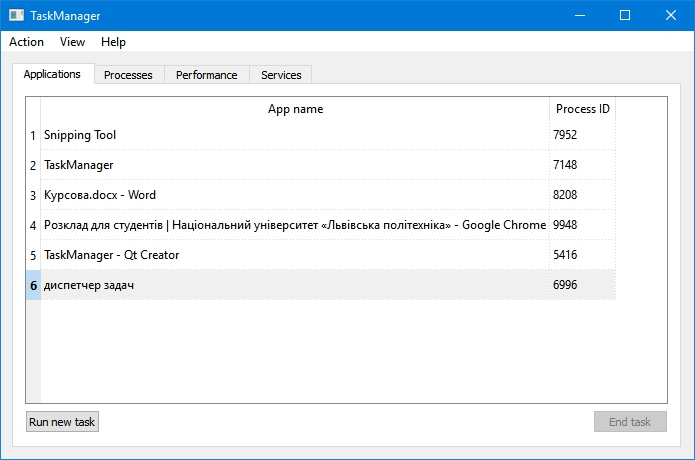
- Перегляд сервісів і драйверів

- Запуск нового процесу на виконання.

- Зупинення процесу або задачі.

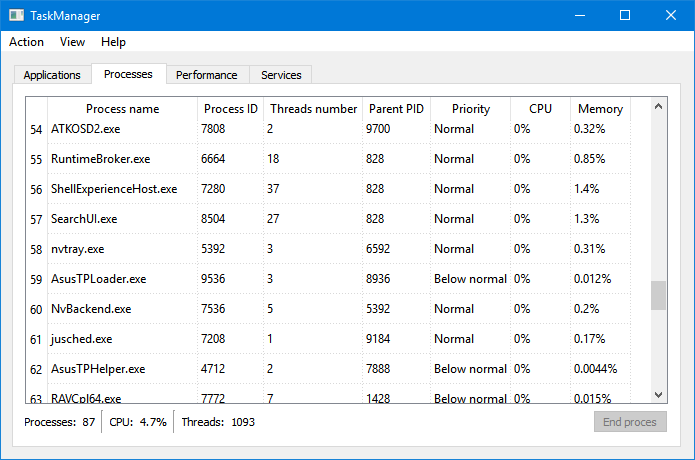
- Перегляд швидкодії комп’ютера.

Вузол «Додатки» з списку процесів вибирає ті процеси, які відображаються на панелі задач.



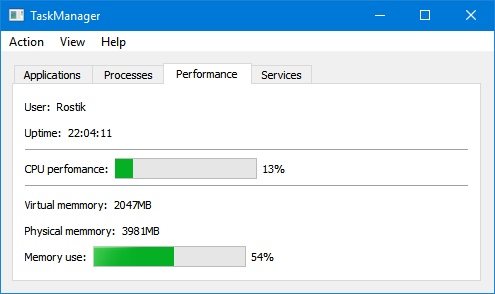
*Рис.6 Вкладка «Додатки»*

Вузол «Процеси» звертається до системи, отримує список всіх процесів, а також визначає процесорний час і пам’ять, які займає процес (подає і відсотках), ID цього процесу кількість потоків даного процеса, ID батьківського процеса, пріоритет. Внизу вкладки відображає загальну кількість процесів, завантаження процесора і загальну кількість потоків.



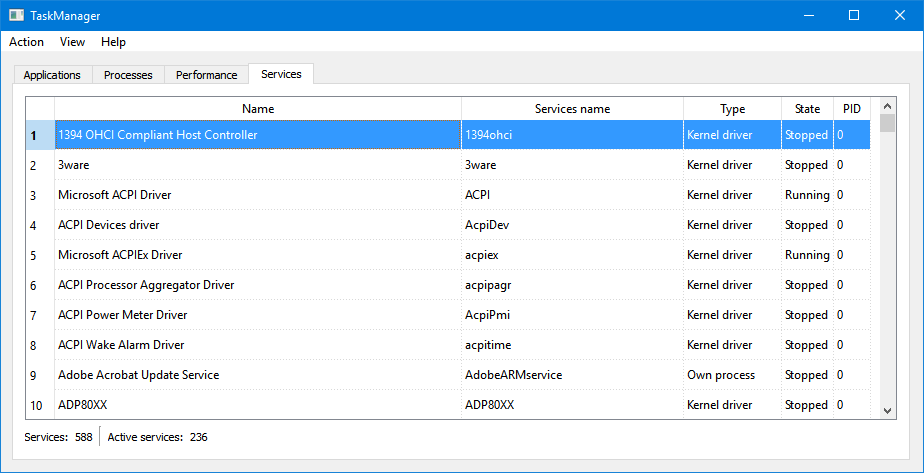
*Рис. 7 Вкладка «Процеси»*

Вузол «Швидкодія» відображає ім’я користувача, час роботи системи, завантаження процесора, обсяг віртуальної пам’яті, загальний обсяг фізичної пам’яті, відсоток використання пам’яті.



*Рис. 8 Вкладка «Швидкодія»*

Вузол «Сервіси» звертається до системи, отримує список всіх служб і описує їх. Внизу вкладки відображає загальну кількість служб і драйверів в системі і кількість запущених служб



*Рис. 9 Вкладка «Сервіси»*

Вузол “Запуск нового процесу на виконання”– запускає процес отримавши адрес до виконуваного файлу.

Вузол “Завершення процесу” – завершує процес.

**3.1. Розробка вузла перегляду процесів.**

Вибір всіх процесів передбачає звернення до системи Windows за допомогою функцій winAPI і отримання списку всіх процесів, а також даних про процесорний час і використання пам’яті.

***3.2 Розробка граф схеми алгоритму вузла перегляду процесів.***

Особливостями даного алгоритму є отриманна всіх процесів і методика обчислення процесорного часу і пам’яті.



