Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

АТЕСТАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Методи резервування обчислювальних ресурсів для забезпечення живучості програмного забезпечення

(тема)

Виконав:
студент II курсу, групи СПм-19-1
Калюжний В.Д.
(прізвище, ініціали)

Спеціальність
123 – Комп'ютерна інженерія
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Системне програмування
(повна назва освітньої програми)

Керівник: проф. Волк М.О.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2020 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 – Комп'ютерна інженерія
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20___ р.

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові Калюжному Валентину Дмитровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Методи резервування обчислювальних ресурсів для забезпечення живучості програмного забезпечення

затверджена наказом по університету від " 30 " жовтня 2020 р. № 1486 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 14 грудня 2020р.

3. Вхідні дані до роботи
Моделі та методи забезпечення живучості програмного забезпечення.
Моделі розподілених програмних систем
Перелік можливих відмов в програмних системах
Методи забезпечення резервування ресурсів в програмних системах

4. Перелік питань, що потрібно опрацювати в роботі
Вступ
1. Аналіз предметної області
2. Моделі та методи резервування обчислювальних ресурсів
3. Реалізація методу резервування обчислювальних ресурсів для забезпечення живучості програмного забезпечення
Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Презентація 13 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання у керівника	02.11.2020	
2	Огляд літератури	05.11.2020	
3	Аналіз предметної області та	15.11.2020	
	постановка задач		
4.	Розробка моделей та методів	20.11.2020	
5.	Впровадження моделей та методів	01.12.2020	
6.	Практичне застосування технології	10.12.2020	
7.	Оформлення пояснювальної записки	15.12.2020	
	та підготовка демонстраційних матеріалів		

Дата видачі завдання 02 листопада 2020 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Волк М.О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка атестаційної роботи: 70 с., 9 рис., 1 дод., 31 джерел.

ПРОГРАММНА СИСТЕМА, ЖИВУЧОСТЬ, РЕЗЕРВУВАННЯ, ОПЕРАЦІЙНА СИСТЕМА, ОБЧИСЛЮВАЛЬНІ РЕСУРСИ

Метою роботи є підвищення ефективності розподілених програмних систем шляхом розробки та впровадження методу резервування комп’ютерних ресурсів для забезпечення живучості програмного забезпечення, який повинен враховувати динамічні властивості середовища виконання реального часу.

В роботі обговорюються основні статичні та динамічні властивості відокремлених комп’ютерних ресурсів, мережних каналів передачі даних, серверів підтримки живучості. Розроблений метод з урахуванням вказаних властивостей дає можливість підвищити ефективність функціонування систем підтримки живучості розподілених програмних систем, які працюють у реальному часі.

ABSTRACT

Master’s thesis: 70 pages, 9 figures, 2 tables, 1 appendices, 31 sources.

SOFTWARE SYSTEM, SURVIVALITY, RESERVATION, OPERATING SYSTEM, COMPUTER RESOURCES

The aim is to increase the efficiency of distributed software systems by developing and implementing a method of backing up computer resources to ensure the survivability of the software, which should take into account the dynamic properties of the real-time runtime.

The paper discusses the main static and dynamic properties of separate computer resources, network data channels, survivors. The developed method taking into account the specified properties gives the chance to increase efficiency of functioning of systems of support of survivability of the distributed software systems working in real time.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

АСУ – автоматизована система управління

ІС – інформаційна система

ІТ – інформаційна технологія

КСМ – комп'ютерні системи та мережі

AOP – aspect-oriented programming

СМО – система масового обслуговування

CBA– Component Based Architecture

CBSN – Computer-based Systems and Networks

DARPA – Defense Advanced Research Projects Agency

IEEE – Institute of Electrical and Electronics Engineers

MDA – Model-driven Architecture

ROC – Recovery-Oriented Computing

SEI – Software Engineering Institute

SLA – Service Level Agreement

SOA – Service-oriented architecture

QoS – Quality of Service

USSD – Unstructured Supplementary Service Data

ВСТУП

Відносно до сучасних систем обробки даних, зокрема тих, що виконують масові обчислення, забезпечення живучості є одним із основних питань набору функціональних можливостей, що демонструються хмарними обчислювальними системами, які часто презентуються терміном живучість, який найчастіше включає підтримку функціональної стійкості, самовідновлення (самолікування) та само балансування навантаження. Основа масових обчислень спирається на різні аналогії, щоб описати обчислювальний ресурс як комп'ютер, функції якого система самовідновлює, таким же чином, як і суспільний або біологічний організм [1] [2].

Компоненти програмного забезпечення для забезпечення живучості забезпечують можливість виявлення, діагностування та виправляння (або принаймні пом'якшення) відмов в послугах, які надає комп’ютерна система. Для великих комп'ютерних систем може існувати багато різних видів несправностей, та їх різноманітна природа часто вимагає розрізнених, оригінальних підходів для знаходження, не кажучи вже про їх виправлення. Отже, для великих систем, підсистема забезпечення живучості також повинна мати можливість реалізовувати різні методи виявлення, діагностики та відновлення функціонування програмних компонентів.

Хмарні системи поширюють вищезазначене поняття живучості, додаючи можливості адаптації до змін операційного середовища, наприклад, для підтримки відновлення його працездатності або пошуку наявного обчислювального ресурсу.

Системи забезпечення живучості являють собою зовсім нову область досліджень, яка займається толерантністю до відмов динамічних систем. Живучість стосується неточних специфікацій, неконтрольованого оточення та конфігурування систем відповідно до їх динамічної поведінки. Термін позначає властивості програмної системи в боротьбі з відмовами та помилками [2].

Допустимість помилок для надійних обчислень полягає не тільки в наданні зазначеної послуги за допомогою вибору надійних апаратних компонент, але й ліквідування цих проблем під час функціонування. Програмне забезпечення, яке здатне виявляти та реагувати на його несправності, називається програмним забезпеченням з підтримкою живучості. Така програмна система має можливість перевіряти свої відмови та вносити відповідні виправлення.

Одним з напрямків забезпечення систем підтримки живучості є резервування комп'ютерних ресурсів для відновлення на їх базі тих програмних компонентів, які відмовляють. Методи резервування можуть бути статичними або динамічними [2]. Більш цікавими є динамічні методи, які виконують балансування навантаження та перерозподіл ресурсів під час роботи програмної системи.

Метою роботи є розробка методу резервування комп'ютерних ресурсів для забезпечення живучості програмного забезпечення, який повинен враховувати динамічні властивості розподілених програмних систем.

