ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І.МЕЧНИКОВА

(повне найменування вищого навчального закладу)

Факультет математики, фізики та інформаційних технологій

(повне найменування інституту, назва факультету (відділення))

Кафедра математичного забезпечення комп'ютерних систем

(повна назва кафедри (предметної, циклової комісії))

Кваліфікаційна робота

на здобуття ступеня вищої освіти «бакалавр»

(освітньо-кваліфікаційний рівень)

на тему «Розробка тренажеру для аналізу алгоритмів планування процесора»

Development of the simulator for analyzing processor planning algorithms

Виконав: студент денної форми навчання

спеціальності 123 – Комп'ютерна інженерія.

(шифр і назва напряму підготовки, спеціальності)

Освітня програма «Комп'ютерна інженерія»

(назва освітньої програми)

Осуський Артем Вікторович

(прізвище, ім'я, по-батькові)

Керівник ст. викл. Лісіцина І. М.

(науковий ступінь, вчене звання, прізвище та ініціали, **підпис**)

Рецензент доц. Шпинарева І.М.

(науковий ступінь, вчене звання, прізвище та ініціали)

Рекомендовано до захисту:

Протокол засідання кафедри

№___ від «___»_____ 2023 р.

Завідувач кафедри

_____ _Євгеній МАЛАХОВ_

(підпис) (ім'я, прізвище)

Захищено на засіданні ЕК № ___

протокол № __ від «_ _» ____ 2023 р.

Оцінка _____/_____/_____

(за національною шкалою, шкалою ECTS, бали)

Голова ЕК

_____ _Алла КОБОЗЄВА_

(підпис) (ім'я, прізвище)

Одеса – 2023

АНОТАЦІЯ

Дипломна робота присвячена розробці тренажеру алгоритмів планування процесора та подальшого аналізу показників роботи цих алгоритмів.

Метою роботи є порівняння ознакових та чисельних характеристик існуючих алгоритмів планування процесора з метою порівняння ціх характеристик для різних алгоритмів або різних модифікацій того самого алгоритму. Для досягнення цієї мети розробляється тренажер, який дозволяє проводити детальний аналіз різних алгоритмів планування процесора та збирати необхідні показники для подальшого порівняння.

У роботі використовуються відомі алгоритми планування процесору. Для порівняння алгоритмів враховуються важливі показники їх роботи, такі як завантаженість ЦП, середній час обороту для завершених процесів, максимальна довжина черги готових процесів та інші.

Отримані показники роботи алгоритмів візуалізуються для подальшого аналізу результатів, який дозволяє порівняти алгоритми за обраними показниками. Розроблений тренажер може бути корисним інструментом для учбового процесу.

ABSTRACT

The thesis is devoted to the development of a simulator for CPU scheduling algorithms and further analysis of the performance of these algorithms.

The aim of the work is to compare the attribute and numerical characteristics of existing CPU scheduling algorithms in order to compare these characteristics for different algorithms or different modifications of the same algorithm. To achieve this goal, a simulator is being developed that allows for a detailed analysis of various CPU scheduling algorithms and collects the necessary indicators for further comparison.

The work uses well-known processor scheduling algorithms. To compare the algorithms, we take into account important indicators of their performance, such as CPU utilization, average turnaround time for completed processes, maximum queue length of finished processes, and others.

The obtained algorithm performance indicators are visualized for further analysis of the results, which allows comparing algorithms by selected indicators. The developed simulator can be a useful tool for the educational process.

ЗМІСТ

Стор.

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

SJF - Shortest Job First

SRT - Shortest Remaining Time

HPF - Highest Priority First

RR - Round Robin

ЦП – центральний процесор

ОП/ОЗУ – оперативна пам’ять

ОС – операційна система

ВСТУП

Актуальність теми "Розробка тренажеру для аналізу алгоритмів планування процесора" визначається значенням оптимізації процесів планування в сучасних комп'ютерних системах. Планування процесора відіграє ключову роль у забезпеченні ефективності виконання задач, таких як обчислення, обробка даних та управління ресурсами.

В сучасному світі швидкість обробки даних є одним з основних факторів успіху в багатьох галузях, включаючи наукові дослідження, виробництво та інформаційні технології. Ефективне планування процесора може суттєво покращити продуктивність комп'ютерних систем і забезпечити оптимальне використання обчислювальних ресурсів.

В сучасній літературі було розроблено різні алгоритми планування процесора, такі як алгоритми з приоритетами, Round Robin, алгоритми з вагами та інші. Відомі світові тенденції розв'язання задач планування процесора показують постійний пошук нових ефективних алгоритмів, які забезпечують оптимальну виконавчу послідовність задач для досягнення максимальної продуктивності.

Підтримуюче програмне забезпечення (а також інструментальний засіб) - це спеціалізоване програмне забезпечення, яке використовується у процесі викладання навчальних дисциплін. У кінці 60-х років почалось застосування програмного забезпечення з метою навчання студентів програмуванню. Поступово його застосування розширилося на інші дисципліни, такі як "Операційні системи" та "Системне програмне забезпечення". Використання програмного забезпечення у цих дисциплінах обумовлено необхідністю вивчення концепцій, що лежать в основі операційних систем.

Симулятори є ефективним інструментом для покращення процесу навчання та сприяють усвідомленню концепцій і технологій. Використання симуляторів операційних систем з візуальним інтерфейсом успішно

застосовується в закордонних університетах для поліпшення якості навчання. Однак, українські університети зіштовхуються з деякими обмеженнями. Більшість доступних симуляторів не є доступними в Україні, а ті, до яких є доступ, мають свої недоліки. Серед цих недоліків можна виділити обмежену кількість реалізованих алгоритмів, відсутність можливості налаштування параметрів цих алгоритмів та обмежену кількість статистичних показників.

Метою даної роботи є розробка програмної системи, яка надасть можливість графічного відображення роботи різноманітних алгоритмів планування процесора з можливістю вибіру параметрів в гіпотетичній операційній системі та збору основних показників роботи цих алгоритмів для подальшого їх порівняння.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- дослідити предметну область;
- провести огляд існуючих аналогів;
- сформулювати вимоги до створюваної системи;
- побудувати загальну архітектуру системи і вибрати програмну платформу для її реалізації;
- спроектувати інтерфейс програми;
- виконати програмну реалізацію системи з урахуванням шаблону проектування MVC;
- візуалізувати порівняння основних показників роботи алгоритмів, сбір яких є однією з вимог до розробляємої системи;
- провести тестування моделі.

Отже, дана робота має на меті розробку тренажеру для аналізу алгоритмів планування процесора, який дозволить проводити порівняльну характеристику алгоритмів на основі збіру показників роботи різних алгоритмів та їх модифікацій. Результати цієї роботи можуть бути корисними та практичними впровадженнями в навчальних матеріалах.

1 ОГЛЯД МЕТОДІВ ТА ЗАСОБІВ НАВЧАННЯ ТЕОРІЇ ОПЕРАЦІЙНИХ СИСТЕМ

Система імітації є звичайною комп'ютерною програмою, яка дозволяє моделювати концепції операційної системи. Більшість симуляторів створюють середовище, в якому студенти можуть активно експериментувати з конкретними поняттями, що сприяє кращому їх розумінню. Імітація полягає у програмуванні моделі комп'ютерної системи, яка поводиться подібно до реальної системи, принаймні щодо планування процесора.

Використання симуляторів має кілька переваг. Вони, як правило, не потребують спеціалізованого обладнання. Багато симуляторів також надають функціональність, яка недоступна в реальних системах.

Однак, використання таких систем також має свої проблеми. Застосувані спрощення зменшують подібність модельованої системи до реальної. Також слід зазначити, що модельовані системи є лише наближенням до реальності, що може призвести до того, що вони не повністю відображають реальну систему. Однак, особливо на початковому етапі навчання, такі системи є незамінними.

На сьогоднішній день існує декілька симуляторів алгоритмів планування процесору, які дозволяють відтворити та досліджувати різні планувальні стратегії. Розглянемо декілька з них.

Симулятор SimPy [1] - це бібліотека для мови програмування Python, яка надає інструменти для моделювання та симуляції подійного протоколу. Вона широко використовується для моделювання систем зі складною взаємодією між компонентами та дослідження їх поведінки. Основна концепція SimPy полягає в тому, що модель подійного протоколу представляється у вигляді генераторів Python, які визначають послідовність подій, що відбуваються в системі.

SimPy моделює систему як послідовність подій, які впливають на стан системи та її компоненти. Події можуть бути специфіковані з певними часовими мітками та пріоритетами. Також він надає підтримку моделювання

ресурсів, таких як обмежені об'єкти, зайнятість яких може змінюватись з часом. Це дозволяє моделювати ситуації, коли ресурси розділяються між процесами або використовуються чергами для керування доступом.

Таблиця 1.1 – Переваги та недоліки SimPy

Переваги	Недоліки
Простота використання	Відсутність графічного інтерфейсу
Підтримка асинхронних процесів	Відсутність підтримки розподіленого моделювання
Гнучкість в моделюванні різних систем	Відсутність вбудованих інструментів аналізу даних
Відкрите джерело	Відсутність вбудованих інструментів візуалізації

SPEC CPU Benchmark Suite [2] – це набір тестів, розроблених для вимірювання продуктивності процесорів та комп'ютерних систем. Він широко використовується для оцінки та порівняння продуктивності різних процесорів та комп'ютерних систем в різних сценаріях використання. Набір тестів SPEC CPU складається з двох основних компонентів: SPEC CPU2006 та SPEC CPU2017. Кожен компонент включає набір бенчмарків, які вимірюють різні аспекти продуктивності процесорів, такі як швидкодія при одно- та багатопоточних обчисленнях, кешування, оптимізація та інші характеристики, наприклад цілочисельні операції, операції з рухомою комою, цілочисельні операції у багатопоточному середовищі, операції з рухомою комою у багатопоточному середовищі, цілочисельні операції з вимірюванням швидкодії та операції з рухомою комою з вимірюванням швидкодії.

