НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Кафедра інформатики та програмної інженерії

Курсова робота

З дисципліни «Моделювання систем»

|  |  |
| --- | --- |
| Керівник  д.т.н., проф. кафедри ІПІ  Стеценко І.В.  «Допущений до захисту»  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (особистий підпис керівника)  « »\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023р.  Захищений з оцінкою  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (оцінка)  Члени комісії:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (особистий підпис)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (особистий підпис) | Виконавець  ст. Кононов М. А.  гр. ЗПІ-зп01  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (особистий підпис виконавця)  « »\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023р.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (розшифровка підпису)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (розшифровка підпису) |

Київ – 2023

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

(назва навчального закладу)

Кафедра ІНФОРМАТИКИ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Дисципліна «Моделювання систем»

Курс 4 Група ЗПІ-зп01 Семестр VII

**ЗАВДАННЯ**

**на курсову роботу студента**

Кононова Максима Анатолійовича\_\_\_\_\_\_\_

(прізвище, ім’я, по батькові)

1.Тема\_роботи Розрахунок та моделювання завдання №15

2. Строк здачі студентом закінченої роботи \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Вихідні дані до роботи: На обробку до інформаційної системи приймаються три класи завдань А, В і С. Завдання класів А і В можуть вирішуватися інформаційною системою одночасно, а завдання класу С вимагають, щоб система була вільна від інших завдань. Завдання класу А надходять через 20±5 хвилин, класу В - через

20 ±10 хвилин і класу С - через 30±10 хвилин і потребують для виконання: клас А - 20±5 хвилин, клас В - 21±3 хвилин і клас С - 28±5 хвилин. Завдання класів А і В можуть дозавантажуватись до задачі, що вирішується.

Визначити завантаження інформаційної системи. Розглянути різноманітні засоби управління чергами і визначити той із них, при якому число завдань у чергах мінімально.

4. Дата видачі завдання\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| №,  п/п | **Назва етапів виконання курсової роботи** | **Строк**  **виконання**  **етапів роботи** | **Підписи або**  **примітки** |
| 1. |  |  |  |
| 2. |  |  |  |
| 3. |  |  |  |
| 4. |  |  |  |
| 5. |  |  |  |
| 6. |  |  |  |
| 7. |  |  |  |
| 8. |  |  |  |
| 9. |  |  |  |
| 10. |  |  |  |
| 11. |  |  |  |
| 12. |  |  |  |
| 13. |  |  |  |
| 14. |  |  |  |
| 15. |  |  |  |

**Студент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  Кононов Максим Анатолійович

(підпис) (прізвище, ім’я, по батькові)

**Керівник\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(підпис) (прізвище, ім’я, по батькові)

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023р.

**Анотація**

Курсовий проект виконується по завданню, що містить текстовий опис об'єкта моделювання і чисельні дані про змінні та параметри об'єкта моделювання, а також про впливи зовнішнього середовища, характеристики процесу функціонування об'єкта, що необхідно оцінити в процесі моделювання.

У ході виконання курсового проекту треба виконати формалізацію опису об'єкта моделювання в термінах відомих математичних схем, розробити алгоритм імітації роботи моделі, виконати перевірку алгоритму імітації, одержати статистичні оцінки заданих характеристик моделі, побудувати і провести факторний експеримент із заданою ціллю.

Після ряду прогонів моделі, яка побудована, і отримання результатів факторного експерименту з необхідною точністю треба провести їх інтерпретацію й аналіз у термінах об'єкта моделювання.

**ЗМІСТ**

ВСТУП

1. Постановка задачі.
2. Концептуальна модель системи.
3. Формалізація моделі системи.
4. Алгоритмізація моделі системи і її машинна реалізація.
5. Верифікація моделі.
6. Дослідження моделі.

ВИСНОВКИ

**ВСТУП**

Інформаційна система – це комунікаційна система, що забезпечує збирання, пошук, обробку та пересилання інформації. До неї можуть належати автоматизовані системи, комп'ютерні мережі або системи зв'язку. Через їх велику розповсюдженість та необхідність використання у майже всіх сферах нашого життя, актуальність досліджень у цій галузі залишатиметься на дуже високому рівні. Проблеми розробки оптимальних алгоритмів впорядкування завдань, організації черг або створення інших рішень для більш продуктивної організації роботи дуже часто постають перед розробниками сучасних інформаційних систем. На продуктивність цих пристроїв впливають багато факторів, які не завжди залежать від обчислювальної потужності та складності завдань. Для побудови високопродуктивних систем слід приділяти велику увагу оптимізації алгоритмів для організації та впорядкування завдань, встановленню пріоритетів.

Проектування ІС завжди розпочинається із визначення мети проекту, яку у загальному вигляді можна визначити як вирішення ряду взаємопов'язаних завдань, що включають в себе забезпечення запуску системи, її експлуатацію протягом певного часу. Серед цих завдань можна виділити такі:

1) визначення необхідної функціональності системи та рівня її адаптивності до постійно змінюваних умов функціонування;

2) необхідної пропускної спроможності системи;

3) необхідного часу реакції системи на запит;

4) безвідмовної роботи системи;

5) необхідного рівня безпеки;

6) простоти експлуатації та підтримки системи.