В роботі обговорюються основні статичні та динамічні властивості відокремлених комп'ютерних ресурсів, мережних каналів передачі даних, серверів підтримки живучості. Розроблений метод з урахуванням вказаних властивостей дає можливість підвищити ефективність функціонування систем підтримки живучості розподілених програмних систем, які працюють у реальному часі [1].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Забезпечення живучості програмних систем

Сучасні широко поширені мережеві системи підвищують ефективність та результативність організацій обчислень, дозволяючи цілком нові рівні організаційної інтеграції. Однак, така інтеграція супроводжується підвищеним ризиком відмов та збоїв. Включення можливості виживання в системах можуть пом'якшити ці ризики. Як нова дисципліна, життєздатність базується на суміжних галузях дослідження (наприклад, безпека, відмовостійкість, надійність, повторне використання, продуктивність, перевірка та тестування) та представляє нові концепції та принципи. Виживання фокусується на збереженні основних послуг, навіть коли у системи відмовляють деякі елементи.

Сучасні моделі життєвого циклу розробки програмного забезпечення не орієнтовані на створення систем, що виживають, і виявляють недоліки, коли метою є розробка систем із високим ступенем гарантія виживання [2]. Якщо взагалі звертатися до них, існують проблеми живучості часто відводиться до окремої нитки проектної діяльності, в результаті чого їх важливість є такою, що розглядається як додаткова властивість. Ця ізоляція міркувань виживання від первинної системи при завданні розробки, призводять до розділення проблем. Живучість повинна бути інтегрованою та оброблятись нарівні з іншими властивостями системи, тобто необхідно розробляти системи з певною функціональністю та продуктивністю, які також можуть протистояти збоям і відмовам.

Важливо, що дизайнерські рішення ускладнюються, коли життєздатність не інтегрована у життєвий цикл первинного розвитку. Окремі шляхи проектування коштовні і трудомісткі, що часто призводить до дублювання зусиль у проектуванні та документації. В додаток, інструменти

підтримки інженерії життєздатності часто не інтегровані в розробку програмного забезпечення середовища виконання. З окремими шляхами діяльності стає важче адекватно вирішувати питання ризику виживання та наслідків невдач. Також, технології, що підтримують цілі виживання, такі як формальна специфікація, архітектура, методи балансування навантаження, аналіз вторгнень та схеми проектування живучості, не застосовуються ефективно в процесі розробки.

Для кожного виду життєвого циклу слід розглядати цілі виживання та способи забезпечення певної живучості [3]. У деяких випадках існуючі методи розробки можуть вдосконалити живучість. Сучасні дослідження створюють нові методи, які можна застосовувати; однак, потрібні додаткові дослідження та експерименти, перш ніж мета виживання може стати реальністю.

У цієї роботі описані концепції живучості, обговорюється життєвий цикл розробки програмного забезпечення, модель життєздатності та ілюструються методи, які можуть застосовуватися під час нового розвитку діяльності на підтримку цілей виживання. Також обговорюється модель життєвого циклу програмного забезпечення та пов'язані з нею діяльність щодо підтримки цілей живучості для обчислюючи систем.

1.2 Основні положення теорії живучості стосовно програмних систем

Дослідження систем, що виживають, за останні кілька років призвело до розвитку концепції та визначення життєздатності, описані в цьому підрозділі. Вони корелюють з роботою групи технологій живучості мереж в Інституті програмного забезпечення Software Engineering Institute (SEI) та його Координаційного центру CERT (CERT / CC) [4].

Роботи було розпочато приблизно 30 років тому, коли комп’ютерні мережі стали критично важливим елементом сучасного суспільства. Ці мережі мають не лише глобальний охоплення; вони також мають вплив

практично щодо кожного аспекту людської діяльності. Мережеві системи є головними агентами, що сприяють у бізнесі, промисловості, уряді та обороні. Основні сектори економіки, включаючи оборону, енергетику, транспорт, телекомунікації, виробництво, фінансові послуги, охорона здоров’я та освіта залежать від широкого кола мереж, що діють на місцевому, національному та глобальному рівнях. Ця всеосяжна соціальна залежність від мереж посилює наслідки вторгнень, аварій та відмов, і підсилює критичну важливість забезпечення живучості комп’ютерних систем.

Оскільки організації прагнуть підвищити ефективність та конкурентоспроможність, є нова парадигма комп’ютерної. Мережі використовуються для досягнення радикально нових рівнів організаційної інтеграції. Ця інтеграція знищує традиційні організаційні межі та трансформує місцеві операції з компонентами комплексних бізнес-процесів, що протикають у мережі. Наприклад, комерційні організації інтегрують операції з бізнес-підрозділами, постачальниками, а клієнти взаємодіють через великі мережі, що покращують зв'язок та послуги. Ці мережі поєднують раніше фрагментовані операції у цілісні процеси, відкриті для багатьох учасників. Ця нова парадигма являє собою перехід від обмежених мереж з центральним контролем до необмежених мереж. Необмежені мережі характеризуються розподіленим адміністративним контролем без центральної влади, обмеженою видимістю поза межами місцевої адміністрації та відсутністю повної інформації про мережу.

1.3 Поняття живучості у сучасних великих системах

Визначимо живучість як здатність системи своєчасно виконувати свою роботу за наявності атак, невдач або аварій. Ми використовуємо термін система якомога ширше, включаючи мережі та широкомасштабні комп’ютерні системи. Термін робота стосується набору вимог чи цілей дуже високого рівня та не обмежується військовими установками, оскільки будь-

яка успішна організація чи проект повинні мати своє бачення його цілі, виражені неявно або як офіційна заява про набір сервісів. Судження щодо того, успішно виконана робота, як правило, робиться в контексті зовнішніх умов, які можуть вплинути на досягнення цієї місії. Наприклад, уявимо, що фінансова система відключається на 12 годин у період широкомасштабних відключень електроенергії, спричинені ураганом. Якщо система зберігає цілісність та конфіденційність своїх даних та відновлює свої основні послуги після закінчення періоду екологічного стресу, система може розумно вважати, що виконала свою роботу. Однак, якщо та сама система вимкнеться тим самим несподівано протягом 12 годин за нормальних умов або незначного екологічного впливу, позбавляючи своїх користувачів основних фінансових послуг, можна обґрунтовано судити, що система не виконала свою роботу, навіть якщо збережені цілісність та конфіденційність даних.

Своєчасність є критичним фактором, який зазвичай включається в (або передбачається) дуже високий рівень вимог, що визначають роботу системи. Однак своєчасність є настільки важливим фактором, що це явно входить у визначення життєздатності.

