МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Державний вищий навчальний заклад

“Нововолинський електромеханічний коледж”

Комп’ютерно-економічне відділення

Циклова комісія комп’ютерних дисциплін

**Пояснювальна записка**

до дипломного проекту молодшого спеціаліста

на тему: Створення програми комплексного тестування апаратних складових обчислювальної системи

Виконав: студент ІV курсу, групи 2-КТ-14

Спеціальності: 5.05010201

"Обслуговування комп'ютерних систем і мереж”

**Джус С. В.**

|  |  |  |
| --- | --- | --- |
| Керівник |  | **Ільїн О.О.** |
| Консультант з економічної частини |  | **Камінська О.Ю.** |
| Нормоконтроль |  | **Ткаченко Ю.І.** |
| Рецензент |  |  |

м. Нововолинськ - 2018 рік

**АНОТАЦІЯ**

У даному дипломному проекті описується за допомогою яких методів можна отримати детальну інформацію про систему, її будову, продуктивність різноманітних апаратних компонентів, описання за яким принципом проводилося тестування.

**ЗМІСТ**

ВСТУП

1 ОГЛЯД ПРОГРАМ ДЛЯ ТЕСТУВАННЯ ТА ВИЗНАЧЕННЯ ПРОДУКТИВНОСТІ ПК………………………………………………….………………………………...5

2 ПОСТАНОВКА ЗАДАЧІ ТА РЕАЛІЗАЦІЯ ПРОГРАМИ………………………………………………………………………...10

2.1 Визначення характеристики та продуктивності компонентів ПК …..10

2.2 Створення звіту……………………………………………………….....11

3 ВИКОРИСТАННЯ ПРОГРАМИ………………………………………..………12

3.1 Тестування компонентів ПК…………………………….……………...12

3.2 Визначення характеристик компонентів ПК………….…………….....12

3.3 Визначення продуктивності компонентів ПК……………………..…..26

3.4 Генерація та відправка звіту роботи програми…...…………………....30

4 ЕКОНОМІЧНА ЧАСТИНА……………………………………………………...32

ВИСНОВОК……………………………………………………………….………..43

ПЕРЕЛІК ПОСИЛАНЬ..……………………………………...………….………...44

**ВСТУП**

Програми для тестування продуктивності ПК є дуже поширеними. Існують різноманітні програми для тестування ПК, як спеціалізовані, так і для широкого застосування. Наприклад, є програми для призначені тільки для виведення загальної інформацію про те, які компоненти містить ПК, є програми, які саме тестують продуктивність компонентів (швидкість оперативної пам’яті, швидкість роботи процесора, швидкість читання/запису жорсткого диску). Також є програми, які містять у собі усі вище згадані функції. Дана дипломна робота передбачає створення програми, яка дозволяє переглянути загальні характеристики ПК, і тестує компоненти ПК.

Однак дана програма призначена для сімейства операційних систем GNU/Linux, яка поєднала у собі детальну інформацію про систему та, зручний графічний інтерфейс.

На сьогоднішній день такі програми є дуже актуальні, так як існує дуже багато різноманітного апаратного забезпечення, яке має свої унікальні характеристики. За допомогою програм для тестування ПК, можна порівнювати будь-які компоненти, та дізнатися про їх продуктивність та, багато іншої цінної інформації.

Вони можуть використовуватися для багатьох різноманітних цілей. Наприклад, у певній компанії, потрібно перевірити на скільки їх обладнання відповідає їхнім потребам, для цього можна скористатися даною програмою, та зібрати інформацію про всі ПК, а потім проаналізувати отримані результати та приймати певні рішення.

**1 Огляд програм для тестування та визначення продуктивності ПК**

На даний момент існує досить багато програм для тестування компонентів ПК. До них входять:

**HWMonitor** — невелика безкоштовна програма для моніторингу показників різних компонентів комп’ютера. HWMonitor виводить показники температури різних компонентів комп’ютера, швидкості обертання вентиляторів і напруги в контрольних точках. Корисна для швидкої діагностики різних параметрів ПК на відповідність рекомендованим параметрам.

**CPU-Z** — це безкоштовна прикладна програма-утиліта для відображення технічної інформації про персональний комп'ютер користувача (характеристик і параметрів роботи компонентів), яка працює під ОС Microsoft Windows починаючи з версії Windows 98 (включаючи Windows 10). Випускається також спеціальна версія яка працює на ОС Android. Програма визначає технічні характеристики центрального процесора, материнської плати, BIOS, оперативної пам'яті, відеокарти, окрім жорсткого диска. Популярна серед IT-фахівців, комп'ютерних техніків і ремонтників, геймерів і оверклокерів.

Програма дозволяє отримувати наступні відомості:

* Назва процесора;
* Архітектура;
* Сокет;
* Техпроцес;
* Напруга живлення ядра;
* Сімейство;
* Степпінг і Ревізія;
* Підтримувані набори інструкцій;
* Тактова частота;
* Об'єм кешу всіх рівнів;
* Фізична організація кешу;
* Кількість процесорів і процесорних ядер;
* Відомості про материнську плату;
* Виробник;
* Модель;
* Чипсет та його ревізія;
* Південний міст;
* Модель;
* Версія BIOS;
* Графічний інтерфейс і кількість ліній (для PCI-Express);
* Відомості про оперативну пам’ять;
* Тактова частота і таймінги;
* Кількість каналів пам’яті;
* Повна інформація, що мітиться в SPD;
* Кількість слотів пам’яті;
* Детальна інформація про модулі, встановленому в кожен слот;
* Тип, об’єм відеопам’яті;
* Частоти відеочіпа, відеопам’яті, шейдерного домену;

Окрім того, програма дозволяє створювати докладні звіти форматів .txt і .html, а також виконувати валідацію: викладати інформацію про систему (в форматі .cvf) на спеціальний сайт CPU-Z Validator містить базу даних про тактових частотах компонентів і іншу інформацію. На сайті також присутній зал слави.

**Everest** — програма для перегляду інформації про апаратну і програмну конфігурації комп'ютера, розроблена компанією Lavalys. Була послідовницею AIDA32, на даний момент, замінена на ринку програмою AIDA64.

Програма аналізує конфігурацію комп'ютера і видає детальну інформацію про:

* Встановлені в системі пристрої - процесори, системні плати, відеокарти, аудіокарти, модулі пам'яті і т. д;
* Їхні характеристики: тактова частота, напруга живлення, розмір кешей, і т. д;
* Підтримувані ними набори команд і режими роботи;
* Їх виробники;
* Встановлене програмне забезпечення;
* Конфігурації операційної системи;
* Встановлені драйвери;
* Автоматично завантажувані програми;
* Запущені процеси;
* Наявні ліцензії;

B програмі є досить широкий набір бенчмаркінгових тестів:

* Читання з пам'яті - тестує швидкість пересилання даних з ОЗП до процесора;
* Запис в пам'ять - тестує швидкість пересилання даних з процесора до ОЗП;
* Копіювання в пам'ять - тестує швидкість пересилання даних з одних осередків пам'яті в інші через кеш процесора;
* Затримка пам'яті - тестує середній час зчитування процесором даних з ОЗП;
* CPU Queen - тестує продуктивність процесора в цілочисельних операціях при вирішенні класичного «Завдання з ферзями»;
* CPU PhotoWorxx - тестує продуктивність блоків цілочисельних арифметичних операцій, множення, а також підсистеми пам'яті при виконанні ряду стандартних операцій з RGB-зображеннями;
* CPU Zlib - тестує продуктивність процесора і підсистеми пам'яті при створенні архівів формату .ZIP, за допомогою популярної відкритої бібліотеки ZLib. Використовує цілочисельні операції;
* CPU AES - тестує швидкість процесора при виконанні шифрування по кріптоалгорітму AES. Здатний використовувати низькорівневі команди шифрування процесорів VIA C3 і C7;
* FPU Julia - тестує продуктивність блоків процесора, що виконують операції з плаваючою комою, в обчисленнях з 32-розрядною точністю. Моделює кілька фрагментів фрактала Жюліа. При можливості використовує інструкції MMX, SSE і 3DNow;
* FPU Mandel - тестує продуктивність блоків процесора, що виконують операції з плаваючою комою, в обчисленнях з 64-розрядною точністю шляхом моделювання декількох фрагментів фрактала Мандельброта. Здатний використовувати інструкції SSE2;
* FPU SinJulia - ускладнений варіант тесту FPU Julia. Тестує продуктивність блоків процесора, що виконують операції з плаваючою комою, в обчисленнях з 80-розрядною точністю. Використовує інструкції x87, призначені для обчислення тригонометричних функцій;

**AIDA32** — в минулому популярна безкоштовна програма для перегляду інформації про апаратну і програмну конфігурації комп'ютера. Її розвитком стали платні програми Everest (2004 — 2010) и AIDA64 (з 2010).

Програма аналізує конфігурацію комп'ютера та видає детальну інформацію про:

* Встановлені в системі пристрої - процесори, системні плати, аудіокарти, модулі пам'яті і т. д;
* Їх характеристики: тактова частота, напруга живлення, розмір кешей, і т. д;
* Температуру центрального процесора (CPU);
* Підтримуваних ними наборах команд і режимах роботи;
* Їх виробників;
* Встановлене програмне забезпечення;
* Конфігурації операційної системи;
* Встановлені драйвери;
* Автоматично завантажуваних програмах;
* Запущені процеси;
* Наявні ліцензії;

У програмі є тест запису в пам'ять і читання з пам'яті з можливістю порівняння їх результатів з еталонними.

Інтерфейс багатомовний і встановлюється автоматично в залежності від встановленої мови.

У Aida32 є 16-бітна версія для використання в DOS.

**2 ПОСТАНОВКА ЗАДАЧІ ТА РЕАЛІЗАЦІЯ ПРОГРАМИ**

**2.1 Визначення характеристик та продуктивності компонентів ПК**

**Визначення характеристик.**

Програма має виконувати наступні задачі:

* Показувати характеристики компонентів ПК;
* Показувати запущені процеси;
* Показувати завантажені модулі;
* Виводити загальну інформацію про компонентну будову;

**Визначення продуктивності компонентів ПК.**

Програма повинна тестувати наступні компоненти:

* Оперативну пам’ять;
* Жорсткий диск;
* Процесор;

Для визначення продуктивності оперативної пам’яті програма використовує наступний алгоритм – копіюється певний об’єм пам’яті з одного місця в інше, та визначає час, за який ця операція здійснюється. Маючи час, та об’єм пам’яті, який було скопійовано, можна дізнатися швидкість копіювання у Мб/сек.

Для тестування швидкості читання/запису на жорсткий диск було створено наступний алгоритм – виділяється певний об’єм пам’яті, та записується файл розміром у 512 мегабайт, аналогічно зчитування. Після чого було отримано час з яким жорсткий диск записує та читає інформацію.

Для визначення продуктивності процесора було створено наступні алгоритми:

* Для визначення продуктивності арифметичних операцій здійснюються наступні дії: виконуються 1024\*1024\*32 ітерацій, та у кожній ітерації циклу виконуються по 17 операції додавання, відніманні, ділення, множення, інкременту, та декременту; та вираховується час, за який виконувались ці операції;
* для визначення продуктивності кешу процесора створюються два циклу з різними стратегіями проходу по оперативній пам’яті:

1. у першому випадку по-пам’яті виконується послідовний прохід — байт за байтом, у цьому випадку промахів кешу процесора є менше ніж у другому випадку, так як процесор кешує пам’ять розташовану поблизу пам’яті, яку в даних момент читають;
2. у другому випадку прохід по-пам’яті зроблений більш хаотичним, що у свою чергу зменшує швидкість звернення до пам’яті, так як є більше промахів кешу;
   1. **Створення звіту**

Звіт складається у вигляді HTML файлу, який надсилається на сервер по заданому IP, та порту. Такий спосіб протоколювання дуже зручний, так як можна запустити сервер, на який буде збиратися інформація про комп’ютери певної компанії. Тоді компанія буде знати, у якому стані знаходяться їхні комп’ютери, та чи потребують вони оновлення, чи заміни.

Для надсилання звіту по мережі, використовується протокол обміну інформацією TCP/IP. Даний протокол було обрано, так як він підходить по усім параметрам, які потрібно для коректної роботи даної програми, а саме:

1. Надійне з’єднання, яке забезпечує надійність цілісності даних, на відміну від протоколу UDP, який не встановлює з’єднання, та не гарантує цілісність даних.
2. Достатня швидкість передачі інформації.

**3 ВИКОРИСТАННЯ ПРОГРАМИ**

**3.1 Тестування компонентів ПК**

Тестування компонентів ПК поділяється на:

1. збір відомостей про комплектуючі – «хороші» програми надають найдетальнішу інформацію, ті, що «відстають» же замість цього виводять на екран набір специфічних позначень, зрозумілих лише фахівцеві з «заліза». Хочете наростити об'єм оперативної пам'яті? За допомогою програми діагностики ви зможете дізнатися, скільки слотів вже зайнято і які модулі пам'яті використовуються в системі.
2. тестування продуктивності ПК – отриманий результат можна порівняти з еталонними системами, щоб прийняти правильне рішення про те, чи варто проводити модернізацію комп'ютера.
3. виведення детальних довідкових відомостей – діагностичні утиліти, окрім інформації про комплектуючі, надають і детальніші дані про той або інший компонент системи або дають посилання на веб-сторінки виробника.

**3.2 Визначення характеристик компонентів ПК**

Дана програма в основному призначена для досвідчених користувачів, але й новачки знайдуть і ній багато цікавого.

У даному пункті та підпунктах плану детально розглянуто використання програми, пояснено як нею користуватися, та розглянуто як аналізувати отримані дані.