Відповідно до сучасної методології, процес створення ІС є процесом побудови і послідовного перетворення ряду узгоджених моделей на всіх етапах життєвого циклу (ЖЦ) ІС. На кожному етапі ЖЦ створюються специфічні для нього моделі (організації, вимог до ІС, проекту ІС, вимог до додатків тощо). Моделі формуються робочими групами команди проекту, зберігаються і накопичуються в репозиторії проекту. Створення моделей, їх контроль, перетворення і надання в колективне користування здійснюється із використанням спеціальних програмних інструментів.

Зазвичай виділяють такі етапи створення ІС: формування вимог до системи, проектування, реалізація, тестування, введення в дію, експлуатація та супровід.

1. Початковим етапом процесу створення ІС є моделювання бізнес-процесів, які мають місце в організації/на підприємстві, реалізують її цілі та завдання. Модель організації, описана в термінах бізнес–процесів і бізнес-функцій, дозволяє сформулювати основні вимоги до ІС.

2. На етапі проектування перш за все формуються моделі даних. Проектувальники в якості вихідної інформації отримують результати аналізу. Побудова логічної і фізичної моделей даних є основною частиною проектування бази даних. Отримана в процесі аналізу інформаційна модель спочатку перетвориться в логічну, а потім у фізичну модель даних. Етап проектування завершується розробкою технічного проекту ІС.

3. На етапі реалізації здійснюється створення ПЗ системи, встановлення технічних засобів, розробка експлуатаційної документації.

4. Етап тестування зазвичай виявляється розподіленим в часі.

Методологічну основу проектування ПЗ складає системний підхід, під час якого реалізують подання складного об'єкта у вигляді ієрархічної системи взаємопов'язаних моделей (останні дозволяють фіксувати цілісні властивості об'єкта, його структуру і динаміку). Проектування ПЗ має вигляд процесу створення специфікацій ПЗ на основі вихідних вимог до нього і зводиться до послідовного уточнення його специфікацій на різних стадіях процесу створення ПЗ. Невід'ємними властивостями ПЗ є складність, узгодженість, змінність і невидимість.

Отже, проблеми проектування оптимальних інформаційних систем є глобальними, головна ціль нашої роботи – дослідити лише незначну кількість з них. Головним кроком для досягнення цієї мети є розробка ПЗ для імітації роботи реальної системи із заданими умовами, аналізу способів збереження завдань, які не можуть виконуватись одночасно, проблеми їх впорядкування, пошук оптимальних рішень для організації черг, які сприяли б зменшенню їх розмірів.

**Постановка задачі**

На обробку до інформаційної системи приймаються три класи завдань А, В і С. Завдання класів А і В можуть вирішуватися інформаційною системою одночасно, а завдання класу С вимагають, щоб система була вільна від інших завдань. Завдання класу А надходять через 20±5 хвилин, класу В - через 20±10 хвилин і класу С - через 30±10 хвилин і потребують для виконання: клас А - 20±5 хвилин, клас В - 21±3 хвилин і клас С - 28±5 хвилин. Завдання класів А і В можуть довантажуватись до задачі, що вирішується.

Визначити завантаження інформаційної системи. Розглянути різноманітні засоби управління чергами і визначити той із них, при якому число завдань у чергах мінімально.

**Концептуальна модель системи**

Цілі нашої роботи визначити завантаження інформаційної системи. Розглянути різноманітні засоби управління чергами і визначити той із них, при якому число завдань у чергах мінімально.

В даній імітаційній моделі (рисунок 1) маємо блоки:

* надходження завдань класу А;
* надходження завдань класу В;
* надходження завдань класу С;
* умова обробки завдань класу А та класу В;
* умова при якій буде оброблятися завдання класу С;
* процеси виконання завдань відповідних класів;
* завдання залишають систему.