*Рис 10. Граф-схема вузла перегляду процесів*

Для того щоб отримати список всіх процесів використовується функція CreateToolhelp32Snapshot(TH32CS\_SNAPPROCESS, 0);

Пріоритет обчислюється так:

HANDLE hProcess = OpenProcess(PROCESS\_QUERY\_INFORMATION |PROCESS\_VM\_READ,FALSE, pe.th32ProcessID); //Получаємо дескриптор процеса

//визначення пріоритета процеса

QString procPriority;

switch (GetPriorityClass(hProcess)) //Получаем пріоритет процеса

{

case HIGH\_PRIORITY\_CLASS:

procPriority = "High";

break;

case IDLE\_PRIORITY\_CLASS:

procPriority = "Idle";

break;

case NORMAL\_PRIORITY\_CLASS:

procPriority = "Normal";

break;

case REALTIME\_PRIORITY\_CLASS:

procPriority = "Realtime";

break;

case ABOVE\_NORMAL\_PRIORITY\_CLASS:

procPriority = "Above normal";

break;

case BELOW\_NORMAL\_PRIORITY\_CLASS:

procPriority = "Below normal";

break;

default:

procPriority = "No access";

}

Процесорний час обчислюється за допомогою наступних конструкцій:

FILETIME ftCreation,ftExit,ftKernel,ftUser;

SYSTEMTIME stCreation,stExit,stKernel,stUser;

double procent=0;//відсоток завантаження процесора

if(GetProcessTimes(hProcess, &ftCreation, &ftExit, &ftKernel, &ftUser))//час виділений процесу системою

{

//приведення часу в нормальний формат

FileTimeToSystemTime(&ftCreation, &stCreation);

FileTimeToSystemTime(&ftExit, &stExit);

FileTimeToSystemTime(&ftUser, &stUser);

FileTimeToSystemTime(&ftKernel, &stKernel);

//час роботи в режимі користувача

double user = ((double)stUser.wHour\*msInHour +

(double)stUser.wMinute\*msInMinute +

(double)stUser.wSecond\*msInSecond +

(double)stUser.wMilliseconds)/updateTime;//час в інтервалі якого знято показники

//час роботи в режимі ядра

double kernel = ((double)stKernel.wHour\*msInHour +

(double)stKernel.wMinute\*msInMinute +

(double)stKernel.wSecond\*msInSecond +

(double)stKernel.wMilliseconds)/updateTime;//час в інтервалі якого знято показники

double res = user + kernel;

QHash<DWORD, double>::iterator it = timeForProces.find(pe.th32ProcessID);

if(it!=timeForProces.end())//якшо такий елемент знайдено

{

procent = (res - it.value())\*100.0;

timeForProces.remove(pe.th32ProcessID);

}

timeForProces.insert(pe.th32ProcessID, res);

}

Пам’ять, яку займає процес:

PROCESS\_MEMORY\_COUNTERS pmc;

GetProcessMemoryInfo(hProcess, &pmc, sizeof(pmc));

**3.3. Розробка вузла перегляду служб.**

Вибір всіх сервісів передбачає звернення до системи Windows за допомогою функції EnumServicesStatusEx і отримання списку всіх сервісів.

***3.4 Розробка граф схеми алгоритму вузла перегляду служб.***



*Рис. 11 Граф-схема алгоритму відображення списку служб*

Для отримання списку всіх служб

hSCM = OpenSCManager(NULL, NULL, SC\_MANAGER\_ALL\_ACCESS);//менеджер доступу до сервісів (повний доступ)

EnumServicesStatusEx(hSCM,

SC\_ENUM\_PROCESS\_INFO,//беремо інформацію про процеси

SERVICE\_WIN32 | SERVICE\_DRIVER | SERVICE\_FILE\_SYSTEM\_DRIVER | SERVICE\_KERNEL\_DRIVER, // пошук типу сервісів

SERVICE\_STATE\_ALL,//Всі сервіси і активні і неактивні

buf,//буфер списку сервісів

bufSize,

&bufSizeNeed,//вертає кількість необхідних байт, якшо замало пам'яті в буфері

&serviceNumber,//кількість сервісів

NULL,

NULL);

Для отримання типу і стану використовуються наступні параметри:

info = (LPENUM\_SERVICE\_STATUS\_PROCESS)buf;

int activeServices=0;

for(ULONG i = 0;i<serviceNumber;i++)

{

//визначення типу

QString type;

switch(info[i].ServiceStatusProcess.dwServiceType)

{

case SERVICE\_FILE\_SYSTEM\_DRIVER:

type = "File system driver";

break;

case SERVICE\_KERNEL\_DRIVER:

type = "Kernel driver";

break;

case SERVICE\_WIN32\_OWN\_PROCESS:

type = "Own process";

break;

case SERVICE\_WIN32\_SHARE\_PROCESS:

type = "Shares a process";

break;

}

//визначення стану

QString state;

switch(info[i].ServiceStatusProcess.dwCurrentState)

{

case SERVICE\_PAUSED:

state = "Paused";

break;

case SERVICE\_RUNNING:

state = "Running";

activeServices++;

break;

case SERVICE\_STOPPED:

state = "Stopped";

break;

}

Отримання імені і назви сервісу, а також ІD:

info[i].lpDisplayName

info[i].lpServiceName

info[i].ServiceStatusProcess.dwProcessId

**3.5. Розробка вузла перегляду задач.**

Даний вузол повинен вибирати тільки ті процеси, які відображаються на панелі задач.