Таблиця 1.2. – Переваги та недоліки SPEC CPU Benchmark Suite

Переваги	Недоліки
Широко використовується для бенчмарк тестування	Доступність лише для обмеженої кількості архітектур
Надає стандартизовані тестові набори	Потребує великих ресурсів для виконання тестів
Дозволяє оцінювати продуктивність алгоритмів планування	Потребує великих зусиль для налаштування тестових сценаріїв
Відкритий інструмент для тестування	Вимагає технічних знань для правильного використання

GEM5 (Full System Simulator) [3] - це потужний та гнучкий симулятор комп'ютерних систем, що використовується для моделювання та дослідження архітектури процесорів, систем на кристалі (SoC), вбудованих систем та інших компонентів обчислювальних систем. Він є відкритим програмним забезпеченням та розроблений з використанням мов програмування C++ та Python. Він надає можливості для моделювання різних рівнів деталей комп'ютерної системи, включаючи процесори, кеші, пам'ять, периферійні пристрої, мережі та багато іншого. Він дозволяє розробникам відтворювати поведінку реальних систем на рівні мікроархітектури та проводити різні експерименти для вивчення продуктивності, енергоефективності, впливу алгоритмів та налаштувань на систему. GEM5 використовується в академічних дослідженнях, розробці архітектури процесорів та систем на кристалі, оптимізації алгоритмів та аналізі продуктивності систем.

Таблиця 1.3. – Переваги та недоліки GEM5

Переваги	Недоліки
Відкрите програмне забезпечення	Складна конфігурація та налаштування
Підтримує широкий спектр архітектур	Великий обсяг коду, що може призводити до складнощів у розумінні
Має гнучку структуру, що дозволяє розширювати його функціональні можливості	Вимагає значних обчислювальних ресурсів
Забезпечує точність симуляції та деталізацію	Потребує високого рівня технічного розуміння для ефективного використання

Як показав огляд аналогів - використання симулятора у навчальному процесі поліпшує ефективність навчання. Однак, використання готових симуляторів не є універсальним рішенням, оскільки вони потребують адаптації під конкретний курс, що часто є неможливим.

Нижче перераховані основні недоліки розглянутих симуляторів:

1) Часто обмежена кількість процесів, що надходять у систему.
2) Використовується лише один спосіб представлення пріоритетної черги.
3) Часто відсутній засіб збору показників роботи алгоритмів планування процесора.
4) Відсутність відкритого коду.

Для вивчення та дослідження алгоритмів планування ресурсів у комп'ютерних системах потрібна система моделювання поведінки процесів, яка відтворює роботу частини операційної системи. У такій системі моделюються лише системні структури, де фіксуються події, пов'язані з процесами, без фактичного створення або виконання процесів.

Враховуючи переваги та недоліки попередніх програмних систем, основні вимоги до такої системи, як симулятор процесів, були сформульовані під час викладання курсів "Операційні системи" та "Системне програмне забезпечення". Основні вимоги включають:

1) Симулятор процесів повинен бути тренажером для освітніх цілей і використовуватися для аналізу поведінки та продуктивності алгоритмів планування процесора.
2) У симуляторі повинна бути реалізована імітація випадкового появи процесів у системі.
3) Симулятор не повинен обмежувати кількість процесів, що надходять у систему.
4) Має бути можливість налаштування параметрів генератора процесів, таких як інтенсивність надходження процесів, обмеження на максимальний час виконання кожного процесу, обмеження на

максимальний розмір адресного простору процесу та кількість допустимих пріоритетів.

5) Має бути можливість вибору алгоритму планування центрального процесора.
6) Має бути можливість порівняння різних стратегій планування шляхом збору і збереження основних показників роботи алгоритмів.
7) Має бути можливість поетапного відстежування стану черг, а також отримання інформації щодо окремих процесів.
8) Має бути можливість планування ресурсів.
9) Система повинна мати відкритий код для можливості розширення.

Основні показники роботи алгоритмів повинні бути візуалізовані для подальшого аналізу.

2 АЛГОРИТМИ ПЛАНУВАННЯ ПРОЦЕСОРУ

Алгоритми планування процесору є важливою складовою операційних систем і використовуються для управління виконанням процесів на процесорі. Їх основна мета полягає в ефективному розподілі обчислювальних ресурсів процесора між різними процесами або потоками.

Коли комп'ютер запускається, у нього можуть бути запущені декілька процесів або потоків, які конкурують за доступ до процесора. Алгоритми планування процесору вирішують, який процес або потік повинен мати пріоритет на виконання і як тривалий час він може використовувати процесор, перш ніж інші процеси отримають свій час.

Основні цілі алгоритмів планування процесору включають:

1) Забезпечення справедливого розподілу процесорного часу між процесами або потоками.
2) Мінімізація часу очікування процесів або потоків.
3) Максимізація продуктивності процесора шляхом мінімізації використання його часу на перемикання між процесами.
4) Забезпечення відповідного рівня пріоритету для важливих процесів або потоків.

Існує багато різних алгоритмів планування процесору, кожен з яких має свої особливості, переваги і недоліки. Далі розглянуто їх основні види [4].

2.1 Види алгоритмів процесору

2.1.1 Планування за тривалистю процесу

SJF (Shortest Job First) є алгоритмом планування процесора, де спочатку виконуються найкоротші задачі. Це означає, що задачі з меншим часом виконання мають пріоритет над довшими задачами. Таке планування забезпечує мінімізацію середнього часу очікування і знижує затримки у виконанні задач. Наприклад, якщо в системі є дві задачі, одна з яких має виконатися за 5 одиниць часу, а інша - за 10 одиниць часу, то першою буде виконана задача, яка потребує меншого часу.

2.1.2 Планування за тривалистю процесу у реальному часі

SRT (Shortest Remaining Time) є варіацією алгоритму SJF, де виконується задача з найменшим залишковим часом виконання. Цей алгоритм вимагає динамічного оновлення залишкового часу виконання кожної задачі. Якщо з'являється нова задача з меншим залишковим часом виконання, вона отримує пріоритет і виконується раніше. Це дозволяє мінімізувати час очікування для коротких задач і прискорює виконання системи в цілому.

2.1.3 Планування за пріоритетом процесу

HPF (Highest Priority First) є алгоритмом планування процесора, де задачі виконуються в порядку їхнього пріоритету. Кожній задачі призначається пріоритет, і задачі з вищим пріоритетом виконуються першими. Цей алгоритм забезпечує пріоритетну обробку важливих задач і може бути корисним в системах, де деякі задачі мають вищий пріоритет або терміновість виконання.

HPF має декілька варіацій, а саме:

2.1.1.1. HPF з витисканням

У цьому варіанті алгоритму HPF, якщо з'являється нова задача з вищим пріоритетом, вона витискає виконувану задачу. Витискнена задача повертається до черги і її виконання продовжується після задач з вищим пріоритетом.

2.1.1.2. HPF з квантуванням

Цей варіант алгоритму HPF використовує кванти часу для кожної задачі. Задача виконується протягом кванту часу, а потім контекст перемикається на наступну задачу з вищим пріоритетом. Це дозволяє чесно розподілити час процесора між задачами з різними пріоритетами.

2.1.1.3. HPF з витисканням та квантуванням

Цей варіант поєднує поняття витискання та квантів часу. Кожна задача має власний квант часу, і нові задачі з вищим пріоритетом можуть витіснити виконувану задачу, якщо вони не вміщуються в поточний квант часу.

2.1.1.4. HPF на основі масиву черг (спрощене хешування)

У цьому варіанті задачі зберігаються в масиві черги, впорядковані за пріоритетом. Задачі з вищим пріоритетом знаходяться в передній частині масиву. Коли з'являється нова задача, вона вставляється відповідно до її пріоритету в масив черги.

2.1.1.5. HPF на основі комбінації відсортованого та невідсортованого списків

У цьому варіанті використовується комбінація двох списків - один відсортований за пріоритетом, а інший невідсортований. Задачі з вищим пріоритетом розміщуються у відсортованому списку, тоді як інші задачі можуть знаходитись у невідсортованому списку. Планування проводиться, спочатку відсортованим списком, а потім невідсортованим списком.

2.1.1.6. HPF на основі комбінації відсортованого та невідсортованого списків з перебудуванням по таймеру

Цей варіант поєднує відсортований та невідсортований список, але також включає перебудування списків за певний проміжок часу. Задачі з вищим пріоритетом спочатку розміщуються в відсортованому списку, а потім після встановленого таймера список перебудовується, щоб врахувати можливі зміни пріоритетів.

2.1.4 Каруселне планування

RR (Round Robin) є алгоритмом планування процесора, де кожна задача отримує фіксований квант часу процесора для виконання. Якщо задача не завершена протягом цього кванту часу, вона переходить до наступної задачі в черзі. Цей алгоритм забезпечує чесний розподіл часу процесора між задачами та уникнення виключно довгих затримок. За допомогою модифікацій, таких як змінний квант часу в залежності від пріоритету, можна досліджувати інші аспекти планування.

2.1.5 Чергове планування

Multilevel Feedback Queue є алгоритмом, де задачі розподіляються у різні рівні черги в залежності від їхнього пріоритету. Задачі вищого пріоритету виконуються першими. Цей алгоритм дозволяє групувати задачі за рівнем важливості та надавати пріоритет виконанню відповідно до цього рівня. Кожна черга може мати власний алгоритм планування, такий як SJF або RR, що робить Multilevel Queue гнучким і придатним для різних сценаріїв планування процесора.

2.2. Планування, його ефективність та вимоги до алгоритмів

Планування є важливою складовою обчислювальних систем, особливо в сучасних процесорах, де ресурси потрібно раціонально розподіляти між багатьма завданнями для досягнення оптимальної продуктивності. Критерії ефективності планування та вимоги до алгоритмів планування грають важливу роль у забезпеченні ефективного використання обчислювальних ресурсів та досягненні поставлених цілей.

При виборі конкретного алгоритму планування процесору важливо враховувати клас завдань, що вирішуються обчислювальною системою, а також поставлені цілі та вимоги до планування. Ось список загальних цілей, які можуть впливати на вибір алгоритму:

1) Мінімізація часу виконання: Основна ціль полягає в мінімізації загального часу виконання всіх задач або процесів. Деякі алгоритми, такі як алгоритми з пріоритетами або згортанням, можуть бути ефективними для досягнення цієї мети.
2) Забезпечення виконання важливих завдань: Деякі системи мають завдання з високим пріоритетом або дедлайнами, які потрібно виконати вчасно. Вимога до алгоритму полягає у забезпеченні дотримання дедлайнів та пріоритетів виконання важливих завдань.
3) Рівномірне використання ресурсів: Деякі системи мають багато задач різної важливості та обчислювальних потреб. Вимога до алгоритму полягає в розподілі ресурсів таким чином, щоб задачі отримували достатньо обчислювальних ресурсів для ефективного виконання.
4) Мінімізація затримок: Деякі системи вимагають найменших можливих задержок між початком та завершенням задач. Алгоритми планування, які розраховані на мінімізацію задержок, можуть бути вибраними для досягнення цієї цілі.
5) Масштабованість: Вимога до алгоритму полягає у здатності ефективно працювати з розширеними системами з багатьма ядрами або процесорами.