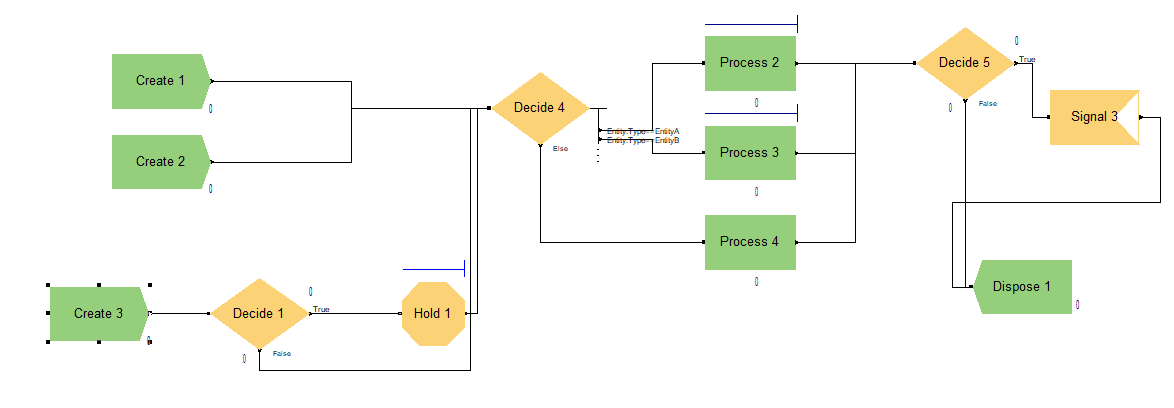


Рисунок 1. Імітаційна модель інформаційної системи

**Формалізація моделі системи**

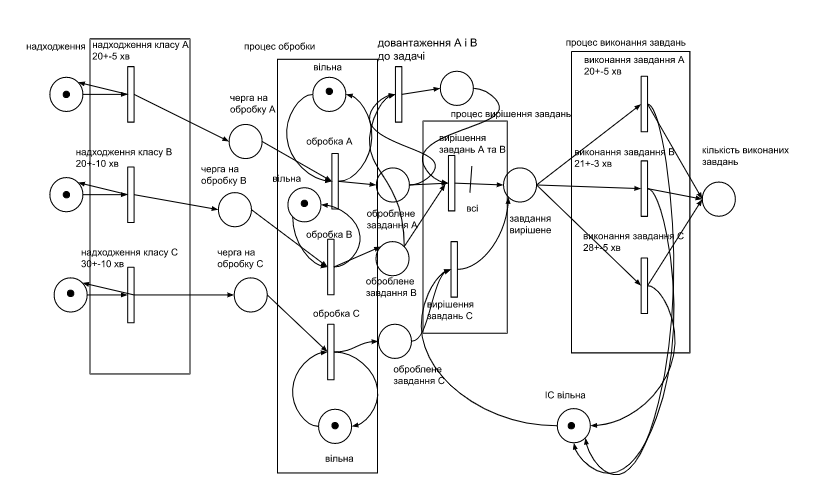


Рисунок 2. Мережа Петрі, що представляє формалізовану модель інформаційної системи

Виділимо події, які відбуваються в інформаційній системі:

* “надходження завдань класу А”;
* “надходження завдань класу В”;
* “надходження завдань класу С”;
* “обробка”;
* “довантаження А і В до задачі”;
* “вирішення задачі”;
* “виконання завдання А”;
* “виконання завдання В”;
* “виконання завдання С”;

Події “надходження” виконуються незалежно від стану системи, тому у вхідних позиціях повинна бути відповідна кількість маркерів. За умовою задачі на обробку до ІС приймається три класи завдань, отже на виході з переходу матимемо три дуги, де об’єктом є наші класи завдань відповідно А, В та С. Подія “обробка” виконується при умовах:

* в чергах є завдання;
* система вільна.

Подія “довантаження А і В до задачі” , якщо є оброблені завдання А і В. Подія “вирішення задачі” спрацює при умові, що в оброблених завданнях є завдання, а якщо це завдання С то ще повинна бути виконана умова, що система вільна. Подія “виконання завдань” виконується при умові, що відповідне завдання вирішене.

Для переходів задають такі часові затримки:

* “надходження завдань класу А” - 205 хв ;
* “надходження завдань класу В” - 2010 хв;
* “надходження завдань класу С” - 3010 хв;
* “обробка” - без затримки;
* “довантаження А і В до задачі” - без затримки;
* “вирішення задачі” - без затримки;
* “виконання завдання А” - 205 хв;
* “виконання завдання В” - 213 хв ;
* “виконання завдання С” - 285 хв .

З’єднуючи події та умови отримуємо мережу Петрі, зображену на рисунку 2.

**Алгоритмізація моделі системи і її машинна реалізація**

Для реалізації нашої імітаційної моделі нами було розроблено спеціальне ПЗ на мові програмування високого рівня C++. Використання саме цієї мови обумовлено наявністю необхідних навичок роботи з нею, потребою створення рішення, яке здатне використовувати ресурси системи найоптимальнішим чином. Наш код, написаний на C++, не потребує наявності віртуальних машин або додаткового ПЗ для виконання, може бути легко оптимізований для використання на будь-якій ОС.

Для просування модельного часу ми використовували спосіб найближчої події, основною ідеєю якого є змінювання стану системи тільки в деякі моменти часу. Реалізація цього принципу вимагала від нас побудови спеціального блоку коду для визначення моменту початку найближчої події, оповіщення користувача про її виникнення та обчислення часу наступного етапу виконання задачі. Для просування стану моделі в часі ми реалізували спосіб, орієнтований на події, тобто ми здійснювали імітацію виконання упорядкованої в часі послідовності логічно взаємозв’язаних подій.

Для вирішення усіх поставлених задач ми використовували об’єктно-орієнтований підхід. Усі об’єкти системи організовані у вигляді класів, які наслідують певні властивості один від одного. Зупинимось на описі кожного з них.