***3.6 Розробка граф схеми алгоритму перегляду задач.***



*Рис. 12 Граф-схема алгоритму відображення списку задач*

Перерахування всіх процесів Windows і їх обробка:

EnumWindows((WNDENUMPROC)EnumWindowsProc, 0);

BOOL CALLBACK EnumWindowsProc(HWND hWnd, LPARAM)//додаткова функція для знаходження активних аплікацій

{

char Buff[255], NameOfClass[255];

GetWindowText(hWnd, (LPWSTR)Buff, 254);

GetClassName(hWnd, (LPWSTR)NameOfClass, 254);

if(IsWindowVisible(hWnd))

{

QString str = QString::fromStdWString((LPWSTR)Buff);

if(!str.isEmpty())

{

bool flag = true;

DWORD dwID ;

GetWindowThreadProcessId(hWnd, &dwID) ;

for(int i = 0; i<tempApps->*rowCount*(); i++)//перевірка чи дане ід вже існує

{

if(tempApps->item(i, 1)->text().toULong() == dwID)

{

flag = false;

break;

}

}

if(flag)

tempApps->appendRow(QList<QStandardItem\*>()<<new QStandardItem(str)<<new QStandardItem(QString::number(dwID)));

}

}

return TRUE;

}

**3.7. Розробка вузла відображення швидкодії.**

Вибір службової інформації про завантаження комп’ютера, імені користувача, використання пам’яті передбачає використання winAPI функцій.

***3.8 Розробка граф схеми алгоритму вузла перегляду швидкодії.***



*Рис. 13 Граф-схема алгоритму відображення інформації про використання ресурсів комп’ютера*

Щоб отримати ім’я користувача використовується:

GetUserNameA((LPSTR)buf, &size);

Щоб отримати час роботи системи використовується наступний код:

GetTickCount()

Обсяг використаної ОП і віртуальної пам’яті, і загальний обсяг ОП:

MEMORYSTATUSEX statex;

statex.dwLength = sizeof (statex);

GlobalMemoryStatusEx (&statex);

statex.dwMemoryLoad

statex.ullTotalVirtual

statex.ullTotalPhys;

**3.9. Розробка вузла запуску нового процесу**

Даний вузол повинен надавати можливість запуску нового процесу шляхом вибору виконуваного файлу.

***3.10 Розробка граф схеми алгоритму вузла запуску нового процесу***



*Рис. 14 Граф-схема алгоритму створення нового процесу*

Щоб запустити новий процес використовується наступний код:

delete newProcess;

newProcess = new QProcess();

newTaskFileName = QFileDialog::getOpenFileName(this, "New task", "\*.exe");

qDebug()<<newTaskFileName;

if(!newTaskFileName.isEmpty())

{

for(int i = 0 ; i < newTaskFileName.length();++i)//зміна шляху (шоб був коректний шлях)

{

if(newTaskFileName.at(i)=='/')

{

newTaskFileName.remove(i,1);

newTaskFileName.insert(i,"\\");

}

}

qDebug()<<newTaskFileName;

newProcess->start("cmd /k " + newTaskFileName);

}

**3.11. Розробка вузла завершення процесу.**

Даний вузол повинен забезпечити вибір процесу, який потрібно зупинити. Отримати доступ до процесу, через його ідентифікатор. Видалити процес з пам’яті.

*3.12 Розробка граф схеми алгоритму вузла “Завершення процесу”.*



*Рис. 15 Граф-схема алгоритму закриття процесу*

Щоб завершити процес потрібно використати наступний блок коду:

BOOL CALLBACK TerminateAppEnum(HWND hwnd, LPARAM lParam )//завершення програми

{

DWORD dwID ;

GetWindowThreadProcessId(hwnd, &dwID) ;

if(dwID == (DWORD)lParam)

{

PostMessage(hwnd, WM\_CLOSE, 0, 0) ;

}

return TRUE ;

}

void TaskManager::on\_endTask\_clicked()//закриття вікна

{

QModelIndex mi = ui->applications->selectionModel()->currentIndex();

if(mi.isValid())

{

EnumWindows((WNDENUMPROC)TerminateAppEnum, (DWORD)apps->item(mi.row(), 1)->text().toULong());

}

else

{

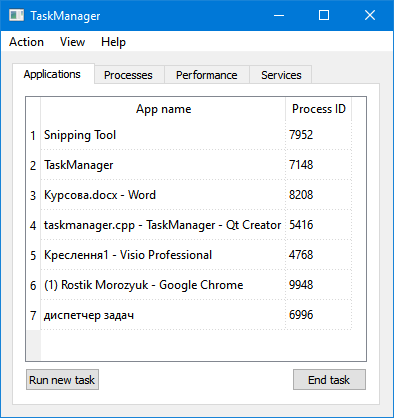
QMessageBox::warning(this, "Error", "Has not select item");

}

}

**4.Опис інтерфейсу та інструкції користувача.**

Програму «TaskManager» можна запустити з виконавчого файлу TaskManager.exe.



*Рис. 16 Головне вікно програми*

В програмі присутні наступні функції:

* + Отримання доступу до процесів, сервісів і служб і службової інформації про стан комп’ютера.
  + Запустити нову програму.
  + Завершити процес.
  + Переглянути список процесів, додатків, служб і драйверів і інформацію про них.
  + Запустити новий процес.
  + Завершити роботу вибраного процеса.
  + Завершити роботу даної програми.
  + Обрати швидкість оновлення інформації.
  + Переглянути інформацію про автора програми.

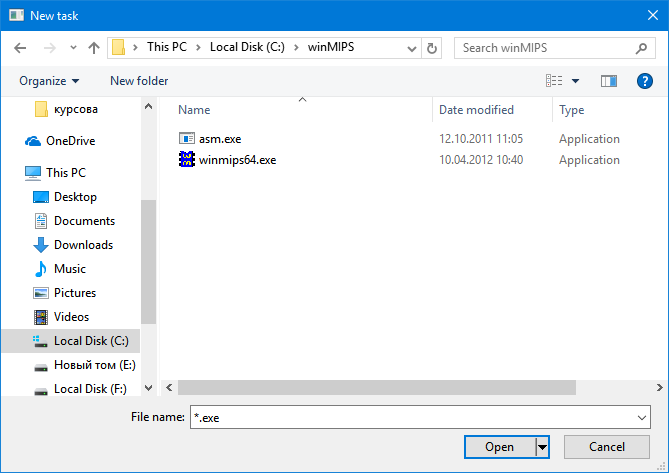
Головне вікно програми містить 4 вкладки, для відображення основної інформаці:

* + Додатки
  + Процеси
  + Швидкодія
  + Служби

На вкладці «Додатки» знаходяться дві функціональні кнопки:

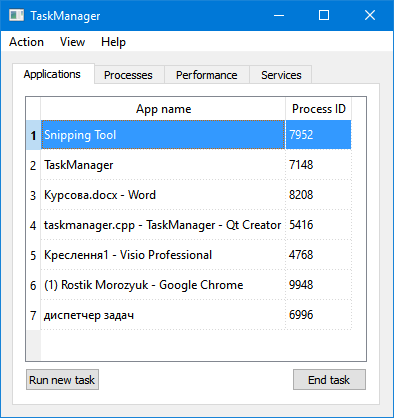
* + Запустити новий процес.
  + Завершити вибраний процес.

