МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМ. В. ДАЛЯ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ЕЛЕКТРОНІКИ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК ТА ІНЖЕНЕРІЇ

До захисту допускається
Завідувач кафедри
_____Скарга-Бандурова І.С.
«____»_____ 2018 р.

ДИПЛОМНИЙ ПРОЕКТ (РОБОТА) БАКАЛАВРА

ПОЯСНЮВАЛЬНА ЗАПИСКА

НА ТЕМУ:

Комплект програм серверу станційного обладнання АКРО-Б

Освітньо-кваліфікаційний рівень "бакалавр"
Напрям 6.050102 – "Комп'ютерна інженерія"

Керівник проекту:	_____ (підпис)	проф. д.т.н. Єлісєєв В. В. (ініціали, прізвище)
Консультант з охорони праці:	_____ (підпис)	ст.викл.Критська Я. О. (ініціали, прізвище)
Здобувач вищої освіти:	_____ (підпис)	Федченко А. С. (ініціали, прізвище)
Група:		КІ-14аД

Сєвєродонецьк 2018

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет Інформаційних технологій та електроніки
Кафедра Комп'ютерних наук та інженерії
Освітньо-кваліфікаційний рівень бакалавр
Напрям підготовки 6.050102 Комп'ютерна інженерія
(шифр і назва)
Спеціальність
(шифр і назва)

ЗАТВЕРДЖУЮ:

Завідувач кафедри
І.С. Скарга-Бандурова
«_____» _____2018 р.

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) БАКАЛАВРА

Федченко Артем Сергійович
(прізвище, ім'я, по батькові)

1. Тема роботи Комплект програм серверу станційного обладнання АКРО-Б

керівник проекту (роботи) Єлісєєв В. В., проф. д.т.н.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "14 " 05 2018р. №
2. Термін подання студентом роботи
3. Вихідні дані до роботи матеріали переддипломної практики

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Огляд об'єкту дослідження. Розробка алгоритмів комплекту програм. Написання комплекту програм. Тестування роботи кожної програми. Охорона праці. Висновки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Електронні плакати

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	ст.викл. кафедри КНІ Критська Я.О.		

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Огляд літератури з теми ДП і постановка задачі	14.05.18-19.05.18	
2	Дослідження матеріалів	20.05.18-25.05.18	
3	Розробка комплекту програм	26.05.18-02.06.18	
4	Тестування комплекту програм	03.06.18-06.06.18	
5	Розробка розділу охорона праці	07.06.18-09.06.18	
6	Оформлення електронних плакатів	10.06.18-12.06.18	
7	Оформлення пояснювальної записки	13.06.18-15.06.18	

Здобувач вищої освіти _____ Федченко А.С.
(підпис) (прізвище та ініціали)

Керівник _____ Єлісєєв В.В.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломного проекту бакалавра: 90 сторінок, 22 рисунка, 14 таблиць, 25 бібліографічних джерел, 6 додатків.

Предметом дослідницької праці є розробка комплекту програм серверу станційного обладнання АКРО-Б.

У вступі представляється об’єкт дослідження з вказанням на його практичну та теоретичну важливість.

У первому розділі представлена інформація про стрктуру об’єкту дослідження та визначаються вимоги до комплексу програмного забезпечення з постановкою технічного завдання.

Другий розділ містить у собі детальний опис інструментів розробки з вказанням галузей їх застосування.

В третьому розділі представлено проектування та розробку кожної з програм комплекту. Проектування велося в декілька етапів: розробка алгоритмів, визначення структур даних, організація програми у вигляді модулів. Розробка кожної програми була описана.

Четвертий розділ описує програмну реалізацію кожної з програм комплекту станційного обладнання з детальними поясненнями.

Висновок являє собою підведення підсумків та перспектив проекту у майбутньому.

Перелік ключових скорочень: АКРО-Б, ПТО, АРМ, ПЗ, ЛПК.

ПЕРЕЛІК СКОРОЧЕННЬ

АКРО-Б	– апаратура дистанційного температурного контролю технічного стану ходових частин рухомих одиниць залізничного транспорту;
АСДК-Б, КТСМ, ДИСК, ПОНАБ	– комплекси контролю буксових вузлів рухомих одиниць залізничного транспорту, що знаходяться в експлуатації в теперішній час;
АРМ	– автоматизоване робоче місце;
ВК	– вимірювальні канали;
ДСП	– черговий по станції;
ЛПК	– лінійний пункт контролю;
ПТО	– пункт технічного огляду вагонів;
ПЗ	– програмне забезпечення
ТСР	– протокол контролю передачі даних

ЗМІСТ

ВСТУП

Автоматизовані системи контролю технічного стану рухомого складу дозволяють своєчасно виявити і усунути несправності ходових частин рухомого складу, що з'являються в процессі експлуатації, і тим самим попередити виникнення незворотних відмов, здатних привести до аварій і катастроф, збільшити швидкості руху потягів, скоротити витрати часу на технічне обслуговування складів, збільшити відстані невпинного пробігу поїздів без технічного обслуговування вагонів.

У зв'язку з цим, в Україні та в ряді зарубіжних країн проводиться розробка і впровадження апаратури контролю найбільш важливих вузлів рухомого складу.

На залізницях багатьох країн світу застосовуються системи контролю та діагностування елементів рухомих одиниць на ходу поїзда. На залізницях України такі задачі протягом останніх десятиліть вирішувалися з використанням пристроїв ПОНАБ, ДИСК, АСДК-Б. В даний час в Україні розробляється і впроваджується система контролю технічного стану ходових частин рухомих одиниць залізничного транспорту АКРО-Б. Вона призначена для заміни на мережі залізниць України застарілих комплексів ПОНАБ, ДИСК, КТСМ-2, АСДК-Б, які використовуються в наш час.

Система є базовою і дозволяє при підключенні додаткових датчиків та відповідної доробки програмного забезпечення (ПЗ) нарощувати її функціональні можливості. Додатково до виявлення перегрітих буксових вузлів і загальмованих колісних пар рухомих одиниць, система може забезпечити виявлення деформації коліс по колу катання, контроль нижнього габариту, автоматичну ідентифікацію кожної рухомої одиниці потягу.

АКРО-Б складається з перегінного обладнання, станційного обладнання і обладнання каналу зв'язку між ними.

Типи обладнання каналу зв’язку можуть розрізнятися для різних комплектів апаратури АКРО-Б в залежності від типу та параметрів виділеної лінії зв’язку між перегінним та станційним обладнанням. Це можуть бути: аналогові модеми, цифрові модеми, оптоволоконні подовжувачі та інші.

Метою дипломного проекту є розробка комплекту програм серверу станційного обладнання АКРО-Б., яке повинно забезпечувати двосторонній зв'язок з кількома комплектами постового обладнання, архівування отриманих даних, надання оперативної та архівної інформації оператору за допомогою графічного інтерфейсу, передачу команд оператора в комплекти постового обладнання. Комплект програм повинен функціонувати під управлінням ОС Linux.

1. ОГЛЯД ФУНКЦІЙ ТА СТРУКТУРИ АКРО-Б. ПОСТАНОВКА ЗАДАЧІ

1.1 Структура та функції апаратури дистанційного температурного контролю

Апаратура дистанційного температурного контролю АКРО-Б розроблена для застосування на залізничному транспорті України та призначена для автоматичного виявлення перегрітих буксових вузлів та загальмованих колісних пар рухомих одиниць під час руху поїзда.

АКРО-Б була розроблена з метою:

- оснащення залізничних доріг України сучасними засобами контролю рухомих одиниць залізничного транспорту, у тому числі заміни апаратури АСДК-Б, КТСМ, ДИСК, ПОНАБ, що знаходиться в експлуатації в теперішній час;
- скорочення експлуатаційних витрат і зниження трудомісткості обслуговування апаратури за рахунок:
    1) автоматичного градуювання вимірювальних каналів (далі – ВК) радіаційної температури за допомогою вбудованих швидкодіючих джерел випромінювання;
    2) автоматичної корекції градуювальних характеристик ВК радіаційної температури за результатами періодичної повірки (калібрування) вбудованих джерел випромінювання зразковим джерелом випромінювання;
    3) автоматичної підтримки оптимальної температури охолодження чутливого шару приймачів інфрачервоного випромінювання при зміні температури навколишнього середовища;

4) автоматичного періодичного самоконтролю апаратури (діагностика основних параметрів апаратури);

5) реалізації конструктивних рішень, спрямованих на зниження трудомісткості операцій монтажу/демонтажу польового обладнання;

6) реалізації конструктивних рішень, спрямованих на спрощення операцій орієнтування лінії візування вимірювальних камер і збільшення інтервалів часу між періодичними перевірками кутів орієнтації;

7) забезпечення можливості дистанційного тестування працездатності апаратури з адресним зазначенням дефектних вузлів за результатами тестування. Концепція діагностики апаратури повинна передбачати адресне виявлення рідко повторюваних, випадкових збоїв;

- забезпечення модульної структури системи контролю рухомих одиниць залізничного транспорту. Модульна структура системи контролю за рахунок додавання до її базового елементу (АКРО-Б) додаткових елементів (модулів) забезпечує виконання наступних функцій:

  1) виявлення деформації коліс по колу катання;

  2) контроль нижнього габариту рухомих одиниць;

  3) автоматична ідентифікація кожної рухомої одиниці поїзда;

- врахування сучасних тенденцій розвитку залізничного транспорту: зростання швидкості руху поїздів, різноманіття типів вагонів, конструкцій візків, буксових вузлів, гальмівних систем, підвищення навантажень на осі вагонів. Створювана апаратура адаптується до мінливих умов шляхом конфігурування її програмного забезпечення.

Структурна схема апратури дистанційного температурного контролю АКРО-Б наведена на рисунку 1.1.

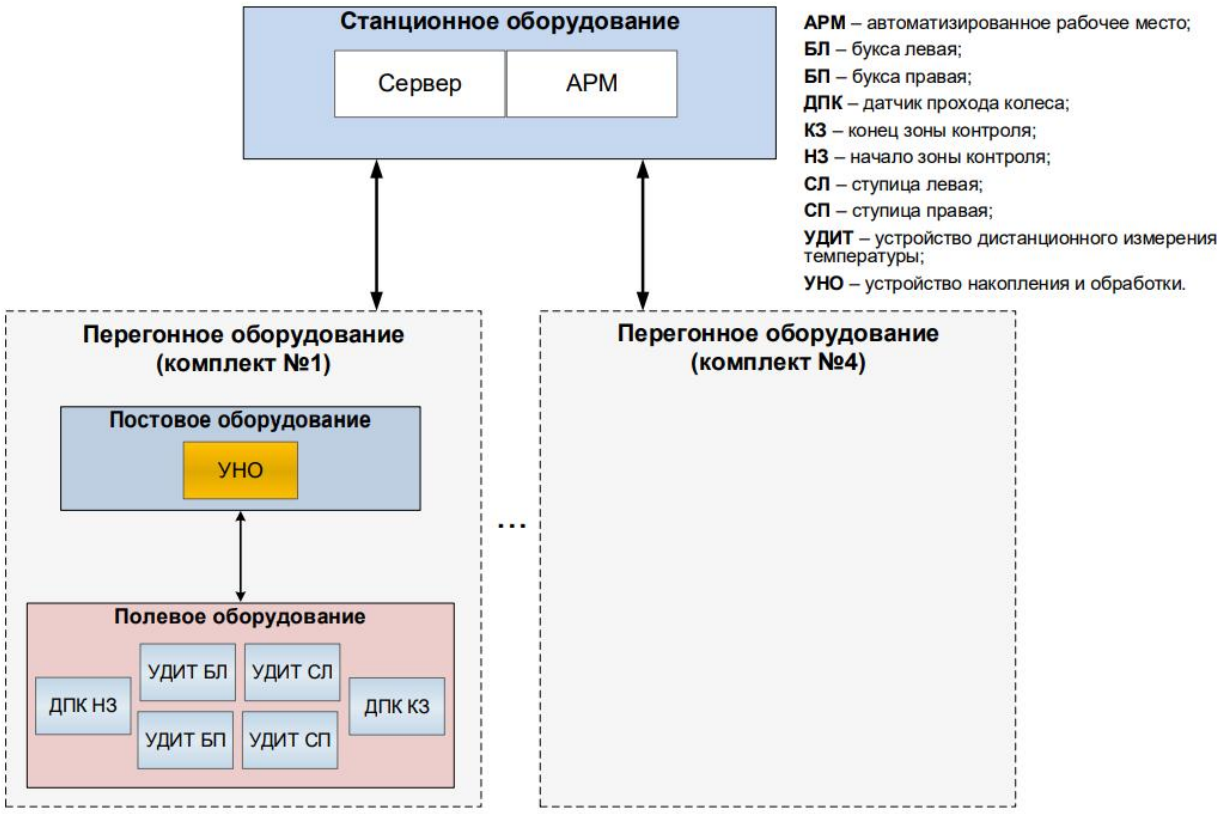


Рисунок 1.1 – Структурна схема АКРО-Б

1.2 Функції об’єкту контролю

АКРО-Б забезпечує дистанційний температурний контроль технічного стану ходових частин всіх рухомих одиниць залізничного транспорту, що слідують через зону контролю ЛПК, де вона встановлена.

Швидкість проходження поїздами зони контролю – від 1 до 350 км/год.

Кількість рухомих одиниць в поїзді – необмежено.

Кількість осей в рухомій одиниці – від 2 до 32.

Кількість осей в поїзді – необмежено.

Максимальна довжина поїзда – необмежена.

Мінімальний інтервал часу між двома поїздами, що слідують через зону контролю (між останньою віссю попереднього поїзда і першою віссю наступного поїзда) – 2 хв.

Максимальна кількість поїздів, що слідують через зону контролю в одному напрямку, за добу – необмежена.

Розташування і орієнтація рухомих одиниць у складі поїзда можуть бути довільними.

1.3 Функції та характеристики серверу станційного обладнання

Сервер станційного обладнання апаратури АКРО-Б побудований на основі промислового комп'ютера ПС5150(рис. 1.2), який складається з:

- процесорного модуля (системного блоку) з інтерфейсом приєднання до АРМ;
- джерела безперебійного живлення;
- накопичувача даних, об'ємом не менше 120 Gb.

Крім того, сервер має зовнішні інтерфейси для забезпечення:

- підключення одного або декількох комплектів перегінного обладнання апаратури АКРО-Б;
- підключення одного або декількох АРМ операторів апаратури АКРО-Б;

Ємність накопичувачів даних серверу станційного обладнання забезпечує зберігання технологічних та діагностичних даних, отриманих від перегінного обладнання протягом останніх 90 діб.

Рисунок 1.2 – ПС5150. Серверна конфігурація

Функції, які виконує сервер станційного обладнання апаратури АКРО-Б:

- прийом від перегінного обладнання даних про результати контролю поїздів та контролю працездатності перегінного обладнання (самоконтролю);
- видача в перегінне обладнання команд та даних;
- у випадку виявлення несправних вузлів рухомих одиниць відбувається сповіщення про результати контролю операторам підключеної до серверу апаратури АРМ;
- виготовлення твердих копій (друк) звітних документів по роботі апаратури АКРО-Б;
- накопичення, зберігання та надання для перегляду даних про результати контролю довільно обраного поїзда за останні 90 діб.

1.4 Актуальність розробки

У зв'язку з розробкою апаратури дистанційного температурного контролю технічного стану ходових частин і необхідності в забезпеченні двобічного зв'язку з декількома його комплектами, архівування отриманих даних, надання оперативної та архівної інформації оператору за допомогою графічного інтерфейсу, виникла необхідність в розробці комплекту програмного забезпечення для серверу станційного обладнання АКРО-Б.

1.5 Постановка задачі

Повинний бути розроблений комплект програм для серверу станційного обладнання АКРО-Б, який має складатися з:

- програми демону зв'язку, що призначена для забезпечення двонаправленного зв'язку з кількома комплектами постового обладнання та для архівування отриманих даних;
- програми демону архіву, що призначена для забезпечення захисту від переповнення архіву серверу станційного обладнання.

Окремо має бути розроблена програма з графічним інтерфейсом для обладнання АРМ, в якій реалізувати отримання та відображення актуальних даних від підключених до серверу комплектів постового обладнання.

Загальними вимогами до комплекту програм являються:

- Інформаційний обмін між технічними засобами апаратури АКРО-Б повинен базуватися на стандартних протоколах обчислювальних систем та локальних мереж;
- Програми повинні мати модульну структуру;

1.5.1 Вимоги до демону зв'язку

Програма демону зв'язку повинна забезпечувати виконання наступних функцій:

- Зв'язок між сервером та комплектами постового обладнання, обладнанням АРМ, повинен здійснюватися по власному протоколу, у якому передбачити контроль цілісності та упорядкованості пакетів;
- Демон зв’язку повинен бути розроблений на мові високого рівня C/C++.

Вхідні дані для програми демону зв'язку:

- Файл конфігурації в форматі INI;
- Повідомлення від комплектів постового обладнання та обладнання АРМ у форматі власного протоколу.