Клас task є базовим для опису всіх подій, які виникають в системі. Його змінні містять інформацію про тривалість завдання (TaskTime), момент виникнення наступної події (NextTime), поточний прогрес виконання (TaskProgress), розмір кроку просування часу (StepSize), після якого змінюється стан системи, «активний» час (ActiveTime), в який накопичується реальна тривалість виконання задачі. Крім того, об’єкти цього класу містять змінні, необхідні для візуалізацій процесу у вигляді «progress bars», за допомогою яких користувач отримує інформацію про поточний стан задачі, що виконується. Основними подіями, які виникають при функціонуванні об’єктів класу task є її надходження в систему, етапи виконання, кожен з яких, становить п’яту частини тривалості задачі, тобто StepSize=TaskTime/5, та завершення виконання задачі, протягом якого індикатор на інтерфейсі користувача переходить у стан «ABSENT», вказуючи на відсутність активних задач певного класу в поточний момент часу.

За умовою завдання даної курсової роботи, одночасно в системі можуть знаходитись лише задачі класів А та В, завдання класу С потребують використання усіх наявних ресурсів системи. Для контролю виконання даної вимоги в головному циклі моделювання нами було передбачено спеціальний блок коду, зовнішній вигляд якого представлений на рисунку 3. Він унеможливлює подальше функціонування системи у випадку порушення цієї фундаментальної вимоги, що супроводжується сповіщенням користувача відповідним надписом на екрані.

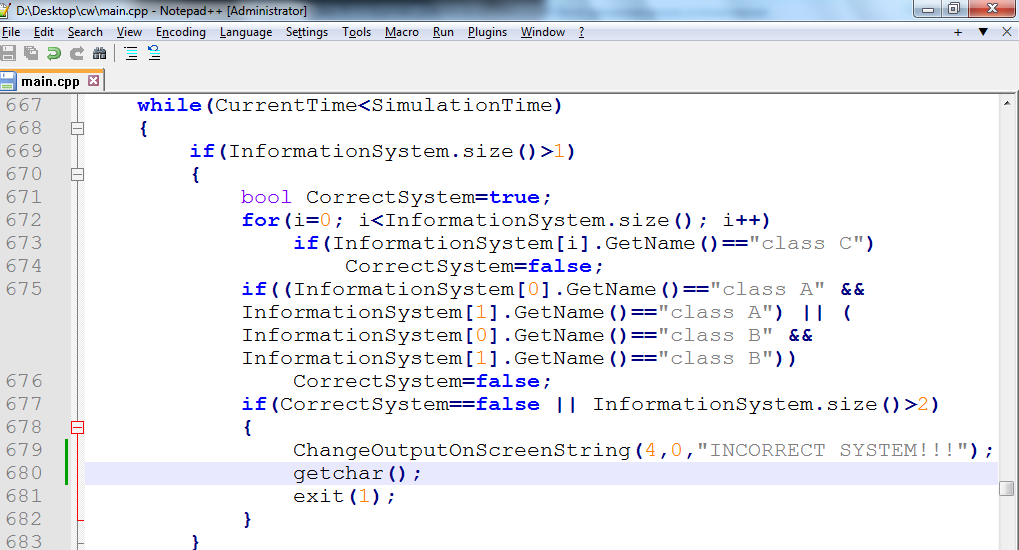


Рисунок 3. Блок коду для перевірки коректності функціонування системи.

Під час розробки програми ми часто стикались з «несумісністю» одночасного виконання названих класів завдань, але на даному етапі розвитку проекту колізії не трапляються, тому користувач не отримує сповіщення про збій.

Джерелами завдань, які надходять до системи, є об’єкти класу TaskCreator. Вони генерують вимоги, тривалість та інтенсивність надходження яких має відповідати вказаним у завданні до курсової роботи. За умовою, завдання класу А надходять через 20±5 хвилин. Отже, інтервал виникнення події, результатом якої є створення екземпляру класу task, нами був описаний як випадкове число з плаваючою точкою в діапазоні 15.0-25.0. Подібним чином були створені «CreatorsOfTasks» для класів В і С, вони зберігаються у контейнері типу vector. Розмір часу, необхідний для виконання відповідних класів завдань генерується подібним чином. Об’єкти класу TaskCreator наслідують властивості класу task, оскільки вони функціонують у системі також за принципом найближчої події.

Особливу увагу в даній роботі було приділено системі управління чергами, а також експериментам із встановленням різних пріоритетів, за якими завдання потрапляють із «сховища» в систему. Нами було реалізовано два основних типи черг, які можуть працювати за різноманітними методами. Базовим класом, який описує структуру даних, в яку потрапляють задачі, що не можуть виконуватись в поточний момент часу, є QueueParent. Він містить змінні, які описують середній та максимальний розміри черги (MaxQueue, AverageQueue виникає на етапі збереження результатів у файл). Крім того, екземпляр даного класу має масив цілочислових змінних, в якому фіксується кількість наборів завдань типів А, В та С, що потрапляли в чергу. Це потрібно для визначення класу, завдання якого частіше потрапляють у чергу, та виконання заходів для оптимізації системи. Чим менше завдань опиняються в сховищі, тим ефективніше працює система та менше ресурсів потрібно для їх збереження. Для обчислення середнього розміру черги використовується vector, в який потрапляють усі розміри, які отримувала черга під час імітації системи.