Щоб додати нову програму потрібно, натиснути кнопку “Запустити новий додаток”, обрати програму як потрібно запустити і нажати кнопку «Відкрити» щоб запустити програму чи копку «Відмінити», щоб відмінити дану дію.



*Рис. 17. Вікно вибору програми для запуску*

Щоб завершити роботу одного з процесів, користувач повинен виділити необхідний процес чи програму і нажати кнопку «Зупинити завдання», яка стає доступною після виділення процесу.



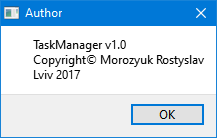
*Рис. 18 Завершення процесу*

Це ж саме користувач може зробити на вкладці «Процеси». На даній кладці Також присутня інформаці про кількість запущених процесів, завантаження процесора, кількість потоків.

Щоб завершити роботу програми користувач в панелі меню обирає «Дія» - «Вихід». Ще один спосіб запустити новий процес – «Дія» - «Запустити нове завдання».

Щоб оновити інформацію на поточній вкладці користувач обчирає з панелі меню кнопку «Перегляд» - «Оновити». Щоб задати швидкість оновлення – «Перегляд» - «Швидкість оновлення» і обирає потрібну швидкість (доступні 3 варіанти: низька(кожні 4 секунди), середня(3), висока(2).

Для перегляду інформації про автора додатку – «Допомога» - «Про автора».



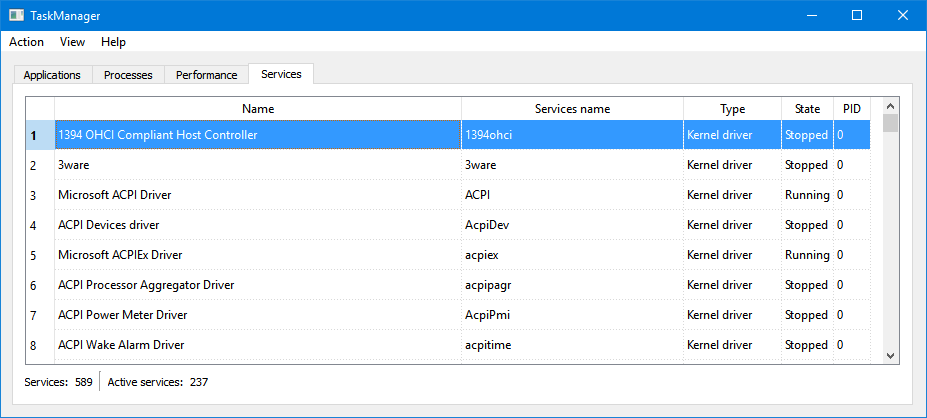
*Рис. 19 Автор програми*

**5. Тестування**

Відлагодження програми відбувається за допомогою автоматизованого відлагоджувача який присутній в середовищі QtCreator (використовується відлагоджувач GNU gdb 7.8), в покроковому режимі перевіряється значення потрібних змінних і вмістиме потрібних структур даних. За допомогою breakpoints відбувається запинка виконання програми в тих місцях де відбулася логічна помилка або у визначених місцях.

**5.1 Виявлення помилок відображення процесів, задач, сервісів і драйверів.**

До помилок на цьому етапі відносяться помилки програмного коду, який супроводжується некоректним відображенням даних. Як видно з рисунку всі дані відображаються коректно:



*Рис. 20 Вікно програми з коректно відображеними даними*

**5.2 Виявлення помилок створення нового процесу.**

До помилок на цьому етапі відносяться помилки неправильного задання шляху процесу. Така ситуація виявляється і виправляється на програмному рівні в даному коді:

if(!newTaskFileName.isEmpty())

{

for(int i = 0 ; i < newTaskFileName.length();++i)//зміна шляху (шоб був коректний шлях)

{

if(newTaskFileName.at(i)=='/')

{

newTaskFileName.remove(i,1);

newTaskFileName.insert(i,"\\");

}

}

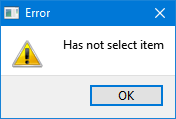
qDebug()<<newTaskFileName;

newProcess->start("cmd /k " + newTaskFileName);

}

**5.3. Виявлення помилок видалення нового процесу**

До помилок на цьому етапі відносяться помилка, коли користувач не вибрав процес і нажимає кнопку «Зупинити завдання». Програма повідомляє, що дана дія не виконана, потрібно обрати процес.



*Рис. 21 Повідомлення про помилку завершення роботи процесу*

**Висновок**

Підчас створення програмного продукту я набув навиуи роботи з процесами і їх керуванням.

На початкових стадіях розробки було проведено аналіз задачі проекту та розроблено граф схеми роботи програмного продукту. Наступні етапи проектування, використовуючи загальнотеоретичні положення сформульовані на ранніх стадіях синтезу, реалізовували проект на програмному рівні.

У курсовій роботі був розроблений «Диспетчер задач» з можливістю роботи з процесами, задачами, службами і драйверами операційної системи Windows. Також було проведене тестування всіх модулів, яке підтвердило, що весь код який міг призвести до помилок опрацьований і дозволяє працювати програмі коректно.