6) Витривалість до відмов: Деякі системи мають вимоги до забезпечення безперебійності та витривалості до відмов. Алгоритми планування повинні бути стійкими до помилок та здатними відновлюватися після відмови.

Враховуючи ці цілі та вимоги, можна вибрати алгоритм планування, який краще відповідає потребам конкретної системи та допомагає досягнути поставлених цілей ефективності та продуктивності.

Високоякісне планування є важливим елементом оптимізації продуктивності систем. Вибір ефективного алгоритму планування, відповідного критерія ефективності та врахування вимог до алгоритмів гарантують оптимальне використання обчислювальних ресурсів, мінімізацію часу виконання та задержок, а також надійне виконання задач та досягнення поставлених цілей у різних сценаріях та умовах.

3 ПОКАЗНИКИ РОБОТИ АЛГОРИТМІВ

Серед можливих показників роботи алгоритмів були обрані ті, які вважались як найбільш повно відражаючі їх роботу [5]. Ці показники наступні:

1) Число кроків експерименту (значення лічильника кроків моделі/годин)
2) Число процесів, що надійшли (число спроб створення процесів)
3) Число відкинутих процесів (невдалих спроб розміщення в пам'яті)
4) Час роботи / простою центрального процесора (ЦП)
5) Завантаженість ЦП (відношення часу роботи процесора до числа кроків експерименту)
6) Продуктивність системи (відношення числа завершених процесів до числа кроків експерименту)
7) Число завершених процесів
8) Середній час очікування завершених процесів (відношення загального часу очікування в черзі готових процесів до числа завершених процесів)
9) Середній час обороту для завершених процесів (відношення загального часу обороту до числа завершених процесів).
10) Максимальна довжина черги готових процесів
11) Середня довжина черги готових процесів (обчислюється за формулою Літтла).

3.1 Збір показників роботи алгоритму

Показники алгоритмів збираються протягом сеансу роботи системи і використовуються у подальшої візуалізації. Вони зв’язані з процесами, але не являють собою характеристиками процесу. У зв’язку з цим, з’явилась необхідність у новому об’єкті, який інкапсулює в собі показники, що збираються. Цей об’єкт можна умовно називати статистикою і в моделі тренажеру він реалізований як окремий клас. Це рішення вигідно відрізняється від альтернативного, яке наділяє процес сторонніми властивостями.

3.2 Аналіз показників системи

Зібрані показники переносяться в електроні таблиці Excel, які використовуються для подальшої візуалізації. Далі стояла задача вибіру такого типу діаграми, який найкраще вирішує задачу порівняння показників.

Візуалізована інформація являє собою часовий ряд [6], тобто ряд кількісних значень, які показують зміну в часі, наприклад, за роками, кварталами, місяцями, тижнями, днями, годинами, хвилинами або навіть секундами.

Для візуалізації часового ряду використовуються:

1) вертикальні столбці (коли потрібно підкреслити окремі значення);
2) лінії (з маркерами або без них, якщо треба показати закономірність змін у часі);
3) тільки маркери (під час відображення значень, зібраних через нерівні проміжки часу);
4) вертикальні блоки (під час відображення розподілу в часі).

Найбільш здатними з вище перерахованих у разі порівняння тих самих показників роботи для різних алгоритмів є вертикальні столбці, таким чином для візуалізації була обрана стовпчата діаграма, приклади якої з інтенсивністю

подання процесів 0.5 приведені на рисунках 3.1 і 3.2, та з інтенсивністю 0.2 на рисунках 3.3 і 3.4. Інші візуалізовані показники наведено у додатку А.

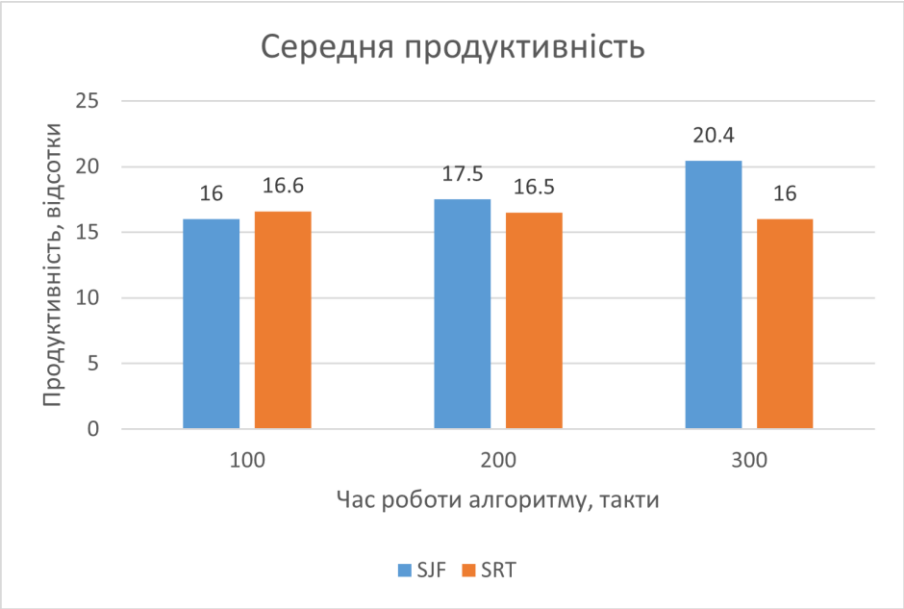

Рисунок 3.1 – Графік середньої продуктивності, інтервал процесів 0.2

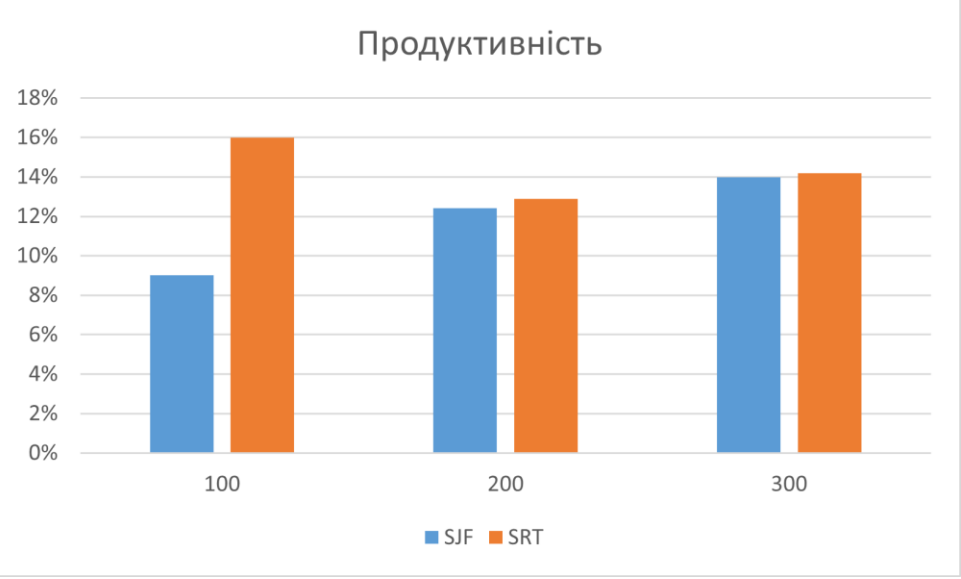

Рисунок 3.2 – Графік середньої продуктивності, інтервал процесів 0.2

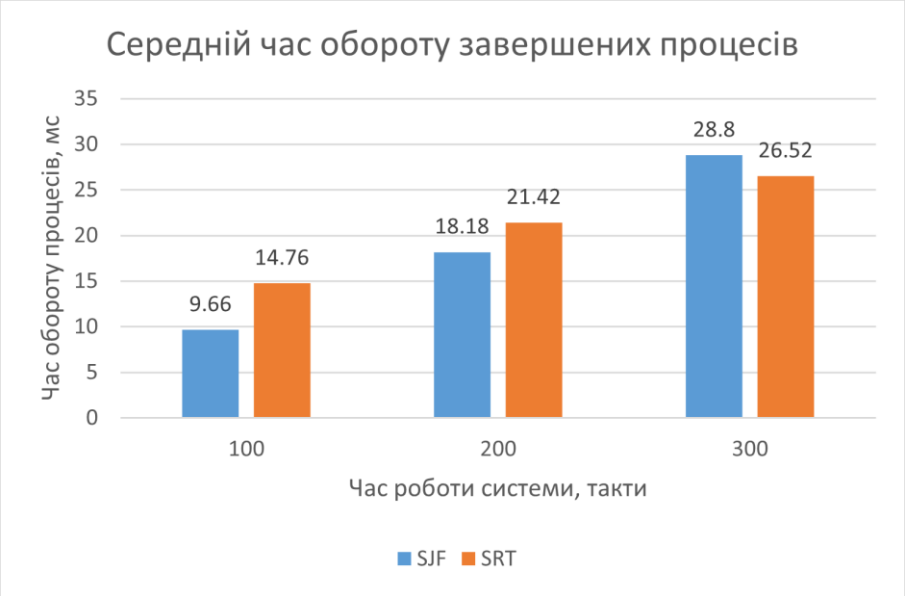

Рисунок 3.3 – Графік середнього часу обороту завершених процесів, інтенсивніть процесів 0.5

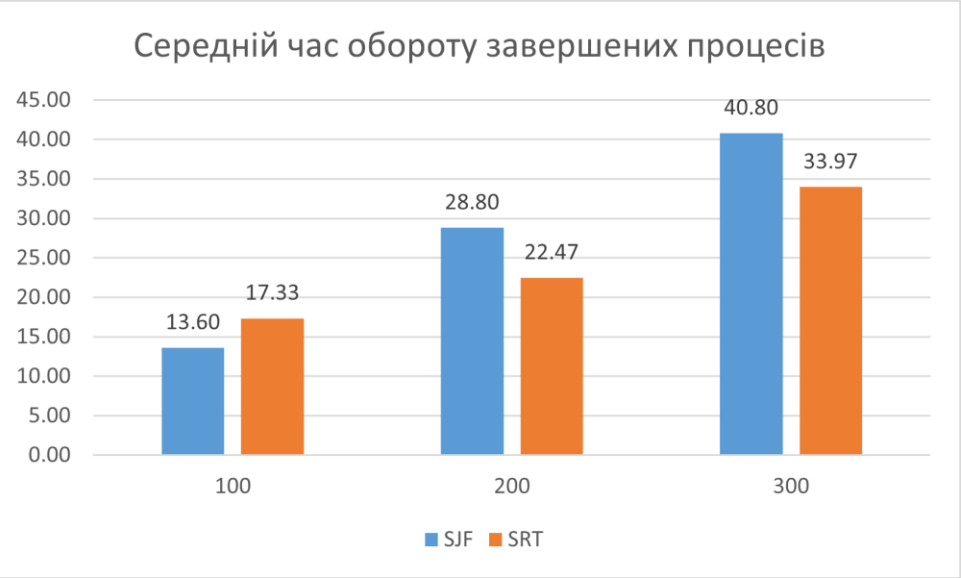

Рисунок 3.4 – Графік середнього часу обороту завершених процесів, інтенсивніть процесів 0.2

4 ПРОГРАМНА РЕАЛІЗАЦІЯ

Кожний алгоритм планування реалізований за допомогою мови програмування C# [8] з використанням шаблону проектування MVC та фреймворку користувацьких інтерфейсів WinForms [9].