Клас OneQueue містить queue<task> QueueOfTasks, в яку при колізії потрапляють завдання усіх типів. Оскільки черга функціонує за принципом «перший прийшов – перший вийшов», такий спосіб організації сховища не є оптимальним через важкість або навіть неможливість оптимізації під поточні потреби системи. Найголовнішими змінними в даному класі є посилання на функції, за допомогою яких здійснюється отримання або видалення завдань з черги. Ми проаналізували результати роботи даного типу черги, використовуючи лише два методи для Get та Remove. Перша комбінація методів, GetTask\_WithoutExtraction та RemoveTask\_WithoutExtraction не виконують нічого, окрім повернення нульових значень. Вони потрібні для отримання даних про можливі стани системи без продуктивного використання черги. Друга група методів, GetTask\_SimpleQueue разом з RemoveTask\_SimpleQueue, повертає та, у випадку успішної інтеграції в систему, видаляє завдання зі сховищ.

Подібним чином працює екземпляр класу OneQueueForEachClass. Він описує окремі сховища для кожного типу завдань як масив черг (vector<queue<task>>). У якості параметрів його конструктор отримує посилання на «користувацькі» методи для отримання та завантаження об’єктів у черги. Така організація сховища є більш придатною для оптимізації, створює більший простір для подальших експериментів. Інші методи, RemoveTask, GetSize та clear не потребують «перезавантаження».

Для додавання завдань певних класів у відповідні черги ми розробили три методи: AddTask\_DefaultMethod, AddTask\_WithMerging, а також AddTask\_WithMergingIncludingClassC. Перший з них просто виконує операцію «push» у відповідні чергу масиву QueuesOfTasks. Другий – виконує ту саму дію, якщо черга порожня, а у випадку існування задач подібного класу здійснює злиття, додаючи новий TaskTime. За умовою, завдання класів А і В можуть довантажуватись до задачі, що вирішується, тому ми провели серію відповідних експериментів із «доповненням» існуючих задач у чергах. Третій метод AddTask\_WithMergingIncludingClassC порушує цю умову, об’єднуючи завдання всіх класів у відповідних чергах. На нашу думку, він також потрібен для прогнозування можливих станів, які можуть виникнути у випадку реорганізації системи.

Для аналізу впливу пріоритетів ми розробили три методи типу «Get», за якими завдання «виймаються» з відповідних черг. Вони мають назви GetTask\_PriorityForClassA, GetTask\_PriorityForClassB та GetTask\_PriorityForClassC. Їх головна ідея ґрунтується на тому, що після отримання запиту першочергово вони намагаються повернути завдання відповідного класу, у гірших випадках виконують операцію «QueuesOfTasks[і].front()» з черг для інших класів. Оскільки завдання класу С виявились найбільш «нестандартними», ми організували дані функції так, щоб вони знаходились на першому, другому та третьому місці у пріоритеті до потрапляння в систему.

Ще два Get-методи, GetTask\_MinQueue та GetTask\_MaxQueue повертають завдання із сховищ максимального або мінімального розміру. Вони також використовуються в інших функціях.

Остання група підпрограм, за допомогою яких отримуються вимоги зі сховищ, базується на поточному стані системи. Оскільки масив InformationSystem є глобальною змінною, відповідні інструкції отримують дані про його вміст, та в залежності від нього формують пріоритет, за яким завдання потрапляють у систему. Нами було реалізовано чотири варіанти даного підходу, скомбінувавши між собою:

1. пріоритет на більшу завантаженість системи: у першу чергу «виймаються» завдання тих класів, які можуть виконуватись паралельно, якщо в поточний момент часу система містить один набір задач класу А або В. У гіршому випадку повертаються вимоги з мінімальної або максимальної черги.
2. Пріоритет на злиття задач після їх отримування системою: у випадку завантаженості системи на 50% метод першочергово надає завдання того самого класу, яке виконуються в даний момент. У гіршому випадку підпрограма повертає задачі із сховища максимального або мінімального розміру.

Розроблена нами програма також надає інформацію про поточну завантаженість системи: у режимі покрокового виконання користувач бачить значення відповідної змінної. У випадку відсутності задач, завантаженість складає 0%, якщо виконується один набір задач класу А або В, load має значення 50%. Одночасне виконання вимог типів А та В або наявність задач класу С призводять до повної завантаженості системи.

У процесі розробки та організації коду ми дійшли висновку, що об’єктно-орієнтовані засоби мови програмування C++, зокрема наслідування, віртуальні та перевантажувальні функції, об’єкти та класи, виявились надзвичайно зручними інструментами для вирішення всіх поставлених задач. Використання цих засобів сприяють зменшенню повторних блоків коду, покращують читабельність, надають засоби для легкої масштабованості продукту. Можна легко інтегрувати нові методи для роботи з чергами, провести додаткові експерименти з використанням їх різних комбінацій, побудувати власну структуру або тип даних. Все це створює величезний простір для нових експериментів.