Терміни атака, невдача та аварія мають включати всі потенційно шкідливі події, але ці терміни не поділяють ці події на взаємовиключні або навіть відмінні набори. Часто важко визначити, чи є певна шкідлива подія наслідком зловмисної атаки, виходу з ладу компонента або аварії. Навіть якщо причина врешті-решт визначена, критична негайна реакція не може залежати від припущень про таке майбутнє знання.

Напади - це потенційно шкідливі події, організовані розумним супротивником. Напади включають вторгнення, зондування та відмову в обслуговуванні. Більше того, загроза нападу може мати як серйозний вплив на систему як фактичне явище. Система, що займає оборонну позицію через загрозу атаки може зменшити її функціональність і перенаправити ресурси на моніторинг навколишнього середовища та захист системних функцій.

Ми включаємо невдачі та аварії у визначення життєздатності. Помилки

та потенційно пошкоджувальні події, спричинені недоліками в системі або зовнішнім елементом, від якого система залежить. Помилки можуть бути пов’язані з помилками в розробці програмного забезпечення, деградацією обладнання, людини або пошкоджених даних. Аварії описують широкий спектр випадків і потенційно шкідливі події, такі як стихійні лиха. Ми схильні сприймати нещасні випадки як зовнішні генеровані події (тобто поза системою) та збої як внутрішньо генеровані події.

Що стосується живучості системи, то різниця між несправністю та аварією менша важливо, ніж вплив події. Крім того, часто можна розрізнити розумно організовані напади та ненавмисні або випадкові згубні події. Наші підхід концентрується на ефекті потенційно шкідливої події. Як правило, для системи щоб вижити, віна повинна реагувати на (і відновлюватись) згубний ефект (наприклад, цілісність бази даних) задовго до встановлення основної причини. Насправді, реакція і відновлення повинні бути успішними незалежно від того, чи буде колись встановлена причина. Основна увага в цій роботі полягає у наданні менеджерам методів, які допоможуть системам вижити вчинки розумних супротивників. Поки основна увага приділяється проникненню, обговорюються методи застосовувати в повному обсязі також до відмов та аварій.

Нарешті, важливо визнати, що саме виконання роботи повинно вижити, а не будь-яке конкретна підсистема або компонент системи. Центральним для поняття виживання є здатність системи для виконання своєї роботи, навіть якщо пошкоджені значні її частини або знищені. Ми використовуємо термін система, що виживає, як скорочення для системи з можливостями виконати визначену роботу перед атаками, невдачами або аваріями. Знову ж таки, це робота, а не конкретна частина системи, яка повинна вижити.

Ключовою характеристикою систем, що виживають, є здатність надавати основні послуги в умовах нападу, невдачі або нещасного випадку. Центральним для надання основних послуг є можливість системи для підтримки основних властивостей (тобто, визначених рівнів цілісності,

конфіденційності, продуктивності та інших показників якості) в несприятливих умовах. Таким чином, це важливо визначити, мінімальні рівні таких атрибутів якості, які повинні бути пов'язані з основними послугами. Наприклад, запуск ракети оборонною системою вже не ефективний, якщо продуктивність системи сповільнюється до такої міри, що ціль виходить за межі діапазону, перш ніж система зможе організувати запуск.

Ці показники якості настільки важливі, що часто висловлюються визначення життєздатності з точки зору підтримання балансу між багатьма атрибутами якості, такими як продуктивність, безпека, надійність, доступність, відмовостійкість, модифікація та доступність. Архітектура проекту аналізу компромісу в SEI використовує цей вигляд балансування атрибутів (тобто компромісу) живучості для оцінки та синтезу систем, що виживають [5]. Атрибути якості представляють широкі категорії пов'язаних вимог, тому атрибут якості може бути складений на основі інших атрибутів якості. Наприклад, атрибут безпеки традиційно включає три субатрибути, а саме конфіденційність, цілісність та доступність.

Здатність надавати основні послуги та підтримувати пов'язані з ними основні властивості повинна бути стійкою, навіть якщо значна частина системи непрацездатна. Крім того, ця можливість не повинна залежати від виживання конкретного інформаційного ресурсу, обчислень, або ліній зв'язку. У військовій обстановці необхідними послугами можуть бути основні послуги підтримувати технічну перевагу, а основні властивості можуть включати цілісність, конфіденційність та рівень ефективності, достатній для досягнення результатів менш ніж за один цикл прийняття рішень ворога. У державному секторі такою може бути фінансова система, що виживає що підтримує цілісність, конфіденційність та доступність важливої інформації та фінансових даних, навіть якщо певні вузли або лінії зв'язку непрацездатні в умовах вторгнення або нещасного випадку, і це своєчасно відновлює скомпрометовану інформацію. Про життєздатність фінансової системи можна судити, використовуючи складений показник

зриву біржових торгів чи банківських операцій (тобто міра зриву суттєвих послуги).

Тоді ключовим для концепції виживання є визначення основних послуг (і основних властивостей, що їх підтримують) в операційній системі. Будемо вважати основні послуги як функції системи, які необхідно підтримувати, коли навколишнє середовище вороже або коли трапляються збої або аварії, які загрожують системі. Щоб зберегти свої можливості надавати основні послуги, системи, що виживають, повинні мати чотири проілюстровані ключові властивості.

Визначимо ряд стратегій виживання, які можна застосувати для протидії загрозам та атакам на систему. Деякі з цих методів підвищення живучості запозичені з інших областей, зокрема безпеки та відказостійкості.

В області опору атаці доступний ряд методів. Аутентифікація користувача є механізмом, який обмежує доступ до системи групою затверджених користувачів. Механізми автентифікації варіюються від простих паролів до комбінацій паролів, аутентифікації, що здійснюється користувачем (самі захищені паролем) та біометричні дані. Можна застосовувати засоби контролю доступу для доступу до системи або до окремих програм та наборів даних. Контроль доступу, що застосовується а надійна операційна система, автоматично застосовує заздалегідь визначену політику для надання або заборони доступу автентифікованому користувачеві. При правильному використанні та впровадженні засоби контролю доступу можуть слугувати як заміна механізмів паролів на рівні програм та набору даних.

1.4. Резервування ресурсів у програмному забезпеченні реального часу

У додатках із вимогами до синхронізації, програмне забезпечення повинно бути спроектоване з урахуванням обмежень часу. Сервери на рівні користувача та системні служби, що використовуються такими програмами,

повинні бути розроблені з урахуванням обмежень часу. Політика управління ресурсами, що лежить в основі цих системних служб, також повинна бути розроблена для підтримки програм із обмеженнями часу.