4.1 Функціонування моделі обчислювальної системи

Модель обчислювальної системи фокусується на кількості та розподілі заявок на планування процесора і величині процесорного часу, необхідного для поточного процесу [7].

Процеси надходять у систему з інтервалом, що визначається заданою інтенсивністю відповідно до експоненціального розподілу. Час обслуговування для чергового процесу також генерується випадковим чином.

Вважається, що в розпорядженні обчислювальної системи є n зовнішніх пристроїв (ресурсів) $R_1, R_2,..., R_n$, звернення до яких переводить процес у стан очікування. Перед першою постановкою завдання в чергу готових процесів (до процесора) імітується розміщення робочої області процесу в оперативній пам'яті. У разі неможливості розміщення процес відкидається, інакше йому виділяється пам'ять.

Вибірка завдання з черги готових процесів відбувається в момент, коли поточний процес вичерпав інтервал безперервної роботи і звільнив центральний процесор.

У разі звернення до зовнішнього пристрою процес поміщається в чергу до нього, причому час використання ресурсу генерується випадковим чином. У разі завершення процес видаляється з обчислювальної системи.

Вважається, що в кожен момент часу процес може звернутися тільки до одного ресурсу. Після закінчення роботи із зовнішнім пристроєм процес знову поміщається в чергу готових завдань, причому генеруються новий інтервал безперервної роботи і причина її припинення.

Модель працює покроково, на кожному кроці збільшується значення таймера і модифікується стан системи, відображаючи діяльність пристроїв, процесів і планувальників. Модифікація здійснюється виконанням наступних дій:

1) викликається генератор процесів (приймається рішення про створення нового процесу залежно від результату, отриманого від генератора випадкових чисел і налаштувань експерименту);
2) у разі появи нового процесу генерується більшість його загальних характеристик (ім'я, ідентифікаційний номер, час обслуговування на центральному процесорі, розмір адресного простору тощо) і за можливості розміщення процесу йому виділяють пам'ять, а потім він розміщується у чергу готових процесів;
3) якщо ресурс (процесор або зовнішній пристрій) зайнятий деяким процесом, перевіряється, чи не вичерпав процес свій інтервал обслуговування, якщо так, то процес звільняє ресурс;
4) для процесу, який звільнив зовнішній пристрій, випадковим чином генерується час обслуговування на центральному процесорі; потім процес поміщається в чергу готових процесів;
5) для процесу, що звільнив процесор, ухвалюється рішення про завершення або запит деякого зовнішнього пристрою; для незавершених процесів випадковим чином генерується подальший час обслуговування на зовнішньому пристрої, потім процес поміщається в чергу до нього;
6) для алгоритмів із витісненням, якщо процесор зайнятий, а першим у черзі готових процесів стоїть більш високопріоритетний процес, процесор звільняється, а поточний процес повертається в чергу готових процесів;
7) для алгоритмів з квантуванням, якщо вичерпано квант часу роботи процесу на процесорі, процесор звільняється, а поточний процес повертається в чергу готових процесів;

8) якщо ресурс вільний, відповідний планувальник вибирає з відповідної черги процес для виконання процесу або дій на зовнішньому пристрої.

Графічне представлення вищесказаного у вигляді схеми життєвого циклу процесу наведено на рисунку 4.1.

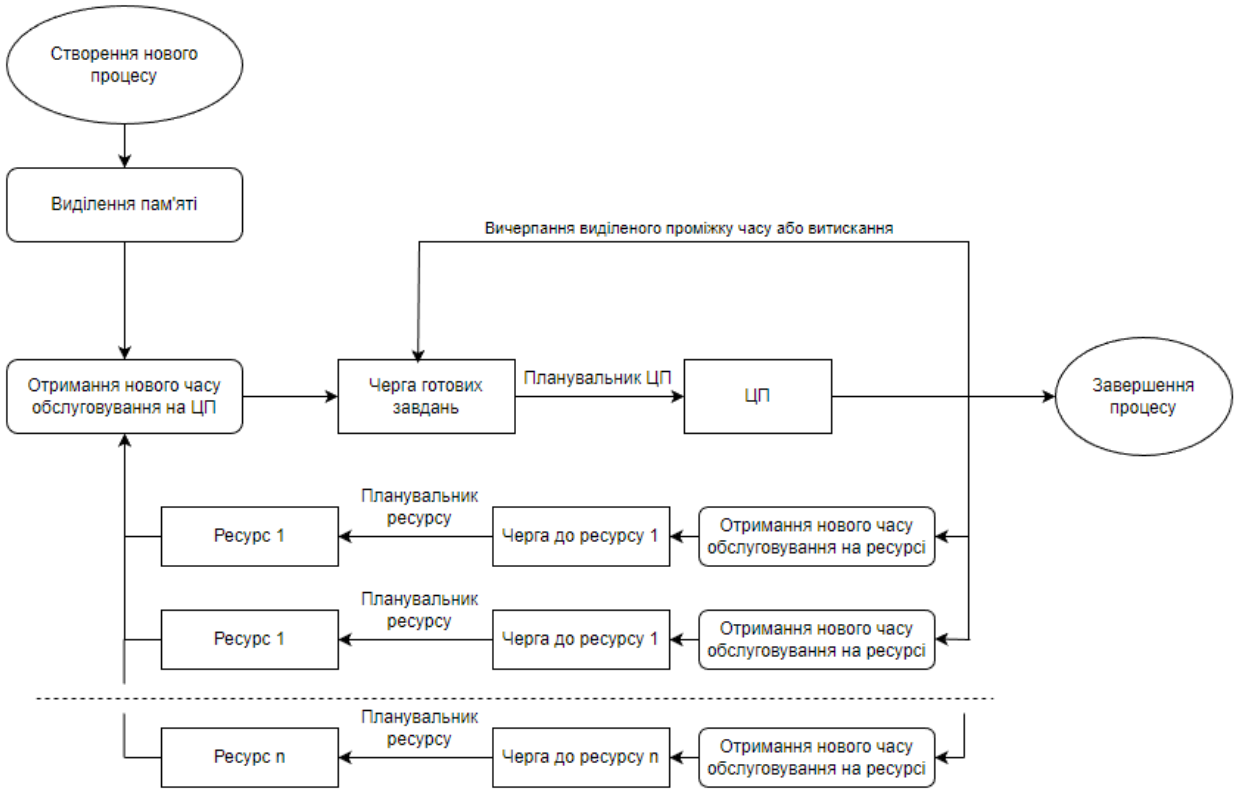

Рисунок 4.1 – Життєвий цикл процесу

Для реалізації моделі обчислювальної системи необхідно виконати такі кроки:

1) Додати в проект клас Model і класи, що реалізують компоненти обчислювальної системи;
2) Додати до проєкту бібліотеки Queues і Structures, у яких реалізовано черги та структури, що в них використовуються
3) Наповнити модель обчислювальної системи апаратними:

а. годинник;

б. центральний процесор (ЦП);

в. зовнішній пристрій;

г. оперативна пам'ять (ОП)

і програмними:

ґ. генератор ідентифікаторів процесів;

д. черга готових процесів;

е. черга до зовнішнього пристрою;

є. планувальник центрального процесора;

ж. планувальник пам'яті;

з. планувальник зовнішнього пристрою;

и. генератор процесів (генератор випадкових чисел);

і. налаштування моделі предметної області

компонентами.

4) Додати в клас Model методи, необхідні для функціонування моделі
5) Забезпечити переміщення процесів в обчислювальній системі за допомогою механізму подій:
6) Побудувати інтерфейс програми.

4.2 Шаблон MVC

Шаблон проектування MVC передбачає поділ даних додатка, призначеного для користувача інтерфейсу та керуючої логіки на три окремі компоненти: модель, представлення та контролер - таким чином, що модифікацію кожного компонента можна здійснювати незалежно [10]. Модель (Model) надає дані предметної області поданням і реагує на команди контролера, змінюючи свій стан. Подання (View) відповідає за відображення даних предметної області (моделі) користувачеві, реагуючи на зміни моделі. Контролер (Controller) інтерпретує дії користувача, сповіщаючи модель про необхідність змін.

MVC задає не стільки правила поділу програми на окремі компоненти, скільки правила їхньої взаємодії [11]. У той час як подання і контролер залежать від моделі, модель не залежить ні від подання, ні від контролера. Це ключова особливість поділу, яка дає змогу працювати з моделлю, а отже, і з бізнес-логікою застосунку, незалежно від візуального представлення.

У реалізації MVC модель може бути або пасивною, або активною. Пасивна модель не обізнана про існування подання, контролера, і навіть про свою участь у MVC-тріаді. Контролер відстежує зміни моделі та сповіщає подання. При цьому або контролер передає поданням інформацію про зміни, або подання самостійно вибирає дані з моделі. Більш витонченим рішенням є активна модель. Активність моделі проявляється в її праві самостійно сповістити подання про зміну свого стану. Щоб не порушити основну вимогу MVC про незалежність моделі від подання і контролера, механізм оповіщення реалізується на основі шаблону проектування Observer (Оглядач).

Шаблон проектування Observer, або Publisher/Subscriber (Видавець/Підписник), використовується у випадку, коли кілька об'єктів (передплатники) повинні знати про зміни стану або деякі події одного об'єкта (видавця), але потрібно підтримати низький рівень зв'язності з передплатниками. Рішення полягає в тому, що передплатники реєструються

як обробники деякої події, яку генерує видавець. Так реалізується незалежність спостережуваного об'єкта (видавця) від оглядачів (передплатників).