**Верифікація моделі**

Перевірку коректності реалізованих алгоритмів ми виконуємо шляхом наочності, виводячи на екран назву та номер експерименту, поточний час, стан трьох класів задач, розмір черги, поточне завантаження, перелік подій, які відбуваються на кожному кроці імітації. Зміни станів системи користувач бачить після натискання клавіші Enter. Зовнішній вигляд вікна роботи системи представлений на рисунках 4-7. Для його коректного відображення в налаштуваннях консолі має бути встановлена ширина 91 символ, у іншому випадку вивід може виявитись некоректним або «поплинути».

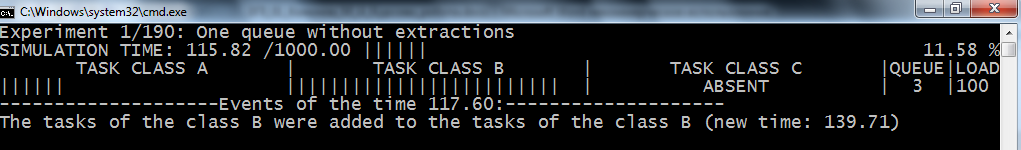


Рисунок 4. Система навантажена на 100%.

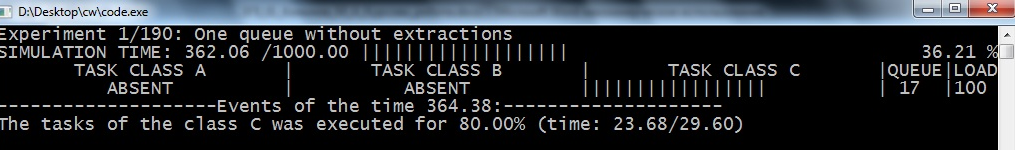


Рисунок 5. Виконання завдань класу С

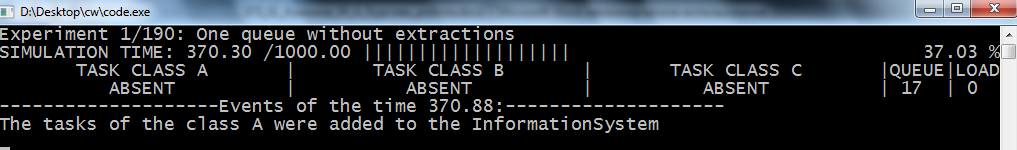


Рисунок 6. У поточний момент часу система вільна, наступним кроком стане надходження задач класу А

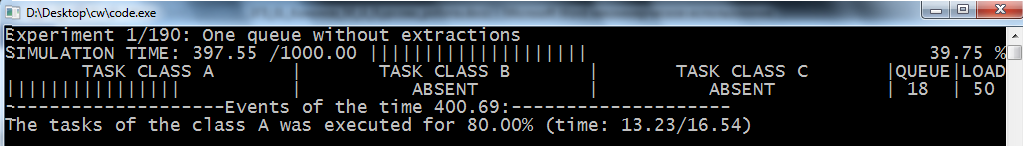


Рисунок 7. Система завантажена лише на 50% виконанням завдань одного класу

Із наданих зображень ми робимо висновок, що візуалізація виконується коректно. У першому рядку можна побачити, що всього програмою має виконатись 190 експериментів. Така кількість пояснюється проведенням кожного з 19 різних випробувань 10 разів. Оскільки виконання всіх алгоритмів при увімкненому виводі на екран займає дуже багато часу, ми проводили всі експерименти в автоматичному режимі без відображення проміжних результатів імітації. У такому режимі ці 19 типів процесів з різними параметрами черги виконуються у двох режимах: із увімкненою можливістю злиття однотипних завдань класів А і В, та без об’єднання вимог, що в певній мірі може порушувати одну з умов даної роботи. Отже, загальна кількість всіх проведених випробувань становить 380.

Результати всіх експериментів записуються у файл statistics.csv. Він представляє собою таблицю, яку легко відкрити у відповідній програмі одного з відомих офісних пакетів для подальшого аналізу даних, побудови графіків тощо. Її зовнішній вигляд показаний на рисунку 8.