Традиційно, при розробці системи в режимі реального часу, програмісти використовують невеликі, прості операційні системи, що забезпечують планування процесорів із фіксованим пріоритетом, черги пріоритетів для різних системних ресурсів, таких як семафори та поштові скриньки, і, можливо, протоколи успадкування пріоритетів. Потім програмісти повинні створити багато власних системних служб у режимі реального часу, таких як системи управління базами даних, файлові системи та мережеві комунікації. Ці програмісти повинні ретельно планувати різні програми, що працюють у системі, та керувати суперечками щодо процесора та інших системних ресурсів. Під час проектування та планування, обчислювальні вимоги програм повинні бути ретельно виміряні та охарактеризовані, а розподіл ресурсів повинен бути ретельно спланований. Тому програми дуже чутливі до неналежної поведінки інших програм. Наприклад, якщо програма високого пріоритету в режимі реального часу має помилку, яка відправляє її у нескінченний цикл, вплив на інші програми та систему в цілому буде руйнівним. Помилкова програма зайняла б процесор і не відмовлялася б від нього, змушуючи збій системи.

Ці надзвичайні особливості серед додатків є результатом відсутності відповідних системних абстракцій для ефективного управління спільними ресурсами в системах реального часу. Багато абстракцій існує в теоріях планування в режимі реального часу, але, як правило, припущення щодо теоретичних результатів неявно вбудовуються в реальні системи. Приклад припущення, що багато теорій вимагають, щоб найгірший час виконання (WCET) для обчислень у додатках реального часу був відомий під час проектування системи. Більшість систем, розроблених з використанням методів аналізу, які мають це припущення, явно не перевіряють і не застосовують найгірший час виконання обчислень. Відповідні абстракції

системи явно вводять припущення в реальну систему, де їх можна перевірити і, в кращому випадку, навіть застосувати.

Відповідні абстракції системи можуть ефективно вирішити кілька проблем, що виникають у системі реального часу. Ці проблеми розподіляються на три великі сфери: політика управління ресурсами, яка може задовольнити обмеження часу, механізми підтримки політики та методи аналізу, засновані на наявних механізмах.

Багато систем не забезпечують політики планування, яка безпосередньо підтримує реальний час управління ресурсами. Для планування процесорів більшість систем надають політику планування з фіксованим пріоритетом або терміном. Обидві ці політики не мають повної інформації про вимоги в режимі реального часу, і тому не вирішують таких важливих проблем, як те, як слід призначати пріоритет чи як запобігати перевантаженню.

При розробці політики управління ресурсами виникають такі проблеми:

1) Проблема призначення пріоритету: прості планувальники пріоритетів важко використовувати, особливо якщо існує багато заходів з обмеженнями часу. Якщо в різних додатках немає глобального сховища знань про часові обмеження різних видів діяльності, немає жодних підстав для прийняття рішення про те, яким має бути пріоритетний порядок дій.

2) Проблема з перевантаженням: щоб запобігти перевантаженню, політика планування вимагає інформації про обмеження в режимі реального часу, такі як вимоги до обчислень та частота виконання. Навіть якщо дизайнер може висловити вимоги до ресурсів та часу для додатків у режимі реального часу, система без політики контролю за допуском не може захистити себе від перевантаження.

3) Вимога до гнучкості: Обмеження часу та вимоги до ресурсів для динамічних додатків реального часу можуть динамічно змінюватися під час виконання. Алгоритм планування повинен підтримувати ефективне

коригування вимог.

Щоб ефективно планувати програми в режимі реального часу, програми, що не можуть монополізувати системні ресурси, мають мати певні механізми вимірювання та забезпечення рівня використання ресурсів. Інформація, отримана за допомогою цих механізмів, може бути використана у політиці планування для прийняття рішень щодо того, як динамічно коригувати пріоритети чи терміни, щоб відображати вимоги та схеми використання програм. Проблеми, що спонукають до використання цих механізмів, коротко описані нижче:

1) Проблема забезпечення: Додаток, який визначає вимоги до ресурсів та часу, може випадково чи навмисно спробувати перевищити його заявлені вимоги. Це може перешкоджати задоволенню обмежень часу для інших зарезервованих видів діяльності, а також може спричинити голод серед безрезервованих видів діяльності.

2) Проблема вимірювання: виконання вимог щодо ресурсів означає, що використання ресурсів кожною програмою має бути точно виміряно та порівняно з розподілом, який було зроблено від її імені. Якщо программа використовує зовнішні модулі, сервери та системні послуги, вимірювання повинно включати також це використання.

3) Проблема координації: багато заходів, обмежених часом, складаються з безлічі підзаходів, реалізованих іншими модулями, сервісами на рівні користувача або системними службами. Виділення ресурсів для однієї діяльності, яка охоплює кілька модулів, можливо, в різних доменах захисту пам'яті, є складним. Однак необхідно мати можливість встановлювати вимоги щодо часу або щодо загальної діяльності та відстежувати використання ресурсів для загальної діяльності.

Інші проблеми стосуються питань вищого рівня, таких як аналізувати поведінку системи, враховуючи більш складні абстракції та механізми планування та управління ресурсами. У цьому контексті виникають такі проблеми:

- Розподілена проблема планування в режимі реального часу: програма, яка використовує зовнішні модулі та послуги, може вимагати використання декількох ресурсів протягом певних часових обмежень. Координація використання кількох ресурсів для задоволення обмежень часу є важкою проблемою.

- Проблема управління якістю: Додатки в режимі реального часу можуть динамічно змінювати вимоги до якості обслуговування (QoS). У системі з багатьма такими додатками для вирішення конфліктів та ведення переговорів між програмами необхідний менеджер ресурсів високого рівня (який іноді називають менеджером QoS).

У традиційному проектуванні систем реального часу багато з цих проблем уникаються на етапі проектування шляхом ретельного вимірювання та аналізу простих обчислень та їх потреби у ресурсах та за допомогою спеціальних методів планування. В результаті такого підходу виникають гнучкі системи, які важко підтримувати [6,7]. Зокрема, підтримка динамічної діяльності в режимі реального часу з обмеженнями часу та вимогами до ресурсів, які можуть вільно змінюватися під час виконання, розширює традиційний підхід за його межі.