Для збору інформації про компонентну будову було використано наступні утиліти:

* **lshw** — [утиліт](https://uk.wikipedia.org/wiki/%D0%A3%D1%82%D0%B8%D0%BB%D1%96%D1%82%D0%B0)а для UNIX-подібних операційних систем, яка дозволяє отримати детальну інформацію про компоненту будову ПК, наприклад: скільки є у наявності оперативної пам’яті, марка процесора, його тактова частота, об’єм жорсткого диску, мережеві інтерфейси, інформацію про відеокарту, та інше.
* **lsusb** — [утиліт](https://uk.wikipedia.org/wiki/%D0%A3%D1%82%D0%B8%D0%BB%D1%96%D1%82%D0%B0)а для відображення USB інтерфейсів, так їх поточне використання;
* **lsmod** — [утиліт](https://uk.wikipedia.org/wiki/%D0%A3%D1%82%D0%B8%D0%BB%D1%96%D1%82%D0%B0)а для виведення списку завантажених модулів у UNIX-подібних системах;
* **fdisk** — [утиліт](https://uk.wikipedia.org/wiki/%D0%A3%D1%82%D0%B8%D0%BB%D1%96%D1%82%D0%B0)а для керування розділами жорсткого [диска](https://uk.wikipedia.org/wiki/%D0%A2%D0%B2%D0%B5%D1%80%D0%B4%D0%B8%D0%B9_%D0%B4%D0%B8%D1%81%D0%BA) чи іншого [носія інформації](https://uk.wikipedia.org/wiki/%D0%9D%D0%BE%D1%81%D1%96%D0%B9_%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D1%97). [Програма](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%B0) використовує консольний [інтерфейс](https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81);
* **ps** — програма в UNIX, Unix-подібних і інших POSIX-сумісних операційних системах, що виводить звіт про працюючі процеси;

На рисунку 3.1 зображене головне меню, за допомогою якого можна переглянути:

1. Інформацію про систему.
2. Результати тестування.

А також – запустити сервер для отримання звітів про тестування, або інформацію системи.

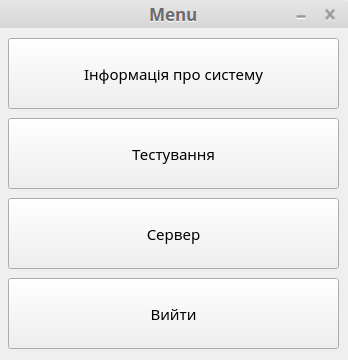


Рисунок 3.1 – Меню

Для того щоб переглянути компоненти ПК потрібно просто натиснути на кнопку «Інформація про систему», у якому можна вибрати, про що саме потрібно дізнатися інформацію (рис. 3.2).

У цьому вікні можна дізнатися інформацію про наступні компоненти:

* Процесор;
* USB;
* Відеокарту;
* DMI — інформація про BIOS та модель ПК;
* Мережеві карти;
* Оперативну пам’ять;
* Жорсткий диск;
* Завантажені модулі;
* Процеси;
* Кількість вільної та доступної пам’яті, розділ підвантаження пам’яті;

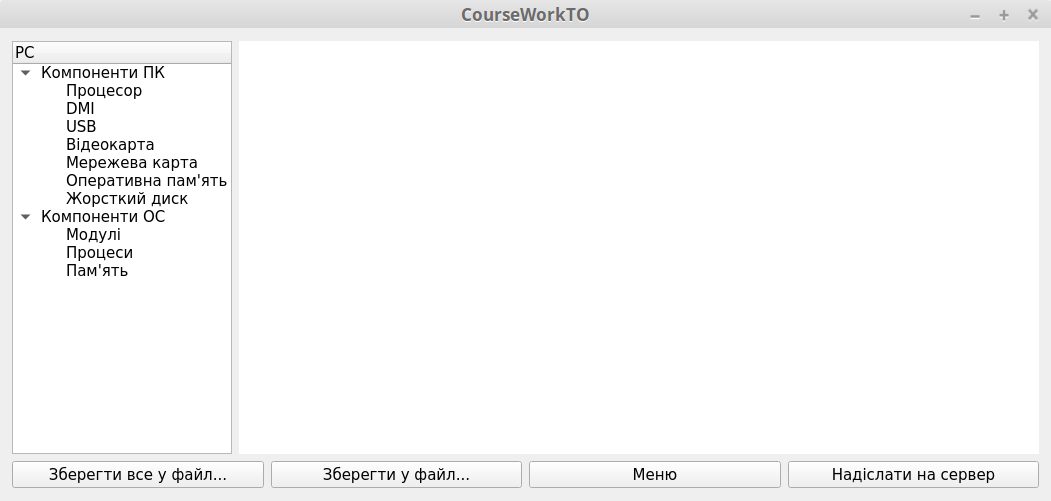


Рисунок 3.2 – вікно для перегляду інформації про систему

**Перегляд інформації про CPU**

При натискані на підпункт “Процесор”, з'явиться інформація про процесор ПК(рис. 3.3), де можна отримати наступну інформацію:

* Назву продукту;
* Постачальника;
* Фізичний ідентифікатор;
* Слот процесора;
* Тактову частоту;
* Розрядність шини;
* Частоту системного таймеру процесора;
* Інформацію про різні рівні кешу;
* Об’єм різних рівнів кешу;
* Частоту системного таймеру кешу;
* Можливості процесора, його особливості, можливості;

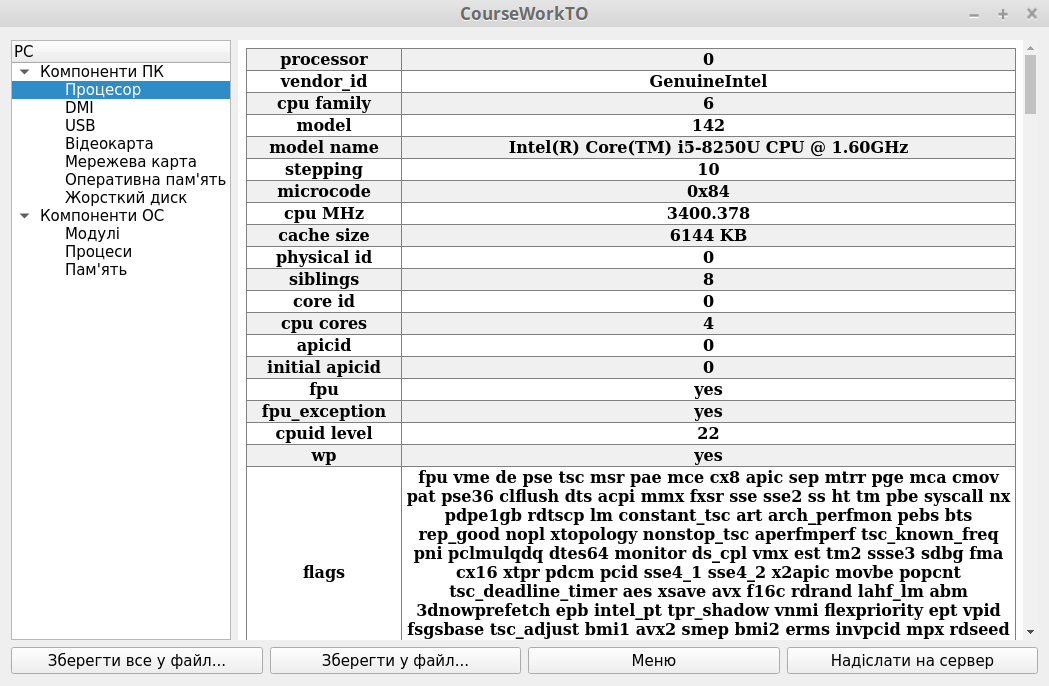
При тестуванні процесора основну увагу варто звернути на тактову частоту процесора, кількість ядер та архітектуру.

Тактова частота процесора – основна одиниця вимірювання частоти тактів у синхронних колах, що визначає кількість елементарних операцій (тактів), що виконуються системою за 1 секунду.

Ядро – частина мікропроцесора, що містить основні функціональні блоки.

Архітектура процесора – з погляду програмістів, під архітектурою процесора мається на увазі його здатність виконувати певний набір машинних кодів. Більшість сучасних десктопних процесорів відносяться до сімейства x86, або Intel-сумісних процесорів архітектури IA32 (архітектура 32-бітових процесорів Intel). Її основа була закладена компанією Intel в процесорі i80386, проте в подальших поколіннях процесорів вона була доповнена і розширена як самою Intel (введені нові набори команд MMX, SSE, SSE2 і SSE3), так і сторонніми виробниками (набори команд EMMX, 3DNow! і Extended 3DNow!, розроблені компанією AMD).

Даної інформації цілком достатньо для точної оцінки можливостей процесора, та визначення його продуктивності, так як найбільш важливі деталі є детально описані.

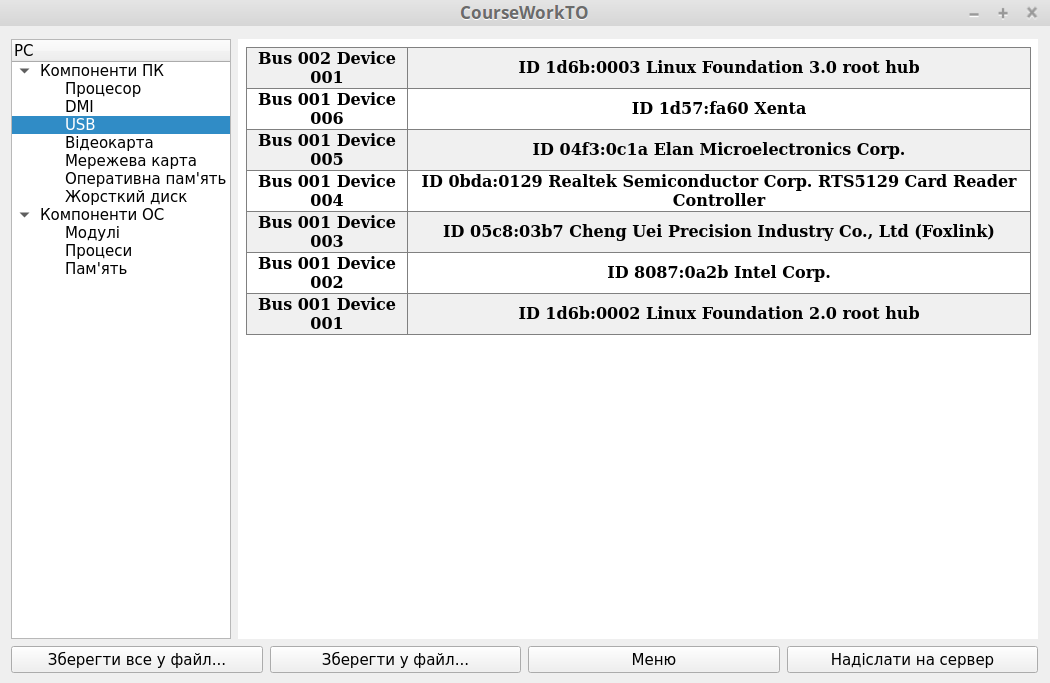
Рисунок 3.3 – інформація про процесор

**Перегляд інформації про USB пристрої**

USB - універсальна послідовна шина, призначена для з'єднання периферійних пристроїв обчислювальної техніки. Символом USB є чотири геометричні фігури: квадрат, трикутник, велике коло та мале коло.

Для перегляду інформації про USB інтерфейси потрібно натиснути на підпункт «USB».

Тут можна подивитися, і зрозуміти, які системні пристрої підключені до USB інтерфейсів (рис. 3.4).

Рисунок 3.4 - Перегляд інформації про USB пристрої

**Перегляд інформації про відеокарту**

Для перегляду інформації про відеокарту потрібно вибрати підпункт «Відеокарта» (рис. 3.5).

Відеокарта - окремий пристрій персонального комп'ютера або ігрової приставки, виконує графічний рендеринг. Сучасні графічні процесори дуже ефективно обробляють і зображують комп'ютерну графіку, завдяки спеціалізованій конвеєрній архітектурі вони набагато ефективніші в обробці графічної інформації, ніж типовий центральний процесор.

Графічний процесор в сучасних відеоадаптерах використовується як прискорювач тривимірної графіки, але в деяких випадках його можна використовувати і для обрахунків (GPGPU). Обрахунковими особливостями в порівнянні із CPU є:

1. архітектура, максимально націлена на збільшення швидкості обчислень текстур та складних графічних об'єктів;
2. обмежений список команд.