Вихідними даними програми демону являються:

- Файл логування;
- Відповіді на повідомлення від комплектів постового обладнання та обладнання АРМ у форматі власного протоколу.

1.5.2 Вимоги до демону архіву

Програмне забезпеченя, яке має виконувати роль демону архіву, повинно бути розроблено на мові високого рівня Python.

Вхідні дані:

- Файл конфігурації cfg.py;

Вихідними даними програми являються:

- Файл логування;

1.5.3 Вимоги до программи для обладнання АРМ

Програмне забезпечення з графічним інтерфейсом для обладнання АРМ повинно функціонувати в операційній системі Linux.

Вхідні дані:

- Файл налаштувань у форматі INI;
- Повідомлення від серверу станційного обладнання у форматі власного протоколу зв'язку.

Вихідні дані:

- Файл логування;
- Відповіді на повідомлення від серверу у форматі власного протоколу зв'язку.

Висновок до розділу 1

У цьому розділі був проведений огляд структури апаратури дистанційного температурного контролю АКРО-Б, описані її функції та характеристики. Були сформовані вимоги до комплекту програм серверу станційного обладнання.

2 АНАЛІЗ ІНСТРУМЕНТІВ РОЗРОБКИ

В даному розділі аналізуються основні інструменти, які були використані для розробки комплекту програм серверу станційного обладнання.

2.1 Мова програмування Сі

Сі - компільована статично типізована мова програмування загального призначення[1]. Згідно дизайну мови, її конструкції близько зіставляються типовим машинним інструкціям, завдяки чому вона знайшла застосування в проектах, для яких була властива мова асемблера, в тому числі як в операційних системах, так і в різному прикладному програмному забезпеченні для безлічі пристроїв - від суперкомп'ютерів до вбудованих систем. Мова Сі призначена для написання компактних та швидких програм[1].

Мова Сі має дуже багату систему типів, яка класифікується наступним чином[2]:

- примітивні типи(цілі числа та числа з плаваючою точкою);
- типи перерахувань;
- тип void;
- виведені типи(вказівники, массиви, структури, об’єднання).

Код на Сі можна легко писати на низькому рівні абстракції, майже як на асемблері. Іноді Сі називають «універсальним асемблером» або «асемблером високого рівня», що відображає відмінність мов асемблера для різних платформ і єдність стандарту Сі, код якої може бути скомпільовано без змін практично на будь-якої моделі комп'ютера. Сі часто називають мовою середнього рівня або навіть низького рівня, з огляду на те, як близько

він працює до реальних пристроїв. Однак в суворої класифікації вона є мовою високого рівня.

Основні особливості мови Сі:

- орієнтація на процедурне і структурне програмування;
- використання препроцессору;
- мінімальна кількість ключових слів;
- безпосередній доступ до пам’яті комп’ютера за допомогою вказівників;
- статична слабка система типів.

Багато елементів Сі потенційно небезпечні, а наслідки неправильного використання цих елементів часто непередбачувані. У зв'язку з порівняно низьким рівнем мови багато випадків неправильного використання небезпечних елементів не виявляються і не можуть бути виявлені ні при компіляції, ні під час виконання. Це часто призводить до непередбачуваної поведінки програми. Іноді в результаті неграмотного використання елементів мови з'являються вразливості в системі безпеки. Необхідно зауважити, що використання багатьох таких елементів можна уникнути.

Недоліки мови Сі:

- Відсутність засобів автоматичного управління пам'яттю;
- Відсутність засобів функціонального програмування;
- Велика кількість потенційних вразливостей;
- Високий поріг входження.

Мова Сі широко використовується як для системного програмування операційних систем та вбудованих системних додатків, завдяки високій кросплатформенності на етапі компіляції, так і для програмування прикладних програм в середовищах з обмеженими ресурсами.

2.2 Мова програмування C++

C++ – компільована, статично типізована мова програмування загального призначення[3][4][5]. Мова C++ підтримує основні концепції і методи. використовувані в реальних комп'ютерних програмах. Підтримує такі парадигми програмування, як процедурне програмування, об'єктно-орієнтоване програмування, узагальнене програмування.

C++ поєднує властивості як високорівневих, так і низькорівневих мов. Мова C++ є однією з найбільш широко поширених мов програмування[6]. Це зумовлено, не в останню чергу, завдяки можливості розробки програм для компіляції на різних платформах[7].

Синтаксис C++ успадкований від мови Сі. Одним з принципів розробки було збереження сумісності з Сі. Проте, C++ не є в строгому сенсі надмножиною Сі; кількість програм, які можуть однаково успішно транслюватися як компіляторами Сі, так і компіляторами C++, досить велике, але не включає всі можливі програми на Сі.

Основні переваги мови C++:

- Підтримка різних стилів програмування:
- об'єктно-орієнтоване програмування, узагальнене програмування, функціональне програмування;
- Висока сумісність з мовою Сі;
- Висока продуктивність компільованого машинного коду;
- Можливість роботи з пам'яттю на низькому рівні.

Недоліки мови C++:

- Високий поріг входження;
- Низька якість засобів функціонального програмування;
- Програмування з використанням шаблонів знижує продуктивність машинного коду;
- Велика кількість потенційних вразливостей;

- Неповна обратна сумісність стандартів C++.

C++ широко використовується для розробки програмного забезпечення, будучи одною з найпопулярніших мов програмування. Область його застосування включає створення операційних систем, різноманітних прикладних програм, драйверів пристроїв, додатків для вбудованих систем, високопродуктивних серверів, а також розважальних програм (ігор). Крім того, існує точний і повний загальнодоступний міжнародний стандарт мови C++, що не захищений правом власності. Якісні та/або безкоштовні реалізації цієї мови доступні для будь-яких комп'ютерів[6].

2.3 Мова програмування Python

Python - скриптова мова загального призначення з відкритим початковим кодом, підтримкою структурного, об'єктно-орієнтованого, функціонального і процедурного програмування[8][9]. Як правило, вона застосовується для написання як автономних програм, так і скриптів, які доповнюють інші програмні продукти. Вважається однією з найбільш широко вживаних мов програмування в світі.

До характерних особливостей Python відносяться акцент на зручності читання початкового коду та функціональних можливостях стандартної бібліотеки[10]. Python дозволяє оптимізувати продуктивність праці розробника, якість програмного забезпечення, переносимість програм та інтеграцію їх компонентів[11]. Програми на Python виконуються на більшості загальновживаних платформ, включаючи Unix і Linux, Windows, MacOS, Android, iOS та ін., але в той же час Python надає великі можливості для використання бібліотек конкретної платформи, на шкоду кроссплатформенності.

Однією з цікавих синтаксичних особливостей мови є виділення блоків коду за допомогою відступів (пробілів чи табуляцій), тому в Python відсутні

операторні дужки «begin / end», як у мові Паскаль, або фігурні дужки, як в Сі. Такий «трюк» дозволяє скоротити кількість рядків і символів в програмі і привчає до «хорошого» стилю програмування. З іншого боку, поведінка і навіть коректність програми може залежати від початкових пробілів в тексті.

Python підтримує динамічну типізацію, тобто тип змінної визначається лише під час виконання. В Python є вбудовані типи: булевий, рядок, Unicode-рядок, ціле число довільної точності, число з плаваючою комою, комплексне число і деякі інші[12]. З колекцій в Python вбудовані: список, кортеж (незмінний список), словник, множина та інші. Всі значення є об'єктами, в тому числі функції, методи, модулі, класи.

Переваги мови Python:

- Підтримка різних стилів програмування;
- Повнота стандартної бібліотеки;
- Можливість взаємодії з кодом на Сі;
- Потужні інтерфейси для конкретних операційних систем;
- Вбудовані структури даних;
- Швидкість розробки програм.

Недоліки мови Python:

- Обмеженість функціонального програмування;
- Невисока швидкодія;
- Важкість розробки великих проектів.

2.4 Фреймворк Qt

Qt являє собою комплексну робочу середу, призначену для розробки міжплатформених додатків з графічним інтерфейсом на C++[13]. Qt дозволяє програмістам використовувати дерево класів з одним джерелом в додатках, які будуть працювати в системах від Windows 95 до 10, Mac OS X, Linux, Solaris, HP-UX та в багатьох інших версіях Unix з X11.

Qt створила собі репутацію засобу розробки міжплатформених додатків, але завдяки своєму інтуїтивному і потужному програмного інтерфейсу в багатьох організаціях Qt використовується для одноплатформенних розробок. Безліч складних систем програмного забезпечення на таких вертикальних ринках, як засоби анімації 3D, цифрова обробка фільмів, автоматизація проектування електронних схем (для проектування чіпів), розвідка нафтових і газових родовищ, фінансові послуги та формування зображень в медицині, будуються за допомогою Qt.

Відмітна особливість Qt - використання метаоб'єктного компілятора - системи попередньої обробки вихідного коду[14]. Розширення можливостей забезпечується системою плагінів, які можливо розміщувати безпосередньо в панелі візуального редактора. Також існує можливість розширення звичної функціональності віджетів, пов'язаної з розміщенням їх на екрані, відображенням, перемальовуванням при зміні розмірів вікна.

Qt може застосовуватися з різними ліцензіями. Якщо ви збираєтеся створювати комерційні додатки, ви повинні придбати комерційну ліцензію Qt; якщо ви збираєтеся створювати програми з відкритим початковим кодом, ви можете використовувати версію з відкритим початковим кодом (з ліцензією GPL). Qt є основою, на якій побудовані K Desktop Environment (KDE) та багато інших додатків з відкритим початковим кодом.

За останню декаду Qt перетворилася з «секретного» програмного продукту, відомого тільки обраної групі професіоналів, в продукт, яким користуються по всьому світу тисячі комерційних замовників і десятки тисяч розробників додатків з відкритим початковим кодом.

2.5 POSIX Threads

Потоки дозволяють одночасно виконувати деякі дії в контексті однієї програми[15][16]. Механізми роботи з потоками реалізовані в Linux в окремій бібліотеці Pthread. Потоки всередині одного процесу ділять секції

коду, даних, а також різні ресурси: дескриптори відкритих файлів, сигнали, значення umask, nice, таймери та інше. Щоб підключити цю бібліотеку до програми, необхідно передати компонувальнику опцію -lpthread і включити заголовний файл <pthread.h> в початковим коді програми.

Потоки дозволяють в рамках однієї програми виконувати одночасно кілька дій, використовуючи при цьому загальні дані. У Linux потоки виконуються так само, як і процеси, тобто незалежно[17].

Реалізація потоку в Linux складається з двох кроків:

1) Створюється функція, яка називається потокової функцією;

2) За допомогою функції pthread_create() створюється потік, в якому починає, паралельно решті програми, виконуватися потокова функція.

Потоки, подібно процесам, працюють з ідентифікаторами. Для їх зберігання передбачений спеціальний тип pthread_t.

Програма зазвичай завершується шляхом повернення з функції main() або через виклик функції exit(). Аналогічним чином працюють і потоки, які можуть завершуватися за допомогою оператора return або викликом спеціальної функції pthread_exit().

Функція pthread_join() дозволяє синхронізувати потоки. Ця функція блокує викликаючий її потік до тих пір, поки не завершиться потік з необхідним ідентифікатором.

Будь-який потік може надіслати запит на завершення будь-якому іншому потоку цього ж процесу. Такий запит називають скасуванням потоку. Для цього передбачена функція pthread_cancel(). Варто відмітити, що виклик функції pthread_cancel() зовсім не означає негайного завершення потоку – в залежності від встановлених атрибутів, потік може проігнорувати запит.

Потоки POSIX є незамінним інструментом при розробці високонавантажених додатків з модульною структурою завдяки своїй простоті, ефективності і швидкості роботи.

2.6 POSIX Sockets

Сокети - універсальний спосіб взаємодії процесів[15][16][17]. Вони можуть використовуватися як для локальної взаємодії, так і для обміну даних між віддаленими системами.

Ядро Linux підтримує десятки родин протоколів, найбільш розповсюдженими з яких є дві: Unix-сокети та Інтернет-сокети. Оскільки Unix-сокети забезпечують взаємодію процесів в рамках однієї операційної системи, їх часто називають локальними сокетами.

За способом взаємодії сокети також діляться на кілька типів. Найбільш розповсюдженими типами є потокові і датаграммні сокети. Перші служать для передачі даних в мережі за допомогою протоколів, що забезпечують контроль цілісності інформації. Найбільш поширеною протоколом такого типу є TCP. Датаграммні сокети, навпаки, орієнтовані на швидку передачу інформації, на шкоду контролю цілісності. У мережі такі дані найчастіше передаються по протоколу UDP.

Для створення сокетів призначений системний виклик socket(), який, при успішному виклику, повертає дескриптор сокета. У разі помилки повертається -1.

Щоб сервер і клієнт могли взаємодіяти, сокету потрібно призначити адресу(ім'я). Залежно від типу сокета адресою може бути:

- ім'я файлу(локальна взаємодія);
- мережева адреса і порт(віддалена взаємодія).

Адреса сокету призначається з боку сервера. Клієнт може використовувати цю адресу для під'єднання до сервера і обміну даними.

При віддаленій взаємодії однієї адреси недостатньо: справа в тому, що сервер може одночасно обмінюватися даними з декількома клієнтами. Крім того, на вузлі одночасно може працювати декілька різних серверів. Для вирішення даної проблеми використовують порти. Порт - число, яке

ідентифікує сеанс взаємодії між процесами. Кожен сеанс взаємодії між процесами ідентифікується окремим портом.

При використанні потокових сокетів між взаємодіючими сторонами має відбутися з'єднання. Для цього клієнт викликає системний виклик connect(). Сервер ж, в свою чергу, повинен завчасно перейти в режим очікування запитів на підключення. Робиться це за допомогою системного виклику listen(), який блокує сервер доти, поки який-небудь клієнт не видасть запит на підключення. Як тільки запит надійшов, сервер «прокидається». Якщо є можливість обслужити питання, сервер викликає системний виклик accept(). У разі успішного з'єднання, клієнт і сервер отримують можливість

обмінюватися даними до тих пір, поки один з процесів не обірветь зв'язок або повідомить про неможливість прийняти дані[17].

Висновок до розділу 2

У розділі 2 був проведений детальний аналіз інструментів розробки з визначеням їх галузей застосування. Були вказані переваги та недоліки для кожного з інструментів.

3 ПРОЕКТУВАННЯ ТА РОЗРОБКА КОМПЛЕКТУ ПРОГРАМ

3.1 Проектування та розробка демону зв'язку

3.1.1 Проектування демону зв'язку

Проектування демону зв'язку проводилося у декілька етапів:

1. Розробка алгоритму демону зв'язку;
2. Розробка протоколу зв'язку ЛПК з сервером;
3. Розробка протоколу зв'язку ПЗ з графічним інтерфейсом для АРМ із сервером;
4. Розробка структур даних.

3.1.1.1 Проектування алгоритму демону зв'язку

Робота демону зв’язку починається з виконання функції main, яка представляє з себе точку входу в програму. Програма переходить до фонового режиму виконання. Далі демон ініціює налаштування та запускає у нових потоках прослуховування сокетів на предмет очікування нових з’єднаннь від ЛПК та программи з графічним інтерфейсом від АРМ. Далі головний поток демона переходить до режиму очікування закриття прослуховування від ЛПК. Алгоритм запуску демону зв’язку наведено на рисунку 3.1.

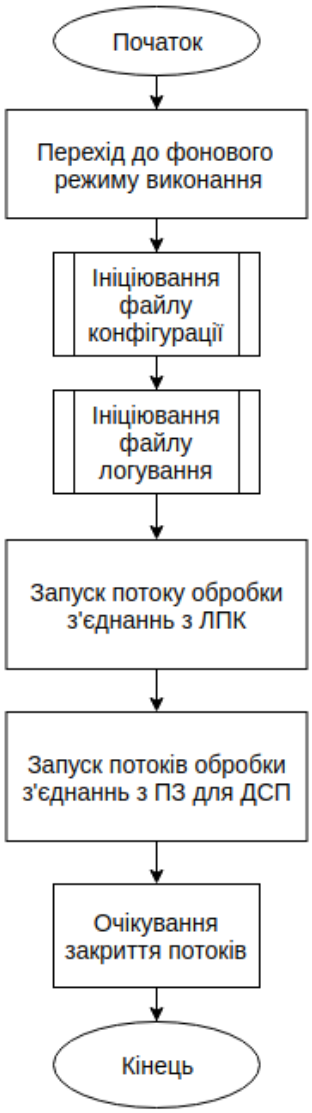


Рисунок 3.1 – Схема алгоритму запуску демону зв'язку

Обробка підключень від ЛПК відбувається в окремому потоці. Під час під’єднання ЛПК до демону зв’язку, функція створює структуру з даними про з’єднання. Створюється нова сесія в новому потоці, після чого функція повертається до режиму прослуховування. Алгоритм обробки підключень від ЛПК зображений нижче на рисунку 3.2.