View (подання) користується екземпляром класу Controller або похідного від нього, для реалізації конкретної стратегії реагування. Щоб реалізувати іншу стратегію, потрібно просто підставити інший контролер. Можна навіть замінити контролер під час виконання програми, змінивши тим самим реакцію на дії користувача. Наприклад, вид можна деактивувати, так що він узагалі не буде ні на що реагувати, якщо передати йому контролер, який ігнорує події введення. Схему роботи шаблону MVC приведено на рисунку 4.2

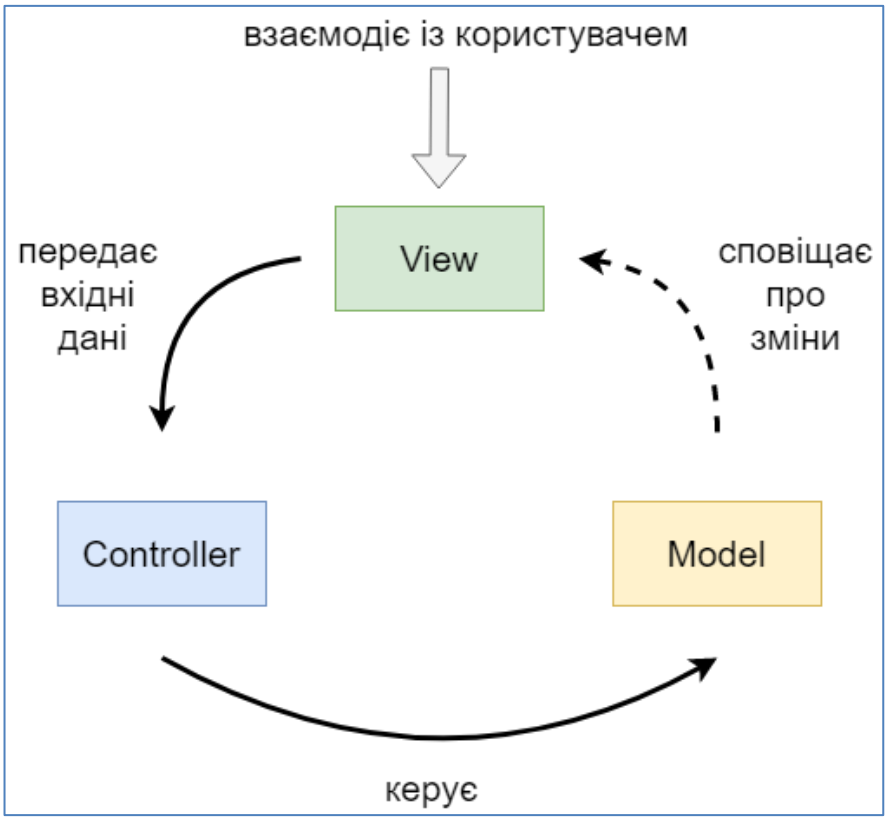

Рисунок 4.2 – Схема роботи патерну MVC

4.3 Класи програми

Деякі алгоритми можуть мати додаткові класі для реалізації свого принципу роботи, наприклад для реалізації квантування, відсортованих та невідсортованих списків, черг, механізм «каруселі» для алгоритму Round Robin та інші, тому далі буде декілька основних класів, які присутні у всіх реалізаціях алгоритму. На рисунку 4.3 розглянуто загальну діаграму класів.

Рисунок 4.3 – Діаграма класів

Далі приведено реалізовані класи апаратних компонентів, вказаних наприкінці пункту 4.1. Реалізація програмних компонентів, їх властивості та методі, наведені у додатку Б.

4.3.1 Клас Resource

Клас Resource представляє ресурс. Це може бути як ЦП так і зовнішній пристрій, в нашій системі вони розрізняються алгоритмами планування і відповідними планувальниками. Але між собою вони схожі, отже узагальнююче поняття ресурс однаково підходить до обох сутностей. Поля та властивості даного класу приведено на риуснку 4.4

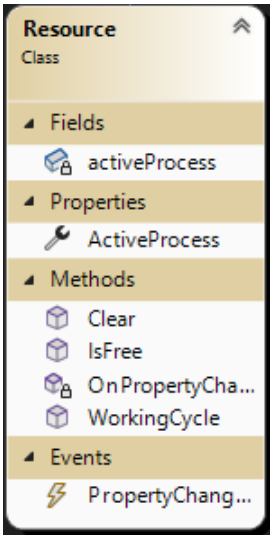

Рисунок 4.4 – поля та властивості класу Resource

Клас містить властивість ActiveProcess типу Process. Воно відображає процес, який зараз знаходиться на обслуговуванні у поточного пристрою. Код наведений на рисунку 4.5

```
public Process ActiveProcess
{
  get; set;
}
```

Рисунок 4.5 – властивість ActiveProcess

На кожному такті, процесу необхідно викликати метод IncreaseWorkTime, для інкременту часу роботи і прийняття подальших дій, стосовно своєї роботи. Але так як процес повинен збільшувати час роботи

тільки тоді, коли він на пристрої, то і сам метод викликається через пристрій, який має посилання на процес на ньому. Код наведений на рисунку 4.6

```
public Resource WorkingCycle()
{
if (ActiveProcess != null) // якщо на пристрої є процес
    ActiveProcess.IncreaseWorkTime(this); // збільш. час роботи
    return this;
}
```

Рисунок 4.6 – функція WorkingCycle

Також присутній метод для перевірки (рис. 4.7), чи зайнятий ресурс процесом:

```
public bool IsFree()
{
   return ActiveProcess == null;
}
```

Рисунок 4.7 – метод IsFree

І зняття поточного процесу (рис. 4.8):

```
public Resource Clear()
{
   ActiveProcess = null;
   return this;
}
```

Рисунок 4.8 – метод Clear

4.3.2 Клас Process

Клас Process (рис. 4.9) представляє узагальнене поняття процесу, головною характеристикою якого є час роботи, яке в реальній ситуації, зі зрозумілих причин, не відомо, але у даній реалізації генерується випадковим чином.

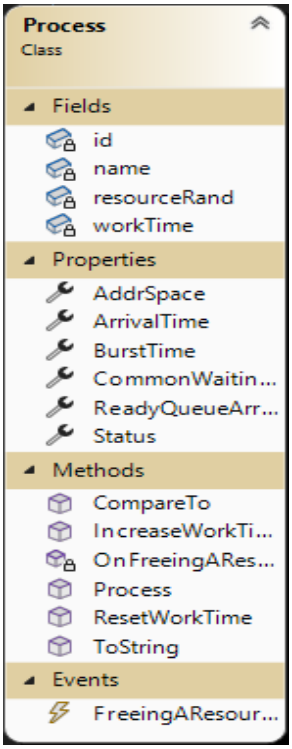

Рисунок 4.9 – Поля та властивості класу Process

У конструкторі (рис 4.10), він приймає інформацію, що генерується системою і ініціалізує процес. Чергою за замовчуванням є перша черга до ЦП.

```
public Process(long pId, long addrSpace)
        {
            id = pId;
            name = "P" + id.ToString();
            AddrSpace = addrSpace;
            Status = ProcessStatus.ready;
        }
```

Рисунок 4.10 – конструктор класу Process

Для інкременту часу роботи використовується метод IncreaseWorkTime (рис. 4.11). Цей метод виконує збільшення часу виконання роботи для об'єкта процесу. Нижче наведений опис його роботи:

1) Умова if (workTime == BurstTime) перевіряє, чи час виконання роботи (workTime) дорівнює загальному часу виконання (BurstTime).
2) Якщо умова виконується, перевіряється статус процесу (Status):
   - Якщо статус рівний ProcessStatus.running, то виконується наступна перевірка:
     - Генерується випадкове число у діапазоні 0-1 за допомогою resourceRand.Next(0, 2).
     - Якщо отримане число дорівнює 0, то статус процесу змінюється на ProcessStatus.terminated.
     - Якщо отримане число дорівнює 1, то статус процесу змінюється на ProcessStatus.waiting.
   - Якщо статус не рівний ProcessStatus.running (тобто ресурс є зовнішнім пристроєм), то статус процесу змінюється на ProcessStatus.ready.
3) Після зміни статусу процесу викликається метод OnFreeingAResource(), який сповіщає про звільнення ресурсу.
4) Якщо умова if (workTime == BurstTime) не виконується, то значення workTime збільшується на одиницю (workTime++).

```
public void IncreaseWorkTime()
{
 if (workTime == BurstTime)
   {
    if (this.Status == ProcessStatus.running)
   {
     this.Status = resourceRand.Next(0, 2) == 0 ?
ProcessStatus.terminated : ProcessStatus.waiting;
   }
    else
  {
     this.Status = ProcessStatus.ready;
  }
     OnFreeingAResource(); // зажечь событие
  } else workTime++;
```

Рисунок 4.11– Функція IncreaseWorkTime

Також у класі Process присутні допоміжні функції (рис. 4.12 – 4.14):

1) ResetWorkTime – зкидує лічильник робочого часу workTime до 0

```
public void ResetWorkTime()
{
  workTime = 0;
}
```

Рисунок 4.12 – функція ResetWorkTime

2) ToString – повертає дані про процес у виді форматованої строки

```
   public override string ToString()
   {
    return $"ID: {id} Name: {name} Status: {Status}
BurstTime: {BurstTime}";
   }
```

Рисунок 4.13 – функція ToString

3) CompareTo – прийимає аргумент у вигляді іншого процесу та повертає результат порівняння різниці BurstTime та workTime введеного процесу з поточним

```
public int CompareTo(Process other)
{
// Якщо аргумент не є посиланням на існуючий об'єкт,
об'єкт вважається більшим за аргумент..
 if (other == null) return 1;

 return (other.BurstTime -
other.workTime).CompareTo(this.BurstTime -
this.workTime);
}
```

Рисунок 4.14 – функція CompareTo

4.3.3 Клас SystemClock

Клас SystemClock (рис. 4.15) використовується для відстеження часу в системі. Його функціональність дозволяє збільшувати значення годинника, скидати його до початкового значення та повідомляти про зміни значень годинника для можливої обробки цих змін в інших частинах програми.

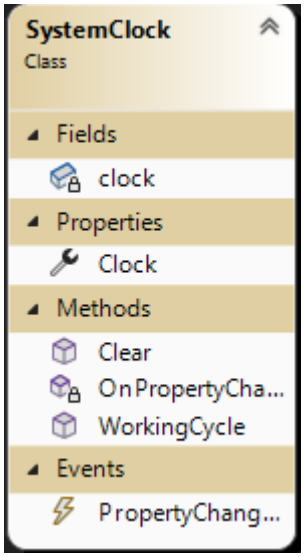

Рисунок 4.15 – Поля та властивості класу SystemClock

Клас SystemClock є реалізацією інтерфейсу INotifyPropertyChanged і представляє системний годинник.