Під час виконання курсової роботи було успішно засвоєно методи розробки системних програм, а зокрема програми керування процесами і задачами.

**Список літератури**

1. Макс Шлее «Qt-5.3 Професійне програмування на срр”
2. Стівен Прата – «Мова програмування С++»
3. Макс Ріхтер «Windows via C++»

4. «Пишем свой системный монитор (Linux)» https://habrahabr.ru/post/186944/

**Додатки**

taskmanager.h

#ifndef TASKMANAGER\_H

#define TASKMANAGER\_H

#include <QMainWindow>

#include <QDebug>

#include <QStandardItemModel>

#include <QStandardItem>

#include <windows.h>

#include <tlhelp32.h>//навантаження процесора

#include <psapi.h>

#include <QString>

#include <QProcess>

#include <QTimer>

#include <QMessageBox>

#include <QFileDialog>

#include <QScrollBar>

#include <QSysInfo>

#include <QTime>

namespace Ui {

class TaskManager;

}

class TaskManager : public QMainWindow

{

Q\_OBJECT

public:

explicit TaskManager(QWidget \*parent = 0);

~TaskManager();

private:

Ui::TaskManager \*ui;

QTimer\* timer;

QTimer\* workerTimer;

int updateTime;//інтервал оновлень (необхідно для обчислення загрузки процесора)

double msInHour, msInMinute, msInSecond;

unsigned long long totalMemorySize;//загальна кількість пам'яті

double cpu;//завантаження процесора

QStandardItemModel\* apps;//модель запущених програм

QStandardItemModel\* process;//модель запущених процесів

QStandardItemModel\* services;//модель запущених процесів

int appsVScrollPos, appsHScrollPos;//позиція скрола аплікацій до оновлення

int procVScrollPos, procHScrollPos;//позиція скрола процесів до оновлення

int servicesVScrollPos, servicesHScrollPos;//позиція скрола процесів до оновлення

QString selectedActiveApps;//ім'я виділеної програми в списку запущених програм

QString selectedActiveProces;//ім'я виділеного процесу в списку запущених процесів

QString selectedActiveService;//ім'я виділеного процесу в списку запущених процесів

QString newTaskFileName;

QProcess\* newProcess;//запускає нові процеси

void activeApps();

void activeProcess();

void mainSystemInfo();

void servicesList();

QHash<DWORD, double> timeForProces;//зберігається час виділений процесу в попередній раз

signals:

void showActiveApps();

private slots:

void indexApp(QModelIndex);

void indexProc(QModelIndex);

void on\_endTask\_clicked();

void timeout();//час оновлення

void setUpdateSpeed();

void on\_newTask\_clicked();

void on\_actionExit\_triggered();

void on\_endTask2\_clicked();

void updateWorkTime();

void on\_actionAuthor\_triggered();

};

#endif // TASKMANAGER\_H

taskmanager.cpp

#include "taskmanager.h"

#include "ui\_taskmanager.h"

TaskManager::TaskManager(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::TaskManager)

{

ui->setupUi(this);

timer = new QTimer();

workerTimer = new QTimer();

newProcess = new QProcess();

appsVScrollPos = 0;

procVScrollPos = 0;

servicesVScrollPos = 0;

appsHScrollPos = 0;

procHScrollPos = 0;

servicesHScrollPos = 0;

//для визначення навантаження процесора

updateTime = 3000;

msInHour = 60.0\*60.0\*1000.0;

msInMinute = 60.0\*1000.0;

msInSecond = 1000.0;

connect(ui->tabWidget, SIGNAL(currentChanged(int)), this, SLOT(timeout()));

connect(ui->applications, SIGNAL(clicked(QModelIndex)), this, SLOT(indexApp(QModelIndex)));

connect(ui->processes, SIGNAL(clicked(QModelIndex)), this, SLOT(indexProc(QModelIndex)));

connect(timer, SIGNAL(timeout()), this, SLOT(timeout()));

connect(ui->actionHigh, SIGNAL(triggered(bool)), this, SLOT(setUpdateSpeed()));

connect(ui->actionMedium, SIGNAL(triggered(bool)), this, SLOT(setUpdateSpeed()));

connect(ui->actionLow, SIGNAL(triggered(bool)), this, SLOT(setUpdateSpeed()));

connect(ui->actionNew\_task, SIGNAL(triggered(bool)), this, SLOT(on\_newTask\_clicked()));

connect(ui->actionRefresh\_now, SIGNAL(triggered(bool)), this, SLOT(timeout()));

connect(workerTimer, SIGNAL(timeout()), this, SLOT(updateWorkTime()));

apps = new QStandardItemModel();

process = new QStandardItemModel();

services = new QStandardItemModel();

ui->applications->setHorizontalHeader(new QHeaderView(Qt::Horizontal));

ui->processes->setHorizontalHeader(new QHeaderView(Qt::Horizontal));

ui->services->setHorizontalHeader(new QHeaderView(Qt::Horizontal));

ui->processes->*setModel*(process);

ui->services->*setModel*(services);

mainSystemInfo();

activeApps();

timer->start(3000);

}

TaskManager::~*TaskManager*()

{

delete ui;

delete apps;

delete timer;

delete workerTimer;

delete process;

delete services;

if(newProcess)

delete newProcess;

}

void TaskManager::timeout()//оновлює відкриту вкладку

{

switch(ui->tabWidget->currentIndex())

{

case 0:

{

appsVScrollPos = ui->applications->verticalScrollBar()->value();

appsHScrollPos = ui->applications->horizontalScrollBar()->value();

QModelIndex curIndex = ui->applications->selectionModel()->currentIndex();//збереження поточно виділеного процеса, щоб виділити його при оновленні списку процесів

if(curIndex.isValid())

selectedActiveApps = apps->item(curIndex.row(), 0)->text();

activeApps();

timer->start(updateTime);

workerTimer->stop();

break;

}

case 1:

{

procVScrollPos = ui->processes->verticalScrollBar()->value();

procHScrollPos = ui->processes->horizontalScrollBar()->value();

QModelIndex curIndex = ui->processes->selectionModel()->currentIndex();

if(curIndex.isValid())

selectedActiveProces = process->item(curIndex.row(), 1)->text();

activeProcess();

timer->start(updateTime);

workerTimer->stop();

break;

}

case 2:

workerTimer->start(1000);

timer->start(updateTime);

activeProcess();

break;

case 3:

{

servicesVScrollPos = ui->services->verticalScrollBar()->value();

servicesHScrollPos = ui->services->horizontalScrollBar()->value();

QModelIndex curIndex = ui->services->selectionModel()->currentIndex();

if(curIndex.isValid())

selectedActiveService = services->item(curIndex.row(), 1)->text();

servicesList();

}

}

}

void TaskManager::setUpdateSpeed()

{

timer->stop();

QAction\* sndr = (QAction\*)sender();

if(sndr==ui->actionHigh)

{

ui->actionHigh->setChecked(true);

ui->actionLow->setChecked(false);

ui->actionMedium->setChecked(false);

timer->start(4000);

updateTime = 4000;

}

else if(sndr==ui->actionLow)

{

ui->actionHigh->setChecked(false);

ui->actionLow->setChecked(true);

ui->actionMedium->setChecked(false);

timer->start(2000);

updateTime = 2000;

}

else if(sndr==ui->actionMedium)

{

ui->actionHigh->setChecked(false);

ui->actionLow->setChecked(false);

ui->actionMedium->setChecked(true);

timer->start(3000);

updateTime = 3000;

}

}

BOOL CALLBACK TerminateAppEnum(HWND hwnd, LPARAM lParam )//завершення програми

{

DWORD dwID ;

GetWindowThreadProcessId(hwnd, &dwID) ;

if(dwID == (DWORD)lParam)

{

PostMessage(hwnd, WM\_CLOSE, 0, 0) ;

}

return TRUE ;

}

void TaskManager::on\_actionExit\_triggered()

{

exit(0);

}

//ВКЛАДКА ЗАПУЩЕНИХ ПРОГРАМ

QStandardItemModel\* tempApps;// НЕ ЧІПАТИ!!!! ДИКИЙ КОСТИЛЬ!!!!

BOOL CALLBACK EnumWindowsProc(HWND hWnd, LPARAM)//додаткова функція для знаходження активних аплікацій

{

char Buff[255], NameOfClass[255];

GetWindowText(hWnd, (LPWSTR)Buff, 254);

GetClassName(hWnd, (LPWSTR)NameOfClass, 254);

if(IsWindowVisible(hWnd))

{

QString str = QString::fromStdWString((LPWSTR)Buff);

if(!str.isEmpty())

{

bool flag = true;

DWORD dwID ;

GetWindowThreadProcessId(hWnd, &dwID) ;

for(int i = 0; i<tempApps->*rowCount*(); i++)//перевірка чи дане ід вже існує

{

if(tempApps->item(i, 1)->text().toULong() == dwID)

{

flag = false;

break;

}

}

if(flag)

tempApps->appendRow(QList<QStandardItem\*>()<<new QStandardItem(str)<<new QStandardItem(QString::number(dwID)));

}

}

return TRUE;

}

void TaskManager::activeApps()

{

apps->clear();

apps->setHorizontalHeaderItem(0, new QStandardItem("App name"));

apps->setHorizontalHeaderItem(1, new QStandardItem("Process ID"));

tempApps = apps;//НАВІТЬ НЕ ПРОБУЙ ЦЕ МІНЯТИ

EnumWindows((WNDENUMPROC)EnumWindowsProc, 0);

ui->applications->*setModel*(apps);

ui->applications->resizeColumnsToContents();

if(!selectedActiveApps.isEmpty())

{

for(int i=0;i<apps->*rowCount*();i++)

{

if(selectedActiveApps == apps->item(i, 0)->text())

{

ui->applications->selectRow(i);

break;

}

}

}

else

ui->applications->selectRow(0);

ui->applications->verticalScrollBar()->setValue(appsVScrollPos);

ui->applications->horizontalScrollBar()->setValue(appsHScrollPos);

}

void TaskManager::indexApp(QModelIndex index)

{

if(index.isValid())

ui->endTask->setEnabled(true);

else

ui->endTask->setEnabled(false);

}

void TaskManager::on\_endTask\_clicked()//закриття вікна

{

QModelIndex mi = ui->applications->selectionModel()->currentIndex();

if(mi.isValid())

{

EnumWindows((WNDENUMPROC)TerminateAppEnum, (DWORD)apps->item(mi.row(), 1)->text().toULong());

}

else

{

QMessageBox::warning(this, "Error", "Has not select item");

}

}

void TaskManager::on\_newTask\_clicked()

{

delete newProcess;

newProcess = new QProcess();

newTaskFileName = QFileDialog::getOpenFileName(this, "New task", "\*.exe");

qDebug()<<newTaskFileName;

if(!newTaskFileName.isEmpty())

{

for(int i = 0 ; i < newTaskFileName.length();++i)//зміна шляху (шоб був коректний шлях)

{

if(newTaskFileName.at(i)=='/')

{

newTaskFileName.remove(i,1);

newTaskFileName.insert(i,"\\");

}

}

qDebug()<<newTaskFileName;

newProcess->start("cmd /k " + newTaskFileName);

}

}

//ВКЛАДАКА ПРОЦЕСІВ!!!!!

void TaskManager::activeProcess()

{

HANDLE hSnapshot = CreateToolhelp32Snapshot(TH32CS\_SNAPPROCESS, 0);//знімок запущених процесів

PROCESSENTRY32 pe; //інформація про знайдений процес

if (hSnapshot == INVALID\_HANDLE\_VALUE)

{

QMessageBox::warning(this, "Find process error", "Can not take snaphot\nINVALID\_HANDLE\_VALUE");

return;