Програма дає наступні відомості:

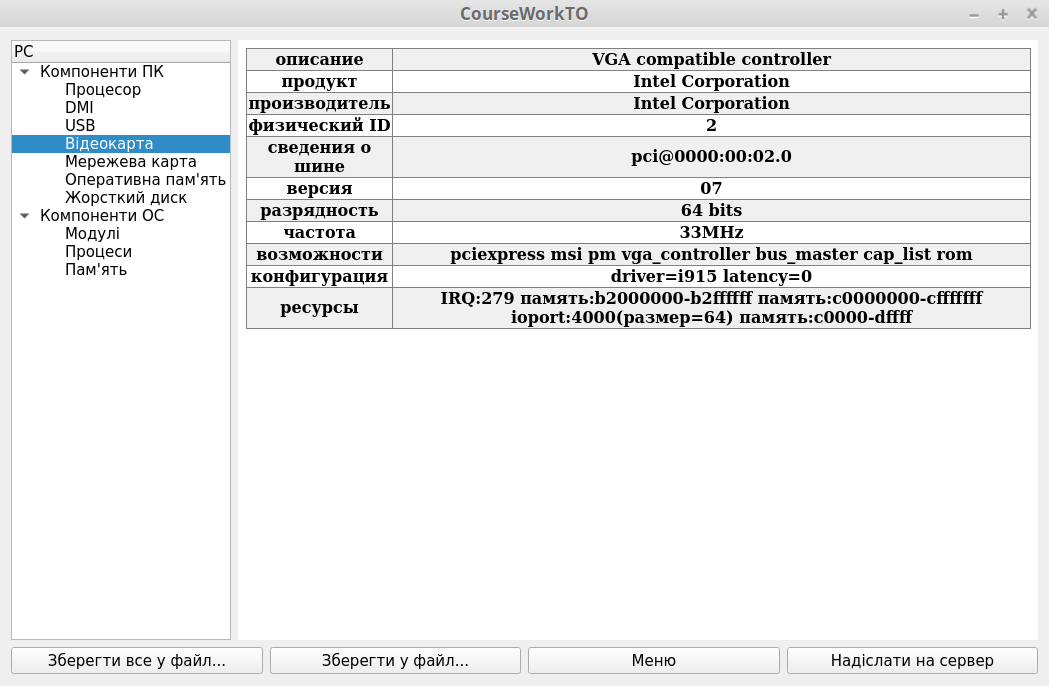
* Назва відеокарти;
* Постачальник;
* Фізичний ідентифікатор.
* Версію.
* Розрядність шини.
* Частоту системного таймера.
* Можливості та особливості, та можливості відеокарти;

Рисунок 3.5 – інформація про відеокарту

**Перегляд інформації про DMI**

Для перегляду базової інформації про материнську плату та BIOS потрібно вибрати пункт «DMI» (рис. 3.6).

DMI **–** інтерфейс програмування додатків (API), що дозволяє програмному забезпеченню збирати дані про характеристики комп'ютера.

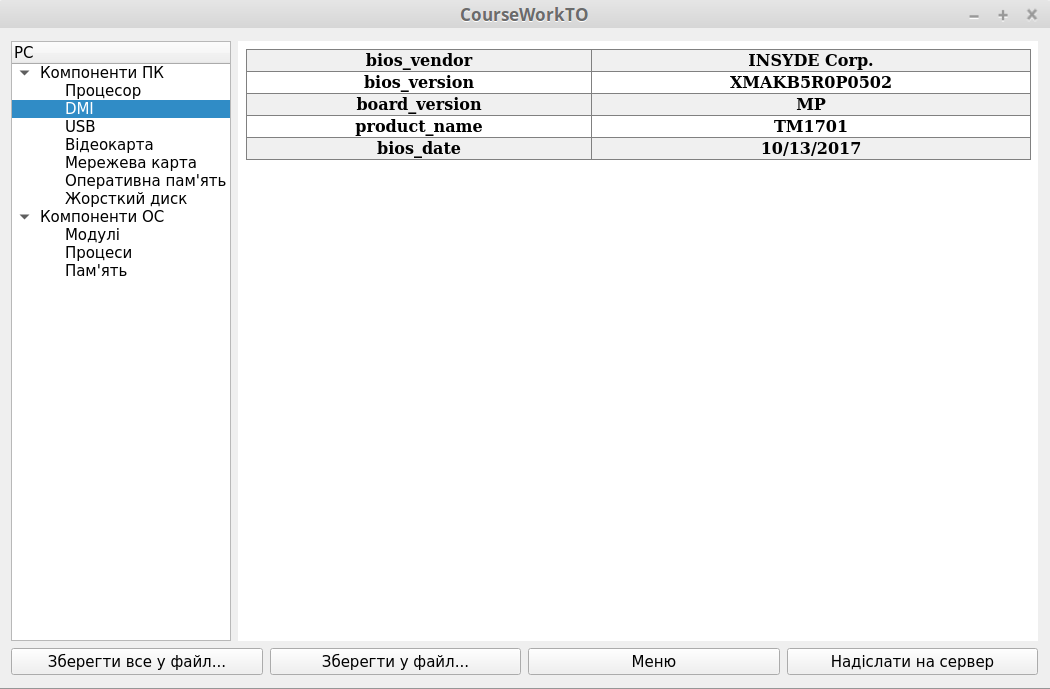
Даний пункт дає наступні відомості:

* Виробник ПК.
* Версія BIOS.
* Версія Motherboard.
* Модель ПК.
* Дата BIOS’а.

BIOS **–** є, насамперед, набором, спеціальних команд-інструкцій, підпрограм обчислювального пристрою для ініціалізації компонентів його персональної платформи, необхідних для первинного завантаження та подальшої роботи. Компонентами, які здатні працювати незалежно від операційної системи, як реагувати і, що робити за певних умов є процесор, системна логіка (чипсет), оперативна пам'ять, пристрої введення-виведення (відеокарта, клавіатура, тощо) та інші.

Фактично, BIOS **–** перше програмне забезпечення, що виконується процесором. Оскільки на початковому етапі завантаження комп'ютера зовнішні пристрої недоступні, BIOS, в загальному випадку, зберігається незалежним від живлення персональної платформи чином **–** в NVRAM-пам'яті (від англ. Non Volatile, **–** не тимчасова). Для цього, як правило, використовується одна або декілька мікросхем пам'яті **–** пристроїв постійного зберігання даних, які розташовані на системній платі.

Даний пункт не є дуже детальним, але основні відомості все ж дозволяє дізнатись.

Рисунок 3.6 – інформація про DMI

**Перегляд інформації про мережеву карту**

Для перегляду детальної інформації про мережеві інтерфейси потрібно вибрати підпункт «Мережева карта» (рис. 3.7).

**Мережева карта**, також відома як мережева плата, мережевий адаптер, Ethernet-адаптер, NIC — периферійний пристрій, що дозволяє [комп'ютеру](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%27%D1%8E%D1%82%D0%B5%D1%80) взаємодіяти з іншими пристроями у [мережі](https://uk.wikipedia.org/wiki/%D0%9B%D0%BE%D0%BA%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0_%D0%BE%D0%B1%D1%87%D0%B8%D1%81%D0%BB%D1%8E%D0%B2%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0_%D0%BC%D0%B5%D1%80%D0%B5%D0%B6%D0%B0). В даний час, особливо в персональних комп'ютерах, мережеві плати досить часто інтегровані в [материнські плати](https://uk.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D1%80%D0%B8%D0%BD%D1%81%D1%8C%D0%BA%D0%B0_%D0%BF%D0%BB%D0%B0%D1%82%D0%B0) для зручності і здешевлення всього комп'ютера в цілому.

Даний підпункт дає можливість дізнатися інформацію про:

* Дротові адаптери;
* Бездротові адаптери;
* Виробника;
* Ширину шини;
* Логічне ім’я;
* Пропускну здатність;
* Серійний номер;
* Ім’я інтерфейсу;
* Частоту системного таймеру;

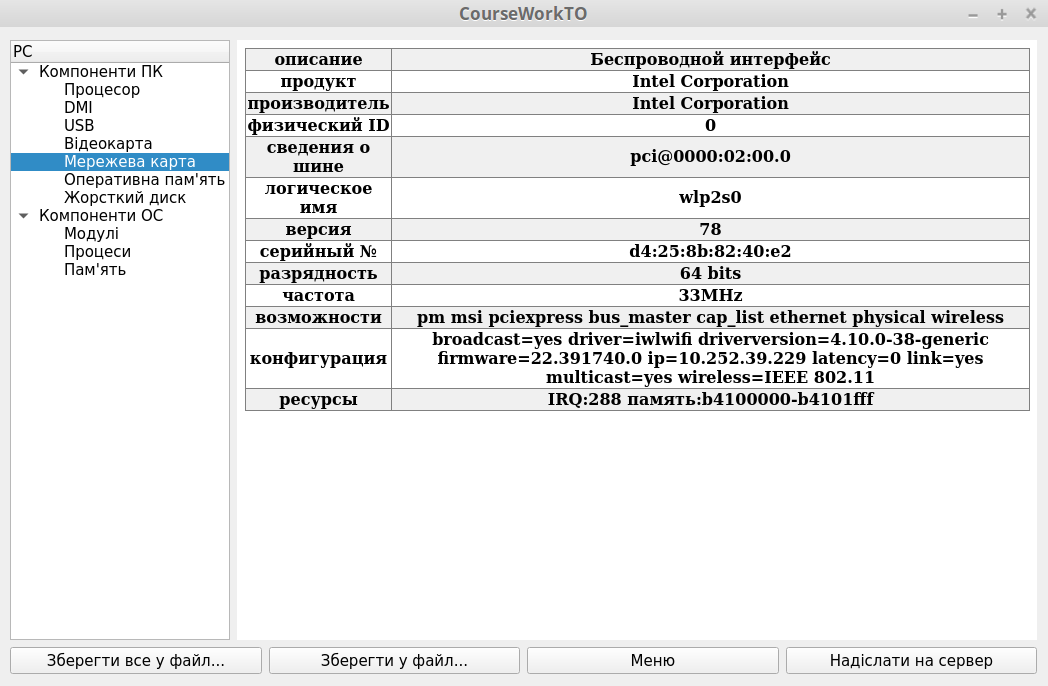
Даний підпункт дає розширену інформацію, яка може бути дуже цінною для спеціалістів, які розуміються що означає кожен із підпунктів.

Рисунок 3.7 – інформація про мережеві адаптери

**Перегляд інформації про оперативну пам’ять**

Для перегляду детальної інформації про оперативну пам’ять потрібно вибрати пункт «Оперативна пам’ять» (рис. 3.8).

Оперативна пам'ять — швидкодіюча пам'ять, призначена для запису, зберігання та читання інформації у процесі її обробки.

В обчисленні, пам'ять відноситься до комп'ютерних пристроїв, що використовуються для зберігання інформації для негайного використання в комп'ютері; вона є синонімом терміна «первинне зберігання». Комп'ютерна пам'ять працює на високій швидкості, наприклад, оперативна пам'ять (RAM), на відміну від зберігання, що забезпечує зберігання даних та повільно-доступних програм, пропонує більш високі можливості. При необхідності, вміст пам'яті комп'ютера може бути переданий у вторинне сховище, за допомогою технології управління пам'яттю під назвою «віртуальна пам'ять». Архаїчний синонім пам'яті — сховище.

Даний підпункт дає можливість дізнатися інформацію про:

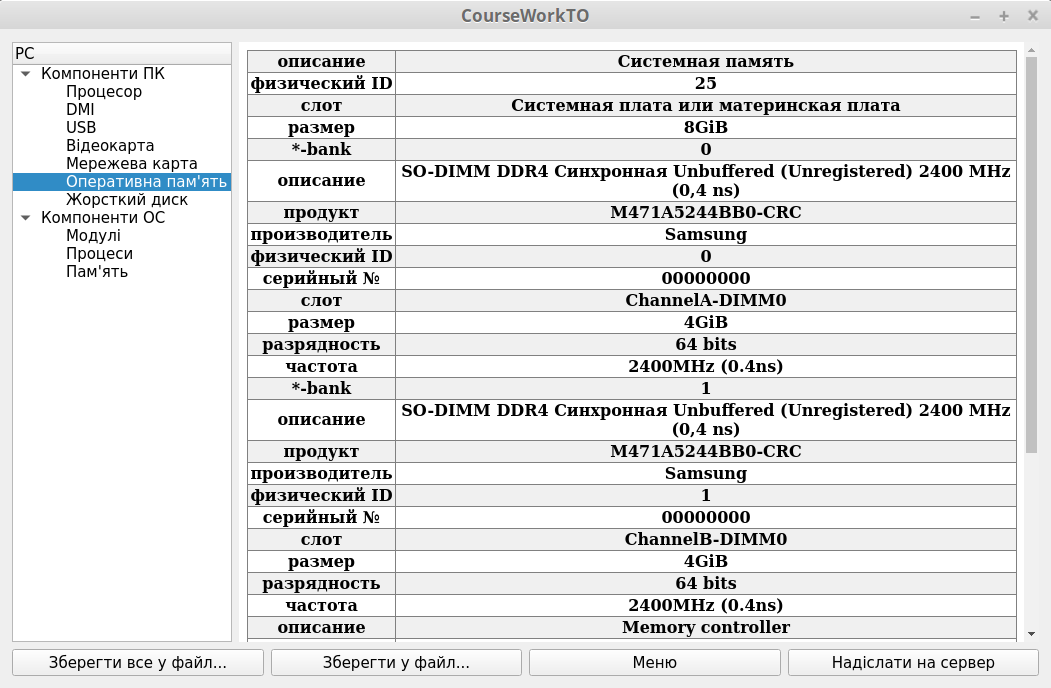
* Об’єм пам’яті;
* Кількість плат оперативної пам’яті;
* Виробника;
* Ширину шини;
* Тип;
* Частоту;
* Номер слоту;
* Частоту системного таймеру;

Рисунок 3.8 – інформація про оперативну пам’ять

**Перегляд інформації про жорсткий диск**

Для перегляду детальної інформації про жорсткий диск потрібно вибрати пункт «HARD\_DRIVE» (рис. 3.9).

Жорсткий диск — у комп'ютерному сленгу — «вінчестер» (від англ. winchester), — магнітний диск, основа якого виконана з твердого матеріалу. У більшості ЕОМ виконує функцію енергонезалежного носія інформації (комп'ютерної пам'яті чи нагромаджувача інформації) з довільним доступом (англ. random access).