Основні елементи класу:

1. Метод WorkingCycle (рис. 4.16) виконує цикл роботи системного годинника, де кожного разу значення годинника (Clock) збільшується на одиницю.

```
public void WorkingCycle()
{
  Clock++;
}
```

Рисунок 4.16 – Метод WorkingCycle

2. Метод Clear (рис. 4.17) встановлює значення годинника (Clock) на 0, тобто скидає його.

```
public void Clear()
{
            Clock = 0;
}
```

Рисунок 4.17 – Метод Clear

3. Властивість Clock (рис. 4.18) є доступною для зчитування і приватної для запису. Звернення до цієї властивості повертає значення годинника (clock), а запис нового значення спричиняє виклик методу OnPropertyChanged() для сповіщення про зміну властивості.

```
public long Clock
{
 get => clock;
 private set
 {
  clock = value;
  OnPropertyChanged();
 }
}
```

Рисунок 4.18 – конструктор Clock

4. Змінна clock відповідає за збереження значення годинника.
5. Клас має подію PropertyChanged, яка виникає при зміні властивостей. Ця подія є важливою для механізму прив'язки даних, що дозволяє іншим об'єктам реагувати на зміни значень властивостей.
6. Метод OnPropertyChanged() викликає подію PropertyChanged, передаючи ім'я зміненої властивості в якості параметру.

4.3.4 Клас Memory

Клас Memory (рис. 4.19) використовується для керування пам'яттю в програмі. Він надає можливість встановлювати загальний розмір пам'яті, визначати зайнятий розмір пам'яті і отримувати вільний розмір пам'яті.

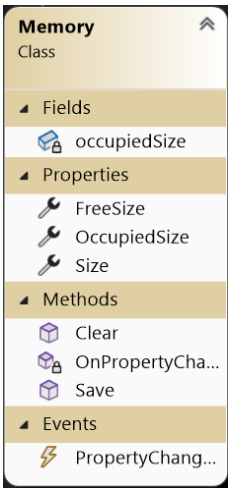

Рисунок 4.19 – Поля та властивості класу Memory

Основні елементи класу:

1) Метод Save (рис. 4.20) встановлює розмір пам'яті (Size) і скидає зайнятий розмір пам'яті (occupiedSize) на початкове значення 0.

```
public void Save(long size)
        {
            Size = size;
            occupiedSize = 0;
        }
```

Рисунок 4.20 – Метод Save

2) Метод Clear (рис. 4.21) скидає розмір пам'яті (Size) і зайнятий розмір пам'яті (occupiedSize) на 0.

```
public void Clear()
        {
            Size = 0;
            occupiedSize = 0;
        }
```

Рисунок 4.21– Метод Clear

3) Властивість Size (рис. 4.22) є доступною для зчитування та приватною для запису і представляє загальний розмір пам'яті.

```
public long Size
        {
            get;
            private set;
        }
```

Рисунок 4.22 – Властивість Size

4) Властивість FreeSize (рис. 4.23) є доступною для зчитування та приватною для запису і повертає вільний розмір пам'яті шляхом обчислення різниці між загальним розміром пам'яті (Size) і зайнятим розміром пам'яті (occupiedSize).

```
public long FreeSize
        {
            get{ return Size - occupiedSize; }
            private set { }
        }
```

Рисунок 4.23 – Властивість FreeSize

5) Властивість OccupiedSize (рис. 4.24) є доступною для зчитування та запису і представляє зайнятий розмір пам'яті. При записі нового значення властивості, виконується оновлення властивості FreeSize і викликається метод OnPropertyChanged().

```
public long OccupiedSize
{
 get { return occupiedSize; }
 set { occupiedSize = value;FreeSize = Size-occupiedSize;
OnPropertyChanged(); }
```

Рисунок 4.24 – Властивість FreeSize

Клас Memory використовується для керування пам'яттю в програмі. Він надає можливість встановлювати загальний розмір пам'яті, визначати зайнятий розмір пам'яті і отримувати вільний розмір пам'яті. За допомогою механізму прив'язки даних, інші частини програми можуть реагувати на зміни властивостей Size, FreeSize і OccupiedSize.

4.4 Графічний інтерфейс

Після запуску програми, користувачу показується вікно вибору алгоритму, симуляцію якого він збирається зробити. Інтерфейс вікна вибору представлено на рисунку 4.25.

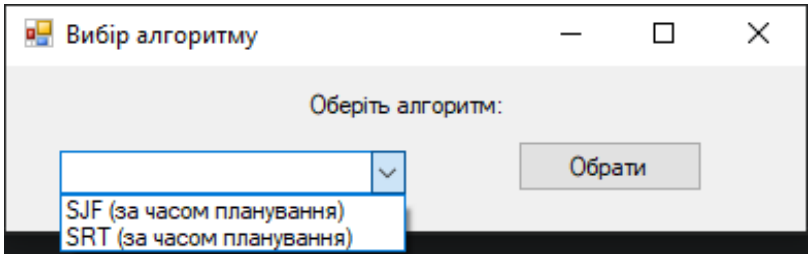

Рисунок 4.25 – Вікно вибору алгоритму

Обравши зі списку один з алгоритмів та натиснувши кнопку «Обрати», користувачу показується вікно симулятору відповідного алгоритму, який він обрав. Загальний інтерфейс симулятору роботи алгоритму планування процесору наведений на рисунку 4.26.

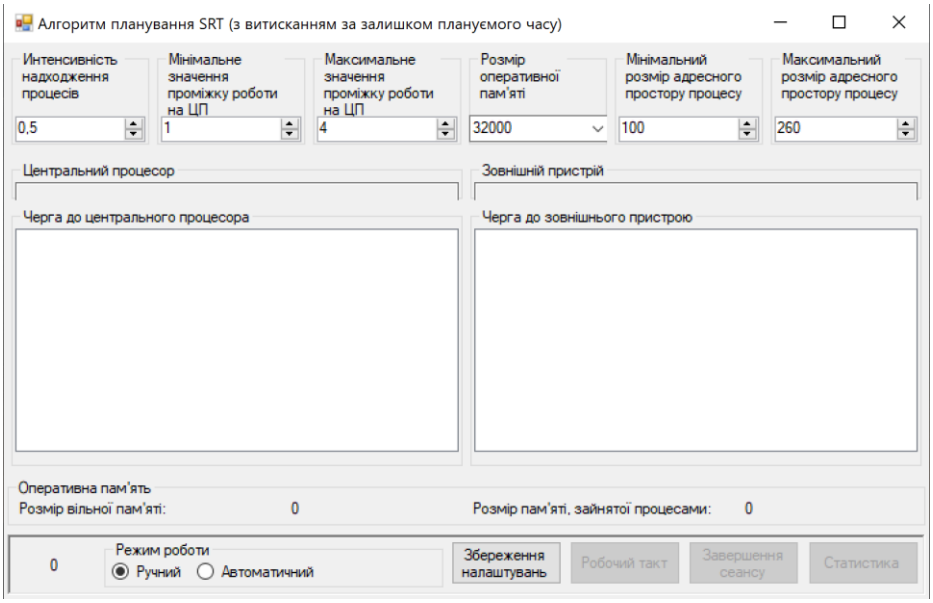

Рисунок 4.26 – інтерфейс симулятору роботи алгоритму SRT

У верхній частину вікна інтерфейсу присутні налаштування роботи алгоритму, які користувач може змінювати заради зміни поведінки алгоритму. Налаштування та їх можливі параметри наступні:

1) Інтенсивність надходження процесів – відповідає за те, з якою інтенсивністю поступають процеси у систему. Діапазон значень від 0.1 до 1.0, де при значенні 1.0 процес буде поступати кожен робочий такт
2) Мінімальне/максимальне значення проміжку роботи на ЦП – відповідає за час безперервного обслуговування процесу, який надійшов до ЦП. Діапазон значень від 1 до 3 для мінімального та від 4 до 10 для максимального
3) Розмір оперативної пам’яті – відповідає за загальну кількість пам’яті системи, куди треба містити процеси. Має два можливих значення – 32000 та 64000 мегабайт.
4) Мінімальний/максимальний розмір адресного простору процесу – відповідає за розмір пам’яті процесу, який треба виділити з оперативної пам’яті. Діапазон значень від 100 до 250 мегабайт для мінімального розміру та від 260 до 500 мегабайт для максимального розміру адресного простору процесу.

У середині вікну інтерфейсу можна побачити вікна ресурсів (ЦП та зовнішній пристрій), вікна черг до цих ресурсів та розмір вільної оперативної пам’яті та розмір пам’яті, який займають процеси на пристроях та у чергах. На рисунку 4.27 наведено, як процеси заповнюють ці ресурси та черги до них.

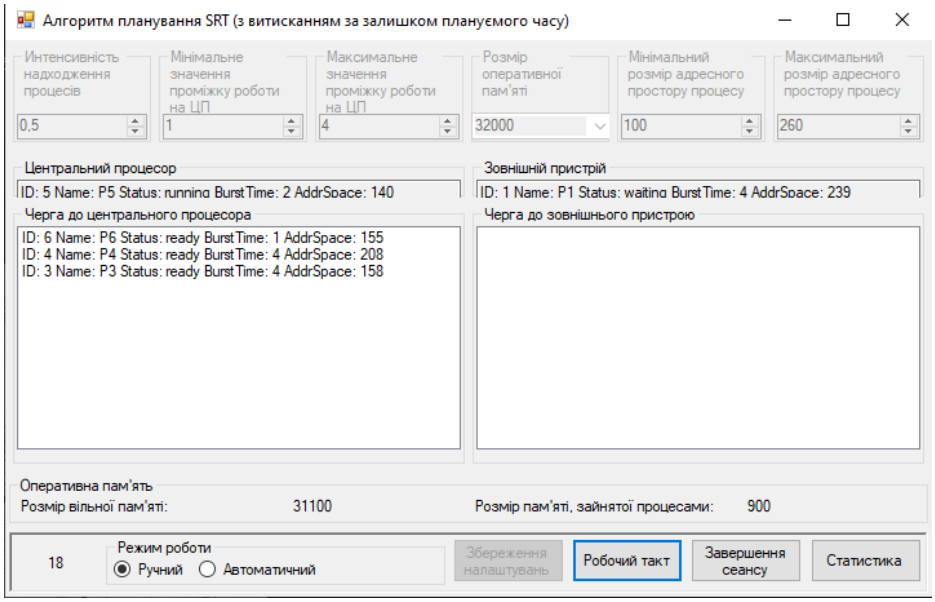

Рисунок 4.27 – наповнення процесами пристроїв та черг

У нижчій частині вікна інтерфейсу видно перемикач ручного та автоматичного режиму роботи симулятору. Різниця між ними полягає у тому, що у ручному режиму процеси будуть створюватись тільки при натисканні кнопки «Робочий такт», а у автоматичному вони будуть генеруватись самі.