}

process->clear();

process->setHorizontalHeaderItem(0, new QStandardItem("Process name"));

process->setHorizontalHeaderItem(1, new QStandardItem("Process ID"));

process->setHorizontalHeaderItem(2, new QStandardItem("Threads number"));

process->setHorizontalHeaderItem(3, new QStandardItem("Parent PID"));

process->setHorizontalHeaderItem(4, new QStandardItem("Priority"));

process->setHorizontalHeaderItem(5, new QStandardItem("CPU"));

process->setHorizontalHeaderItem(6, new QStandardItem("Memory"));

pe.dwSize = sizeof(PROCESSENTRY32);

int procCount = 0;

int threadCount = 0;

cpu = 0.0;

if (Process32First(hSnapshot, &pe)) //Пошук першого знімку в снепшоті

{

do

{

procCount++;

//if (!pe.th32ProcessID) continue; // Пропуск [System process]

HANDLE hProcess = OpenProcess(PROCESS\_QUERY\_INFORMATION |PROCESS\_VM\_READ,FALSE, pe.th32ProcessID); //Получаємо дескриптор процеса

//визначення пріоритета процеса

QString procPriority;

switch (GetPriorityClass(hProcess)) //Получаем пріоритет процеса

{

case HIGH\_PRIORITY\_CLASS:

procPriority = "High";

break;

case IDLE\_PRIORITY\_CLASS:

procPriority = "Idle";

break;

case NORMAL\_PRIORITY\_CLASS:

procPriority = "Normal";

break;

case REALTIME\_PRIORITY\_CLASS:

procPriority = "Realtime";

break;

case ABOVE\_NORMAL\_PRIORITY\_CLASS:

procPriority = "Above normal";

break;

case BELOW\_NORMAL\_PRIORITY\_CLASS:

procPriority = "Below normal";

break;

default:

procPriority = "No access";

}

//визначення загрузки процесора

FILETIME ftCreation,ftExit,ftKernel,ftUser;

SYSTEMTIME stCreation,stExit,stKernel,stUser;

double procent=0;//відсоток завантаження процесора

if(GetProcessTimes(hProcess, &ftCreation, &ftExit, &ftKernel, &ftUser))//час виділений процесу системою

{

//приведення часу в нормальний формат

FileTimeToSystemTime(&ftCreation, &stCreation);

FileTimeToSystemTime(&ftExit, &stExit);

FileTimeToSystemTime(&ftUser, &stUser);

FileTimeToSystemTime(&ftKernel, &stKernel);

//час роботи в режимі користувача

double user = ((double)stUser.wHour\*msInHour +

(double)stUser.wMinute\*msInMinute +

(double)stUser.wSecond\*msInSecond +

(double)stUser.wMilliseconds)/updateTime;//час в інтервалі якого знято показники

//час роботи в режимі ядра

double kernel = ((double)stKernel.wHour\*msInHour +

(double)stKernel.wMinute\*msInMinute +

(double)stKernel.wSecond\*msInSecond +

(double)stKernel.wMilliseconds)/updateTime;//час в інтервалі якого знято показники

double res = user + kernel;

QHash<DWORD, double>::iterator it = timeForProces.find(pe.th32ProcessID);

if(it!=timeForProces.end())//якшо такий елемент знайдено

{

procent = (res - it.value())\*100.0;

timeForProces.remove(pe.th32ProcessID);

}

timeForProces.insert(pe.th32ProcessID, res);

cpu+=procent;

}

//визначення пам'яті процеса

PROCESS\_MEMORY\_COUNTERS pmc;

GetProcessMemoryInfo(hProcess, &pmc, sizeof(pmc));

process->appendRow(QList<QStandardItem\*>()<<new QStandardItem(QString::fromStdWString(pe.szExeFile))

<<new QStandardItem(QString::number(pe.th32ProcessID))

<<new QStandardItem(QString::number(pe.cntThreads))

<<new QStandardItem(QString::number(pe.th32ParentProcessID))

<<new QStandardItem(procPriority));

if(procPriority=="No access")

{

process->setItem(process->*rowCount*()-1, 5, new QStandardItem("No access"));

process->setItem(process->*rowCount*()-1, 6, new QStandardItem("No access"));

}

else

{

process->setItem(process->*rowCount*()-1, 5, new QStandardItem(QString::number(procent, 'g', 2)+"%"));

process->setItem(process->*rowCount*()-1, 6, new QStandardItem(QString::number(((double)pmc.WorkingSetSize/totalMemorySize)\*100, 'g', 2)+"%"));

}

threadCount+=pe.cntThreads;

CloseHandle(hProcess);

} while (Process32Next(hSnapshot, &pe)); //пошук наступного процеса

}

else

QMessageBox::warning(this, "Show process error", "Cannot showproceses");

CloseHandle(hSnapshot);

if(!selectedActiveProces.isEmpty())//якшо закинути вище, не буде працювати виділення через постійне оновлення моделі

{

for(int i=0;i<process->*rowCount*();i++)

{

if(selectedActiveProces == process->item(i, 1)->text())

{

ui->processes->selectRow(i);

break;

}

}

}

else

ui->processes->selectRow(0);

ui->processes->*setModel*(process);

ui->processes->verticalScrollBar()->setValue(procVScrollPos);

ui->processes->horizontalScrollBar()->setValue(procHScrollPos);

ui->procNum->setText(QString::number(procCount));

ui->threadsNum->setText(QString::number(threadCount));

ui->cpuNum->setText(QString::number(cpu, 'g', 2)+"%");

ui->processes->resizeColumnsToContents();

ui->cpuPerf->setValue(cpu);

}

void TaskManager::indexProc(QModelIndex index)

{

if(index.isValid())

ui->endTask2->setEnabled(true);

else

ui->endTask2->setEnabled(false);

}

void TaskManager::on\_endTask2\_clicked()