Даний підпункт дає можливість дізнатися інформацію про:

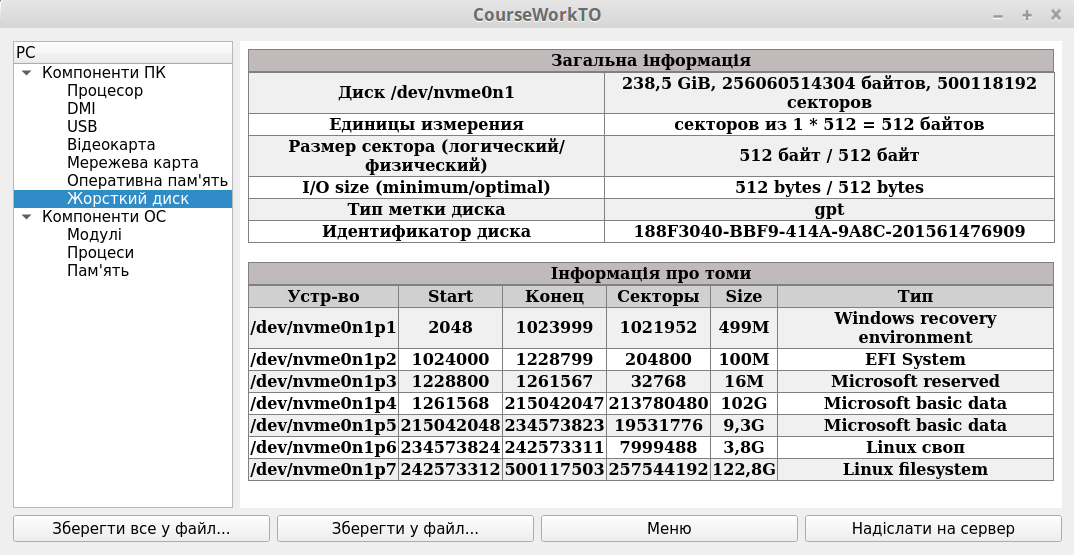
* Об’єм пам’яті;
* Переглянути розділи, та їх об’єм;
* Типи розділів;

Рисунок 3.9 – інформація про жорcткий диск

**Перегляд інформації про завантажені модулі**

Для перегляду детальної інформації про модулі потрібно вибрати пункт «Модулі» (рис. 3.10).

Модулі — (драйвери пристроїв) зберігаються на диску у вигляді об'єктних файлів (\* .o). Під час роботи системи ми самі можемо завантажити потрібний нам модуль. Для цього потрібно використовувати програму insmod, що входить до складу пакету modutils.

Даний підпункт дає можливість дізнатися інформацію про:

* Завантажені в даний момент модулі;
* Ресурси, які вони використовують;

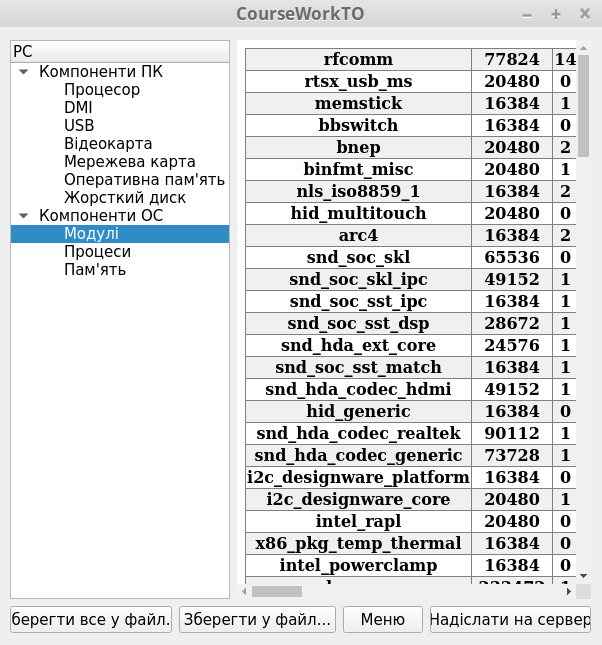


Рисунок 3.10 – інформація про модулі

**Перегляд інформації про завантажені процеси**

Для перегляду детальної інформації про процеси потрібно вибрати пункт «Процеси» (рис. 3.11).

Термін "процес" вперше з'явився при розробці операційної системи Multix і має кілька визначень, які використовуються в залежності від контексту:

* Програма на стадії виконання;
* Об'єкт, якому виділений процесорний час;
* Асинхронна робота;

Для опису станів процесів використовується кілька моделей. Найпростіша модель - це модель трьох станів:

* Стан виконання — це активний стан, під час якого процес володіє всіма необхідними йому ресурсами. У цьому стані процес безпосередньо виконується процесором.
* Стан очікування — це пасивний стан, під час якого процес заблокований, він не може бути виконаний, бо чекає якусь подію, наприклад, введення даних або звільнення потрібного йому пристрою.
* Стан готовності — це теж пасивний стан, процес теж заблокований, але на відміну від стану очікування, він заблокований не через внутрішні причини (адже очікування введення даних - це внутрішня, "особиста" проблема процесу - він може і не очікувати введення даних і вільно виконуватися - ніхто йому не заважає), а за зовнішніми, що не залежать від процесу, причин.

Даний підпункт дає можливість дізнатися інформацію про:

* Завантажені в даний момент процеси;
* Ресурси, які вони використовують;
* PID процеса;

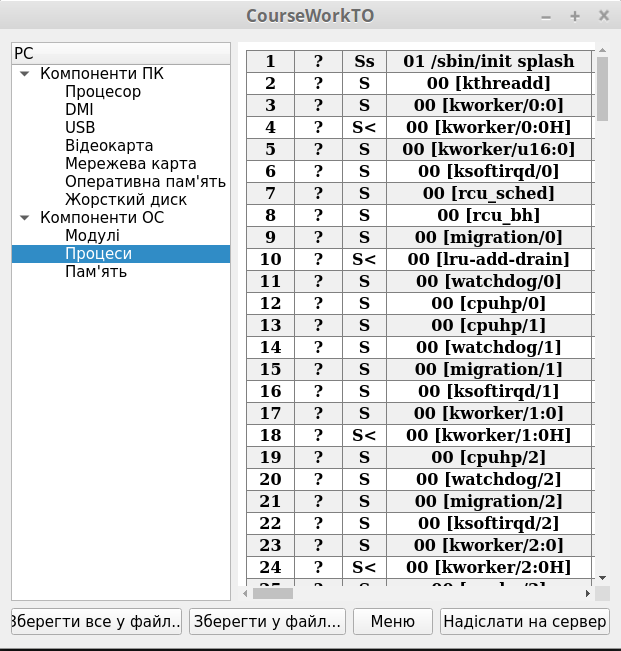


Рисунок 3.11 – інформація про процеси

**Перегляд інформації про пам’ять**

Для перегляду детальної інформації про пам’ять потрібно вибрати пункт «Пам’ять» (рис. 3.12).

Даний підпункт довзоляє переглянути інформацію про:

* Вільну пам’ять;
* Доступну пам’ять;
* Кешовану пам’ять;
* Розмір SWAP розділу;
* Розмір доступної пам’яті на SWAP розділі;
* Тощо;

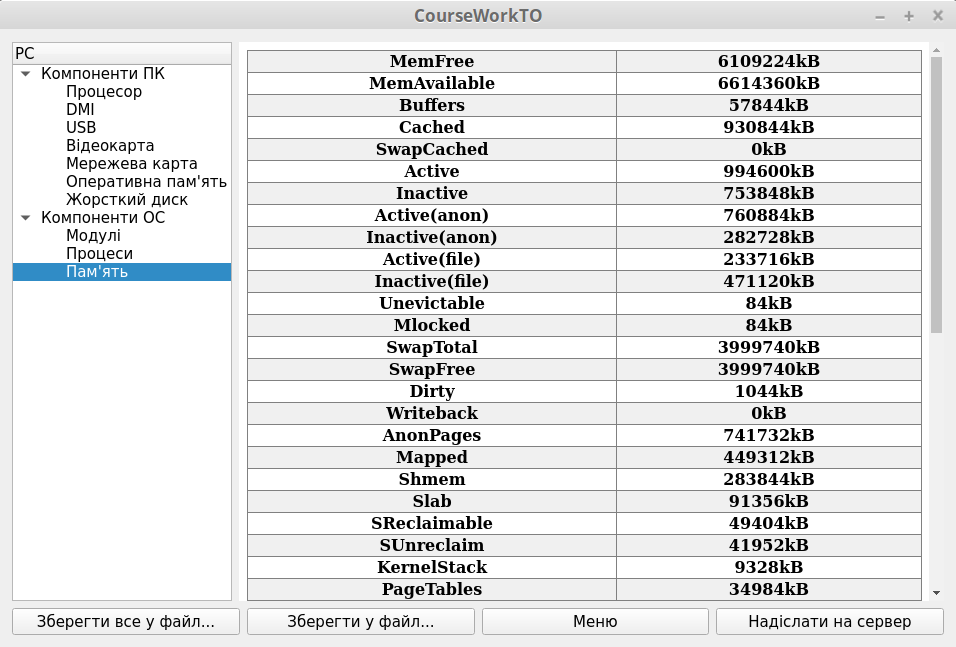


Рисунок 3.11 – інформація про пам’ять

**3.3 Визначення продуктивності компонентів ПК**

Основною задачею програми є реалізація визначення компонентів з яких складається ПК, та визначення їх характеристик. У даному пункті розглянуто, як саме виконується визначення продуктивності.

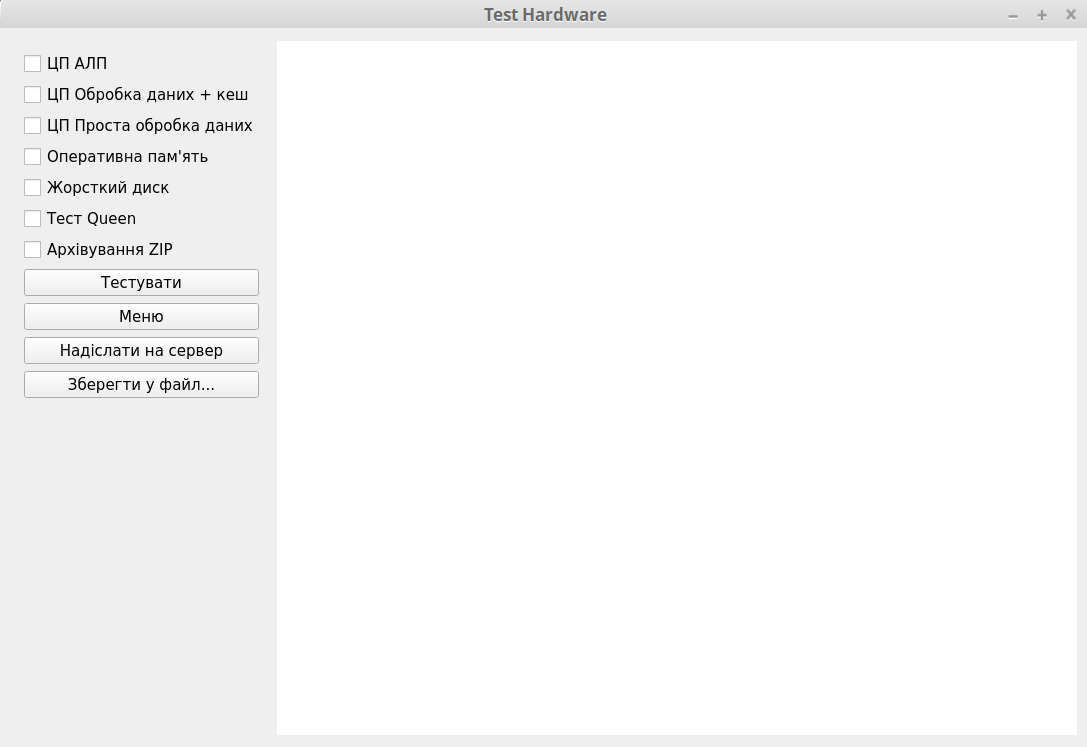
Для визначення продуктивності, потрібно у пункті меню натиснути на «Тестування». Далі потрібно вибрати компоненти, які потрібно протестувати та натиснути на кнопку «тестувати», після чого появиться результат тестування(рис. 3.10).