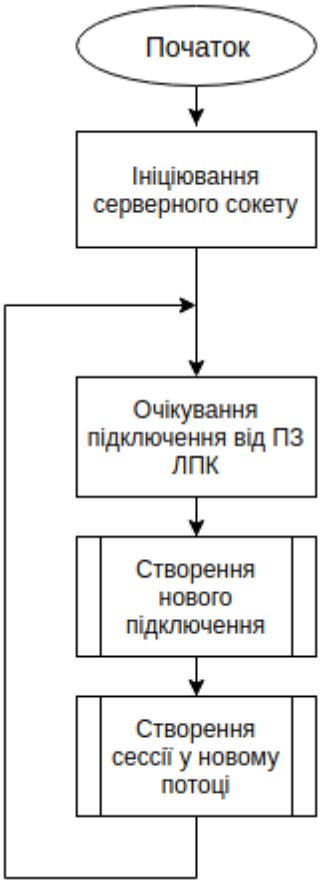


Рисунок 3.2 – Схема алгоритму обробки підключеннь від ЛПК

Обробка підключень від АРМ відбувається в окремому потоці. Функція приймає параметр, який відповідає за тип сокету, в залежності від якого створюються різні налаштування для серверного сокету. Під час під’єднання додатку від АРМ, створюється сессія у новому потоці. Схема алгоритму обробки підключень від АРМ зображена на рисунку 3.3.

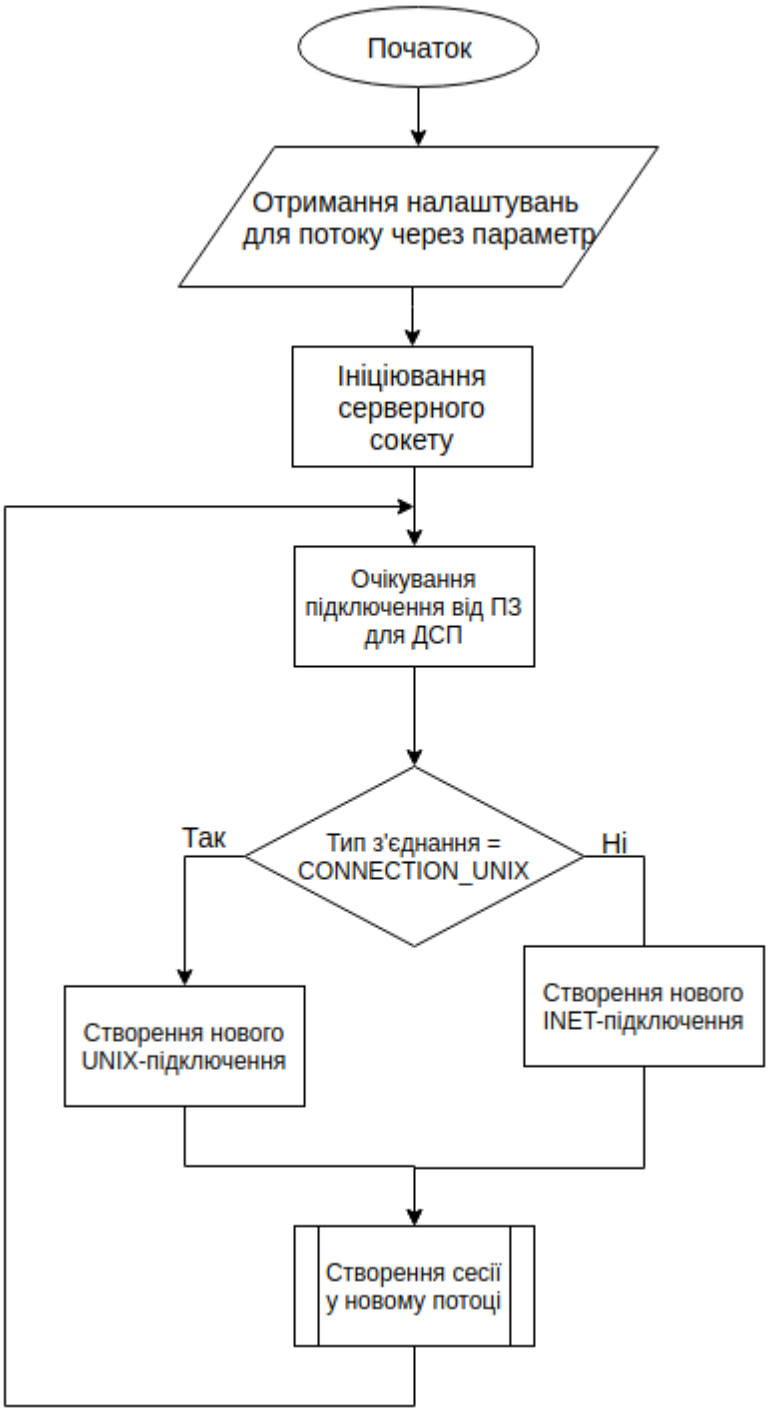


Рисунок 3.3 – Схема алгоритму обробки підключень від АРМ

Сессія обміну даними між ЛПК та демоном зв’язку полягає в отриманні даних від ЛПК, перевірки отриманих даних на цілісність і, в залежності від заголовку повідомлення від ЛПК, виконується або передача отриманих даних до підключень від АРМ, або архівація інформації. Схема алгоритму обміну даними демону зв'язку з ЛПК зображена на рисунку 3.4.

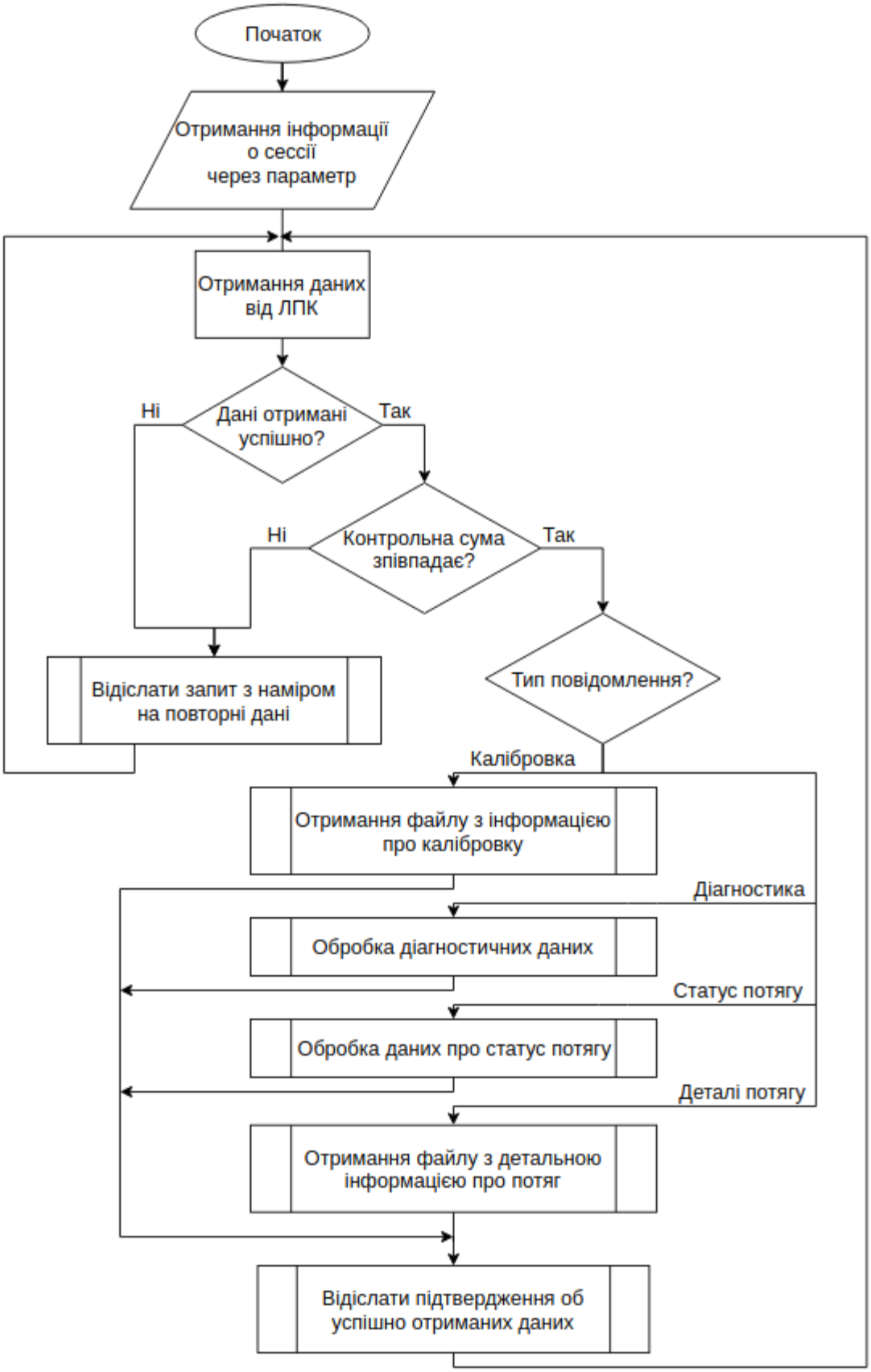


Рисунок 3.4 – Схема алгоритму обміну даними демону зв'язку з ЛПК

Сессія обміну даними між АРМ та демоном зв’язку полягає в відсиланні масиву останніх актуальних даних від підключених ЛПК при першому підключенні додатку АРМ до демону зв’язку та послідовній передачі оновлень до підключених АРМ надалі. Схема сесії обміну даними з АРМ зображена на рисунку 3.5.

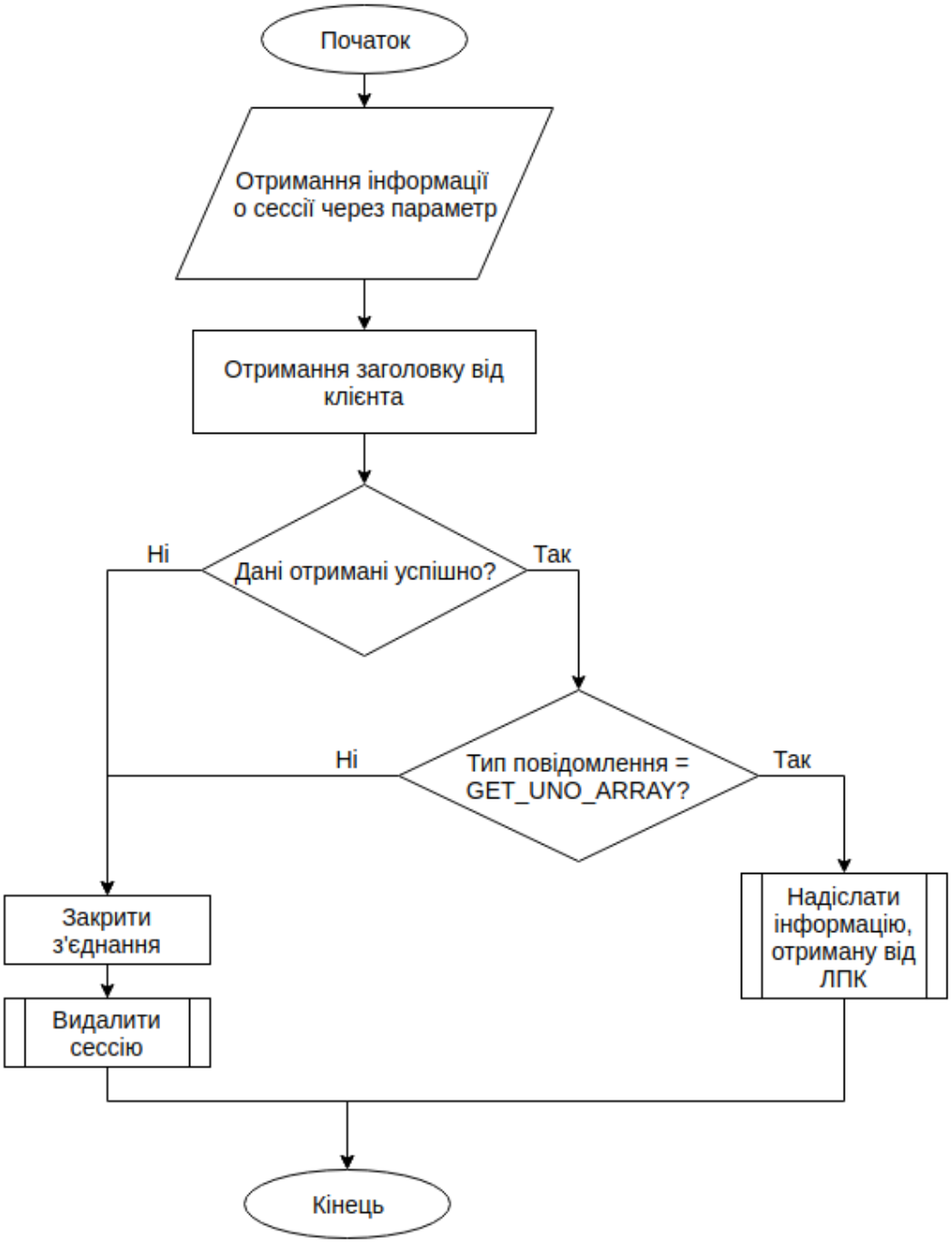


Рисунок 3.5 – Схема алгоритму обміну даними демону зв'язку з АРМ

3.1.1.2 Проектування протоколу зв'язку ЛПК з сервером

Зв'язок між комплектами постового обладнання та сервером відзначається наступними характеристиками:

- Передача даних здійснюється невеликими за розміром пакетами(за виключенням файлів);
- велика частота передачі даних;
- дані являють собою сукупність показників вимірювальних приладів.

Зважаючи на дані умови, до протоколу висуваються наступні вимоги:

- протокол не повинен створювати навантаження на мережу;
- необхідно забезпечити контроль цілістності та упорядкованості даних;
- в протокол необхідно вбудувати можливість повторної передачі даних при необхідності.

Проаналізувавши й узагальнивши цю інформацію, для протоколу передачі даних між сервером ПТО та постовим обладнанням був створений заголовок для кожної одиниці передачі даних, який представлено на таблиці 3.1.

Таблиця 3.1 – Структура заголовка протоколу «ЛПК – сервер»

Байт	0-1	2-3	4-5	6-7	8-9	10-11	12-15
0	Маркер пристрою, **Bif Marker**					ver	time
16	in_idx	out_idx	dtype	dsize	crc16	Дані	
32+	Дані						

Протокол базується поверх TCP, який, в збиток швидкодії, забезпечує вбудовані перевірки на цілісність та упорядкованість даних. Окрім цього, створений протокол має власні перевірки на останнє вірно прийняте повідомлення та перевірку цілістності надісланих даних за допомогою контрольної суми.

Процес зв'язку вібувається за ініціативи постового програмного забезпечення шляхом підключення до серверу з передачею заголовка, поле *in_idx* якого заповняється нулями. Сервер має перевірити заголовок на корректність та, в залежності від поля *dtype* заголовку, обробити запит, після чого сформувати відповідь. Відповідь складається із заголовку, поле *dtype* якого заповняється нулями, поле *in_idx* збільшується на одиницю, а *out_idx* дорівнює значенню *in_idx* отриманого заголовку.

Наявність даних услід за заголовком визначається в залежності від поля *dtype*. Розмір даних(в байтах) вказується в полі *dsize*.

Перевірка цілістності даних(у разі їх наявності) здійснюється шляхом звірки значення поля *crc16* від постового обладнання з фактичним значенням контрольної суми, отриманої у наслідок застосування алгоритму находження контрольної суми CRC16(додаток А).

Підключення між постовим програмним забезпеченням та сервером зберігається до моменту відключення одного з вузлів. Подальшу передачу даних ініціює пост за сценарієм, що був описаний вище.

3.1.1.3 Проектування протоколу зв'язку АРМ з сервером

Необхідність розробки окремого протоколу зв'язку виникла внаслідок різної поведінки та функціоналу ПЗ ЛПК з ПЗ для АРМ.

Взаємодія між сервером ПТО та АРМ була визначена наступними правилами:

- Сервер ПТО надсилає невеликі пакети даних;
- АРМ надсилає запити та підтвердження;

Беручи до уваги встановлені правила, було вирішено створити протокол максимально простим з одним сценарієм взаємодії. Тим не менш, протокол має потенціал до розширення.

Структура заголовку протоколу «сервер – АРМ» представлена на таблиці 3.2.

Таблиця 3.2 – Структура заголовку протоколу «сервер – АРМ»

Байт	0	2	3	4-5	6-9	10-13	14-15
0	mtype	ver	cmd	dsize	val	val2	Дані
16+	Дані						

Протокол побудований поверх протоколу TCP, який здійснює перевірку цілістності та упорядкованості даних. Крім цього, протокол «сервер – АРМ» має власні перевірки на цілісність та упорядкованість.

Взаємодія ініціюється додатоком для АРМ шляхом передачі заголовка, поле mtype якого заповняється значенням константи GET_UNO_ARRAY, поле ver заповнюється константою значенням актуальної версії протоколу. Всі інші поля попереднью заповнюються нулями.