Останні роботи в системах реального часу вирішують деякі з цих проблем. Кілька систем (наприклад, [3,8,9]) дозволяють вказувати вимоги до синхронізації, а не змушувати програміста визначати відповідне призначення пріоритету. Деякі системи мають он-лайн політику контролю за допуском [3,7,10], але багато інших використовують офлайн-аналіз [11,12,13]. Треті взагалі не мають контролю за вступом [14,15,16]. Деякі обмежені вимоги до гнучкості для жорсткого реального часу були розглянуті нещодавно [17]. Ця робота зосереджена на зміні режиму, корінні, але нечасті зміни у наборах завдань системи реального часу. Наприклад, система реального часу в літаку може відчувати зміну режиму після зльоту, коли вона перемикається з наземного завдання на набір завдань на основі повітря.

Дуже мало програмних систем вирішують проблеми вимірювання та забезпечення використання ресурсів, хоча більшість систем можуть виявляти та реагувати на пропущені терміни [17]. У мережах поняття примусового виконання чи правоохоронних органів є набагато більш поширеним [18]. Робота над пріоритетними протоколами успадкування [18,19] стосується деяких аспектів проблеми координації.

Проблема розподіленого планування в режимі реального часу є активною сферою досліджень [20]. Проблема якості програмних систем - ще одна активна область досліджень. Деякі дослідження операційної системи в цій галузі зосереджені на найкращих підходах [2].

2 МОДЕЛІ ТА МЕТОДИ РЕЗЕРВУВАННЯ ОБЧИСЛЮВАЛЬНИХ РЕСУРСІВ

2.1 Основні вимоги та особливості моделі резервування ресурсів

Визначимо модель резервування ресурсів, яка забезпечує абстракцію операційної системи, яка називається резервом. Резерви явно представляють резервування ресурсів, таких як процесори, сторінки пам'яті, диски та мережеві пристрої. Зокрема, підтримка резервів

- уточнення параметрів резервування;
- контроль прийому;
- планування на основі обмежень часу та вимог використання;
- забезпечення резервування;
- розповсюдження резерву в механізмі управління обчислювальним процесом;
- гнучке прив’язування програмних компонентів до резервів.

Резерви не дозволяють програмам надмірно запускати дозволене використання ресурсів та втручатися в інші зарезервовані ресурси або утриматись від надмірного їх використання. Програми резервують потужність ресурсів, необхідних для їх обчислень. Наприклад, програма може зарезервувати 10 мс обчислювального часу на процесорі за кожні 100 мс реального часу. Потім додаток прив'язується до резерву, і планувальник процесора використовує інформацію, пов'язану з резервом, для управління плануванням програми. Система також проводить контрольний тест на вступ перед наданням бронювання, щоб переконатися, що ресурс може підтримати замовлене бронювання. Механізм забезпечення гарантує, що програма не використовує більше зарезервованого часу, якщо це заважатиме іншим зарезервованим заходам.

Параметри бронювання, пов'язані з резервами програми, не обов'язково фіксуються протягом усього терміну дії програми. Динамічний додаток у режимі реального часу повинен бути підготовлений для зміни своїх вимог щодо поведінки та часу на основі змін вимог користувачів та, можливо, зміни доступності ресурсів. Користувач може захотіти змінити частоту кадрів на відеопрогравачі або змінити роздільну здатність, і програма повинна бути готова до відповідного регулювання рівня резервування ресурсів. Аналогічним чином, резерви повинні підтримувати операцію, яка змінює параметри бронювання, відповідно до політики контролю за допуском.

Резерви - це об’єкти першого класу в операційній системі; резерв асоціюється з певним потоком (або процесом) шляхом явного прив’язування потоку до резерву. Це надає дозвіл програме для резервування всіх ресурсів, необхідних для його обчислення, включаючи ресурси, які будуть потрібні різним модулям, серверам та системним службам, які він має намір викликати. Потім програма може передавати посилання на свої резерви ресурсів модулям або серверам разом із викликом операції. Модуль або сервер може прив’язати свій потік до цього резерву під час виконання операції, а потім може скористатися перевагами ресурсів, зарезервованих клієнтом. Якщо модулі та сервери беруть плату за резерв абонента за роботу, виконану від імені абонента, також зберігається послідовний погляд на використання ресурсів, що споживається від імені цього клієнта.

Резервна модель, представлена в цій роботі, стосується проблем, визначених у попередньому розділі. Політики планування, вбудовані в модель резерву, вирішують проблему призначення пріоритетів та проблему перевантаження, залишаючись гнучкими з точки зору динамічних змін вимог до ресурсів програми, як обговорюється нижче.

- Проблема призначення пріоритету: Резерви уникають проблеми присвоєння пріоритету, приймаючи специфікації бронювання, які включають обмеження часу та вимоги до використання.

- Проблема з перевантаженням: Політика контролю за допуском

механізму бронювання запобігає перевантаженню. Це можливо, оскільки політика планування має інформацію як про вимоги до обчислювальних ресурсів, так і про обмеження часу (наприклад, про період виклику) для кожної програми.

- Вимога до гнучкості: Зміни параметрів бронювання можуть бути внесені в будь-який час відповідно до політики контролю за живучостю системи.

Резервна модель використовує декілька системних механізмів, що підтримують абстракцію. Ці механізми надають інформацію про використання ресурсів та координують споживання ресурсів для діяльності, яка переходить межі захисту пам’яті наступним чином:

- Проблема забезпечення: Забезпечення резервування ізолює забронювані дії від надмірного втручання з боку інших зарезервованих видів діяльності.

- Проблема вимірювання: Гнучкість прив'язки резервів дає змогу накопичувати плату за використання ресурсів за певну діяльність, навіть коли роботу виконують зовнішні модулі, сервери або системні служби.

- Проблема координації: Модель резерву підтримує розповсюдження резервів, що включає механізм успадкування «пріоритету» і який використовує гнучкість прив'язки резервів до потоків.

Резервна модель забезпечує абстракцію, яка може бути використана для розподіленого планування в режимі реального часу та управління якістю послуг. Підходи до цих проблем коротко описані нижче:

- Розподілена проблема планування в режимі реального часу: резервна модель підтримує резервування віддалених ресурсів, а також підтримує архітектуру програмного забезпечення в стилі конвеєра. Це не є оптимальним, але подальша робота над цією проблемою може використати резервну модель як основу.

- Проблема управління якістю: рівень управління якістю ЯС високого рівня може використовувати резерви для прийняття рішень щодо політики

розподілу ресурсів. Менеджери ЯП можуть виконувати рішення щодо розподілу, маніпулюючи параметрами резервування для керованих додатків.