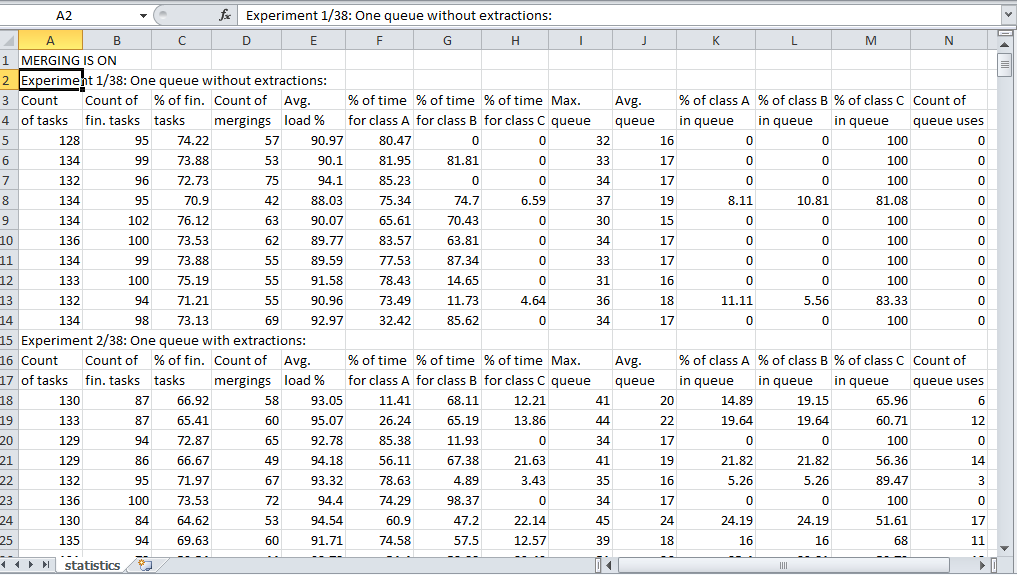


Рисунок 8. Результати експериментів у вікні табличного процесора

У таблицях ми фіксуємо такі показники як кількість створених та виконаних завдань, їх процентне співвідношення, кількість операцій злиття, середня завантаженість системи, процент часу, який було витрачено на виконання завдань усіх трьох класів, максимальний та середній розміри черги, процентна частка різних класів задач, які надходили до черги, кількість її використань.

Над єдиною чергою ми виконували лише два експерименти, які були детально описані в попередньому розділі. Результати верифікації даних типів сховищ представлені на рисунках 9-10.

Рисунок 9. Максимальний розмір черги

Рисунок 10. Середній розмір черги

Ми бачимо, що у випадках невикористання черги її розміри менші за тих, які виникають при роботі «класичного» метода роботи з нею. На перший погляд ми мали побачити протилежну картину, тому що використання черги сприяє зменшенню її розмірів. Але не слід забувати, що завдання видаляються з черги лише у випадку успішної інтеграції в систему, а отримати доступ до більш підходящих завдань при такому підході неможливо. При роботі методу GetTask\_WithoutExtraction нові завдання не потрапляють в систему зі сховищ, рідше породжуючи колізії та зменшуючи ймовірність потрапляння новоствореної задачі в чергу. GetTask\_WithExtraction виконує протилежні дії, частіше породжуючи колізії, що сприяє частішому потраплянню нових задач у сховище.

Рисунок 11. Порівняння завантаженості системи

Ми бачимо, що використання традиційного алгоритму роботи з чергою сприяє більшій завантаженості системи, але меншій продуктивності через більший розмір черги, що видно на рисунку 12.

Рисунок 12. Процент завершених завдань

Було встановлено, що менший розмір черги сприяє більшій продуктивності. Отже, ми робимо висновок, що потрапляння нових завдань лише з одного джерела, тобто невикористання черги для покращення продуктивності, більш позитивно впливає на стан системи.

Спробуємо проаналізувати вплив пріоритету, за яким завдання потрапляють на виконання зі сховища. Було встановлено, що пріоритет на отримання завдання класу С сприяє збільшенню розмірів черги. Це видно на рисунку 13.

Рисунок 13. Максимальний розмір черги

Такий феномен пояснюється тим, що задачі класу С не можуть бути об’єднані та не допускають паралельного виконання інших типів завдань. При появі нових вимог це призводить до колізій, сприяє більш ймовірному потраплянню у сховище.

Діаграму процентного співвідношення різних класів завдань, які потрапляють у чергу ми бачимо на рисунку 14.

Рисунок 14. Процентне співвідношення класів завдань, які потрапляють в чергу

Ми бачимо, що завдання класу С найчастіше потрапляють у чергу через їх вже названі обмеження. Це підтверджує й наступний графік на рисунку 15.

Рисунок 15. Співвідношення часу, витраченого на виконання різних типів завдань

Ми бачимо, що завдання класів А і В у найбільшій мірі завантажують систему, клас С займає дуже незначний процент модельного часу.

Дослідження системи за допомогою методів, в основі яких лежать орієнтованість на поточний стан системи у більшості випадків показують подібні результати. Але комбінація push-методу, який базується на злитті завдань класів А і В при додаванні у чергу, з get-методами, в основі яких лежить пріоритет як на злиття завдань, так і на паралельне виконання вимог типів А і В, що в гіршому випадку отримують завдання з черги максимального розміру, дають суттєве зменшення середнього та максимального розмірів сховищ. Це видно на рисунку 16.

Рисунок 16. Максимальний розмір черг при використанні певної комбінації методів, орієнтованих на поточний стан системи

Це свідчить про те, що злиття задач у чергах, та подальше їх завантаження із сховищ максимальних розмірів дають суттєве зменшення черги та сприяють більшій продуктивності. Підвищення проценту виконаних завдань видно на рисунку 17.