Сервер повинен перевірити заголовок на корректність та у відповідь відіслати отриманний заголовок, модифікований наступним чином: поле mtype заповнюється значенням константи UNO_ARRAY; in_idx нарощується на одиницю; поле out_idx заповнюється попереднім значенням in_idx; поле val заповнюється значенням, яке дорівнює кількості активних підключень ЛПК. У слід за переданим заголовком, сервер повинен передати стільки блоків даних, скільки має активних підключень. Блок даних має наступну стрктуру: перші 2 біти дорівнюють IP-адресі ЛПК, наступна кількість біт дорівнюється розміром даних діагностики та стану потягу(обчислюється на стадії компіляції).

Додаток, здійснивши перевірки на цілісність даних, відправляє заголовок зі значенням m_type, дорівнюючим константі SUCCESS, попереднью встановивши нові in_idx та out_idx.

3.1.1.4 Структура демону зв'язку

Перелік модулів демону зв'язку серверу ПТО із вказанням їх призначення наведено у таблиці 3.3. Виходячи зі специфіки виконуваних задач демону, було прийнято рішення написати програмні модулі на мові програмування С.

Таблиця 3.3 – Склад модулів демону зв'язку серверу ПТО

Назва файлу	Функції та призначення
akrobsrv.c	Головна програма, ініціалізація файлу конфігурації та файлу логування. Робота з підключеннями від ЛПК та АРМ. Демонізація процесу.
akrobtypes.h iface_ptokmp.h	Бібліотеки, що представляють типи даних та структури ЛПК та заголовки протоколу «ЛПК – сервер»
log.h	Логування важливих подій
ini.h	Інтерфейс для доступу файлів конфігурації
guitypes.h	Бібліотека з типами даних, що представляють заголовки та структури для взаємодії по протоколу «сервер – АРМ»
listen.h	Обробка підключень від ЛПК та АРМ з виділенням окремих потоків

3.1.1.5 Склад структур даних демону зв'язку

Перелік основних розроблених структур та їх призначення наведено у таблиці 3.4.

Таблица 3.4 – Структури даних демону зв'язку

Назва структури	Призначення
UnoHeader	Відображення заголовку протоколу «ЛПК – сервер»
LpkDiagnostic	Діагностичні дані від польового обладнання
TrainControlState	Дані по останньому потягу від польового обладнання
GuiHeader	Відображення заголовку протоколу «АРМ – сервер»
UnoData	Відображення діагностичних даних і даних по останному потягу, зафіксованих конкретним ЛПК

Продовження таблиці 3.4

UnoStateData	Відображення даних по останньому потягу, зафіксованих конкретним ЛПК
UnoDiagnosticData	Відображення діагностичних даних, зафіксованих конкретним ЛПК
LpkConnection	Відображення даних з’єднання з ЛПК
LpkConnectionList	Організація списку з’єднань з ЛПК
GuiConnection	Відображення даних з’єднання з АРМ
GuiConnectionList	Організація списку з’єднань з АРМ

3.1.2 Розробка демону зв’язку

3.1.2.1 Склад основних функцій демону зв’язку

Перелік основних розроблених функцій думону зв’язку серверу наведений у таблиці 3.5

Таблиця 3.5 – Перелік основних функцій демону зв’язку.

Назва функції	Опис
void* listen_lpk(void *)	Прослуховування порту для підключеннь ЛПК та створення сокет-з’єднань
void* listen_gui(void *)	Прослуховування порту для підключеннь АРМ та створення сокет-з’єднань
void* session_lpk(void *)	Сессія взаємодії з ЛПК по протоколу «сервер – ЛПК»
Void* session_gui(void*)	Сессія взаємодії з додатком для АРМ з сервером по протоколу «сервер – АРМ»
uint16_t crc16_checksum(char*, size_t)	Визначення контрольної суми алгоритмом циклічного надлишкового коду
void send_lpkd_update(uint32_t, struct LpkDiagnostic)	Надсилання даних діагностики ЛПК додаткам для АРМ
void send_tcs_update(uint32_t, struct TrainControlState)	Надсилання даних статусу контролю потягу додаткам для АРМ

3.1.2.2 Реалізація алгоритмів роботи демону зв’язку

Під час запуску програми демона зв’язку відбувається запуск нових потоків з функціями lisen_lpk та listen_gui, які розташовані у модулі listen.h(Додаток А).

Під час отримання сигналів підключення від ЛПК та АРМ, демон зв’язку формує нові з’єднання, та створює нові потоки з функціями session_lpk() та session_gui() відповідно. Лістинг функцій розташований у модулі akrobsrv.c(Додаток Б).

3.2 Проектування та розробка демону архіву

3.2.1 Проектування демону архіву

Проектування програми демону-архіву проводилося в два етапи:

1. Розробка алгоритму роботи;
2. проектування програмних модулів.

3.2.1.1 Розробка алгоритму роботи демону архіву

Основною задачею демону архіву є контроль за станом директорій з файлами інформації від ЛПК. Програма повинна працювати незалежно від демону зв'язку та ніяк з ним не взаємодіяти.

Алгоритм роботи демону архіву полягає в переході до фонового режиму роботи, перевірці наповненості архіву станційного обладнання з файлами від ЛПК на предмет перевищення допустимих розмірів директорій. Поведінка демону архіву стосовної кожної частини архіву визначається в файлі налаштування. Частота перевірки архіву демоном архіву визначається файлом налаштувань. Алгоритм роботи демону архіву зображений на рисунку 3.6.

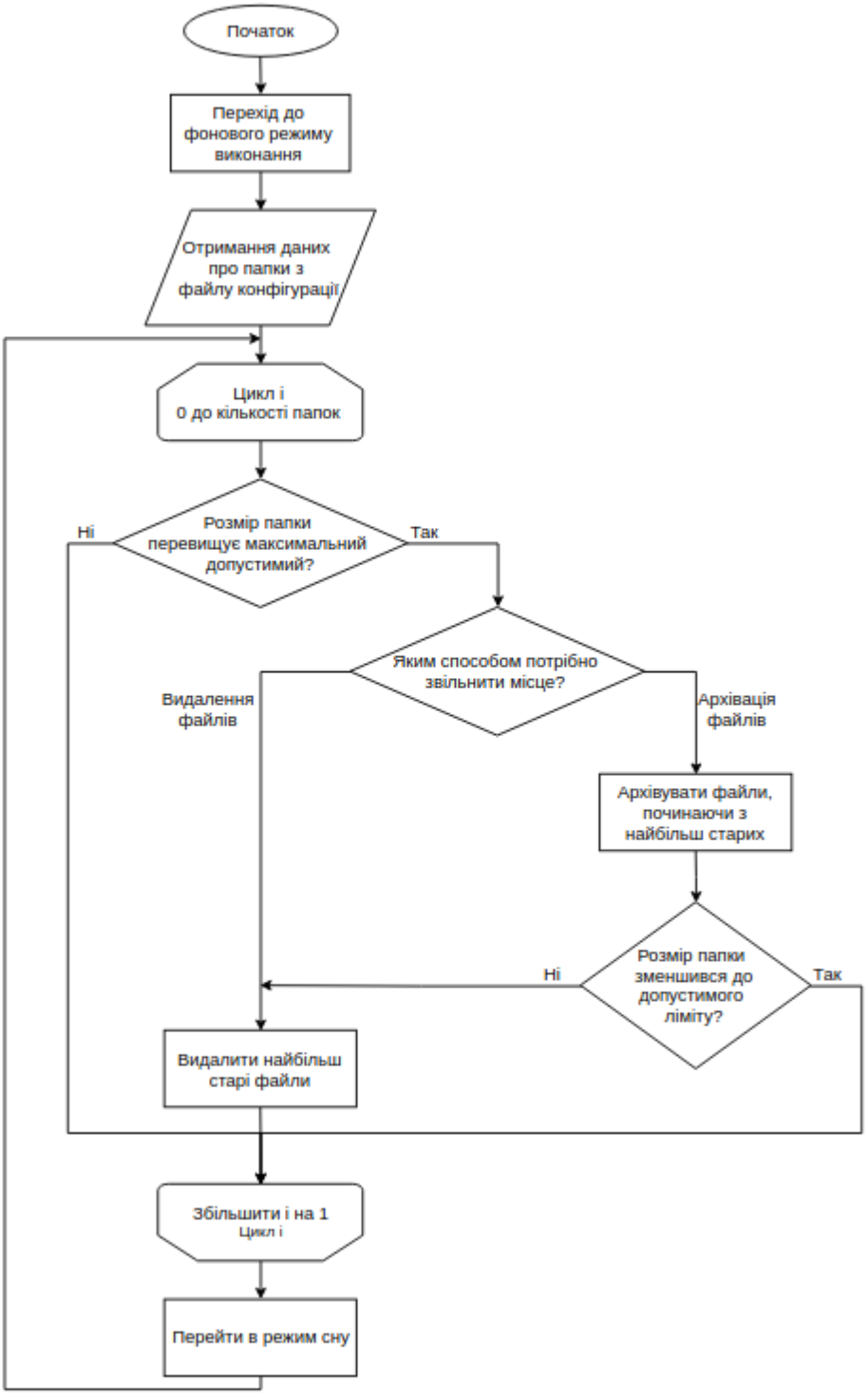


Рисунок 3.6 – схема алгоритму роботи демону архіву серверу ПТО

3.2.1.2 Проектування програмних модулів демону архіву

Демон архіву має виконувати фіксований обсяг задач між заданими проміжками часу. Демон архіву представляє собою компонент ПЗ серверу ПТО. Тому було принято рішення розробити программу, виконуючу роль контролю за станом директорій з файлами від ЛПК, за допомогою мови програмування Python версії 3. Програма буде являти собою невеликий скрипт з виконанням у фоновому режимі. Список модулів демону архіву приведений нижче на таблиці 3.6.

Таблиця 3.6 – Склад модулів демону архіву серверу ПТО

Назва файлу	Функції та призначення
archd.py	Точка входу в програму, виконання функцій демону архіву
cfg.py	Файл конфігурації демону архіву

3.2.2 Розробка демону архіву

3.2.2.1 Склад основних функцій демону архіву

Перелік основних розроблених функцій для демону архіву наведений нижче у таблиці 3.7

Таблиця 3.7 – Перелік та опис основних функцій демону-архіву

Сигнатура функції	Опис
float dirsz(path, recursive)	Визначення розміру директорії у мегабайтах
list dir_files_by_date(dir_path)	Отримання списку з іменами файлів у директорії, сортованого за актуальністю
void dir_compress_oldest(path, limit)	Архівування та видалення найменш актуальних даних при необхідності зменшити розмір директорії.
void dir_delete_oldest(path, limit)	Видалення найменш актуальних даних при необхідності зменшити розмір директорії
void main()	Запуск алгоритму роботи

3.2.2.2 Програмна реалізація демону архіву

Під час запуску програми здійснюється перехід до фонового режиму роботи. демону архіву проводиться імпортування модулю *logging* стандартної бібліотеки та ініціація логування. Імпортується файл конфігурації модулю cfg.py(Додаток В) для визначення налаштувань демона.

Реалізація основного алгоритму роботи наведена у функції main, реалізація якої навадена нижче.

```
def main(): while True:
      for key, value in cfg.folders.items():
          max_size = value.get('max_size', cfg.folder_max_size)
          if dirsz(value['path']) > max_size:
              cleanup_action = value.get('cleanup_action',
cfg.cleanup_action)
              if cleanup_action == cfg.ACTION_COMPRESS_OLDEST:
                  dir_compress_oldest(value['path'], max_size)
              elif cleanup_action == cfg.ACTION_DELETE_OLDEST:
                  dir_delete_oldest(value['path'], max_size)
      time.sleep(cfg.freq_seconds)
```

Очищення директорії від застарілих файлів реалізована в функції dir_delete_oldest.

Архівація застарілих файлів реалізована у функції dir_compress_oldest.

Отримання отсортованого по застарілості списку файлів реалізовано в функції dir_files_by_date().

Повний лістинг демону архіву наведено у додатку Г.

3.3 Проектування та розробка додатку для АРМ

В цілях перевірки та відладки протоколу зв’язку серверу з АРМ було прийняте рішення в проектуванні та подальшій реалізації прототипу додатку для АРМ з графічним інтерфейсом.

3.3.1 Проектування додатку для АРМ

Проектування додатку для АРМ було проведено у декілька етапів:

1. Розробка алгоритму роботи;
2. Визначення структур та модулів, застосовуваних сумісно з демоном зв'язку;
3. Визначення модулів для реалізації.

3.3.1.1 Розробка алгоритму роботи

Алгоритм роботи додатку для АРМ полягає у створенні віджету, відповідного за відображення списку підключених до ЛПК, який, в свою чергу, відповідний за отримання актуальних даних від ЛПК та керує віджетами, відповідними за показ актуальних даних кожного постового обладнання.

Программа повинна мати наявність інтерфейсу користувача. Було прийнято рішення розробити додаток на основі фреймворку Qt з використанням мови програмування C++, яка надає можливість перевикористання деяких модулів демону зв'язку.

Додаток розділяється на декілька модулів:

- Модуль інтерфейсу, відповідного за встановлення та підтримку зв’язку з сервером;
- Модуль відображення списку підключень ЛПК та відповідний за отримання інформації від інтерфейсу;
- Модуль, відповідний за відображення інформації про конкретний ЛПК та обробку подій користувача.

Алгоритм роботи віджету контролю постових віджетів полягає в першочерговому створенні постових віджетів, налаштування яких містяться

в файлі. Під час отримання сигналу від класу, що відповідає за зв’язок з сервером ПТО, віджет контролю віджетів ЛПК надсилає сигнал з оновленнями конкретному віджету ЛПК. Алгоритм роботи віджету контролю постових віджетів відображений на рисунку 3.7.

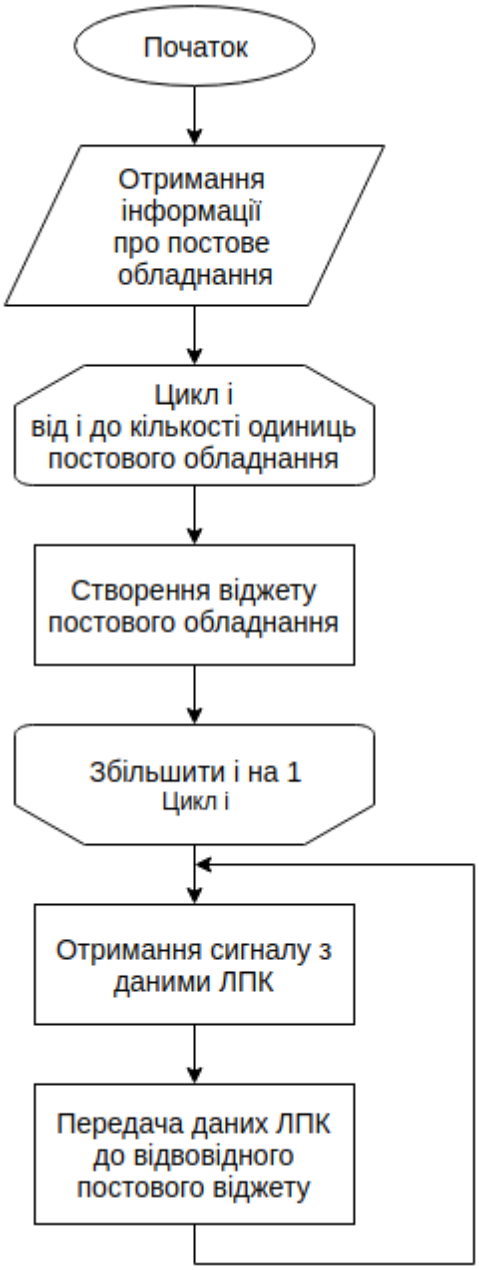


Рисунок 3.7 – Схема алгоритму роботи віджету контролю постових віджетів

Інтерфейс зв’язку з сервером, в залежності від налаштувань, підключається до серверу з різним типом сокетного з’єднання. Після першого підключення до сервера, інтерфейс передає віджету контролю за

відметами ЛПК актуальну інформацію про підключене до серверу постове обладнання шляхом сигналу. Далі інтерфейс прослуховує сервер на предмет оновлень та відсилає кожне отримане оновлення до віджету контролю за відметами ЛПК. Алгоритм наведений на рисунку 3.8.

Рисунок 3.8 – Схема алгоритму роботи віджету інтерфейсу зв’язку з сервером

3.3.1.2 Визначення сумісності з модулями демона зв’язку

Мова програмування C++ являється надмножиною над мовою C, за допомогою якої реалізується демон зв’язку серверу. Наявність протоколу зв’язку АРМ з сервером, написанному на мові C, а також необхідність у використанні діагностичних даних від ЛПК та даних по стану потязів дозволяє перевикористати структури даних, які відповідають за реалізацію представлення вищенаведенної інформації. Список структур даних демону зв’язку серверу, які знайшли своє використання в додатку для АРМ, відображений в таблиці 3.8.