2.2 Модель резервування обчислювальних ресурсів

Визначена у попередньому підрозділі особливості моделі резервування, використовуються в концепції резерву щодо конкретного ресурсу операційної системи. Резерв – це об’єкт певного класу, який представляє частину потужності або кількість ресурсу, яка відводиться для обчислення, яка представляє цей резерв разом із запитом на використання ресурсу. Слово «ємність» тут використовується в широкому значенні; резервування частини потужності ресурсу означає, що потік матиме доступ до ресурсу за умови деяких детальних параметрів резервування, а параметри є специфічними для ресурсу та реалізації системи резервування. Як приклад, резерв може вказати, що 30 мс часу обчислення на процесорі резервується з кожних 100 мс реального часу.

Резервування ресурсної ємності означає, що ресурс може мультиплексироваться між кількома обчисленнями, і модель фокусується на мультиплексних ресурсах, таких як процесори та пропускна здатність мережі. Інші типи ресурсів, такі як сторінки фізичної пам'яті та буфери, не піддаються надзвичайно дрібному мультиплексуванню, і їх називають дискретними ресурсами. У цій моделі дискретні ресурси резервуються на одиницю, і резервація виділяє одиниці ресурсів необмежено довго, а не передбачає мультиплексне використання потужності з часом, але більш складні системи резервування можуть використовувати інші підходи.

Ключовою вимогою до пропонування абстракції ресурсного резерву в системі є наявність зарезервованої ємності ресурсів, залежно від параметрів резервування, для розрахунку, який представляє резерв і вимагає ресурс. Якщо система не може гарантувати що потужність ресурсів буде доступна, як обіцяно, корисність системи обмежена. Тому забезпечення резервування

ресурсів є надзвичайно важливим. Застосування є важливим не лише для захисту від зловмисних користувачів, що може створити проблему в системах, де багатьма розподіляються ресурси, а й для звільнення окремі додатки від тягаря забезпечення строго передбачуваної та контрольованої їхньої роботи. Система повинна терпіти програми, які можуть намагатись використовувати більше, ніж дозволяє їхнє резервування, ізолюючи не пов’язані між собою програми від такої поведінки. Цей вид тимчасової ізоляції схожий за концепцією на ізоляцію, що забезпечується системою віртуальної пам'яті, що дозволяє процесу намагатись отримати доступ до місць пам'яті за власним бажанням, втручатися лише тоді, коли доступ до пам'яті заборонений. Ні в якому разі система віртуальної пам’яті не повинна дозволяти процесу доступ до захищеної пам’яті іншого процесу, і також ні в якому разі система резервування не повинна дозволяти потоку впливати на зарезервований ресурс іншого потоку.

Таким чином, система гарантує, що ресурсна потужність, задана параметрами резервування, буде доступна для потоку, який має резерв. Однак програміст програми повинен переконатись, що потік може скористатися можливостями ресурсу, коли він стає доступним. Сама система бронювання не претендує на те, що конкретна програма буде відповідати своїм обмеженням часу. Наприклад, якщо програма блокує на невизначений час очікування повідомлення, можливо, вона не зможе скористатися можливостями ресурсу, коли воно доступне. Щоб програма мала передбачувану продуктивність у реальному часі, вона повинна мати належні резерви ресурсів, і вона повинна мати можливість використовувати ці резерви ресурсів таким чином, щоб задовольняти тимчасові обмеження програми.

Резервна абстракція може містити різні рамки для контролю за вступом, плануванням та забезпеченням виконання. Більшість особливостей, зокрема операції, доступні в абстракції програми, залишаються незмінними, навіть якщо рамки планування змінюються. Основними відмінностями в

інтерфейсі до іншої структури є специфікація параметрів запиту на бронювання для контролю за допуском та статистикой використання ресурсів, доступним від механізму забезпечення.

2.3 Стилі та специфікації програмування з резервами для забезпечення живучості

Резерви можна використовувати в двох різних стилях програмування в режимі реального часу. Резерви підтримують суворі жорсткі програми реального часу і можуть однаково добре підтримувати більш гнучкі програми м'якого реального часу. Основна відмінність між жорстким та м'яким програмуванням у реальному часі при обговоренні моделі бронювання:

- чи ретельно вимірюються та чітко визначаються вимоги щодо використання ресурсів, що гарантують ефективність до фактичного розгортання програми (жорсткий режим реального часу);
- чи динамічно коригуються вимоги щодо використання ресурсів та резервування ресурсної потужності на основі вимірювань часу використання, замість того, щоб точно відповідати їм на етапі проектування.

У будь-якому випадку резерви ресурсів гарантують запити, які приймаються до системи, і чи будуть ці резерви використовуватися для жорсткого програмування в реальному часі чи легкого програмування в реальному часі, залежить від того, як самі програми використовують резерви.

Інша відмінність у програмуванні в режимі реального часу з використанням резервів полягає в тому, чи резервуються ресурси локально для кожного потоку, чи вони зарезервовані глобально для діяльності, яка може охоплювати кілька потоків у різних доменах захисту та навіть на різних машинах. У моделі використання резервів на основі живучості, ініціатор діяльності набуває резерви ресурсів для неї, а потім передає ці резерви разом із будь-якими запитами, поданими на різні сервери. За допомогою цієї моделі

спрощується облік використання ресурсів між клієнтами та серверами, а також може бути спрощено узгодження параметрів якості обслуговування. Резерви для кожного запиту надходять із запитом, і сервер зараховує використання ресурсів за ці резерви під час обслуговування відповідного запиту.

Підводячи підсумок, декілька особливостей резервів корисні як для жорсткого, так і для м’якого програмування в режимі реального часу:

1. Необходимо дбайти про глобальні рішення щодо контролю за вступом, звільняючи дизайнера від аналізу глобального планування.

2. Планувати потоки ресурсів відповідно до їх запасів.

3. Накопичувати інформацію про використання, яка може бути корисною під час розробки, особливо для адаптації в програмних системах реального часу.

4. Запобігати втручанню з боку інших програм у реальному часі та системних служб операційних систем, які працюють не в реальному часі, і які можуть конкурувати за ті самі ресурси.

5. Резерви можуть служити або для відокремлення розподілу ресурсів та управління модулями один від одного, або для інтеграції розподілу ресурсів та управління модулями, дозволяючи програмним компонентам охоплювати декілька потоків та захисні домени однієї діяльності.

Основний резерв забезпечує абстракцію потужності певного ресурсу. Це твердження про основні резерви не гарантує, що програми будуть відповідати їхнім вимогам щодо часу. Єдина гарантія полягає в тому, що ємність буде зарезервована та доступна для використання.