{

QModelIndex mi = ui->processes->selectionModel()->currentIndex();

if(mi.isValid())

{

EnumWindows((WNDENUMPROC)TerminateAppEnum, (DWORD)process->item(mi.row(), 1)->text().toULong());

}

else

{

QMessageBox::warning(this, "Error", "Has not select item");

}

}

//ВКЛАДКА PERFOMANCE

void TaskManager::mainSystemInfo()

{

//ім'я системи

char buf[20];

DWORD size = 20;

GetUserNameA((LPSTR)buf, &size);

ui->userName->setText(QString(buf));

//час роботи системи

workerTimer->start(1000);

updateWorkTime();

//розмір пам'яті

MEMORYSTATUSEX statex;

statex.dwLength = sizeof (statex);

GlobalMemoryStatusEx (&statex);

totalMemorySize = statex.ullTotalPhys;

ui->physMemory->setText(QString::number(totalMemorySize/(1024\*1024), 'g', 6)+"MB");//фізична пам'ять

}

void TaskManager::updateWorkTime()

{

QTime systemWorkTime(0,0,0);

systemWorkTime = systemWorkTime.addMSecs(GetTickCount());

ui->uptime->setText(systemWorkTime.toString("hh:mm:ss"));

MEMORYSTATUSEX statex;

statex.dwLength = sizeof (statex);

GlobalMemoryStatusEx (&statex);

ui->memUse->setValue(statex.dwMemoryLoad);

ui->virtMemmory->setText(QString::number(statex.ullTotalVirtual/(1024\*1024), 'g', 6) + "MB");//віртуальна пам'ять об'єм

}

//СПИОК СЕРВІСІВ

void TaskManager::servicesList()

{

services->clear();

services->setHorizontalHeaderItem(0, new QStandardItem("Name"));

services->setHorizontalHeaderItem(1, new QStandardItem("Services name"));

services->setHorizontalHeaderItem(2, new QStandardItem("Type"));

services->setHorizontalHeaderItem(3, new QStandardItem("State"));

services->setHorizontalHeaderItem(4, new QStandardItem("PID"));

unsigned char buf[256000];

DWORD bufSize = 256000;

DWORD bufSizeNeed=0, serviceNumber=0;

SC\_HANDLE hSCM = NULL;

LPENUM\_SERVICE\_STATUS\_PROCESS info = NULL;//структури сервісів

hSCM = OpenSCManager(NULL, NULL, SC\_MANAGER\_ALL\_ACCESS);//менеджер доступу до сервісів (повний доступ)

if(hSCM==NULL)

{

if(GetLastError()==ERROR\_ACCESS\_DENIED)

QMessageBox::warning(this, "Acces error", "The requested access was denied.");

else if(GetLastError()==ERROR\_DATABASE\_DOES\_NOT\_EXIST)

QMessageBox::warning(this, "Acces error", "The specified database does not exist.");

timer->stop();

return;

}

EnumServicesStatusEx(hSCM,

SC\_ENUM\_PROCESS\_INFO,//беремо інформацію про процеси

SERVICE\_WIN32 | SERVICE\_DRIVER | SERVICE\_FILE\_SYSTEM\_DRIVER | SERVICE\_KERNEL\_DRIVER, // пошук типу сервісів

SERVICE\_STATE\_ALL,//Всі сервіси і активні і неактивні

buf,//буфер списку сервісів

bufSize,

&bufSizeNeed,//вертає кількість необхідних байт, якшо замало пам'яті в буфері

&serviceNumber,//кількість сервісів

NULL,

NULL);

qDebug()<<bufSizeNeed;

qDebug()<<serviceNumber;

info = (LPENUM\_SERVICE\_STATUS\_PROCESS)buf;

int activeServices=0;

for(ULONG i = 0;i<serviceNumber;i++)

{

//визначення типу

QString type;

switch(info[i].ServiceStatusProcess.dwServiceType)

{

case SERVICE\_FILE\_SYSTEM\_DRIVER:

type = "File system driver";

break;

case SERVICE\_KERNEL\_DRIVER:

type = "Kernel driver";

break;

case SERVICE\_WIN32\_OWN\_PROCESS:

type = "Own process";

break;

case SERVICE\_WIN32\_SHARE\_PROCESS:

type = "Shares a process";

break;

}

//визначення стану

QString state;

switch(info[i].ServiceStatusProcess.dwCurrentState)

{

case SERVICE\_PAUSED:

state = "Paused";

break;

case SERVICE\_RUNNING:

state = "Running";

activeServices++;

break;

case SERVICE\_STOPPED:

state = "Stopped";

break;

}

services->appendRow(QList<QStandardItem\*>()<< new QStandardItem(QString::fromStdWString(info[i].lpDisplayName))

<< new QStandardItem(QString::fromStdWString(info[i].lpServiceName))

<< new QStandardItem(type)

<< new QStandardItem(state)

<< new QStandardItem(QString::number(info[i].ServiceStatusProcess.dwProcessId)));

}

if(!selectedActiveService.isEmpty())

for(int i=0;i<services->*rowCount*();i++)

{

if(selectedActiveService == services->item(i, 1)->text())

{

ui->services->selectRow(i);

break;

}

}

else

ui->services->selectRow(0);

ui->services->*setModel*(services);

ui->services->verticalScrollBar()->setValue(servicesVScrollPos);

ui->services->horizontalScrollBar()->setValue(servicesHScrollPos);

ui->servicesNum->setText(QString::number(serviceNumber));

ui->activeServicesNum->setText(QString::number(activeServices));

ui->services->resizeColumnsToContents();

CloseServiceHandle(hSCM);

}

void TaskManager::on\_actionAuthor\_triggered()

{

QMessageBox::about(this, "Author", "TaskManager v1.0\nCopyright© Morozyuk Rostyslav\nLviv 2017");

}

main.cpp

#include "taskmanager.h"

#include <QApplication>

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

{

QApplication a(argc, argv);

TaskManager w;

w.show();

return a.exec();

}