Рисунок 3.10 – загальний вигляд

**Визначення продуктивності процесора**

Для визначення продуктивності процесора компонентів було створено наступні алгоритми:

* Для визначення продуктивності арифметичних операцій було здійснено наступні дії: виконати 1024\*1024\*32 ітерацій, та у кожній ітерації циклу виконувались по 17 операції додавання, відніманні, ділення, множення, інкременту, та декременту для різноманітних типів даних; та вираховувався час, за який виконувались ці операції;
* для визначення продуктивності кешу процесора було створено два циклу з різними стратегіями проходу по оперативній пам’яті:

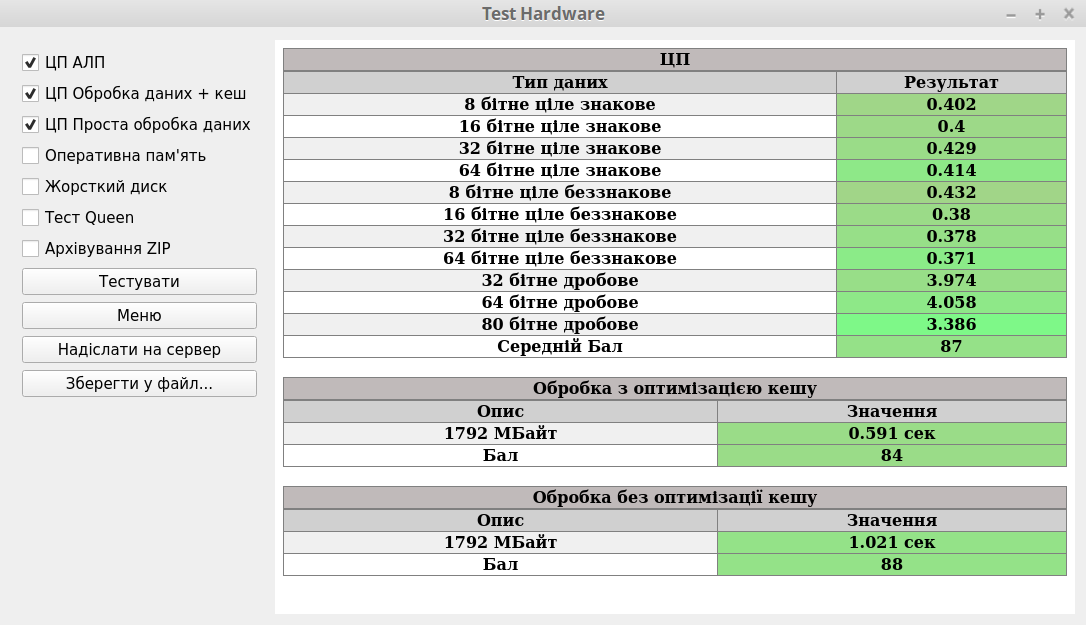
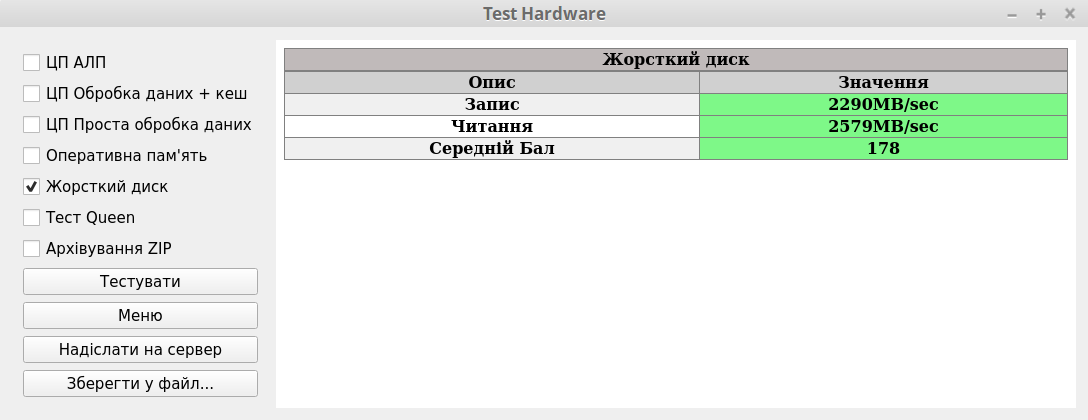
1. у першому випадку по-пам’яті прохід був послідовним — байт за байтом, у цьому випадку промахів кешу процесора було мало, так як процесор кешує пам’ять розташовану поблизу пам’яті, яку в даних момент читають;
2. у другому випадку прохід по пам’яті був більш хаотичний, що у свою чергу зменшило швидкість звернення до оперативної пам’яті;

Рисунок 3.11 – результат тестування ЦП

**Визначення продуктивності жорсткого диску**

Для тестування швидкості читання/запису на жорсткий диск було створено наступний алгоритм: було виділено певний об’єм пам’яті, та записано туди файл розміром 512 мегабайт, і так само із зчитуванням. Після чого було отримано час з яким жорсткий диск записує та читає інформацію.

 Рисунок 3.12 – результат тестування жорсткого диску

**Визначення продуктивності оперативної пам’яті**

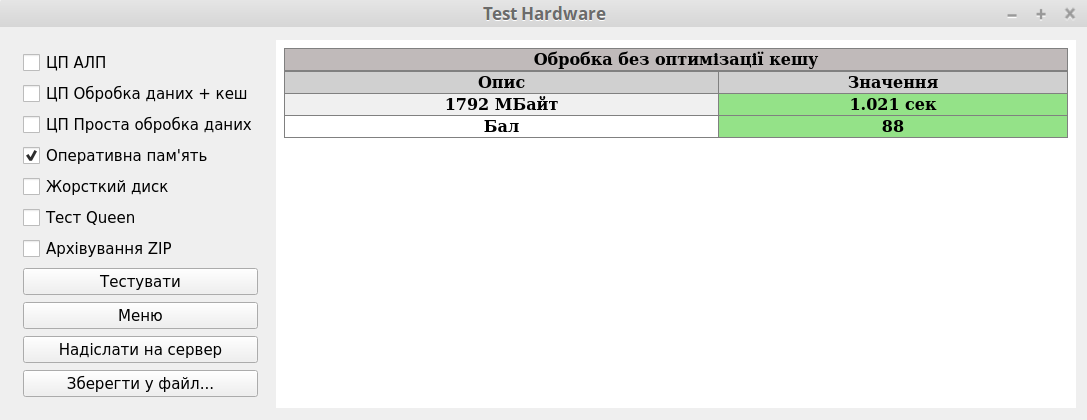
Для тестування швидкості оперативної пам’яті був створений простий алгоритм, який просто записує в оперативну пам’ять, та читає з неї, та вираховує час, який був для цього затрачений.

Рисунок 3.13 – результат тестування оперативної пам’яті

**Визначення продуктивності тестом “Queen”**

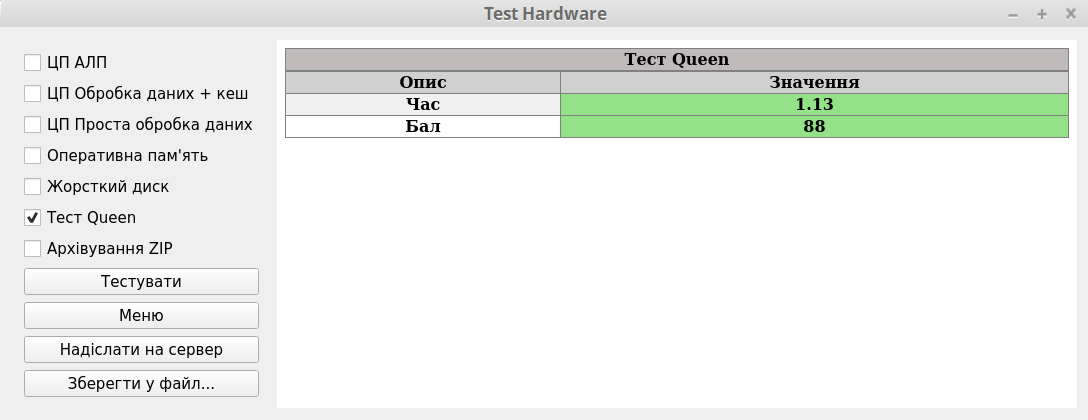
Цей простий тест оцінює, наскільки добре прогнозує розгалуженя процесор і, здійснюється помилкові прогнози віток. Виконується видача рішень для головоломки з N ферзями, розташованими на шахівниці розміром NxN.

Рисунок 3.14 – результат тесту “Queen”

**Визначення продуктивності за допомогою архіватора “Zip”**

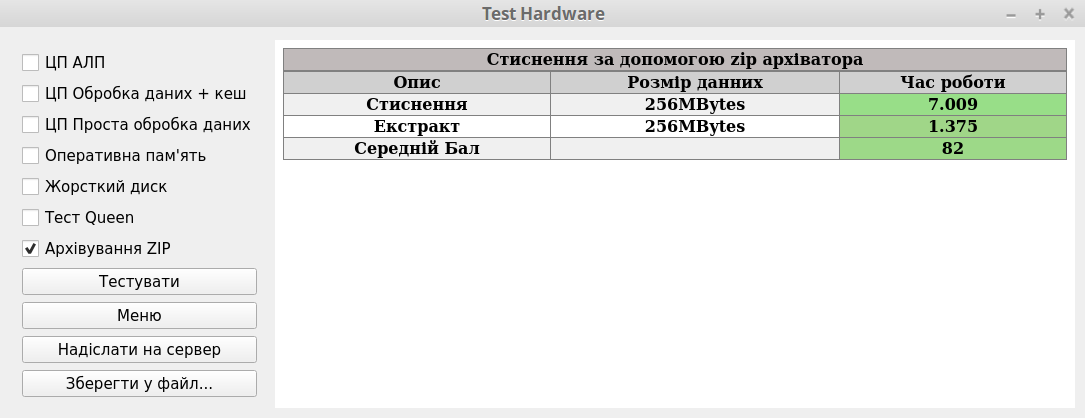
ZIP - популярний формат архівації файлів і стиснення даних без втрат. Архів ZIP може містити один або кілька файлів і каталогів, які можуть бути стиснуті різними алгоритмами. Найбільш часто в ZIP використовується алгоритм стиснення Deflate. Формат був створений в 1989 році Філом Кацем і реалізований в програмі PKZIP компанії PKWARE

Рисунок 3.11 – результат тесту архівування “Zip”

**3.4 Генерація та відправка звіту роботи програми**

Звіт створюється у вигляді .HTML файлу, я якому може знаходитися як інформація про компонентну будову ПК, так і інформацію про результам тестування компонентів, після чого звіт може бути надісланий на сервер, який збирає інформацію про ПК. Користувач має сам ввести IP адресу, та порт, на який надсилати звіт.

Для передачі звіту по мережі використовується протокол TCP/IP, щоб забезпечити цілісність отримуваного файлу.

Звіт є досить детальним, для того щоб точно оцінити на що здатний даний комп’ютер.

Для того, щоб відправити звіт про інформацію компонентної будови потрібно:

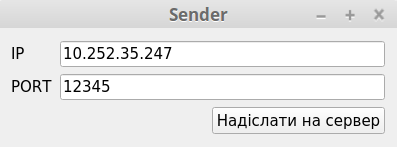
1. У меню натиснути на пункт «Інформація про систему».
2. У вікні яке з’явилося (рис. 3.2) натиснути кнопку «Надіслати на сервер».

Також потрібно, щоб на машині, на яку відправляється звіт був запущений сервер, у якого ip-адреса, та порт вказані такі ж самі, як і у клієнта, який надсилає звіт. Звіти будуть зберігатися у папці, з якої була запущена програма-сервер.

Для того, щоб запустити сервер потрібно:

1. У меню натиснути на пункт «Сервер».
2. У вікні яке з’явилося (рис. 3.12) ввести ip-адресу та порт, та натиснути кнопку «Запустити сервер».

Для зупинки сервера, потрібно натиснути на кнопку «зупинити сервер».

Рисунок 3.11 – Запуск клієнта для надсилання звіту

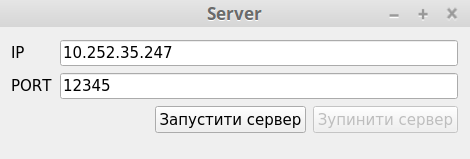


Рисунок 3.12 – запуск сервера для приймання звітів

# 5. ЕКОНОМІЧНА ЧАСТИНА

Метою даного розділу є обґрунтування доцільності надання послуги створення програми комплексного тестування апаратних складових обчислювальної системи.

*1. Обчислюємо баланс робочого часу працівника*

Ефективний фонд часу розраховується за формулою:

(5.1)

де Фном – номінальний фонд робочого часу працівників

h – плановий коефіцієнт витрат часу, приймається в межах 2 – 5%.

Припускаємо витрати часу 3%.

Номінальний фонд часу визначається за формулою:

(5.2)

де Дк – кількість календарних днів в році, дн.;

Дсв – кількість святкових днів в році, дн.;

Дв – кількість вихідних днів в році, дн.;

Двід – середня кількість днів у році наданих робітнику під відпустку, дн.;

Дп.св – передсвяткові, дн.;

Tзм1, tзм2 – тривалість зміни у звичайні та передсвяткові дні, год.;

Для розрахунку наведеної формули її дані визначаємо за 2018 р. і вони становлять:

– дні календарні – 365;

– вихідні та святкові дні – 116 ;

– дні відпустки – 24;

– тривалість зміни – 8 годин, в передсвятковий день - 7 годин.

Звідси фонд номінальний дорівнює:

Фонд ефективний становить:

*2. Розраховуємо трудомісткість робіт з надання послуги створення програми*

Спочатку визначають трудомісткість робіт щодо 1 послуги, тобто час за формулою:

(5.3)

де tн/год – загальна сума норми часу згідно технологічного процесу, н/год.

Трудомісткість обчислюємо у таблиці 5.1

Таблиця 5.1 – Трудомісткість робіт

|  |  |
| --- | --- |
| Найменування операції | Час на 1 операцію, год. |
| Пошук середовища розробки | 2 |
| Встановлення та налаштування середовища | 8 |
| Розробка архітектури програми | 24 |
| Написання коду | 120 |
| Тестування | 10 |
| Виправлення помилок | 16 |
| **РАЗОМ** | 180 |

Далі визначаємо загальну трудомісткість робіт, тобто всю кількість робочого часу, яку необхідно витратити на планову річну кількість послуг за формулою:

(5.4)

де Тод – трудомісткість 1 послуги створення програми;

N – річна кількість планових послуг. Приймаємо кількість послуг створення програми – 4 в рік.

Звідси:

Припускаємо, що даний вид роботи не є єдиним на підприємстві і становить 20 % від решти наданих послуг.

Отже на весь обсяг:

*3. Розрахунок необхідної кількості працюючих.*

Основних працівників підприємства визначаємо з розрахунку планової трудомісткості всіх наданих послуг фірми протягом планового року за формулою:

(5.5)

де Тзаг – загальна трудомісткість робіт на підприємстві;

Феф – ефективний фонд, год.;

kвн – коефіцієнт виконання норм, що приймаються рівним від 1 до 1,2. В нашому випадку припускаємо перевиконання на рівні 5 %, звідси kвн = 1.1

Обчислюємо чисельність основних робітників для надання даної послуги:

Крім того, приймаємо інші категорії персоналу згідно штатного розкладу у розмірі:

1. Директор – 1 чол.