Сам запрос резерву є абстракцією операційної системи, яка є ортогональною для управління такими структурами, як потоки. Потік може прив'язуватися до резерву ресурсів, і в цьому випадку планувальник використовуватиме інформацію з резерву під час прийняття рішень щодо планування виконання потоку. Багато потоків можуть бути прив'язані до одного резерву, але зазвичай резерв матиме лише один потік, прив'язаний до

нього одночасно. Планувальник завжди знайде відповідний резерв, хоча іноді цей резерв буде резервом за замовчуванням, який не має фактичного резервування і служить для накопичення використання ресурсів усіх потоків, які безоговорочно використовують ресурс.

Специфікація бронювання залежить від типу ресурсу. Мультиплекс резервування ресурсів включає інформацію про обсяг роботи, яку потрібно виконати за період реального часу. Вони можуть також включати вимогу про затримку. Цей параметр визначає максимальну кількість часу після початку кожного періоду, коли потоку доведеться чекати, перш ніж отримати зарезервований час на ресурсі. Бронювання дискретних ресурсів просто вказує кількість одиниць необхідних дискретних ресурсів; вони не включають поняття часу.

Незважаючи на різницю між мультиплексованим та дискретним резервуванням ресурсів, вони мають однакову базову структуру. Вони обидва вимагають:

- інтерфейс специфікації резервування;
- політика контролю за допуском;
- політика планування та;
- механізм забезпечення.

Для дискретних резервів, таких як сторінки пам’яті та мережеві буфери, специфікація резервування дає ряд одиниць ресурсу, що запитується. Політика контролю за допуском до дискретних ресурсів просто перевіряла б наявність необхідної кількості одиниць ресурсу. Оскільки дискретні резерви за визначенням не мультиплексуються, вони не потребують планування.

Важливо зазначити, що основна абстракція резерву не залежить від політики контролю та планування прийому, яка використовується. Для мультиплексованих ресурсів резерви надають основу для запитування резервування ресурсної потужності, контролю за допуском, планування обчислень та забезпечення резервування потужності. Вибір політики контролю та планування прийому буде впливати на те, як визначаються

запити на бронювання та на те, як механізм забезпечення відстежує їх поведінку, але рамки є достатньо загальними, щоб враховувати різні політики.

Система бронювання повинна надавати додаткам можливість вказати ресурсну потужність, яку вони хотіли б зарезервувати. Форма специфікації відрізняється від ресурсу до ресурсу, і різні політики контролю та планування прийому можуть вимагати різних параметрів специфікації відновлення. У найзагальнішому розумінні застереження визначають тривалість використання з певними часовими обмеженнями, доступними, використовуваними та поповнюваними якимись конкретними режимами.

Як приклад типу параметрів, які можуть відображатися в специфікації, резервування ресурсу може вказати кількість часу, який потрібно витратити на ресурс за період реального часу, а також може вказати час початку для періодів. Наприклад, запит на бронювання може вказати 30 мс кожні 100 мс, починаючи з 13:00.

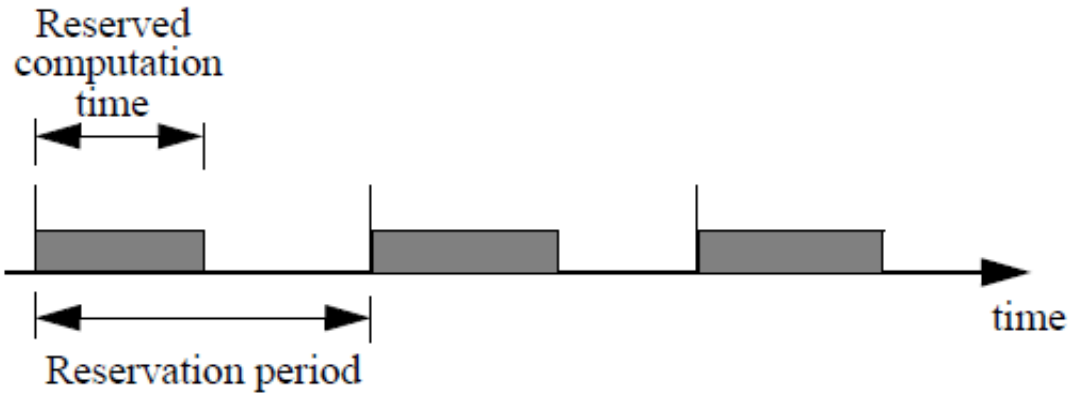

Рисунок 2.1 – Розклад періодичного планування

Рисунок 2.1 ілюструє, як зарезервований час може бути витрачений за час простого обчислення за період реального часу для резервів. Зарезервований час обчислень можна використовувати протягом кожного періоду бронювання. Час обчислень гарантовано буде доступним у певний

момент протягом періоду; не гарантується перебування в якомусь конкретному місці, наприклад, на початку періоду або на кінці періоду.

Існує багато різних стратегій планування та методів аналізу планування, які можуть бути використані для забезпечення резервної абстракції, кожна з яких потребуватиме відповідного контролю, політики планування та механізму забезпечення.

2.4 Методи планування резервів та виконання плану резервування

Алгоритм планування для ресурсу приймає рішення щодо порядку, в якому потоки отримують час на ресурсі. Планувальник переглядає резерв, що належить кожному потоку, який готовий до запуску, і використовує інформацію в резерві, щоб визначити, який потік отримає доступ до ресурсу. Алгоритм підтримує рішення, прийняте політикою контролю за допуском.

Планувальник також повинен узгодити з механізмом забезпечення, щоб переконатись, що він не намагається планувати потоки, пов'язані з резервом, який вже використав свій зарезервований час ресурсу для певного періоду резервування. Таким чином, програма, у якої ще залишився час для бронювання, перебуває в «зарезервованому режимі», а та, у якої закінчився час, тимчасово перебуває в «незарезервованому режимі». Це означає суттєвий відхід від інших алгоритмів планування в режимі реального часу, які, як правило, припускають, що вимоги щодо використання ресурсів програми точно характеризуються і не потребують виконання [21].

Система резервування повинна забезпечити, щоб процеси не використовували більше, ніж їх зарезервована потужність або зарезервовані одиниці ресурсу. Виконання застережень щодо дискретних ресурсів є простим; система гарантує, що ресурс, присвячений одному процесу, не перерозподіляється іншому процесу. Застосування мультиплексованого бронювання вимагає від системи збереження точних цифр використання, які описують, скільки ємності було витрачено на кожне бронювання. Якщо потік

намагається використовувати деяку ємність поза резервуванням, система повинна це визнати і активно запобігти процесу споживання будь-якої додаткової ємності в зарезервованому режимі (може бути дозволено споживання додаткової ємності в режимі без резервування).