Рисунок 17. Процент виконаних завдань

Такі результати пояснюються тим, що з максимальної черги отримуються завдання класу С і в цьому випадку вони використовують найбільшу кількість системних ресурсів, що видно на рисунку 18.

Рисунок 18. Процент модельного часу, який використовується для виконання задач класу С

На основі таких даних можна сформулювати висновок, що для підвищення кількості виконаних задач класу С слід виконувати злиття вимог класів А і В у момент їх потрапляння в чергу, а при виникненні події типу «get» завантажувати завдання із черги максимального розміру, яка у переважній більшості випадків належить класу С. Більшість результатів проведених досліджень є неочевидними, та інколи наводять на думку, що реалізовані нами алгоритми функціонують неправильно та в реальній системі ми мали отримати протилежні результати. Але дослідивши сутність процесів можна дійти висновку, що отримання задач із черги сприяє збільшенню кількості колізій та більш ймовірному потраплянню нових вимог у сховище. Найбільш часто це трапляється із класом С. Але певна комбінація названих методів та реакція черги на поточний стан системи інколи дають суттєве покращення продуктивності, сприяють скороченню кількості задач у сховищах.

**Дослідження моделі**

Продуктивність системи та кількість завдань у чергах у найбільшій мірі залежать від реалізованих алгоритмів та способів організації сховищ, користувач не має доступу до параметрів, зміна яких може впливати на стан системи. Інтервали та час, необхідний для виконання вимог певних типів є випадковими величинами, які задаються лише в чітко фіксованому діапазоні. Тому ми робимо висновок, що розроблена нами модель є найбільш чутливою до методів, які становлять алгоритми управління чергами. Продуктивність деяких з них зумовлена поточним станом системи. При дослідженні єдиної черги головним фактором, який впливає на стан системи є перший елемент у сховищі. Саме від його класу залежить те, чи буде задача завантажена в систему зі сховища та чи не призведе наявність цієї задачі до виникнення подальших колізій при регулярному надходженні нових завдань різних типів. Схожу картину спостерігаємо із дослідженням іншого способу організації черг, який базується на тому, що вимоги відповідних класів мають власні сховища. Подальший стан, а, отже, й продуктивність всієї системи повністю обумовлені змістом перших об’єктів черг.

Тривалість завдань, інтервал їх надходження в систему, ймовірність діставання набору вимог того чи іншого класу, які в залежності від станів системи можуть у різній мірі впливати на продуктивність, становлять фактори, від яких залежать показники системи. У попередньому розділі нами була встановлена комбінація методів, за допомогою яких ми отримуємо найменшу кількість завдань в системі. Головною рисою цих функцій є орієнтованість на злиття подібних завдань класів А і В, врахування поточного стану системи та першочергове повернення задач найбільш «проблемного» класу. Детальні статистичні дослідження та факторні експерименти потребують можливість зміни параметрів. У нашому випадку зміна параметрів призведе до порушення умов поставленої задачі. Ми дійшли висновку, що найголовніший вплив на продуктивність системи, мінімальні чи максимальні розміри черг, здійснюють саме обрані способи організації сховищ, а також ймовірність надходження з черги більш придатних завдань.

**ВИСНОВКИ**

У ході виконання даної роботи ми побудували імітаційну модель інформаційної системи, яка є орієнтованою на події. Ми дослідили два основних способи організації черг, до яких завдання потрапляють при неможливості виконання в поточний момент часу. Головною метою дослідження було дати відповідь на запитання, в якому випадку система отримує мінімальний розмір черги завдань, які надходять із заданою регулярністю. Нами була встановлена комбінація методів, які призводять до цього результату. Окрім стандартних експериментів у рамках заданих вимог нами були виконані випробування, які порушують задані умови. Зокрема був реалізований метод для злиття у чергах завдань класу С. Це призвело до отримання найнижчих показників максимального розміру черги – лише 3 одиниці. На нашу думку, такий експеримент дає відповідь на запитання, яких результатів може досягти система у випадку її реорганізації та уможливлення об’єднання задач класу С. Ми можемо стверджувати, що зміна такої умови призведе до значного скорочення ресурсів, необхідних для організації сховища вимог.

Наступною важливою серією експериментів є виконання імітації без здійснення злиття вимог класів А і В після їх породження об’єктом відповідного класу. При використанні такої опції ми не помітили суттєвого впливу об’єднання однотипних завдань на продуктивність інформаційної системи. Це пояснюється тим, що час, необхідний для виконання завдань, не зменшується. У кількох випадках було помічено збільшення максимального та середнього розміру черги, що є результатом більш частого виникнення колізій у зв’язку з неможливістю паралельного виконання деяких класів задач.

Для покращення результатів функціонування системи ми могли б дати пораду розглянути можливість об’єднання завдань класу С, або уможливити їх виконання паралельно з іншими класами задач, що може призвести до збільшення необхідних обчислювальних потужностей. На нашу думку, така реорганізація призведе до значного спрощення алгоритмів управління чергами, а також відкриє новий простір для подальших експериментів з оптимізації системи.