2. Бухгалтер – 1 чол.

3. Допоміжний робітник – 1 чол.

4. Прибиральниця – 1 чол.

*4. Обчислення витрат на оплату праці*

Заробітна плата основних робітників визначається за відрядною заробітною платою за формулою:

(5.6)

де N – річна кількість послуг;

Рзв – це звичайний розцінок за 1 послугу створення програми, що визначається за формулою:

(5.7)

де Тод – трудомісткість робіт щодо створення програми;

Тстс – тарифна ставка робітників, що зайняті у наданні цієї послуги, грн.

Згідно законодавством на 1.01.18 мінімальна заробітна плата встановлена на рівні 3723 грн. Отже тарифну ставку 1 розряду приймаємо у сумі 22,41 грн. за 1 годину.

Середній коефіцієнт складності робіт з надання послуги приймаємо 1.70

Визначаємо середню тарифну ставку працівника за формулою:

(5.8)

де Кс – середній коефіцієнт складності робіт з надання послуги;

Тст1 - тарифна ставка 1 розряду.

Обчислюємо розцінок звичайний:

Обчислюємо зарплату основних робітників:

Отже прямий фонд оплати праці основних робітників, що надаватимуть послугу створення програми за рік становить 27429,84 грн. А на весь обсяг робіт фірми:

Заробітну плату іншим категоріям персоналу розраховуємо за формулою:

(5.9)

де Оі – оклад одного працівника певної категорії, грн.;

Чі – чисельність працівників;

М – кількість місяців у році.

Премії визначаю за формулою:

(5.10)

де ЗП – пряма заробітна плата працівників, грн;

Кд– коефіцієнт премії.

Річний фонд оплати праці за кожною категорією визначаємо за формулою:

(5.11)

де ЗП – пряма заробітна плата, грн;

П – премія, грн.

Нарахування на соціальні потреби містять: відрахування на соціальне страхування, у пенсійний фонд, фонд зайнятості, та інші фонди. На сьогодні єдиний соціальний внесок становить 22%

Розрахунки здійснюємо у зведеній відомості фонду оплати праці підприємства.

Таблиця 5.2 – Зведена відомість фонду оплати праці

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Категорія персоналу** | **Оклад, грн.** | **Прямий ФЗП, грн.** | **Премії.** | | **ФЗП, грн.** | **Сума нарахувань, грн** |
| **%** | **грн.** |
| Основні робітники | - | 137149,2 | 15 | 20572,38 | 157721,58 | 34698,75 |
| Допоміжний працівник | 3950 | 47400 | 5 | 3318 | 50718 | 11157,96 |
| Директор | 6500 | 78000 | 15 | 11700 | 89700 | 19734 |
| Бухгалтер\* | 5150 | 30900 | 10 | 3090 | 33990 | 7477,8 |
| Прибиральниця\* | 4000 | 24000 | 6 | 1440 | 25440 | 5596,8 |
| **РАЗОМ** | - | 317449,2 | - | 40120,38 | 357569,58 | 78665,31 |

\* бухгалтера та прибиральницю приймаємо на 0,5 ставки

*5. Розрахунок матеріальних витрат на послугу.*

Розрахунок по елементам кошторису матеріальних витрат здійснюємо за формулами:

(5.12)

де Ці – ціна купованого виробу або матеріалу, грн.,

nі – кількість купованих виробів та матеріалів, одиниць.

В нашому випадку до матеріальних витрат відносимо:

Таблиця 5.3 – Розрахунок матеріальних витрат на 1 послугу

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Найменування  елементів | Кількість  штук | Ціна за одиницю, грн. | Сума, грн. | Джерело доступу |
| Qt Creator\* | 1 | 11934 | 2983,5 | https://www.qt.io |

\*В розрахунку на 1 послугу

*6. Розрахунок загальновиробничих витрат.*

Підприємство планує орендувати приміщення з виробничою площею 54 м2. Вартість оренди в місяць приймаємо: 37 грн/м2. Загальну вартість оренди в рік визначаємо за формулою:

(5.13)

де Sзаг – загальна площа орендованого приміщення підприємства, м2;

Вор.1м2 – вартість оренди 1 м2 в місяць, грн.;

М - кількість місяців.

грн.

Визначаємо суму комунальних платежів. Щоб визначити кількість енергії та освітлення використовуємо формулу:

(5.14)

де Sзаг – загальна площа орендованого приміщення підприємства, м2;

Феф – ефективний фонд, год.;

Nосв – це норма інтенсивності освітлення одного м? площі. Приймаємо – 40 Вт/ м2

кВт – коефіцієнт втрат електроенергії в мережі 13%.

Вартість освітлення визначаємо за формулою:

Ц1 – ціна однієї кВт години, яка з 1.03.2018 р. встановлена 2,20грн. за кВтгод.;

грн.

Витрати на опалення визначаються за формулою:

(5.15)

де Sзаг – загальна площа, м2;

Ц2 – вартість опалення одного м2 площі, грн..

Повний сезонний тариф 35,2 грн.(з ПДВ) з 01.10.2017

М – кількість опалюваних місяців у році.

грн.

Витрати на воду визначаються за формулою:

(5.16)

де Ч – чисельність персоналу;

Ц3 – вартість за витрачання одного м3 води, грн.;

Централізоване водопостачання та водовідведення - 13.33 грн. (з ПДВ)

0,5 м3 – орієнтовна норма витрачання води на 1 працівника в місяць.

М – кількість місяців у році.

грн.

Знаходимо амортизацію на кожен елемент основних фондів за формулою:

(5.17)

де На – норма амортизації;

Воф – вартість основних фондів, грн.

Визначаємо амортизаційні відрахування за податковим методом у таблиці 5.4

Таблиця 5.4 – Розрахунок амортизаційних відрахувань

|  |  |  |  |
| --- | --- | --- | --- |
| Категорія основних фондів | Вартість ОФ, грн. | %  відрахувань | Амортизація, грн. |
| Обладнання | 82000 | 10 | 8200 |
| Інструмент | 750 | 25 | 187,5 |
| Інвентар | 970 | 27 | 261,9 |
| **РАЗОМ** | **83720** | **-** | **8649,4** |

Витрати на ремонт основних фондів приймаємо в розмірі 13% від вартості основних фондів. Розраховуємо витрати на ремонт

Обчислюємо витрати на охорону праці, які приймаємо у розмірі 170 грн.:

Інші витрати приймаємо в сумі 3 % від всіх загальновиробничих витрат. Всі елементи витрат зводимо у таблицю 5.5

Таблиця 5.5 – Загальновиробничі витрати

|  |  |
| --- | --- |
| Елементи витрат | Сума, грн. |
| 1. Оренда | 23976 |
| 2. Електроенергія | 9477,6 |
| 3. Опалення | 11404,8 |
| 4. Вода | 479,88 |
| 5. Амортизація | 8649,4 |
| 6. Ремонт основних фондів | 10883,6 |
| 7. Охорона праці | 1020 |
| Всього | 65891,28 |
| 8. Інші витрати | 1984,24 |
| **РАЗОМ** | **67875,52** |

*7. Визначення собівартості та ціни послуги створення програми*

До статей калькуляції відносяться: матеріальні витрати, витрати на транспортування, витрати на оплату праці, загальновиробничі витрати.

Калькуляцію собівартості здійснюємо у таблиці 5.6.

Таблиця 5.6 – Розрахунок собівартості 1 послуги

|  |  |  |
| --- | --- | --- |
| Статті калькуляції собівартості | Сума | Примітки |
| 1.Матеріальні витрати | 2983,5 | З таб. 5.3 з розрахунку на 1 |
| 2.Транспортні витрати |  | Відсутні |
| 3.Пряма ЗП | 15872,46 | З таб. 5.2 з розрахунку на 1 |
| 4. Премії | 2006,02 | З таб. 5.2 з розрахунку на 1 |
| 5.Нарахування | 3933,27 | З таб. 5.2 з розрахунку на 1 |
| 6.Загальновиробничі витрати | 3393,78 | З таб. 5.2 з розрахунку на 1 |
| 7.Позавиробничі витрати | 221,41 | 10% від пункту 6 |
| Повна собівартість | 28410,43 | Сума пунктів 1-7 |

Обчислюємо прибуток згідно планової норми рентабельності за формулою:

(5.18)

де С – собівартість 1 послуги, грн.;

kр − плановий коефіцієнт рентабельності.

Приймаємо плановий коефіцієнт рентабельності в розмірі 10%.

Здійснюємо розрахунки:

Обчислюємо ціну послуги (без ПДВ) за формулою:

(5.19)

Здійснюємо розрахунки:

Виходячи з розрахунків, проведених в даному розділі, можна зробити висновок, що вартість послуги створення програми в середньому буде 30151,47 грн.

**ВИСНОВОК**

В результаті, програма виконує задачі, які були поставлені. Були розроблені алгоритми для тестування продуктивності різноманітних компонентів персонального комп’ютера. Впродовж виконання завдання було також досліджено, які саме є оптимізації у роботі процесора, як працює кеш процесора, та наскільки він важливий, у швидкодії обробки інформації, та особливості файлової систем ext у GNU/Linux.

Було виявлено, які фактори впливають на результати тестів, до них належать:

1. Температура компонентів під час тестування.
2. Навантаженість компонентів під час тестування.
3. Розрядність процесора.
4. Швидкість обміну інформацією між процесором, та оперативною пам’яттю.
5. Кешування файлів у оперативну пам’ять до яких звертались останім часом - що дуже впливає на тест читання із жорсткого диску.

Також на тести впливали оптимізації компілятора, який не виконував пусті цикли, у тесті швидкості ітерацій циклу. Тому потрібно було виконувати компіляцію у режимі “Debug”.

Також слід відмітити, що усі тести проводилися у одно-поточному режимі, для уникання надмірного перегріву комп’ютера, це потрібно враховувати у оцінювані ПК.

Також було замічено, що отримання більш детальної та повної інформації, програму потрібно запускати з правами супер-користувача(root). Наприклад – для звичайного користувача неможливо отримати інформація про кеш процесора, та кількість слотів оперативної пам’яті.

**ПЕРЕЛІК ПОСИЛАНЬ**

1. Линус Торвальдс, Дэвид Даймонд — Just for fun.
2. Д. Н. Колиснеченко, В. Аллен — LINUX Полное Руководство.
3. Джеймс Армстронг — Секреты Unix.
4. Дэниел Барретт «Карманный путеводитель по Linux» 3-е изд. (2016).
5. Сэм Алапати «Современное администрирование Linux» (2016).
6. Брайан Уорд «Внутреннее устройство Linux» (2016).
7. http://www.linux.org.ru/
8. http://www.opennet.ru/

**Додаток А**

#include "SystemOverview.h"

#include "SysInfo/All.h"

#include "Utils/HtmlUtils.h"

#include "senderfile.h"

#include "SaverFile.h"

#include "forms.h"

#include "utils.h"

#include <fstream>

#include <list>

#include <map>

MyTreeView::MyTreeView(QWidget \*parent) : QTreeWidget(parent)

{

setHeaderLabel("PC");

init();

}

void MyTreeView::init()

{

setFixedWidth(200);

#define create\_node(VarName, Device) \

auto VarName = new QTreeWidgetItem(this, QStringList(AllModulesMap[DEVICE:: Device]))

#define add\_child(Parent, ChildDevice) \

Parent->addChild(new QTreeWidgetItem(QStringList(AllModulesMap[DEVICE:: ChildDevice])));

#define add\_exist\_child(VarName, NewVar) \

(VarName) ->addChild( NewVar );

create\_node(ROOT\_HW, HARDWARE);

add\_child(ROOT\_HW, CPU);

add\_child(ROOT\_HW, DMI);

add\_child(ROOT\_HW, USB);

add\_child(ROOT\_HW, GPU);

add\_child(ROOT\_HW, NETWORK);

add\_child(ROOT\_HW, RAM);

add\_child(ROOT\_HW, HARD\_DRIVE);

create\_node(OS, OS);

add\_child(OS, MODULES);

add\_child(OS, PROCESSES);

add\_child(OS, RAM\_MEMORY);

expandItem(ROOT\_HW);

expandItem(OS);

}

SystemOverview::SystemOverview(QWidget \*parent) :

QWidget(parent),

htmlPage(HtmlUtils::getEmptyHtmlPage()),

treeView(new MyTreeView(parent)),

lockedUpdateInfo(false),

wasSentFile(false)

{

init();

}

void SystemOverview::init()

{

setMinimumSize(QSize(600, 400));

auto mainLayout = new QVBoxLayout;

auto horisontal = new QHBoxLayout;

horisontal->addLayout(initTreeWidget());

horisontal->addLayout(initWebView());

mainLayout->addLayout(horisontal);

mainLayout->addLayout(initButtons());

setLayout(mainLayout);

initTimerUpdater();

initConnections();

}

QLayout \*SystemOverview::initButtons()

{

auto l = new QHBoxLayout;

gotoMenu = new QPushButton("Меню", this);

sendToServer = new QPushButton("Надiслати на сервер", this);

buttonSaveAsFile = new QPushButton("Зберегти у файл...", this);

buttonSaveAllAsFile = new QPushButton("Зберегти все у файл...", this);

l->addWidget(buttonSaveAllAsFile);

l->addWidget(buttonSaveAsFile);

l->addWidget(gotoMenu);

l->addWidget(sendToServer);

return l;

}

QLayout \*SystemOverview::initTreeWidget()

{

treeView->setMinimumWidth(220);

auto layout = new QHBoxLayout;

layout->addWidget(treeView);

return layout;

}

QLayout \*SystemOverview::initWebView()

{

auto l = new QHBoxLayout;

webView = new QWebEngineView;

l->addWidget(webView);

return l;