Також є 4 кнопки, які відповідають за наступне:

1) Збереження налаштувань – зберігає параметри симулятору, які задав користувач. До її натискання ніякі кнопки неактивні.
2) Робочий такт – симулює робочий такт системи. Кнопка неактивна при автоматичному режиму роботи.
3) Завершення сеансу – скидає всі черги, пристрої та налаштування симулятору. Кнопка активна тільки після зберігання налаштувань
4) Статистика – відкриває вікно інтерфейсу статистики (рисунок 4.28)

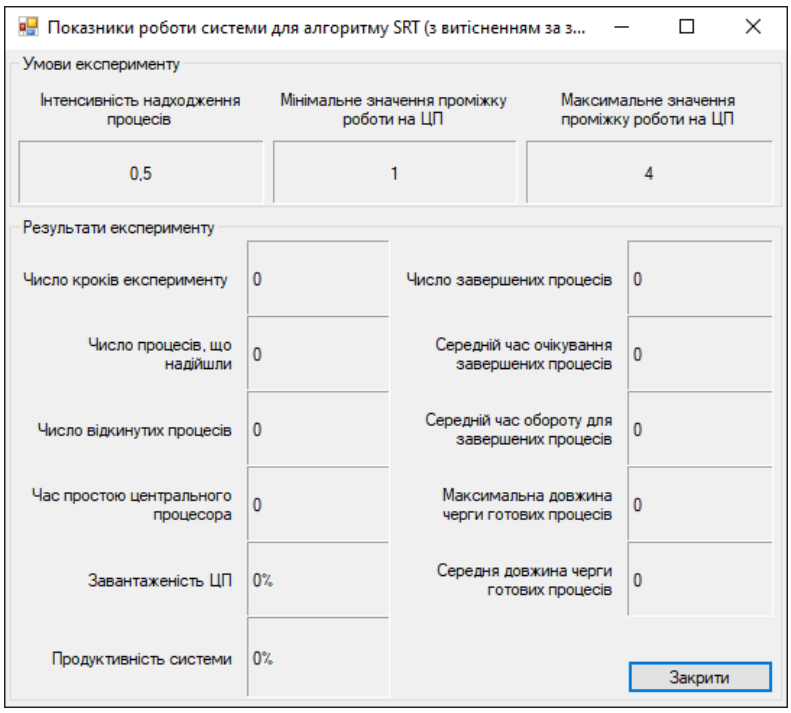

Рисунок 4.28 – інтерфейс вікна статистики

На цьому інтерфейсі видно налаштування симулятору, які задав користувач, та показники роботи системи, які оновлюються з кожним робочим тактом симулятору (рисунок 4.29). Показники, які збираються в цьому вікні, наведені у пункті 3.1.

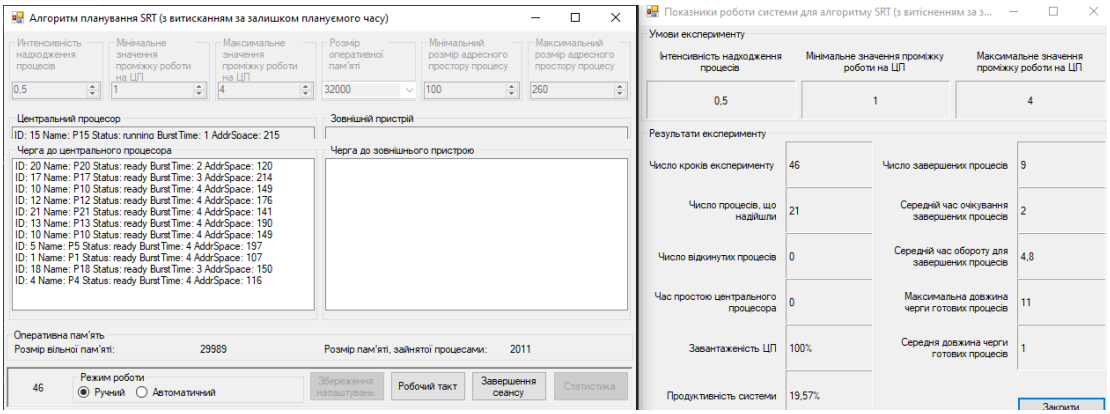

Рисунок 4.29 – демонстрація оновлення статистики

ВИСНОВКИ

В процесі виконання дипломної роботи було досліджено існуючи алгоритми планування процесору та принцип їх роботи, розглянуто існуючи симулятори роботи даних алгоритмів. Сформульовано вимоги до створення тренажеру для алгоритмів планування процесору.

Вирішено питання вибіру програмного забезпечення для розробки системи та схеми функціонування моделі обчислювальної системи.

Були зібрані показники роботи алгоритмів та візуалізовані за допомогою стовпчатої діаграми, яка надає можливість порівняння показників роботи алгоритмів

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. SimPy – Discrete event simulation for Python [Електроний ресурс] – Режим доступу: https://simpy.readthedocs.io/en/latest/
2. SPEC CPU Benchmark Suite [Електроний ресурс] – Режим доступу: https://www.spec.org/benchmarks.html#cpu
3. GEM5: The GEM5 simulator system [Електроний ресурс] – Режим доступу: https://www.gem5.org/
4. ДОСЛІДЖЕННЯ АЛГОРИТМІВ ДИСПЕТЧЕРИЗАЦІЇ В КОМП'ЮТЕРНИХ СИСТЕМАХ / В.Б. Кропивницька, Б.В. Клим, А.Г. Романчук, М.О. Слабінога // Івано-Франківск, ISSN 1993—9973, 2011. № 2(39)
5. Analysis and Comparison of CPU Scheduling Algorithms [Electronic book] / Pushpraj Singh, Vinod Singh, Anjani Pandey // Режим доступу - www.ijetae.com
6. Stephen F. Show Me The Numbers, Designing Tables and Graphs to Enlighten / Burlingame, California
7. Трубіна Н.Ф. Системне програмне забезпечення: конспект – О.: ОНУ, 2015
8. Мова програмування C# та платформа .NET [Електроний ресурс] – Режим доступу: https://metanit.com/sharp/
9. Desktop Guide (Windows Forms .NET) [Електроний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/overview/
10. Get started with ASP.NET Core MVC [Електроний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/start-mvc
11. MVC для веб: простіше нікуди [Електроний ресурс] – Режим доступу: https://habr.com/ru/articles/181772/

ДОДАТОК А

Візуалізовані показники роботи системи

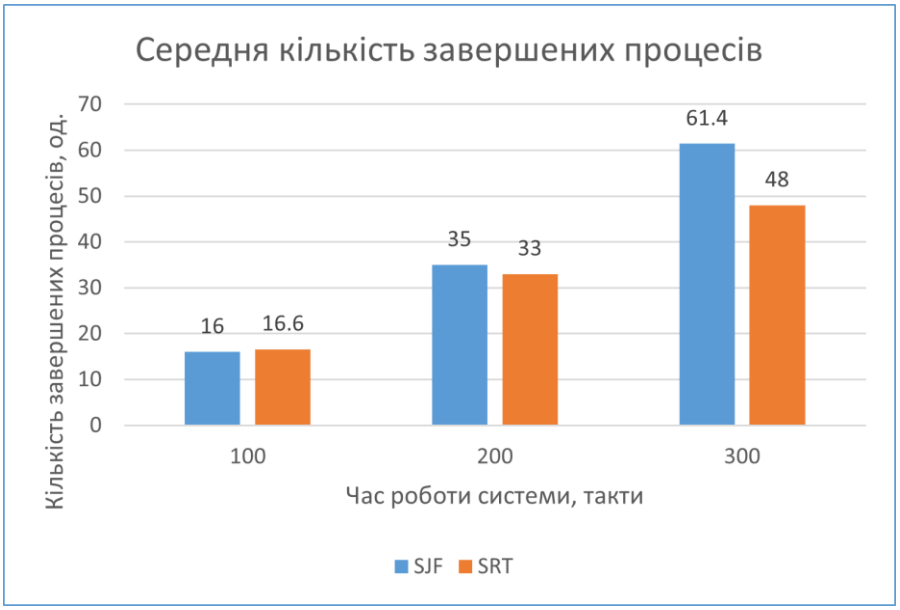

Рисунок А.1 – Графік середньої кількості завершених процесів, інтенсивність процесів 0.5

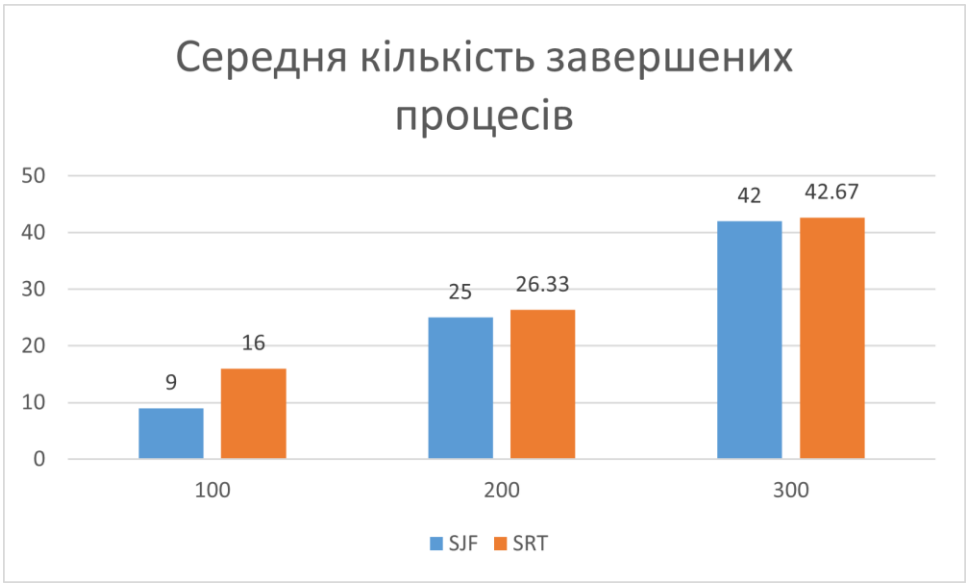

Рисунок А.2 – Графік середньої кількості завершених процесів, інтенсивність процесів 0.2