Таблиця 3.8 – Структури даних із проекту серверу в додатку для АРМ

Назва структури	Призначення
LpkDiagnostic	Діагностичні дані від польового обладнання
TrainControlState	Дані по останньому потягу від польового обладнання
GuiHeader	Відображення заголовку протоколу «АРМ – сервер»
UnoData	Відображення діагностичних даних і даних по останному потягу, зафіксованих конкретним ЛПК
UnoStateData	Відображення даних по останньому потягу, зафіксованих конкретним ЛПК
UnoDiagnosticData	Відображення діагностичних даних, зафіксованих конкретним ЛПК

3.3.1.3 Визначення модулів для реалізації

Було прийнято рішення розподілити кожен класс в окремий модуль(за винятком точки входу в програму) з метою забезпечити «гнучкість» проекту з можливістю легкого розширення за необхідністю. Всі модулі написані на мові C++(за винятком модулів, які містять структури даних демону зв’язку серверу ПТО) з використанням фреймворку Qt.

Список модулів додатку для АРМ наведений нижче на таблиці 3.9.

Таблиця 3.9 – Склад модулів додатку для АРМ

Назва файлу	Функції та призначення
main.cpp	Точка входу в програму
Mainwindow.h Mainwindow.cpp	Віджет головного вікна додатка
Lpkmanager.h lpkmanager.cpp	Віджет зв’язку з інтерфейсом та контролю за віджетами, представляючими конкретний ЛПК
Lpkwidget.h lpkwidget.cpp	Віджет відображення інформації про конкретний ЛПК
Ifaceserver.h ifaceserver.cpp	Інтерфейс для взаємодії з демоном зв’язку серверу ПТО

3.3.2 Розробка додатку для АРМ

3.3.2.1 Склад основних функцій додатку для АРМ

Перелік основних розроблених функцій для додатку для АРМ наведений нижче у таблиці 3.10

Таблиця 3.10 – Перелік та опис основних функцій додатку для АРМ

Назва функції	Опис
void IfaceServer::start()	Ініціація підключення до серверу та початок його прослуховування на предмет оновлень
void LpkManager::onStateUpdate(struct UnoStateData *)	Функція, що викликається під час отримання оновлення конрольного стану потягу від ЛПК
Void LpkManager::onDiagnosticUpdate(struct UnoDiagnosticData *)	Функція, що викликається під час отримання оновлення даних діагностики від ЛПК

3.3.2.2 Програмна реалізація додатку для АРМ

Під час запуску додатку, першим чином створюється головне вікно, на якому розміщується віджет контролю за відображеннями інформації від ЛПК LpkManager.

Створюється екземпляр класу IfaceServer, який зв’язується з функціями менеджеру віджетів.

```
    connect(iface, SIGNAL(sendStateData(UnoStateData*)), manager,
SLOT(onStateData(UnoStateData*)));
    connect(iface, SIGNAL(sendDiagnosticData(UnoDiagnosticData*)), manager,
SLOT(onDiagnosticData(UnoDiagnosticData*)));
```

Менеджер віджетів має функції обробки оновлень від інтерфейсу IfaceServer::startPolling(). Лістинг початкового коду класу IfaceServer наведений у додатку Ґ.

Вигляд графічного інтерфейсу програми-додатку представлений нижче (рис. 3.9).

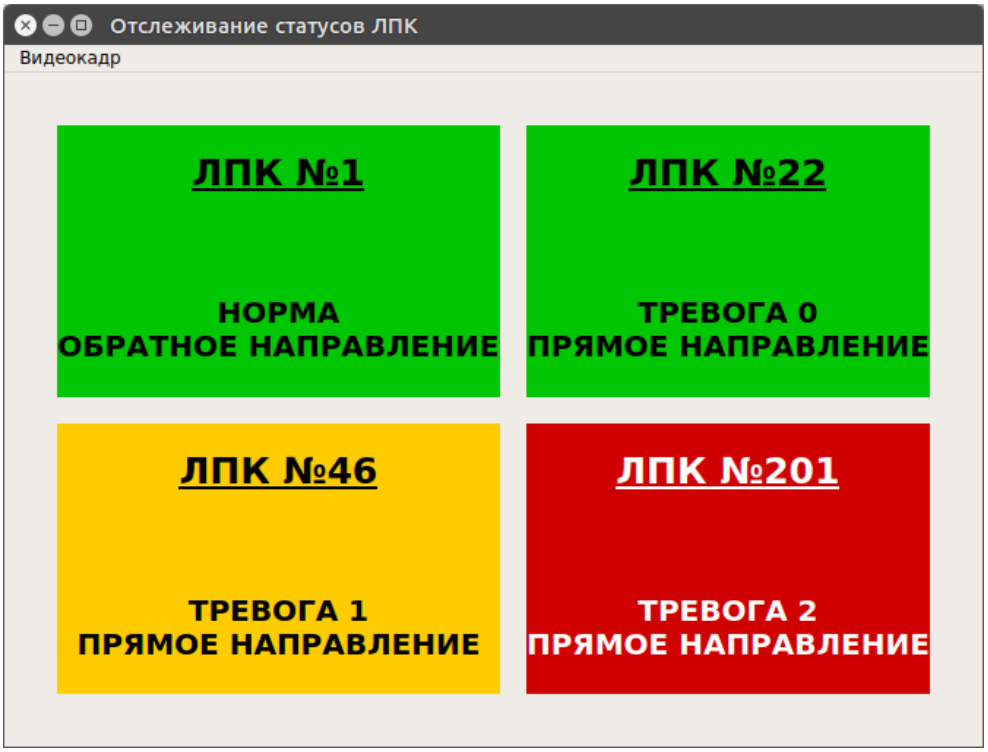


Рисунок 3.9 – Вигляд головного віджету програми-додатка

Висновок до розділу 3

У цьому розділі було розроблено алгоритми, модулі, структури даних для комплекту ПЗ серверу станційного обладнання АКРО-Б. Були розроблені й описані протоколи зв'язку серверу з АРМ та постовим обладнанням. Реалізовані програми комплекту серверу станційного обладнання АКРО-Б.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

В даному розділі проведено аналіз потенційних небезпечних та шкідливих виробничих факторів, причин пожеж. Розглянуті заходи, які дозволяють забезпечити гігієну праці і виробничу санітарію. На підставі аналізу розроблені заходи з техніки безпеки та рекомендації з пожежної профілактики.

Завданням даної роботи бакалавра було проектування комплекту програм станційного обладнання АКРО-Б, і як результат був спроектований комплект програм станційного обладнання. Так як в процесі проектування використовувався персональний комп'ютер, то аналіз потенційно небезпечних і шкідливих виробничих чинників виконується для персонального комп'ютера на якому буде розроблятися та використовуватися розроблена система.

4.1 Аналіз стану умов праці

4.1.1 Вимоги до приміщення

Таблиця 4.1 - Розміри приміщення.

Найменування	Значення
Довжина, м	18
Ширина, м	6
Висота, м	3,5
Площа, $м^2$	108
Об'єм, $м^3$	378
Кількість робочих місць	8

Згідно з ДСН 3.3.6.042-99 [18] розмір площі для одного робочого місця оператора персонального комп’ютера має бути не менше 6 кв. м, а об’єм — не менше 20 куб. м. Отже, дане приміщення цілком відповідає зазначеним нормам.

Для забезпечення потрібного рівного освітленості кімната має вікна та систему загального рівномірного освітлення, що встановлена на стелі. Для дотримання вимог пожежної безпеки встановлено порошковий вогнегасник та систему автоматичної пожежної сигналізації.

4.1.2 Вимоги до організації місця праці

При порівнянні відповідності характеристик робочого місця нормативним основні вимоги до організації робочого місця за ДСанПіН 3.3.2.007-98 [19] (табл. 4.2) і відповідними фактичними значеннями для робочого місця, констатуємо повну відповідність.

Таблиця 4.2 - Характеристики робочого місця

Найменування параметра	Фактичне Значення	Нормативне Значення
Висота робочої поверхні, мм	700	680 ÷ 800
Висота простору для ніг, мм	650	не менше 600
Ширина простору для ніг, мм	540	не менше 500
Глибина простору для ніг, мм	660	не менше 650
Висота поверхні сидіння, мм	420	400 ÷ 500
Ширина сидіння, мм	410	не менше 400
Глибина сидіння, мм	420	не менше 400
Висота поверхні спинки, мм	500	не менше 300
Ширина опорної поверхні спинки, мм	400	не менше 380
Радіус кривини спинки в горизонтальній площині, мм	400	400
Відстань від очей до екрану дисплея, мм	750	700 ÷ 800

4.2 Виробнича санітарія

На підставі аналізу небезпечних та шкідливих факторів при виробництві (експлуатації), пожежної безпеки можуть бути надалі вирішені питання необхідності забезпечення працюючих достатньою кількістю освітлення, вентиляції повітря, організації заземлення, тощо.

4.2.1 Аналіз небезпечних та шкідливих факторів при виробництві (експлуатації) виробу

Аналіз небезпечних та шкідливих виробничих факторів виконується у табличній формі (табл. 4.3). Роботу, пов'язану з ЕОП з ВДТ, у тому числі на тих, які мають робочі місця, обладнані ЕОМ з ВДТ і ПП, виконують із забезпеченням виконання НПАОП 0.00-1.28-10 [22], яке встановлюють вимоги безпеки до обладнання робочих місць, до роботи із застосуванням ЕОМ з ВДТ і ПП. Переважно роботи за проектами виконують у кабінетах чи інших приміщеннях, де використовують різноманітне електрообладнання, зокрема персональні комп’ютери (ПК) та периферійні пристрої. Основними робочими характеристиками персонального комп'ютера є:

– робоча напруга U=+220В +-5%;

– робочий струм I=2А;

– споживана потужність P=350 Вт.

Таблиця 4.3 – Аналіз небезпечних і шкідливих виробничих факторів

Небезпечні і шкідливі виробничі фактори	Джерела факторів (види робіт)	Кількісна Оцінка	Нормативні Документи
1	2	3	4
Фізичні:			
підвищена або	-//-	3	[18]

знижена вологість повітря			
підвищений рівень напруги електричної мережі, замикання якої може відбутися через тіло людини	-//-	3	[20] [21]
Психофізіологічні:			
1	2	3	4
нервово-психічна перевантаження (розумове, перенапруження аналізаторів-зорових)	- формулювання теми; - пошук інформацію про предметну область; - проектування структур та алгоритмів; - виконання роботи; - оформлення записки.	4	[21] [22]
фізичні (статичне - сидіння)	порушення умов організації робочого часу (безперервна робота)	2	[19] [22]

Робочі місця в обов'язковому порядку повинні відповідати вимогам до санітарних правил і норм роботи з візуальними дисплейними терміналами електронно-обчислювальних машин, що затверджені постановою Головного державного санітарного лікаря України від 10.12.98 N 7 [19].

4.2.2 Пожежна безпека

Висока щільність елементів в електронних схемах призводить до значного підвищення температури окремих вузлів (80…100 ℃). При проходженні електричного струму по провідниках і деталей виділяється тепло, що в умовах їх високої щільності може привести до перегріву, і може

служити причиною запалювання ізоляційних матеріалів. Слабкий опір ізоляційних матеріалів дії температури може викликати порушення ізоляції і привести до короткого замикання між струмоведучими частинами обладнання (шини, електроди).

Заземлені конструкції, що знаходяться в приміщеннях, де розміщені робочі місця (батареї опалення, водопровідні труби, кабелі із заземленим відкритим екраном), надійно захищені діелектричними щитками та/або сітками з метою недопущення потрапляння працівника під напругу.

В приміщенні наявна затверджена «План-схема евакуації з кабінету (приміщення)».

Горючими матеріалами в приміщенні, де розташовані ЕОМ, є:

1) поліамід - матеріал корпусу мікросхем, горюча речовина, температура самозаймання 420℃;

2) полівінілхлорид - ізоляційний матеріал, горюча речовина, температура запалювання 335℃, температура самозаймання 530℃;

3) склотекстоліт ДЦ - матеріал друкарських плат, важкогорючий матеріал, показник горючості 1.74, не схильний до температурного самозаймання;

4) пластикат кабельний №489 - матеріал ізоляції кабелів, горючий матеріал, показник горючості більше 2.1;

5) деревина - будівельний і обробний матеріал, з якого виготовлені меблі, горючий матеріал, показник горючості більше 2.1, температура запалювання 255℃, температура самозаймання 399℃.

Простори усередині приміщень в межах, яких можуть утворюватися або знаходиться пожежонебезпечні речовини і матеріали відповідно до НАПБ Б.03.002-2007 [23] відносяться до пожежонебезпечної зони класу П-IIа. Це обумовлено тим, що в приміщенні знаходяться тверді горючі та важкозаймисті речовини та матеріали. Приміщенню, у якому розташоване робоче місце, присвоюється II ступень вогнестійкості.

Причинами можливого загоряння і пожежі можуть бути:

1) несправність електроустановки;

2) конструктивні недоліки устаткування;

3) коротке замикання в електричних мережах;

4) запалювання горючих матеріалів, що знаходяться в безпосередній близькості від електроустановки.

Продуктами згорання, що виділяються на пожежі, є: окис вуглецю; сірчистий газ; окис азоту; синильна кислота; акромін; фосген; хлор і ін. При горінні пластмас, окрім звичних продуктів згорання, виділяються різні продукти термічного розкладання: хлорангідридні кислоти, формальдегіди, хлористий водень, фосген, синильна кислота, аміак, фенол, ацетон, стирол [24].

4.2.3 Електробезпека

Виконуються наступні вимоги електробезпеки: ПК, периферійні пристрої та устаткування для обслуговування, електропроводи і кабелі за виконанням та ступенем захисту відповідають класу зони за ПУЕ (правила улаштування електроустановок), мають апаратуру захисту від струму короткого замикання та інших аварійних режимів. Лінія електромережі для живлення ПК, периферійних пристроїв і устаткування для обслуговування, виконана як окрема групова три провідна мережа, шляхом прокладання фазового, нульового робочого та нульового захисного провідників. Нульовий захисний провідник використовується для заземлення (занулення) електроприймачів. Штепсельні з'єднання та електророзетки крім контактів фазового та нульового робочого провідників мають спеціальні контакти для підключення нульового захисного провідника. Електромережа штепсельних розеток для живлення персональних ПК, укладено по підлозі поруч зі стінами відповідно до затвердженого плану розміщення обладнання та

технічних характеристик обладнання. Металеві труби та гнучкі металеві рукави заземлені. Захисне заземлення включає в себе заземлюючих пристроїв і провідник, який з'єднує заземлюючий пристрій з обладнанням, яке заземлюється - заземлюючий провідник.

4.3 Гігієнічні вимоги до параметрів виробничого середовища

4.3.1 Мікроклімат

Мікроклімат робочих приміщень - це клімат внутрішнього середовища цих приміщень, що визначається діючої на організм людини з'єднанням температури, вологості, швидкості переміщення повітря. В даному приміщенні проводяться роботи, що виконуються сидячи і не потребують динамічного фізичного напруження, то для нього відповідає категорія робіт 1а. Отже оптимальні значення для температури, відносної вологості й рухливості повітря для зазначеного робочого місця відповідають ДСН 3.3.6.042-99 [18] і наведені в табл. 4.4:

Таблиця 4.4 – Норми мікроклімату робочої зони об’єкту

Період Року	Категорія Робіт	Температура C^0	Відносна вологість %	Швидкість руху повітря, м/с
Холодна	Легка-1а	22-24	40-60	0,1
Тепла	Легка-1а	23-25	40-60	0,1

4.3.2 Освітлення

Для виробничих та адміністративних приміщень світловий коефіцієнт приймається не менше -1/8, в побутових - 1/10:

$$S_b = \left(\frac{1}{5}\Big/\frac{1}{10}\right) * S_n \qquad (4.1)$$

де S_b – площа віконних прорізів, м^2;

S_n – площа підлоги, м^2 .

$S_n = a \cdot b = 18 \cdot 6 = 108$ м^2 ,

$S = 1/10 \cdot 25 = 1{,}232$ м^2 .

Приймаємо 1 вікно площею S=1,6 м^2.

Світильники загального освітлення розташовуються над робочими поверхнями в рівномірно-прямокутному порядку. Для організації освітлення в темний час доби передбачається обладнати приміщення, довжина якого складає 18 м, ширина 6 м, світильниками ЛПО2П, оснащеними лампою типа ЛБ (одна - 80 Вт) з світловим потоком 5400 лм. Розрахунок штучного освітлення виробляється по коефіцієнтах використання світлового потоку, яким визначається потік, необхідний для створення заданої освітленості при загальному рівномірному освітленні. Розрахунок кількості світильників n виробляється по формулі (4.2):

$$n = \frac{E * S * Z * K}{F * U * M} \qquad (4.2)$$

де E - нормована освітленість робочої поверхні, визначається нормами – 300 лк;

S - освітлювана площа, м^2 ; S = 108 м^2 ;

Z - поправочний коефіцієнт світильника (Z = 1,15 для ламп розжарювання та ДРЛ; Z = 1,1 для люмінесцентних ламп) приймаємо рівним 1,1;

K - коефіцієнт запасу, що враховує зниження освітленості в процесі експлуатації – 1,5;

U - коефіцієнт використання, залежний від типу світильника, показника індексу приміщення і т.п.