}

void SystemOverview::initTimerUpdater()

{

timerForUpdateInfo = new QTimer(this);

timerForUpdateInfo->start(1000);

timerWatchdogCompleteSend = new QTimer(this);

timerWatchdogCompleteSend->start(100);

}

void SystemOverview::mousePressEvent(QMouseEvent \*event)

{

lockedUpdateInfo = true;

}

void SystemOverview::mouseReleaseEvent(QMouseEvent \*event)

{

lockedUpdateInfo = false;

}

void SystemOverview::initConnections()

{

connect(this->treeView, SIGNAL(itemClicked(QTreeWidgetItem\*,int)), this, SLOT(slotClickTreeView(QTreeWidgetItem\*,int)));

connect(this->gotoMenu, SIGNAL(clicked(bool)), this, SLOT(on\_pushButton\_clicked()));

connect(this->sendToServer, SIGNAL(clicked(bool)), this, SLOT(slotSendCurrentDataToServer()));

connect(this->buttonSaveAsFile, SIGNAL(clicked(bool)), this, SLOT(slotSaveAsFile()));

connect(this->buttonSaveAllAsFile, SIGNAL(clicked(bool)), this, SLOT(slotSaveAllAsFile()));

connect(this->webView->page(), SIGNAL(scrollPositionChanged(QPointF)), this, SLOT(slotScrollChangePosition(QPointF)));

connect(this->timerWatchdogCompleteSend, SIGNAL(timeout()), this, SLOT(slotCheckIsSend()));

}

DEVICE SystemOverview::decodeTextNameItem(const QString &name) const

{

if (name == "Процесор") return DEVICE::CPU;

if (name == "DMI") return DEVICE::DMI;

if (name == "Оперативна пам'ять") return DEVICE::RAM;

if (name == "Відеокарта") return DEVICE::GPU;

if (name == "USB") return DEVICE::USB;

if (name == "Мережева карта") return DEVICE::NETWORK;

if (name == "Жорсткий диск") return DEVICE::HARD\_DRIVE;

if (name == "Процеси") return DEVICE::PROCESSES;

if (name == "Модулі") return DEVICE::MODULES;

if (name == "Пам'ять") return DEVICE::RAM\_MEMORY;

return DEVICE::UNDEFINED;

}

Subsystem \*SystemOverview::getDevice(DEVICE dev) const

{

switch (dev) {

case DEVICE::CPU: return SubsystemCPU::inst();

case DEVICE::DMI: return SubsystemDMI::inst();

case DEVICE::RAM: return SubsystemRAM::inst();

case DEVICE::GPU: return SubsystemGPU::inst();

case DEVICE::USB: return SubsystemUSB::inst();

case DEVICE::NETWORK: return SubsystemNet::inst();

case DEVICE::HARD\_DRIVE: return SubsystemFilesystem::inst();

case DEVICE::PROCESSES : return OS::Processes::inst();

case DEVICE::MODULES : return OS::Modules::inst();

case DEVICE::RAM\_MEMORY: return OS::RAMMemory::inst();

default : return nullptr;

};

}

void SystemOverview::applySettings()

{

currentView = getDevice(DEVICE::CPU);

}

void SystemOverview::printResultTest()

{

}

void SystemOverview::slotClickTreeView(QTreeWidgetItem \*item, int)

{

qDebug() << "click tree view..." << ": " << item->text(0);

currentView = getDevice(decodeTextNameItem(item->text(0)));

slotUpdateInfoFromDevice();

}

void SystemOverview::slotUpdateInfoFromDevice()

{

if (currentView) {

qDebug() << currentView->getPrintableInfo().c\_str();

htmlPage = currentView->getHTMLCode();

webView->setHtml(QString(htmlPage.c\_str()));

QScroller::scroller(webView->page())->scrollTo(QPointF(-100, -100));

}

}

void SystemOverview::slotScrollChangePosition(QPointF newPosition)

{

qDebug() << "scroll";

currentPosition = newPosition;

}

void SystemOverview::slotSaveAllAsFile()

{

string allHtmlPage;

for (Subsystem \*subsystem : std::initializer\_list<Subsystem\*>{

SubsystemCPU::inst(),

SubsystemDMI::inst(),

SubsystemRAM::inst(),

SubsystemGPU::inst(),

SubsystemUSB::inst(),

SubsystemNet::inst(),

SubsystemFilesystem::inst(),

OS::Processes::inst(),

OS::Modules::inst(),

OS::RAMMemory::inst()

})

{

allHtmlPage += subsystem->getHTMLCode();

allHtmlPage += HtmlUtils::getBRTag();

}

SaverFile::saveLocalFile((const char \*)allHtmlPage.c\_str(), allHtmlPage.length());

}

void SystemOverview::slotSaveAsFile()

{

SaverFile::saveLocalFile((const char \*)htmlPage.c\_str(), htmlPage.length());

}

void SystemOverview::slotCheckIsSend()

{

if (wasSentFile) {

wasSentFile.store(false);

QMessageBox::information(nullptr, "", "Звiт надіслано");

}

}

void SystemOverview::slotSendCurrentDataToServer()

{

cout << "slotSendToServer" << std::endl;

QString namefile = Utils::generateFilename().c\_str();

SaverFile::saveLocalFile(namefile, htmlPage.c\_str(), htmlPage.length());

(new SenderFile(namefile.toStdString(), wasSentFile))->show();

}

void SystemOverview::slotSendAllDataToServer()

{

}

void SystemOverview::on\_pushButton\_clicked()

{

Forms::inst()->showMenu();

}

void SystemOverview::on\_buttonSend\_clicked()

{

string result;

for (DEVICE dev : {

DEVICE::CPU,

DEVICE::DMI,

DEVICE::RAM,

DEVICE::GPU,

DEVICE::USB,

DEVICE::NETWORK,

DEVICE::HARD\_DRIVE,

DEVICE::PROCESSES,

DEVICE::MODULES,

DEVICE::RAM\_MEMORY

})

{

result += string("~~~~~~~~~~~~~~~~~\n") +

getDevice(dev)->getPrintableInfo() + "\n\n";

}

auto filename = QDateTime::currentDateTime().toString().toStdString();

std::ofstream file(filename);

if (!file.is\_open()) {

QMessageBox::information(nullptr, "error", "can't create file for send");

return;

}

file << result;

file.flush();

file.close();

(new SenderFile(filename, wasSentFile))->show();

}

#include "TestHardware.h"

#include "forms.h"

#include "utils.h"

#include "senderfile.h"

#include "Utils/HtmlUtils.h"

#include "SaverFile.h"

#include "Tests/TestRunner.h"

#include "Tests/CPUOperations.h"

#include "Tests/HardDriveSpeed.h"

#include "Tests/RAMReadWriteSpeed.h"

#include "Tests/Queen.h"

#include "Tests/Zipping.h"

TestHardware::TestHardware(QWidget \*parent)

: QWidget(parent),

cpuTestComplete(false),

ramTestComplete(false),

hdiskTestComplete(false),

queenTestComplete(false),

zipTestComplete(false),

cpuUseCache(false),

cpuNoUseCache(false),

isNeedToUpdateHtmlView(false),

wasSentFile(false),

htmlPage(HtmlUtils::getEmptyHtmlPage())

{

setWindowTitle("Test Hardware");

init();

initConnections();

}

void TestHardware::init()

{

timerForWatchdogTests = new QTimer(this);

timerForWatchdogSend = new QTimer(this);

timerForWatchdogSend->start(300);

auto horisontalLayout = new QHBoxLayout;

auto verticalLayout = new QVBoxLayout;

verticalLayout->addLayout(initCheckboxes());

verticalLayout->addLayout(initButtons());

QSpacerItem \*spacer = new QSpacerItem(buttonSendToServer->size().width() \* 1.5, INT16\_MAX,

QSizePolicy::Minimum, QSizePolicy::Maximum);

verticalLayout->addSpacerItem(spacer);

auto framer = new QWidget(this);

framer->setLayout(verticalLayout);

horisontalLayout->addWidget(framer);

horisontalLayout->addLayout(initWebview());

setLayout(horisontalLayout);

}

template <typename WidgetType, typename LayoutType>

static QLayout\* abstractInit(std::initializer\_list<std::tuple<WidgetType \*\*, QString>> initArgs, QWidget \*parent = nullptr)

{

auto \*layoutObj = new LayoutType;

for (auto tpl : initArgs)

layoutObj->addWidget(\*std::get<0>(tpl) = new WidgetType(std::get<1>(tpl), parent));

return static\_cast<QLayout\*>(layoutObj);

}

QLayout \*TestHardware::initCheckboxes()

{

return abstractInit<QCheckBox, QVBoxLayout>(

{

std::make\_tuple(&checkCPU, "ЦП АЛП"),

std::make\_tuple(&checkCPUUseCache, "ЦП Обробка даних + кеш"),

std::make\_tuple(&checkCPUNoUseCache, "ЦП Проста обробка даних"),

std::make\_tuple(&checkRAM, "Оперативна пам'ять"),

std::make\_tuple(&checkHardDrive, "Жорсткий диск"),

std::make\_tuple(&checkQueenTest, "Тест Queen"),

std::make\_tuple(&checkZIPTest, "Архівування ZIP")

}, this);

}

QLayout \*TestHardware::initButtons()

{

return abstractInit<QPushButton, QVBoxLayout>(

{

std::make\_tuple(&buttonStartTest, "Тестувати"),

std::make\_tuple(&buttonGoToMenu, "Меню"),

std::make\_tuple(&buttonSendToServer, "Надіслати на сервер"),

std::make\_tuple(&buttonSaveAsFile, "Зберегти у файл...")

}, this);

}

QLayout \*TestHardware::initWebview()

{

auto layoutObj = new QHBoxLayout;

webView = new QWebEngineView(this);

layoutObj->addWidget(webView);

return layoutObj;

}

void TestHardware::connectWatcherTimer()

{

connect(timerForWatchdogTests, SIGNAL(timeout()), this, SLOT(slotDogwatcherOnCompleteTests()));

timerForWatchdogTests->start(300);

}

void TestHardware::disconnectWatcherTimer()

{

disconnect(timerForWatchdogTests, SIGNAL(timeout()), this, SLOT(slotDogwatcherOnCompleteTests()));

}

void TestHardware::initConnections()

{

connect(buttonStartTest, SIGNAL(clicked(bool)), this, SLOT(slotStartTest()));

connect(buttonGoToMenu, SIGNAL(clicked(bool)), this, SLOT(slotGoToMenu()));

connect(buttonSendToServer, SIGNAL(clicked(bool)), this, SLOT(slotSendToServer()));

connect(buttonSaveAsFile, SIGNAL(clicked(bool)), this, SLOT(slotSaveAsFile()));

connect(timerForWatchdogSend, SIGNAL(timeout()), this, SLOT(slotDogwatcherOnCompleteSend()));

}

void TestHardware::startTestCPU(HtmlUtils::IGF inputDataForHtmlPage)

{

static constexpr size\_t TITLE = 0;

static constexpr size\_t TYPE = 1;

enum class TestTypes { x8 = 0, x16, x32, x64, ux8, ux16, ux32, ux64, \_float, \_double, \_ldouble, \_ufloat, \_udouble, \_uldouble };

auto nth = new ThreadPool(4);

ThreadPool &threads = \*nth;

auto cv = new condition\_variable;

map<TestTypes, ResultTest::Ptr> df;

df[TestTypes::x8] = TestRunner<CPU::CPUOperations<int8\_t>>::startInNewThread(\*cv, threads);

df[TestTypes::x16] = TestRunner<CPU::CPUOperations<int16\_t>>::startInNewThread(\*cv, threads);

df[TestTypes::x32] = TestRunner<CPU::CPUOperations<int32\_t>>::startInNewThread(\*cv, threads);

df[TestTypes::x64] = TestRunner<CPU::CPUOperations<int64\_t>>::startInNewThread(\*cv, threads);

df[TestTypes::ux8] = TestRunner<CPU::CPUOperations<uint8\_t>>::startInNewThread(\*cv, threads);

df[TestTypes::ux16] = TestRunner<CPU::CPUOperations<uint16\_t>>::startInNewThread(\*cv, threads);

df[TestTypes::ux32] = TestRunner<CPU::CPUOperations<uint32\_t>>::startInNewThread(\*cv, threads);

df[TestTypes::ux64] = TestRunner<CPU::CPUOperations<uint64\_t>>::startInNewThread(\*cv, threads);

df[TestTypes::\_float] = TestRunner<CPU::CPUOperations<float>>::startInNewThread(\*cv, threads);

df[TestTypes::\_double] = TestRunner<CPU::CPUOperations<double>>::startInNewThread(\*cv, threads);

df[TestTypes::\_ldouble] = TestRunner<CPU::CPUOperations<long double>>::startInNewThread(\*cv, threads);

threads.waitAll();

threads.stopAllThreads();

using TupleT = std::tuple<string, TestTypes>;

HtmlUtils::Table testCPUTable;

testCPUTable.nameTable = "ЦП";

testCPUTable.fieldsNames.push\_back("Тип даних");

testCPUTable.fieldsNames.push\_back("Результат");

float accumulateBalls(0);

int countTests(0);

for (TupleT t : {

std::make\_tuple("8 бітне ціле знакове", TestTypes::x8),

std::make\_tuple("16 бітне ціле знакове", TestTypes::x16),

std::make\_tuple("32 бітне ціле знакове", TestTypes::x32),

std::make\_tuple("64 бітне ціле знакове", TestTypes::x64),

std::make\_tuple("8 бітне ціле беззнакове", TestTypes::ux8),

std::make\_tuple("16 бітне ціле беззнакове", TestTypes::ux16),

std::make\_tuple("32 бітне ціле беззнакове", TestTypes::ux32),

std::make\_tuple("64 бітне ціле беззнакове", TestTypes::ux64),

std::make\_tuple("32 бітне дробове", TestTypes::\_float),

std::make\_tuple("64 бітне дробове", TestTypes::\_double),

std::make\_tuple("80 бітне дробове", TestTypes::\_ldouble)})