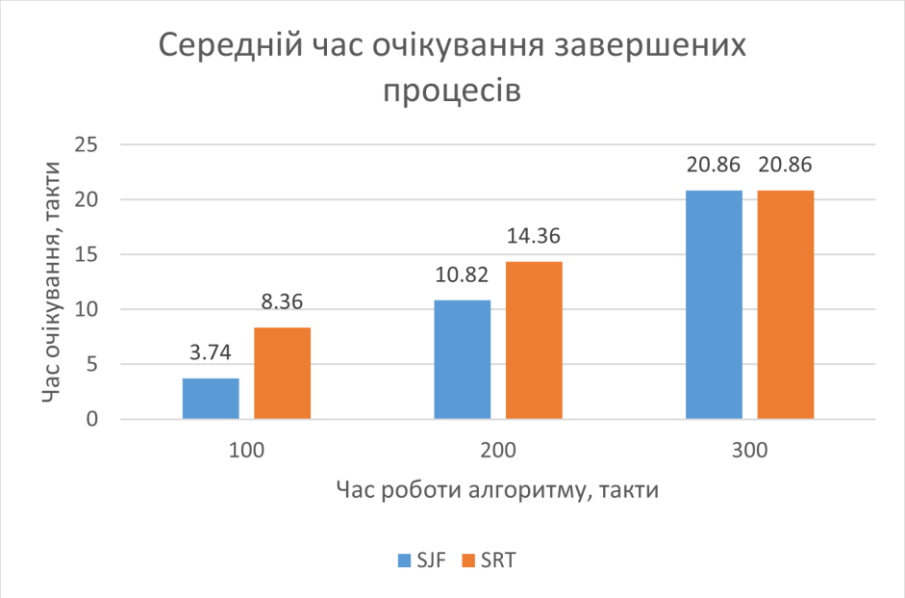

Рисунок А.3 – графік середнього часу очікування завершених процесів, інтенсивність процесів 0.5

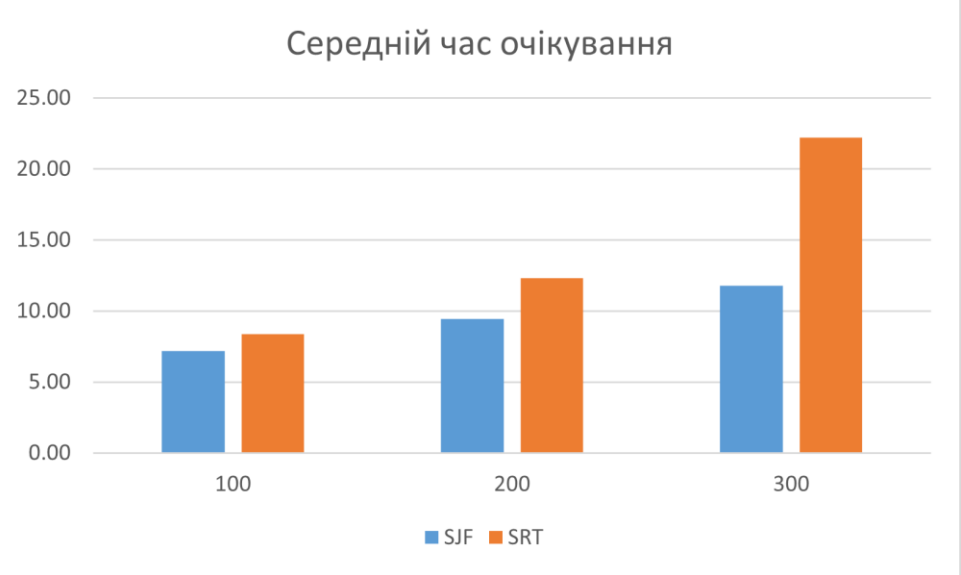

Рисунок А.4 - графік середнього часу очікування завершених процесів, інтенсивність процесів 0.2

ДОДАТОК Б

Моделі класів системи, їх властивості та методи

Таблиця Б.1 – Поля класу Statistics

Характеристика	Назва	Момент зміни
Умови експерименту		
Обрані параметри моделі	`Settings settings`	Незмінні протягом такту
Результати експерименту		
Число кроків експерименту	`long commonTime`	Кожен такт
Число процесів, що надійшли	`long arrivalProcessesCount`	Створення процесу
Число відкинутих процессов	`long rejectedProcessesCount`	При неможливості розміщення процесу у пам’яти
Час простою ЦП	`long cpuIdleTime`	Перевірка на кожному такті. Зміна, якщо процесор вільний
Число завершених процесів	`long terminatedProcessesCount`	Завершення роботи процесу
Загальний час очікування завершених процесів	`long commonWaitingTime`	Завершення роботи процесу
Загальний час обороту для завершених процесів	`long commonTurnAroundTime`	Завершення роботи процесу
Максимальна довжина черги готових процесів	`long maxReadyQueueLength`	Постановка процесу в чергу готових процесів

Таблиця Б.2 – Обчислювані властивості класу Statistics

Характеристика	**Назва**	**Момент зміни**
Завантаженість ЦП (відношення часу роботи процесора до числа кроків експерименту)	`double CpuUtilization`	Кожен такт
Продуктивність системи (відношення числа завершених процесів до числа кроків експерименту)	`double SystemPerformance`	Кожен такт
Середній час очікування завершених процесів (відношення загального часу очікування в черзі готових процесів до числа завершених процесів)	`double AvgWaitingTime`	Завершення роботи процесу
Середній час обороту для завершених процесів (відношення загального часу обороту до числа завершених процесів)	`double AvgTurnAroundTime`	Завершення роботи процесу
Середня довжина черги готових процесів (обчислюється за формулою Літтла)	`long AvgReadyQueueLength`	Прибуття процесу на процесор

Таблиця Б.3 – Методи класу Statistics

Назва	**Тип**	**Параметри**	**Опис**
Save	*void*	*Resource cpu double Intensity*	Ініціалізує поля cpu і intensity, необхідні для обчислення статистичних показників "час простою ЦП" і "середня довжина черги готових процесів"
Clear	*void*	Відсутні	Скидає значення всіх показників у 0

Таблиця Б.4 – Методи класу Model

Назва	**Тип**	**Параметри**	**Опис**
Model (конструктор)	-	Відсутні	Створює компоненти обчислювальної системи
SaveSettings	*void*	Відсутні	Запам'ятовує розмір оперативної пам'яті відповідно до налаштувань моделі та ініціалізує планувальник пам'яті
WorkingCycle	*void*	Відсутні	Моделює робочий такт обчислювальної системи: 1. виконується робочий такт годинника, якщо на цьому кроці процес має надійти в систему 2. створюється процес, якщо процесу виділяється пам'ять

Продовження таблиці Б.4

Назва	Тип	Параметри	Опис
WorkingCycle	*void*	Відсутні	3. генерується передбачуваний проміжок часу роботи процесу на процесорі (BurstTime) 4. процес стає в чергу готових процесів якщо процесор вільний 5. виклик планувальника процесора (метод Session поміщає процес із черги на пристрій) 6. робочий крок процесора (його метод WorkingCycle) 7. робочий крок зовнішніх пристроїв (їхній метод WorkingCycle)
Clear	*void*	Відсутні	Очищення (підготовка обчислювальної системи до наступного сеансу роботи) 1. видалення процесу з процесора 2. видалення процесу із зовнішнього пристрою 3. очищення пам'яті 4. очищення черги готових процесів 5. очищення черги до зовнішнього пристрою 6. скидання налаштувань

Таблиця Б.5 – Властивості класу FIFOQueue

Назва	**Тип**	**Опис**
Count	int	Повертає кількість елементів в черзі.

Таблиця Б.6 – Методи класу FIFOQueue

Назва	**Тип**	**Параметри**	**Опис**
FIFOQueue (конструктор)	-	structure: TStruct	Створює новий об'єкт черги, приймаючи екземпляр структури TStruct.
Item()	*TItem*	-	Повертає перший елемент з черги. Перевіряється, що черга не є порожньою перед доступом до елементу.
Remove()	*IQueueable <TItem>*	-	Видаляє перший елемент з черги. Перевіряється, що черга не є порожньою перед видаленням.
Put(TItem t)	*IQueueable <TItem>*	t: TItem	Додає новий елемент t до черги.
Empty()	bool	-	Перевіряє, чи є черга порожньою.
Clear()	*IQueueable <TItem>*	-	Очищає чергу, видаляючи всі елементи.
ToArray()	TItem[]		Повертає масив, який містить всі елементи черги.

Таблиця Б.7 – Методи класу MemoryManager

Назва	Тип	Параметри	Опис
Save()	void	Memory memory	Зберігає посилання на об'єкт типу Memory для подальшого використання.
Allocate()	*Memory*	long size	Виділяє пам'ять заданого розміру. Перевіряється, чи є вільна пам'ять достатньою для виділення. Якщо так, то збільшує зайнятий розмір пам'яті та повертає об'єкт типу Memory. Якщо недостатньо вільної пам'яті, повертає null.
Free()	*Memory*	long size	Звільняє пам'ять заданого розміру. Зменшує зайнятий розмір пам'яті та повертає об'єкт типу Memory.

Таблиця Б.8 – Поля та методи класу IdGenerator

Назва	Тип	Параметри	Опис
Id	*Властивість*	-	Повертає наступний доступний ідентифікатор. Якщо поточний ідентифікатор досягне максимального значення long.MaxValue, то повертається 0, інакше ідентифікатор збільшується на 1 та повертається.
Clear()	*Метод*	-	Скидає поточний ідентифікатор на 0. Повертає посилання на поточний об'єкт типу IdGenerator.

Таблиця Б.9 – Властивості класу Settings

Назва	**Тип**	**Опис**
Intensity	double	Інтенсивність поступлення процесів. Дозволяє отримати або встановити значення інтенсивності.
MinValueOfBurstTime	int	Мінімальне значення інтервалу обслуговування процесу процесором. Дозволяє отримати або встановити мінімальне значення.
MaxValueOfBurstTime	int	Максимальне значення інтервалу обслуговування процесу процесором. Дозволяє отримати або встановити максимальне значення.
MinValueOfAddrSpace	int	Мінімальне значення адресного простору процесу. Дозволяє отримати або встановити мінімальне значення.
MaxValueOfAddrSpace	int	Максимальне значення адресного простору процесу. Дозволяє отримати або встановити максимальне значення.
ValueOfRAM	int	Розмір оперативної пам'яті. Дозволяє отримати або встановити розмір пам'яті.