- 0,575 M - число люмінесцентних ламп в світильнику - 1;

F - світловий потік лампи - 5400лм (для ЛБ-80).

Підставивши числові значення у формулу (4.2), отримуємо:

$$n = \frac{300 * 108 * 1,15 * 1,5}{5400 * 0,575 * 1} = 18,0$$

Приймаємо освітлювальну установку, яка складається з 18 світильників, оснащених лампами типа ЛБ (одна - 80 Вт) зі світловим потоком 5400 лм.

4.4 Вентилювання

У приміщенні, де знаходяться ЕОМ, повітрообмін реалізується за допомогою природної організованої вентиляції (вентиляційні шахти), тобто при V приміщення > 40 м^3 на одного працюючого допускається природна вентиляція. Цей метод забезпечує приток потрібної кількості свіжого повітря, що визначається в СНіП. Також має здійснюватися провітрювання приміщення, в залежності від погодних умов, тривалість повинна бути не менше 10 хв. Найкращий обмін повітря здійснюється при наскрізному провітрюванні.

4.5 Заходи з організації виробничого середовища та попередження виникнення надзвичайних ситуацій

1) Заходи безпеки під час експлуатації персонального комп’ютера та периферійних пристроїв передбачають:

- правильне організування місця праці та дотримання оптимальних режимів праці та відпочинку під час роботи з ПК;
- експлуатацію сертифікованого обладнання;
- дотримання заходів електробезпеки;

- забезпечення оптимальних параметрів мікроклімату;

- забезпечення раціонального освітлення місця праці (освітленість робочого місця не перевищувала 2/3 нормальної освітленості приміщення);

- облаштовуючи приміщення для роботи з ПК, потрібно передбачити припливно-витяжну вентиляцію або кондиціювання повітря:

а) якщо об'єм приміщення 20 м^3 , то потрібно подати не менш як 30 м^3 /год повітря;

б) якщо об'єм приміщення у межах від 20 до 40 м^3, то потрібно подати не менш як 20 м^3/год повітря;

в) якщо об'єм приміщення становить понад 40 м^3, допускається природна вентиляція, у випадку, коли немає виділення шкідливих речовин.

2) Заходи безпеки під час експлуатації інших електричних приладів передбачають дотримання таких правил:

- постійно стежити за справним станом електромережі;

- постійно стежити за справністю ізоляції електромережі та мережевих кабелів, не допускаючи їхньої експлуатації з пошкодженою ізоляцією;

- не тягнути за мережевий кабель, щоб витягти вилку з розетки;

- не закривати меблями, різноманітним інвентарем вимикачі, штепсельні розетки;

- не підключати одночасно декілька потужних електропристроїв до однієї розетки, що може викликати надмірне нагрівання провідників, руйнування їхньої ізоляції, розплавлення і загоряння полімерних матеріалів;

- не залишати включені електроприлади без нагляду;

4.5.1 Розрахунок захисного заземлення (забезпечення електробезпеки будівлі).

Згідно з класифікацією приміщень за ступенем небезпеки ураження електричним струмом [25], приміщення в якому проводяться всі роботи

відноситься до першого класу (без підвищеної небезпеки). Під час роботи використовуються електроустановки з напругою живлення 36 В, 220 В, та 360 В. Опір контуру заземлення повинен мати не більше 4 Ом.

Послідовність розрахунку.

1) Визначається необхідний опір штучних заземлювачів $R_{шт.з}$.:

$$R_{шт.з} = \frac{R_д * R_{пр.з}}{R_{пр.з} - R_д} \quad (4.3)$$

де $R_{пр.з}$. - опір природних заземлювачів;

$R_д$ - допустимий опір заземлення.

Якщо природні заземлювачі відсутні, то $R_{шт.з}$.=$R_д$.

Підставивши числові значення у формулу (А.3), отримуємо:

$$R_{шт.з} = \frac{4 * 40}{40 - 4} \approx 4Ом$$

2) Опір заземлення в значній мірі залежить від питомого опору ґрунту ρ, Ом·м. Приблизне значення питомого опору глини приймаємо ρ=40 Ом*м (табличне значення).

3) Розрахунковий питомий опір ґрунту, Ррозр, Ом·м, визначається відповідно для вертикальних заземлювачів ρрозр.в, і горизонтальних Ррозр.г, Ом·м за формулою:

$$P_{розр.} = \Psi * p \quad (4.4)$$

де ψ - коефіцієнт сезонності для вертикальних заземлювачів I кліматичної зони з нормальною вологістю землі, приймається для вертикальних заземлювачів $P_{розр.в}$=1,7 і горизонтальних $P_{розр.в}$ =5,5 Ом·м

$P_{озр.в}$ = 1, 7 * 40 = 68 Ом/м

$P_{озр.г}$ = ·5,5 * 40 = 220 Ом/м

4) Розраховується опір розтікання струму вертикального заземлювача Rв, Ом, за (4.5).

$$R_В = \frac{P_{розр.в}}{2 * \pi * 1_В} * \left(\ln \frac{2 * 1_В}{d_{СТ}} + \frac{1}{2} * \ln \frac{4 * t + 1_В}{4 * t - 1_В}\right) \quad (4.5)$$

де $l_в$ - довжина вертикального заземлювача (для труб - 2−3 м; $l_в$=3 м);

$d_{ст}$ - діаметр стержня (для труб - 0,03−0,05 м; $d_{ст}$=0,05 м);

t - відстань від поверхні землі до середини заземлювача, яка визначається за ф. (4.6):

$$t = h_в + \frac{1_в}{2} \quad (4.6)$$

де $h_в$ - глибина закладання вертикальних заземлювачів (0,8 м); тоді $t = 0.8 + \frac{3}{2} = 2{,}3м$

$$R_В = \frac{68}{2 * \pi * 3} * \left(\ln\frac{2 * 3}{0{,}05} + \frac{1}{2} * \ln\frac{4 * 2{,}3 + 3}{4 * 2{,}3 - 3}\right) = 18{,}5Ом$$

5) Визначається теоретична кількість вертикальних заземлювачів n штук, без урахування коефіцієнта використання ηв:

$$n = \frac{2 * R_В}{R_д} = \frac{2 * 18{,}5}{4} = 9{,}25 \quad (4.7)$$

6) Визначається необхідна кількість вертикальних заземлювачів з урахуванням коефіцієнта використання $n_в$, шт:

$$n_В = \frac{2 * R_В}{R_д * \eta_В} = \frac{2 * 18{,}5}{4 * 0{,}57} = 16{,}2 \approx 16 \quad (4.8)$$

7) Визначається довжина з'єднувальної стрічки горизонтального заземлювача lc, м:

$$l_c = 1{,}05 * L_В * (n_В - 1) \quad (4.9)$$

де $l_в$ - відстань між вертикальними заземлювачами, (прийняти за $L_в$ = 3м);

$n_в$ - необхідна кількість вертикальних заземлювачів.

$$l_c = 1{,}05 * 3 * (16 - 1) \approx 48м$$

8) Визначається опір розтіканню струму горизонтального заземлювача (з'єднувальної стрічки) $R_г$, Ом:

$$R_Г = \frac{P_{розр.г}}{2 * \pi * l_c} * \ln\frac{2 * l_c^2}{d_{см} * h_Г} \quad (4.10)$$

де $d_{см}$ - еквівалентний діаметр смуги шириною b, $d_{см}$ =0,95b, b = 0,15 м;

$h_г$ - глибина закладання горизонтальних заземлювачів (0,5 м);

$l_с$ - довжина з'єднувальної стрічки горизонтального заземлювача $l_с$, м

$$R_Г = \frac{220}{2 * \pi * 48} * \ln\frac{2 * 48^2}{0{,}95 * 0{,}15 * 0{,}5} = 8{,}1Ом$$

9) Визначається коефіцієнт використання горизонтального заземлювача η_c. відповідно до необхідної кількості вертикальних заземлювачів $n_в$. Коефіцієнт використання з'єднувальної смуги η_c=0,3 (табличне значення).

10) Розраховується результуючий опір заземлювального електроду з урахуванням з'єднувальної смуги:

$$R_{заг} = \frac{R_В * R_Г}{R_В * \eta_c + R_Г * n_В * \eta_В} \quad (4.11)$$

Висновок: дане захисне заземлення буде забезпечувати електробезпеку будівлі, так як виконується умова: $R_{заг}$ Ом, а саме:

$$R_{заг} = \frac{18,5 * 8,1}{18,5 * 0,3 + 8,1 * 16 * 0,57} = 1,9 \leq R_д$$

Висновки до розділу 4

В результаті проведеної роботи було зроблено аналіз умов праці, шкідливих та небезпечних чинників, з якими стикається робітник. Було визначено параметри і певні характеристики приміщення для роботи над запропонованим проектом написаному в кваліфікаційній роботі, описано, які заходи потрібно зробити для того, щоб дане приміщення відповідало необхідним нормам і було комфортним і безпечним для робітника.

Приведені рекомендації щодо організації робочого місця, а також важливу інформацію щодо пожежної та електробезпеки. Були наведені розміри приміщення та значення температури, вологості й рухливості повітря, необхідна кількість ламп та інші параметри, значення яких впливає на умови праці робітника, а також – наведені інструкції з охорони праці, техніки безпеки при роботі на комп'ютері.

ВИСНОВКИ

У даному дипломному проекті був розроблений комплект програм сервер станційного обладнання апаратури дистанційного температурного контролю технічного стану ходових частин рухомих одиниць залізничного транспорту.

Для виконання цієї задачі були задіяні архітектурні, структурні та програмні методи її вирешення.

З перспектив розвитку програмного комплексу слід відмітити можливу реалізацію централізованої мережі для зв'язку між станціями ПТО.

До вимог експлуатації в реальних умовах також належить реалізація можливості віддаленого контролю та ручної діагностики ЛПК за допомогою додатку для АРМ, що належить до подальших перспектив проекту.

Розроблене програмне забезпечення являється прототипним, яке, в перспективі, буде являти собою основу реального комплекту ПЗ для станційного обладнання АКРО-Б.

ПЕРЕЛІК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ

1. David Griffiths, Dawn Griffiths. Head First C: A Brain-Friendly Guide, 1st Edition. 2012.
2. Peter Prinz, Tony Crawford. C in a Nutshell: The Definitive Reference, 2nd Edition. 2015.
3. Bjarne Stroustrup. The C++ Programming Language, 4th Edition. 2013.
4. Ivor Horton, Peter Van Weert. Beginning C++17 From Novice to Professional. 2018.
5. Scott Meyers. Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14, 1st Edition. 2014.
6. Bjarne Stroustrup. Programming: Principles and Practice Using C++, 2nd Edition. 2014.
7. Tony Gaddis. Starting Out with C++ from Control Structures to Objects 9th Edition. 2017.
8. Mark Lutz. Python Pocket Reference: Python In Your Pocket, 5th Edition. 2014.
9. C. H. Swaroop. A Byte of Python. 2013.
10. Paul Barry. Head First Python: A Brain-Friendly Guide, 2nd Edition. 2016.
11. Dan Bader. Python Tricks: A Buffet of Awesome Python Features. 2017.
12. David Beazley, Brian K. Jones. Python Cookbook, 3rd Edition. 2013.
13. Ray Rischpater. Application Development with Qt Creator, 2nd Edition. 2014.
14. Lee Zhi Eng. Qt 5 C++ Cookbook. 2016.

15. Иванов Н.Н. Программирование в Linux. Самоучитель. Второе издание. 2013.

16. W. Richard Stevens, Stephen A. Rago. Advanced Programming in the UNIX Environment, 3rd Edition. 2013.

17. Robert Love. Linux System Programming: Talking Directly to the Kernel and C Library, 2nd Edition. 2013.

18. ДСН 3.3.6.042-99. Санітарні норми мікроклімату виробничих приміщень. Міністерство охорони здоров'я України (МОЗ). Постанова № 42 від 01.12.1999.

19. ДСанПІН 3.3.2.007-98. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин. Міністерство охорони здоров'я України (МОЗ).

20. ГОСТ 12.1.030-81 ССБТ. Електробезпека. Захисне заземлення. Занулення.

21. ГОСТ 13109-97. Норми якості електричної енергії в системах електропостачання загального призначення.

22. НПАОП 0.00-1.28-10. Про погодження матеріалів правил охорони праці під час експлуатації електронно-обчислювальних машин.

23. НАПБ Б.03.002-2007. Норми визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою.

24. ГОСТ 12.1.044-89. Система стандартів безпеки праці. Вогнестійкість. Номенклатура показників і методи їх визначення (ICO 4589-84).

25. НПАОП 40.1-1.01-97. Правила безпечної експлуатації електроустановок.

ДОДАТОК А

Файл listen.h

```
#ifndef _ART_LISTEN_H
#define _ART_LISTEN_H

#ifdef __cplusplus
extern "C" {
#endif

#include <stdio.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/un.h>
#include <unistd.h>
#include <pthread.h>
#include <arpa/inet.h>
#include <stdint.h>
#include <inttypes.h>

#include "lpktypes.h"
#include "guitypes.h"
#include "../rxi/ini.h"
#include "../rxi/log.h"

void *listen_lpk(void *fn) {
        int listen_sock, client_sock;
        int *sockp;
        struct sockaddr_in server_addr, client_addr;
        int client_addr_len = 32;
        uint16_t max_queue;
        uint16_t port_lpk;

        memset(&server_addr, 0, sizeof(server_addr));
        ini_sget("CONNECTION", "max_queue", "%u", &max_queue);
        ini_sget("CONNECTION", "port_lpk", "%u", &port_lpk);

        server_addr.sin_family = AF_INET;
        server_addr.sin_port = htons(port_lpk);
        server_addr.sin_addr.s_addr = inet_addr(ini_get("CONNECTION", "ip"));

        if ((listen_sock = socket(PF_INET, SOCK_STREAM, 0)) == -1)
                return (void *)-1;
        if (setsockopt(listen_sock, SOL_SOCKET, SO_REUSEADDR,
                &(int){1}, sizeof(int)) == -1)
                return (void *)-2;
        if ((bind(listen_sock, (struct sockaddr *)&server_addr,
                        sizeof(server_addr))) == -1)
                return (void *)-3;
        if (listen(listen_sock, max_queue) == -1)
```

```
                return (void *)-4;

        while (1) {
                pthread_t thread_session_lpk;
                struct LpkConnection *conn;

                client_sock = accept(listen_sock,
                        (struct sockaddr *)&client_addr, (socklen_t *)&client_addr_len);
                if (client_sock == -1)
                        continue;

                conn = lpk_connection_new(client_addr.sin_addr.s_addr, client_sock);
                if (conn == NULL) {
                        close(client_sock);
                        continue;
                }

                if (pthread_create(&thread_session_lpk, NULL, ((void *)(void *)fn), (void *)conn) != 0)
        }
        close(client_sock);
        return (void *)0;
}

struct ListenGuiSettings {
        void *(*callback)(void *);
        GuiConnectionType connection_type;
};

void *listen_gui(void *s)
{
        struct ListenGuiSettings settings = *(struct ListenGuiSettings *)s;
        int listen_sock, client_sock;
        int client_addr_len = 32;
        uint16_t max_queue;
        ini_sget("CONNECTION", "max_queue", "%u", &max_queue);

        if (settings.connection_type == CONNECTION_INET) {
        struct sockaddr_in server_addr, client_addr;
        uint32_t port;
        ini_sget("CONNECTION", "port_gui_updates", "%lu", &port);

        memset(&server_addr, 0, sizeof(server_addr));

        server_addr.sin_family = AF_INET;
        server_addr.sin_port = htons(port);
        server_addr.sin_addr.s_addr = inet_addr(ini_get("CONNECTION", "ip"));

        if ((listen_sock = socket(PF_INET, SOCK_STREAM, 0)) == -1) return (void *)1;

        if (setsockopt(listen_sock, SOL_SOCKET, SO_REUSEADDR, &(int){1}, sizeof(int)) == -1)
return (void *)1;

        if ((bind(listen_sock, (struct sockaddr *)&server_addr,
                        sizeof(server_addr))) == -1) return (void *)1;
```

```
		if (listen(listen_sock, max_queue) == -1)
			return (void *)1;

		while (1) {
			pthread_t thread_session_gui;
			struct GuiConnection *conn;

			client_sock = accept(listen_sock, (struct sockaddr *)&client_addr, (socklen_t
				*)&client_addr_len);
			if (client_sock == -1) continue;

			conn = gui_connection_new(CONNECTION_INET, client_addr.sin_addr.s_addr,
					client_sock);
			if (conn == NULL) {
				close(client_sock);
				continue;
			}
			if (pthread_create(&thread_session_gui, NULL, (settings.callback), (void *)conn) != 0) {
				free(conn);
				close(client_sock);
			}
		}

		return (void *)0;
	}
	else if (settings.connection_type == CONNECTION_UNIX)
	{
		struct sockaddr_un saddr, saddr_client;

		memset(&saddr, 0, sizeof(saddr));
		saddr.sun_family = AF_UNIX;

		strcpy(saddr.sun_path, ini_get("CONNECTION", "sock_gui_updates"));

		if ((listen_sock = socket(AF_UNIX, SOCK_STREAM, 0)) == -1)
			return (void *)1;

		unlink(saddr.sun_path);
		if (bind(listen_sock, (struct sockaddr *)&saddr, sizeof(saddr)) == -1)
			return (void *)1;

		if (listen(listen_sock, max_queue) == -1)
			return (void *)1;

		while (1) {
			pthread_t thread_session_gui;
			struct GuiConnection *conn;

			client_sock = accept(listen_sock,
				(struct sockaddr *)&saddr_client, (socklen_t *)&client_addr_len);
			if (client_sock == -1)
				continue;
```

```
                        conn = gui_connection_new(CONNECTION_UNIX, 0, client_sock);
                        if (conn == NULL) {
                                close(client_sock);
                                continue;
                        }

                        if (pthread_create(&thread_session_gui, NULL, ((void *)(void *)settings.callback), (void *)conn)
!= 0) {
                                free(conn);
                                close(client_sock);
                        }
                }
                return (void *)0;
}

#ifdef __cplusplus
} // extern "C"
#endif

#endif // _ART_LISTEN_H
```