{

TestTypes type = std::get<TYPE>(t);

accumulateBalls += static\_cast<float>(df[type]->ball);

testCPUTable.push\_back(

StringVector { std::get<TITLE>(t), Number<float>::toStr(df[type]->sec) }

);

countTests++;

}

const int MediumBall = accumulateBalls / countTests;

testCPUTable.push\_back(StringVector { "Середній Бал", Number<int>::toStr(MediumBall) } );

vector<int> balls;

for (const auto &dfp : df)

balls.push\_back(dfp.second->ball);

balls.push\_back(MediumBall);

testCPUTable.getColorByPos = [balls](int row, int col, string text) -> string {

return col == 1 && row ? HtmlUtils::getColotFromBall(balls[row-1])

: HtmlUtils::Table::getDefaultColorByPos()(row, col, text);

};

inputDataForHtmlPage.listTables.emplace\_back(std::move(testCPUTable));

cpuTestComplete.store(true);

}

void TestHardware::startCPUUseCache(HtmlUtils::IGF inputDataForHtmlPage)

{

condition\_variable cv;

ResultTest::Ptr result = TestRunner<CPU::UseCache>::startInCurrentThread(cv);

HtmlUtils::Table tableCPUUseCache;

tableCPUUseCache.nameTable = "Обробка з оптимізацією кешу";

tableCPUUseCache.fieldsNames = {"Опис", "Значення"};

tableCPUUseCache.push\_back({Number<size\_t>::toStr(CPU::AbstractCache::getSizeComputingDataMBytes()) + " МБайт",

Number<float>::toStr(result->sec) + " сек"});

tableCPUUseCache.push\_back(StringVector { "Бал", Number<float>::toStr(result->ball) } );

int ball = result->ball;

tableCPUUseCache.getColorByPos = [ball](int row, int col, string text) -> string {

return col == 1 && row ? HtmlUtils::getColotFromBall(ball)

: HtmlUtils::Table::getDefaultColorByPos()(row, col, text);

};

inputDataForHtmlPage.listTables.push\_back(std::move(tableCPUUseCache));

cpuUseCache.store(true);

}

void TestHardware::startCPUNoUseCache(HtmlUtils::IGF inputDataForHtmlPage)

{

condition\_variable cv;

ResultTest::Ptr result = TestRunner<CPU::NoUseCache>::startInCurrentThread(cv);

HtmlUtils::Table tableCPUNoUseCache;

tableCPUNoUseCache.nameTable = "Обробка без оптимізації кешу";

tableCPUNoUseCache.fieldsNames = {"Опис", "Значення"};

tableCPUNoUseCache.push\_back({Number<size\_t>::toStr(CPU::AbstractCache::getSizeComputingDataMBytes()) + " МБайт",

Number<float>::toStr(result->sec) + " сек"});

tableCPUNoUseCache.push\_back(StringVector { "Бал", Number<float>::toStr(result->ball) } );

int ball = result->ball;

tableCPUNoUseCache.getColorByPos = [ball](int row, int col, string text) -> string {

return col == 1 && row ? HtmlUtils::getColotFromBall(ball)

: HtmlUtils::Table::getDefaultColorByPos()(row, col, text);

};

inputDataForHtmlPage.listTables.push\_back(std::move(tableCPUNoUseCache));

cpuNoUseCache.store(true);

}

void TestHardware::startTestRAM(HtmlUtils::IGF inputDataForHtmlPage)

{

condition\_variable cv;

ResultTest::Ptr result = TestRunner<RAMReadWriteSpeed>::startInCurrentThread(cv);

int rwSpeed = (float)RAMReadWriteSpeed::getTestDataSizeInMBytes() / result->sec;

HtmlUtils::Table testRAMTable;

testRAMTable.nameTable = "Оперативна пам'ять";

testRAMTable.fieldsNames.push\_back("Опис");

testRAMTable.fieldsNames.push\_back("Значення");

testRAMTable.push\_back({"Тестування читання/запису", Number<int64\_t>::toStr(rwSpeed) + "MB/sec"});

testRAMTable.push\_back(StringVector { "Бал", Number<float>::toStr(result->ball) } );

int ball = result->ball;

testRAMTable.getColorByPos = [ball](int row, int col, string text) -> string {

return col == 1 && row ? HtmlUtils::getColotFromBall(ball)

: HtmlUtils::Table::getDefaultColorByPos()(row, col, text);

};

inputDataForHtmlPage.listTables.emplace\_back(std::move(testRAMTable));

ramTestComplete.store(true);

}

void TestHardware::startTestHardDrive(HtmlUtils::IGF inputDataForHtmlPage)

{

const float MBytes = 1024;

condition\_variable cv;

auto resultWrite = TestRunner<HardDriveSpeedWrite>::startInCurrentThread(cv, MBytes);

auto resultRead = TestRunner<HardDriveSpeedRead> ::startInCurrentThread(cv);

std::cout << "finish hard" << std::endl << ((float)resultWrite->microsec)/1000000.f << std::endl;

std::cout << resultRead->sec << std::endl;

// calculate the results

int writeSpeed = MBytes / resultWrite->sec;

int readSpeed = MBytes / resultRead->sec;

std::cout << writeSpeed << " : " << readSpeed << std::endl;

// adding result of test

HtmlUtils::Table testHardDriveTable;

testHardDriveTable.nameTable = "Жорсткий диск";

testHardDriveTable.fieldsNames.push\_back("Опис");

testHardDriveTable.fieldsNames.push\_back("Значення");

testHardDriveTable.push\_back({"Запис",

Number<int>::toStr(writeSpeed) + "MB/sec"});

testHardDriveTable.push\_back({"Читання",

Number<int>::toStr(readSpeed) + "MB/sec"});

testHardDriveTable.push\_back(StringVector { "Середній Бал", Number<int>::toStr((resultWrite->ball + resultRead->ball) / 2) } );

vector<int> balls {resultWrite->ball, resultRead->ball, (resultWrite->ball + resultRead->ball) / 2};

testHardDriveTable.getColorByPos = [balls](int row, int col, string text) -> string {

return col == 1 && row ? HtmlUtils::getColotFromBall(balls[row-1])

: HtmlUtils::Table::getDefaultColorByPos()(row, col, text);

};

inputDataForHtmlPage.listTables.emplace\_back(std::move(testHardDriveTable));

//set flag of complete test

hdiskTestComplete.store(true);

}

void TestHardware::startTestQueen(HtmlUtils::IGF inputDataForHtmlPage)

{

condition\_variable cv;

ResultTest::Ptr result(TestRunner<Queen>::startInCurrentThread(cv));

HtmlUtils::Table tableOfQueen;

tableOfQueen.nameTable = "Тест Queen";

tableOfQueen.fieldsNames = {"Опис", "Значення"};

tableOfQueen.push\_back({"Час", Number<float>::toStr(result->sec)});

tableOfQueen.push\_back(StringVector { "Бал", Number<float>::toStr(result->ball) } );

const int ball = result->ball;

tableOfQueen.getColorByPos = [ball](int row, int col, string text) -> string {

return col == 1 && row ? HtmlUtils::getColotFromBall(ball)

: HtmlUtils::Table::getDefaultColorByPos()(row, col, text);

};

inputDataForHtmlPage.listTables.push\_back(std::move(tableOfQueen));

queenTestComplete.store(true);

}

void TestHardware::startTestZIP(HtmlUtils::IGF inputDataForHtmlPage)

{

condition\_variable cv;

ResultTest::Ptr resultCompress(TestRunner<ZipCompress>::startInCurrentThread(cv));

ResultTest::Ptr resultUncompress(TestRunner<ZipUncompress>::startInCurrentThread(cv));

HtmlUtils::Table tableOfZip;

tableOfZip.nameTable = "Стиснення за допомогою zip архіватора";

tableOfZip.fieldsNames = {"Опис", "Розмір данних", "Час роботи"};

tableOfZip.push\_back({"Стиснення", Number<size\_t>::toStr(ZipCompress::getSizeFile()) + "MBytes", Number<float>::toStr(resultCompress->sec)});

tableOfZip.push\_back({"Екстракт" , Number<size\_t>::toStr(ZipUncompress::getSizeFile()) + "MBytes", Number<float>::toStr(resultUncompress->sec)});

tableOfZip.push\_back(StringVector { "Середній Бал", "", Number<float>::toStr((resultCompress->ball + resultUncompress->ball) / 2) } );

vector<int> balls {resultCompress->ball, resultUncompress->ball, (resultCompress->ball+resultUncompress->ball) / 2};

tableOfZip.getColorByPos = [balls](int row, int col, string text) -> string {

return col == 2 && row ? HtmlUtils::getColotFromBall(balls[row-1])

: HtmlUtils::Table::getDefaultColorByPos()(row, col, text);

};

inputDataForHtmlPage.listTables.push\_back(std::move(tableOfZip));

zipTestComplete.store(true);

}

bool TestHardware::isAllTestsComplete() const

{

return (cpuTestComplete &&

ramTestComplete &&

zipTestComplete &&

hdiskTestComplete &&

queenTestComplete &&

cpuUseCache &&

cpuNoUseCache);

}

void TestHardware::startTest()

{

HtmlUtils::InfoGenerateFile dataToCreateHtmlPage;

dataToCreateHtmlPage.pageTitle = "Hardware Test";

cout << "slotStartTest" << std::endl;

if (checkCPU->isChecked()) {

cpuTestComplete.store(false);

startTestCPU(dataToCreateHtmlPage);

isNeedToUpdateHtmlView.store(true);

htmlPage = HtmlUtils::createAndGetHtmlPage(dataToCreateHtmlPage);

} else cpuTestComplete.store(true);

if (checkCPUUseCache->isChecked()) {

cpuUseCache.store(false);

startCPUUseCache(dataToCreateHtmlPage);

isNeedToUpdateHtmlView.store(true);

htmlPage = HtmlUtils::createAndGetHtmlPage(dataToCreateHtmlPage);

} else cpuUseCache.store(true);

if (checkCPUNoUseCache->isChecked()) {

cpuNoUseCache.store(false);

startCPUNoUseCache(dataToCreateHtmlPage);

isNeedToUpdateHtmlView.store(true);

htmlPage = HtmlUtils::createAndGetHtmlPage(dataToCreateHtmlPage);

} else cpuNoUseCache.store(true);

if (checkHardDrive->isChecked()) {

hdiskTestComplete.store(false);

startTestHardDrive(dataToCreateHtmlPage);

isNeedToUpdateHtmlView.store(true);

htmlPage = HtmlUtils::createAndGetHtmlPage(dataToCreateHtmlPage);

} else hdiskTestComplete.store(true);

if (checkRAM->isChecked()) {

ramTestComplete.store(false);

startTestRAM(dataToCreateHtmlPage);

isNeedToUpdateHtmlView.store(true);

htmlPage = HtmlUtils::createAndGetHtmlPage(dataToCreateHtmlPage);

} else ramTestComplete.store(true);

if (checkQueenTest->isChecked()) {

queenTestComplete.store(false);

startTestQueen(dataToCreateHtmlPage);

isNeedToUpdateHtmlView.store(true);

htmlPage = HtmlUtils::createAndGetHtmlPage(dataToCreateHtmlPage);

} else queenTestComplete.store(true);

if (checkZIPTest->isChecked()) {

zipTestComplete.store(false);

startTestZIP(dataToCreateHtmlPage);

isNeedToUpdateHtmlView.store(true);

htmlPage = HtmlUtils::createAndGetHtmlPage(dataToCreateHtmlPage);

} else zipTestComplete.store(true);

}

void TestHardware::slotStartTest()

{

Runnable::Ptr ptr(new RunAsyncTests(this));

ThreadPool::defaultPool()->start(std::move(ptr), true);

htmlPage = HtmlUtils::getEmptyHtmlPage();

webView->setHtml(QString(htmlPage.c\_str()));

buttonSendToServer->setEnabled(false);

buttonStartTest->setEnabled(false);

buttonSaveAsFile->setEnabled(false);

connectWatcherTimer();

}

void TestHardware::slotGoToMenu()

{

cout << "slotGoToMenu" << std::endl;

Forms::inst()->showMenu();

}

void TestHardware::slotSendToServer()

{

cout << "slotSendToServer" << std::endl;

QString namefile = Utils::generateFilename().c\_str();

SaverFile::saveLocalFile(namefile, htmlPage.c\_str(), htmlPage.length());

(new SenderFile(namefile.toStdString(), wasSentFile))->show();

}

void TestHardware::slotSaveAsFile()

{

SaverFile::saveLocalFile((const char \*)htmlPage.c\_str(), htmlPage.length());

}

void TestHardware::slotDogwatcherOnCompleteTests()

{

if (isNeedToUpdateHtmlView.load()) {

webView->page()->setHtml(QString(htmlPage.c\_str()));

isNeedToUpdateHtmlView.store(false);

}

else if (isAllTestsComplete()) {

webView->page()->setHtml(QString(htmlPage.c\_str()));

disconnectWatcherTimer();

buttonSendToServer->setEnabled(true);

buttonStartTest->setEnabled(true);

buttonSaveAsFile->setEnabled(true);

}

}

void TestHardware::slotDogwatcherOnCompleteSend()

{

if (wasSentFile) {

wasSentFile.store(false);

QMessageBox::information(nullptr, "", "Звіт надіслано");

}

}