ДОДАТОК Б

Файл akrobsrv.c

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <time.h>
#include <stdlib.h>
#include <locale.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/un.h>
#include <arpa/inet.h>
#include <linux/limits.h>
#include <pthread.h>
#include <signal.h>
#include <inttypes.h>

#define DEBUG 1
#define LPK_BUFSIZ 10240
#define CRC16 0x8005

#include "src/mad/akrobtypes.h" // Осн. типы данных АКРО-Б
#include "src/mad/iface_ptokmp.h" // Осн. составные типы данных АКРО-Б и
интерфейс взаимодействия с сервером
#include "src/rxi/ini.h"
#include "src/rxi/log.h"
#include "src/art/types.h"
#include "src/art/guireq.h"
#include "src/art/utils.h"
#include "src/art/listen.h"


void *listen_gui_inet(void *fn);
void *listen_gui_unix(void *fn);
void *listen_lpk(void *fn);
void *session_gui(void *connp);
void *session_lpk(void *connp);



int main(void) {
daemon(0, 0);
      log_start(LOG_TRACE);
      log_info("[main] Program started.");

      int result;
      pthread_t thread_listen_gui_inet_upd;
      pthread_t thread_listen_gui_inet_cmd;
      pthread_t thread_listen_gui_unix_upd;
```

```
	pthread_t thread_listen_gui_unix_cmd;
	pthread_t thread_listen_lpk;
	pthread_t thread_tick_lpk_timers;

	struct ListenGuiSettings listen_gui_inet_upd = {
			session_gui
			CONNECTION_INET};
	struct ListenGuiSettings listen_gui_unix_upd = {
			session_gui,
			CONNECTION_UNIX};

	if (pthread_create(&thread_listen_gui_inet_upd, NULL, listen_gui, (void
*)&listen_gui_inet_upd) != 0)
	{
		log_error("[main] pthread_create(thread_listen_gui_inet_upd, ...)
error: %s", strerror(errno));
		return 1;
	}
	if (pthread_create(&thread_listen_gui_unix_upd, NULL, listen_gui, (void
*)&listen_gui_unix_upd) != 0)
	{
		log_error("[main] pthread_create(thread_listen_gui_unix_upd, ...)
error: %s", strerror(errno));
		return 1;
	}
	if (pthread_create(&thread_listen_lpk, NULL, listen_lpk, (void
*)session_lpk) != 0)
	{
		perror("[main] pthread_create(thread_listen_lpk, ...) error");
		return 1;
	}

	log_info("[main] Threads created.");
///=====================================JOINING=============================
===///


	if (pthread_join(thread_listen_lpk, (void *)&result) != 0)
	{
		log_error("[main] pthread_join(thread_listen_lpk, ...) error: %s",
		strerror(errno));
		return 1;
	}

	return 0;
}

size_t recv_from_lpk(const struct LpkConnection *const c, struct UnoHeader
*const h,
char *const buf, const size_t bufsz)
{
	ssize_t recvd;

	recvd = recv(c->socket, buf, bufsz, 0);
	if (recvd < sizeof(*h))
		return 0;

	memcpy(h, buf, sizeof(*h));
```

```
	return (recvd - sizeof(*h));
}

ssize_t send_retry_to_lpk(const struct LpkConnection *const c, struct
UnoHeader *const h)
{
	h->output_ind = h->input_ind;
	h->datablock_type = 0;
	h->datablock_size = 0;
	h->crc16 = 0;

	return send(c->socket, h, sizeof(*h), 0);
}

ssize t send_ok_to_lpk(const struct LpkConnection *const c, struct UnoHeader
*const h)
{
	h->output ind = c->input_ind;
	h->datablock_type = 0;
	h->datablock_size = 0;
	h->crc16 = 0;

	return send(c->socket, h, sizeof(*h), 0);
}


void crc16_checkfile(u16 crc16, const char *fpath)
{
	long sz;
	FILE *f;
	char *buf;

	f = fopen(fpath, "r");
	if (!f) {
		log_error("No such file '%s' or error opening!", fpath);
		return;
	}

	fseek(f, 0L, SEEK_END);
	sz = ftell(f);
	rewind(f);

	buf = (char*)malloc(sz);
	if (!buf) {
		log_error("Error allocating %l bytes for buffer", sz);
		fclose(f);
		return;
	}

	if (fwrite(buf, 1, sz, f) != sz) {
		log error("Error writing to buf!");
		fclose(f);
		free(buf);
		return;
	}

	if (crc16_checksum(buf, sz) != crc16) {
		log_error("File '%s' corrupted!");
		fclose(f);
```

```
                free(buf);
                return;
        }
}

u16 crc16_checksum(const char *data, size_t sz)
{
        u16 out = 0;
        u16 crc;
        int bit_flag;
        int bits_read = 0;
        int i, j;

        if (data == NULL) return 0;

        while (sz > 0) {
                bit_flag = out >> 15;

                out <<= 1;
                out |= (*data >> bits_read) & 1;

                if (++bits_read > 7) {
                        bits_read = 0;
                        data++;
                        sz--;
                }

                if (bit_flag) out ^= CRC16;
        }

        for (i = 0; i < 16; i++) {
                bit_flag = out >> 15;
                out <<= 1;
                if (bit_flag) out ^= CRC16;
        }

        crc = 0;
        i = 0x8000;
        j = 0x0001;
        for (; i >>= 1; j <<= 1)
                if (i & out) crc |= j;

        return crc;
}

int check_datablock(const struct UnoHeader *h, const char *const buf, const
size_t bufsz)
{
        size_t datasz = bufsz - sizeof(*h);

        if (crc16_checksum(buf, bufsz) != h->crc16)
                return 0;

        switch (h->datablock_type) {
        case TRAIN:
                return h->datablock_size == sizeof(struct TrainControlState);
        case DIAGNOSTIC:
                return h->datablock_size == sizeof(struct LpkDiagnostic);
        case TRAIN_DETAILS: case SIGNALS: case CALIBRATION:
```

```
            return 1;
      default:
            return 0;
}
}

void handle_tcs(struct LpkConnection *c, char *data)
{
      struct TrainControlState tcs;
      memcpy(&tcs, data, sizeof(tcs));

      lpk_connection_update_tcs(c, tcs);
}

void handle_diagnostic(struct LpkConnection *c, char *data)
{
      struct LpkDiagnostic diag;
      memcpy(&diag, data, sizeof(diag));

      lpk_connection_update_lpkd(c, diag);
}

void handle_tdetails(struct LpkConnection *c, struct UnoHeader *h, char
*offp)
{
      char buf[LPK BUFSIZ];
      char fname[80];
      char path[PATH_MAX];
      ssize t recvd;
      FILE *f;

      memcpy(fname, offp, 80);
      sprintf(path, "%s/%s", ini_get("FOLDERS", "TRAINS"), fname);
      f = fopen(path, "wb");
      if (h->datablock_size <= (LPK_BUFSIZ-sizeof(*h))) {
            fwrite(offp, 1, h->datablock_size, f);
            fclose(f);
            return;
      } else {
            fwrite(offp, 1, LPK_BUFSIZ-sizeof(*h), f);
      }

      while ((recvd = recv(c->socket, buf, LPK_BUFSIZ, 0)) > 0)
            fwrite(buf, 1, recvd, f);
      fclose(f);

      crc16_checkfile(h->crc16, path);
}

void handle_calibration(struct LpkConnection *c, struct UnoHeader *h, char
*offp)
{
      char buf[LPK_BUFSIZ];
      char fname[80];
      u8 bif_idx;
      u8 irtc_idx;
      char path[PATH_MAX];
      ssize_t recvd;
      FILE *f;
```

```
	memcpy(&bif_idx, offp++, 1);
	memcpy(&irtc_idx, offp++, 1);
	log_info("Received SIGNAL: bif_idx = %u, irtc_idx = %u", bif_idx,
irtc_idx);

	memcpy(fname, offp, 80);
	sprintf(path, "%s/%s", ini_get("FOLDERS", "CALIBRATION"), fname);
	f = fopen(path, "wb");
	if (h->datablock_size <= (LPK_BUFSIZ-sizeof(*h)-82)) {
		fwrite(offp, 1, h->datablock_size, f);
		fclose(f);
		return;
	} else {
		fwrite(offp, 1, LPK_BUFSIZ-sizeof(*h), f);
	}

	while ((recvd = recv(c->socket, buf, LPK_BUFSIZ, 0)) > 0)
		fwrite(buf, 1, recvd, f);
	fclose(f);

	crc16_checkfile(h->crc16, path);
}

void *session_lpk(void *connp)
{
	struct LpkConnection *conn = // Информация(ip и сокет) о соединении
		(struct LpkConnection *)connp; // ^
	ssize_t bytes;
	struct UnoHeader unoh; // Заголовок
	char buf[LPK_BUFSIZ];
	char *offp;

	if (conn == NULL) {
		log_error("[session_lpk] Error with allocating memory for
LpkConnection");
		return (void*)1;
	}

	while (1) {
		memset(&unoh, 0, sizeof(unoh));
		memset(buf, 0, LPK_BUFSIZ);

		if ((bytes = recv_from_lpk(conn, &unoh, buf, LPK_BUFSIZ) <= 0) ||
				!check_datablock(&unoh, buf, bytes)) {
			send_retry_to_lpk(conn, &unoh);
			continue;
		}
		lpk_connection_refresh(conn);

		if (conn->input_ind + 1 == unoh.input_ind)
		log_warn("[session_lpk] Received previous package!");

		offp = buf+sizeof(unoh);
		if (unoh.datablock_type == TRAIN)
			handle_tcs(conn, offp);
		else if (unoh.datablock_type == DIAGNOSTIC)
			handle_diagnostic(conn, offp);
		else if (unoh.datablock_type == TRAIN_DETAILS)
```

```
                handle_tdetails(conn, &unoh, offp);
            else if (unoh.datablock_type == CALIBRATION)
                handle_calibration(conn, &unoh, offp);
            else {
                log_error("Unrecognized UnoHeader type - %d! Closing socket
connection with error!", unoh.datablock_type);
                close(conn->socket);
                lpk_connection_delete(conn);
                return (void *)1;
            }
            send_ok_to_lpk(conn, &unoh);
      }
}

void *session_gui(void *connp)
{
      struct GuiConnection *conn = (struct GuiConnection *)connp;
      struct GuiHeader guih; // Заголовок
      ssize_t recvd;

      if (conn == NULL) return (void*)1;

      if (conn->socket_type == SOCKET_GUI_UPDATES) {
            memset(&guih, 0, sizeof(guih));

            // Получаем initial message
            recvd = recv(conn->socket, &guih, sizeof(guih), 0);
            // Если тип сообщения верный(пока что единственный при
SOCKET_GUI_UPDATES), то отправляем массив и оставляем в списке соединений
            if (guih.message_type == GET_UNO_UPDATES)
            {
                send_uno_array(conn);

                // Получаем ответ на UNO_ARRAY
                memset(&guih, 0, sizeof(guih));
                recvd = recv(conn->socket, &guih, sizeof(struct GuiHeader),
0);
                if (guih.message_type == DISCONNECT || guih.message_type ==
STOP_UNO_UPDATES) {
                    log_debug("[session_gui] message_type is DISCONNECT or
STOP_UNO_UPDATES. Closing socket connection");
                    close(conn->socket);
                    gui_connection_delete(conn);
                    return (void *)0;
                }
            else if (guih.message_type == SUCCESS) {
                log_debug("[session_gui] message_type is SUCCESS. Signed for
updates");
                return (void *)0; // Не закрываем соединение и возвращаем
код возврата без ошибок
            }
      }
      close(conn->socket);
      gui_connection_delete(conn);
      return (void *)1;
}
```

ДОДАТОК В

Файл cfg.py

```
import logging


#====================== Константы методов очищения папок
================================#
"""
Удаление самых старых файлов в директории
пока размер папки больше допустимого
"""
ACTION_DELETE_OLDEST = 1

"""
Сжимание самых старых файлов в директории
пока размер папки больше допустимого
(если сжали все файлы - удалять самые старые)
"""
ACTION_COMPRESS_OLDEST = 2

"""
Отсылать главному серверу самые старые файлы
в директории(и удалять у себя) пока
размер папки больше допустимого
"""
ACTION_SEND_OLDEST = 3
####====================== Константы, отв. за очищение папок:
==============================#
cleanup_action = ACTION_DELETE_OLDEST # Выбранный метод очищения папок по
умолчанию
folder_max_size = 4.096 # (в Мб)Макс размер папки по умолчанию
#================================================================================
==============#

# Папки с файлами(можно указывать сколько угодно)
folders = {
      'trains': {
            'path': '/home/julia/Desktop/archd/trains', # (обяз.)Папка с
файлами
            'max_size': 2.22, # (опц.)Перезапис. max_size для этой папки
            'cleanup_action': ACTION_SEND_OLDEST # (опц.)Перезапис.
cleanup_action    для этой папки
      },
      'signals': {
            'path': '/home/julia/Desktop/archd/signals'
      },
      'diagnostic': {
            'path': '/home/julia/Desktop/archd/diagnostic',
            'max_size': 10.2,
            'cleanup_action': ACTION_DELETE_OLDEST
```

```
        },
        'calibration': {
                'path': '/home/julia/Desktop/archd/calibration',
                'max_size': 10.0,
                'cleanup_action': ACTION_COMPRESS_OLDEST
        }
}

log_file = "/home/julia/Desktop/archd.log" # Куда писать логи
log_level = logging.DEBUG # Уровень логгирования
log_msgfmt = "[%(asctime)s %(levelname)s] %(message)s" # Формат лог сообщений
log_datefmt = "%b %d %H:%M" # Формат даты

# (в сек) Как часто демон-архив проверяет папки
freq_seconds = 30
```

ДОДАТОК Г

Файл archd.py

```
# -*- coding: utf-8 -*-
#!/usr/bin/python3

import time
import datetime
import os
import logging as log
import zipfile
from stat import S_ISREG, ST_CTIME, ST_MODE
import daemon # for Python 3.X only

import cfg # Конфиг файл из соседнего файла


def dirsz(path='.', recursive=True):
      """
      Возвращает суммарный размер файлов в папке
      path - Путь к папке
      recursive - Включать ли подпапки
      """
      if not os.path.isdir(path):
            return 0

      if not recursive:
            return sum(os.path.getsize(f)
                        for f in os.listdir(path) if os.path.isfile(f)) /
1000000

      total_size = 0

      for dirpath, dirnames, filenames in os.walk(path):
            for f in filenames:
                  fp = os.path.join(dirpath, f)
                  total_size += os.path.getsize(fp)
      return total_size / 1000000


      def dir_files_by_date(dir_path, from_oldest_to_newest=True):
            data = (os.path.join(dir_path, fn) for fn in os.listdir(dir_path))
            data = ((os.stat(path), path) for path in data)
            data = ((stat[ST_CTIME], path)
                              for stat, path in data if
S_ISREG(stat[ST_MODE]))

            files = []
            for cdate, path in sorted(data, reverse=not
from_oldest_to_newest):
                  files.append(path)
```

```
            return files


def dir_send_oldest(path, limit):
      log.info("TODO: Sending oldest files from '{}' to server".format(path))
      files = dir_files_by_date(path)


def dir_compress_oldest(path, limit):
      log.info("Compressing oldest files from '{}'".format(path))
      files = dir_files_by_date(path)
      files_compressed = 0

      for f in files:
            if f.endswith(".zip"):
                  continue

            zout = zipfile.ZipFile("{}.zip".format(f), "w",
                        zipfile.ZIP_DEFLATED)
            zout.write(f)
            zout.close()
            os.remove(f)

            log.info("Compressed '{}'!".format(os.path.basename(f)))
            files_compressed += 1

      if dirsz(path) <= limit:
            log.info("Successfully freed disk space on '{}'. Files compressed:
{}".format(path, files_compressed))
            return 0

      log.warning("All files({}) compressed on '{}' and still too "
"many space folder occupying! Cleaning up".format(files_compressed, path))

      dir_delete_oldest(path, limit)


def dir_delete_oldest(path, limit):
      log.info("Deleting oldest files from '{}'".format(path))
      files = dir_files_by_date(path)
      files_deleted = 0

      while len(files) > 0 and dirsz(path) > limit:
            to_del = files.pop(0)
            os.remove(to_del)

            log.info("Deleted '{}' from '{}'".format(os.path.basename(to_del),
path))
            files_deleted += 1
      log.info("'{}' cleaned up successfully! Files deleted: {}".format(path,
files_deleted))


def main():
      log.info("Archive daemon started.")

      while True:
```

```
            log.debug("Checking folders for needs in cleaning..")
            for key, value in cfg.folders.items():
                  max_size = value.get('max_size', cfg.folder_max_size)
                  actual_size = dirsz(value['path'])

                  if actual_size > max_size:
                        cleanup_action = value.get('cleanup_action',
cfg.cleanup_action)

                        log.warning("Folder {} is too large(max_size={}Mb, "
"actual_size={}Mb)!".format(key, max_size, actual_size))

                        if cleanup action == cfg.ACTION SEND OLDEST:
                              dir send oldest(value['path'], max size)
                        elif cleanup action == cfg.ACTION COMPRESS OLDEST:
                              dir compress oldest(value['path'], max size)
                        elif cleanup action == cfg.ACTION DELETE OLDEST:
                              dir delete oldest(value['path'], max size)
                        log.debug("Check successfull! Sleeping for {}
                              seconds..".format(cfg.freq_seconds))
            time.sleep(cfg.freq_seconds)

      log.info("Archive daemon stopped.")


with daemon.DaemonContext():
      log.basicConfig(
            filename=cfg.log file,
            level=cfg.log level,
            format=cfg.log msgfmt,
            datefmt=cfg.log_datefmt
      )
      main()
```

ДОДАТОК Ґ

Файл ifaceserver.cpp

```
#include "ifaceserver.h"

IfaceServer::IfaceServer(QObject *parent, GuiConnectionType conn_type) :
QObject(parent)
{
	m_connection_type = conn_type;
}

void IfaceServer::startPolling()
{
	int sock;
	int i;
	struct GuiHeader header;
	struct UnoStateData diag;
	struct UnoDiagnosticData state;
	struct sockaddr_in saddr_in;
	struct sockaddr_un saddr_un;

	if (m_connection_type == CONNECTION_UNIX)
	{
		sock = socket(AF_UNIX, SOCK_STREAM, 0);
		memset(&saddr_un, 0, sizeof(saddr_un));
		saddr_un.sun_family = AF_UNIX;
		strcpy(saddr_un.sun_path, ini_get("CONNECTION",
"sock_gui_updates"));
		::connect(sock, (struct sockaddr *)&saddr_un, sizeof(saddr_un));
	}
	else
	{
		saddr_in.sin_family = AF_INET;
		ini_sget("CONNECTION", "port_gui_updates", "%u",
&saddr_in.sin_port);
		saddr_in.sin_addr.s_addr = inet_addr(ini_get("CONNECTION", "ip"));
		sock = socket(PF_INET, SOCK_STREAM, 0);
		::connect(sock, (struct sockaddr *)&saddr_in, sizeof(saddr_in));
	}
	sendInitialMessage(sock, &header);
	for (i = 0; i < header.value; i++)
	{
		memset(&diag, 0, sizeof(struct UnoDiagnosticData));
		memset(&state, 0, sizeof(struct UnoStateData));
		recv(sock, &diag, sizeof(struct UnoDiagnosticData), 0);
		emit sendDiagnosticData(&diag);
		recv(sock, &state, sizeof(struct UnoStateData), 0);
		emit sendStateData(&state);
	}
	sendSuccessMessage(sock, &header);
	while (1)
	{
```

```
            memset(&header, 0, sizeof(struct GuiHeader));
            recvUpdateHeader(sock, &header);
            if (header.mtype == UNO_STATE_UPDATE)
            {
                  recv(sock, &state, sizeof(struct UnoStateData), 0);
                  emit sendStateData(&state);
                  sendSuccessMessage(sock, &header);
            }
            else if (header.mtype == UNO_DIAGNOSTIC_UPDATE)
            {
                  recv(sock, &diag, sizeof(struct UnoDiagnosticData), 0);
                  emit sendDiagnosticData(&diag);
                  sendSuccessMessage(sock, &header);
            }
            else
            {
                  sendDisconnectMessage(sock, &header);
                  close(sock);
                  return;
            }
      }
      close(sock);
      return;
}
```

ДОДАТОК Д

Комп’ютерна презентація

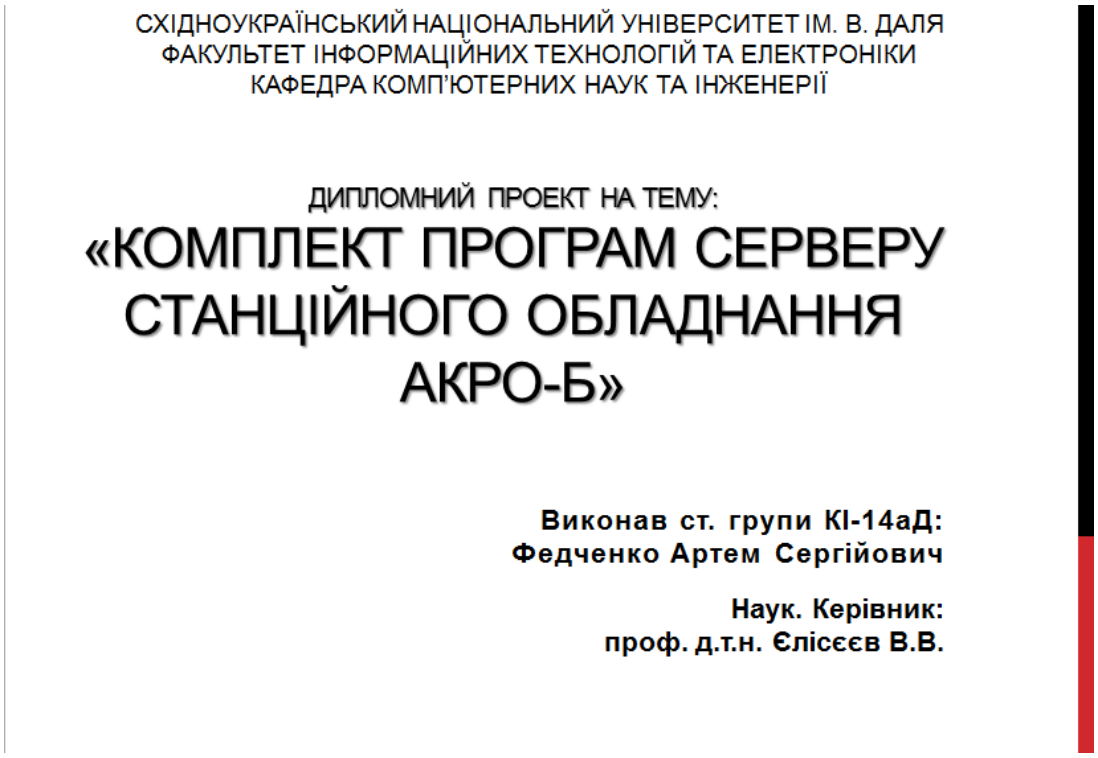


Рисунок Д.1 – Титульний слайд

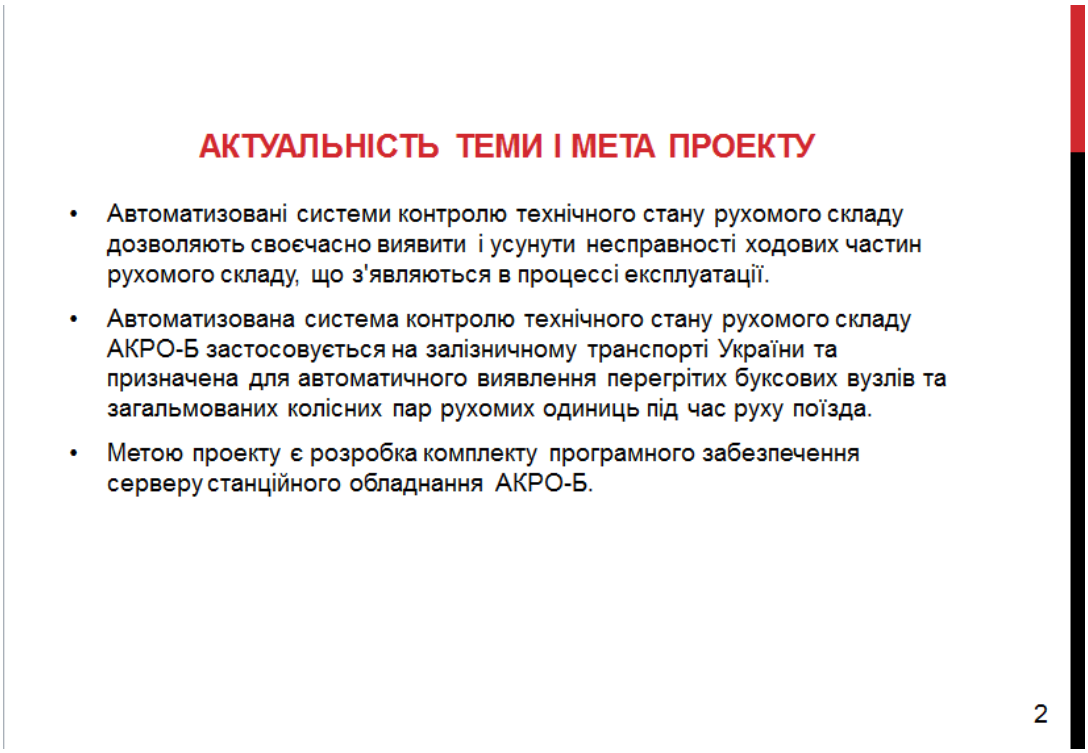


Рисунок Д.2 – Слайд «Актуальність і мета»

СТРУКТУРА АКРО-Б

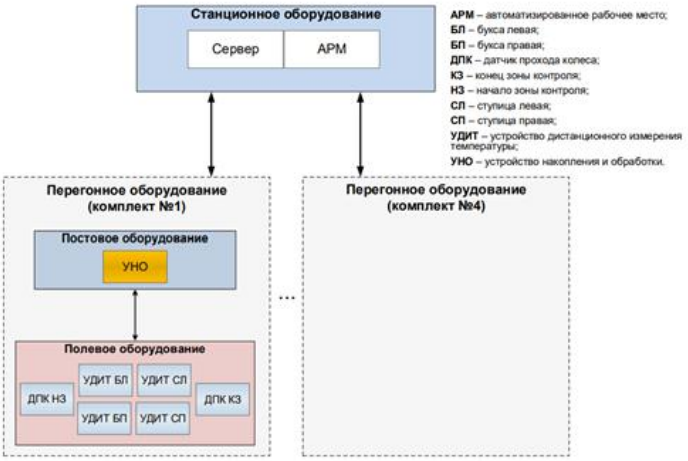
- Підлогове обладнання – включає 2 інфрачервоні вимірювальні камери для дистанційного вимірювання температури корпусів буксових вузлів та температури дисків коліс рухомого складу.
- Постове обладнання – представляє собою пристрій накопичення та оборобки в шафовій конструкції, розташовуваний на відстані до 30 м від вимірювальних камер підлогового обладнання.
- Станційне обладнання – робоча станція ПС5150, яка являє собою сервер ПТО, у сукупі з АРМ на базі промислового або персонального комп'ютера Поєднуються зовнішніми інтерфейсами. Розміщуються в приміщенні ПТО або чергового по станції.

ПС5150. Серверна конфігурація

3

Рисунок Д.3 – Слайд «Структура АКРО-Б»

СТРУКТУРНА СХЕМА АКРО-Б



4

Рисунок Д.4 – Слайд «Структурна схема АКРО-Б»

ПОСТАНОВКА ЗАДАЧІ

- Вивчення та аналіз роботи перегінного обладнання системи контролю технічного стану ходових частин рухомих одиниць залізничного транспорту АКРО-Б.
- Розробка протоколу зв'язку серверу з комплектами постового обладнання.
- Розробка протоколу зв'язку серверу з обладнанням АРМ.
- Розробка програми-демону зв'язку з комплектами постового обладнання та обладнанням АРМ.
- Розробка програми-демону архіву, виконуючу роль контролю за станом накопичувачу даних серверу.
- Розробки програми з графічним інтерфейсом для обладнання АРМ.

5

Рисунок Д.5 – Слайд «Постановка задачі»

ІНСТРУМЕНТИ РОЗРОБКИ

- Мова програмування С – компільована мова програмування високого рівня, орієнтована на створення швидкодіючих програм.
- Мова програмування С++ у зв'язці з використанням фреймворку Qt являються основою для міжплатформенних та швидких програм з графічним інтерфейсом.
- Python – високорівнева скриптова мова загального призначення, яка орієнтована на вирішення задач невеликої ступені складності.

6

Рисунок Д.6 – Слайд «Інструменти розробки»

ДЕМОН ЗВ'ЯЗКУ

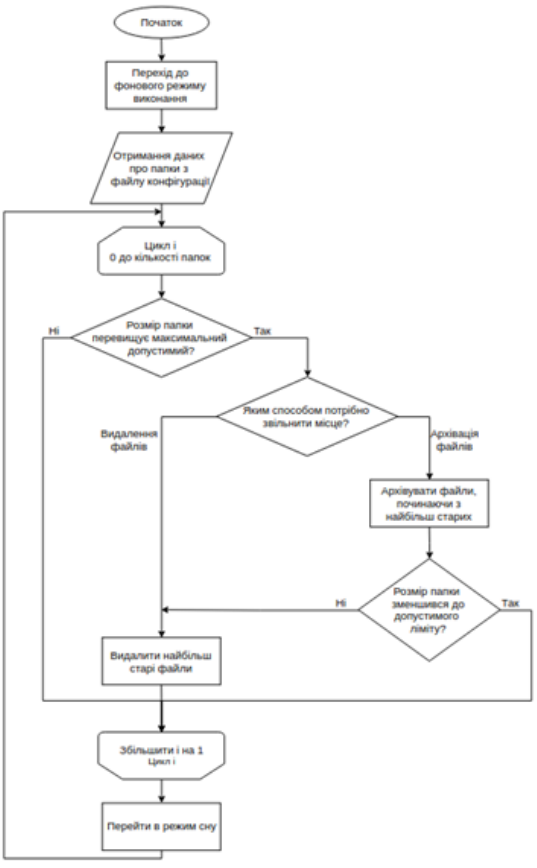
Для забезпечення зв'язку між комплектами постового обладнання та програми з графічним інтерфейсом для обладнання АРМ був розроблений демон зв'язку за допомогою мови програмування С з використанням бібліотек POSIX Threads та POSIX Socket.

7

Рисунок Д.7 – Слайд «Демон зв'язку»

ДЕМОН АРХІВУ

Для забезпечення контролю за станом директорій архіву серверу станційного обладнання був розроблений демон архіву



8

Рисунок Д.8 – Слайд «Демон архіву»

ПРОГРАМА ДЛЯ ОБЛАДНАННЯ АРМ

Була реалізована програма для обладнання АРМ, основною задачею якої є підключення до серверу станційного обладнання для отримання оновлень інформації від комплектів постового обладнання з її подальшим відображенням.

Інтерфейс програми

9

Рисунок Д.9 – Слайд «Програма для обладнання АРМ»

РЕЗУЛЬТАТ РОБОТИ

- В результаті розробки був реалізований комплект програмного забезпечення для серверу станційного обладнання АКРО-Б. Дана система реалізує зв'язок комплектів постового обладнання з сервером станціонного обладнання. Основною частиною розробленої системи є демон зв'язку.
- Комплект програмного забезпечення серверу станційного обладнання був побудований за допомогою мов програмування C, C++, Python з використанням фреймворку Qt та бібліотек POSIX Threads, POSIX Socket.
- Комплект програмного забезпечення серверу станційного обладнання має модульну структуру, за рахунок чого програмне забезпечення даного комплекту може бути легко розширено та змінено шляхом введення або розширення нових функцій.

10

Рисунок Д.10 – Слайд «Результат роботи»

ВИСНОВКИ

- У даному дипломному проекті був розроблений комплект програм сервер станційного обладнання апаратури дистанційного температурного контролю технічного стану ходових частин рухомих одиниць залізничного транспорту.
- Розроблене програмне забезпечення являється прототипним, яке, в перспективі, буде являти собою основу реального комплекту ПЗ для станційного обладнання АКРО-Б.

11

Рисунок Д.11 – Слайд «Висновки»