Якщо з якихось причин зарезервований час на ресурсі не використовується власником програмної системи в даний період резервування, такий розподіл часу втрачається власником. Власник може не мати можливості використовувати ресурс, якщо він заблокований, чекаючи, коли стане доступним якийсь інший ресурс, або для синхронізації або зв'язку з іншим обчисленням. Ресурс не обов'язково буде простоювати протягом такого періоду часу, оскільки політика планування вільна, щоб дозволити безрезервному обчисленню використовувати ресурс (за умови, що безрезервоване обчислення може бути виключено, щоб дозволити власнику резерву використовувати ресурс). Це означає, що обчислення можуть не економити зарезервований час (не використовуючи його), а потім використовувати все одразу в серії. Розподіл ресурсного часу доступний протягом кожного періоду, але не може бути перенесений після закінчення періоду.

На рисунку 2.2 показано зарезервований час, доступний для певного ресурсу для одного бронювання. На початку кожного періоду бронювання розподіл зарезервованого часу поповнюється, і потік, який має це резервування, використовує ресурс у зарезервованому режимі. Після того, як зарезервований час, виділений на цей період, вичерпується, механізм забезпечення генерує подію планувальника, щоб вказати, що зарезервований час був витрачений на цей період. Планувальник відповідає за використання цієї інформації при прийнятті рішень щодо планування. На малюнку потік продовжує використовувати ресурс у «режимі без резервування», витрачаючи більше часу на ресурс на розсуд планувальника. Історія виконання, показана на малюнку, базується на припущенні, що жоден інший потік не конкурує за ресурс і що політика дозволяє йому працювати в режимі

спільного використання часу після закінчення терміну його резервування, і тому потік може отримувати час у режимі без резервування. На початку наступного періоду відновлення резерв поповнюється, і потік може знову працювати в зарезервованому режимі. Основним пунктом цього показника є те, що механізм забезпечення відстежує використання ресурсів і викликає цю подію вичерпаного резерву для планувальника. Потім планувальник може використовувати цю інформацію для прийняття майбутніх рішень щодо планування. Наприклад, він може надавати перевагу іншим зарезервованим ресурсам або дозволяти безрезервний розподілу часу. На початку наступного періоду відновлення резерв поповнюється, і потік може знову працювати в зарезервованому режимі. Основним пунктом цього показника є те, що механізм забезпечення відстежує використання ресурсів і викликає цю подію вичерпаного резерву для планувальника. Потім планувальник може використовувати цю інформацію для прийняття майбутніх рішень щодо планування. Наприклад, він може надавати перевагу іншим зарезервованим діяльностям або дозволяти безрезервному розподілу часу використовувати цей ресурс.

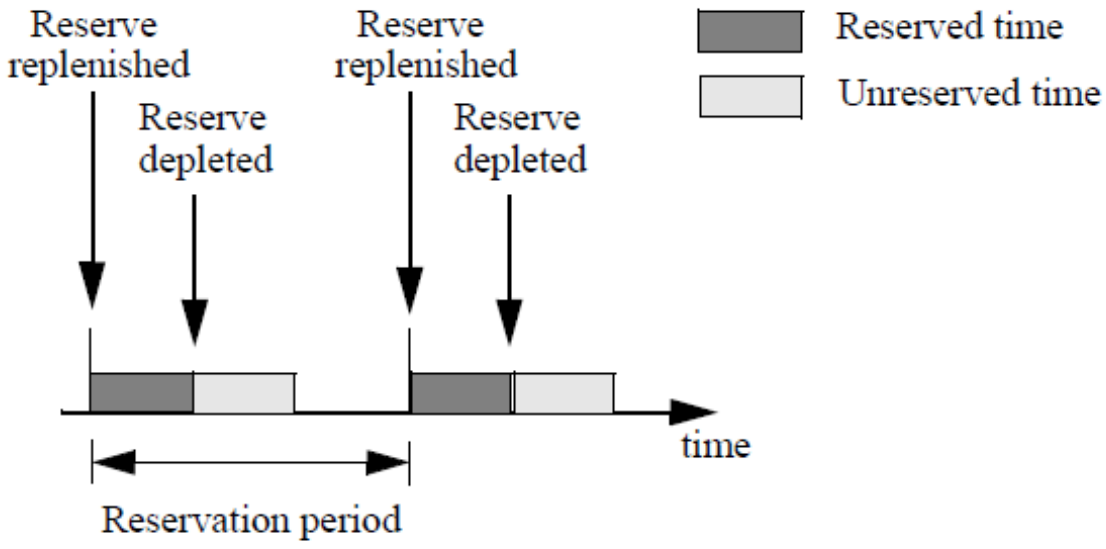

Рисунок 2.2 – Ілюстрація примусового виконання

На початку наступного періоду відновлення резерв поповнюється, і потік може знову працювати в зарезервованому режимі. Основним пунктом цього показника є те, що механізм забезпечення відстежує використання ресурсів і викликає цю подію вичерпаного резерву для планувальника. Потім планувальник може використовувати цю інформацію для прийняття майбутніх рішень щодо планування. Наприклад, він може надавати перевагу іншим зарезервованим діяльностям або дозволяти безрезервному розподілу часу використовувати цей ресурс.

Три важливі питання виникають при розробці механізму забезпечення:

- як точно акумулювати використання ресурсів;
- як помітити, що потік вичерпав свій резерв ресурсу;
- як дізнатися, коли поповнити виділений резерв.

Щоб точно накопичити використання ресурсів для кожного резерву, система реєструє час кожного перемикання резервів. Перемикання резерву відбувається в двох випадках: коли виконується перемикання контексту потоку або коли потік змінює резерв, з якого він буде використовувати час обчислення. У разі перемикання контексту потоку перемикач резерву реєструє поточний час, скасовує таймер перевитрати (який сигналізує про подію вичерпання резерву, як описано нижче), обчислює час запуску старого потоку і додає цей час до накопиченого використання сарого потоку. Потім механізм резервного перемикача економить поточний час для подальшого використання при обчисленні часу обчислення нового потоку та встановлює таймер перевитрати. Перемикач резерву, викликаний потоком, що змінює резерв, до якого він хоче зарядити час обчислення, працює так само, різниця лише в тому, як спрацьовує перемикач резерву. Рисунок 2.3 ілюструє, як таймери використовуються для забезпечення виконання. В історії виконання на часовій шкалі він показує зарезервовану діяльність (пов'язану з іншими резервами) у смужку. Резервні перемикачі (у цьому прикладі також перемикачі контексту) між цими діями відбуваються там, де зустрічаються смугасті ящики та штрихові коробки.

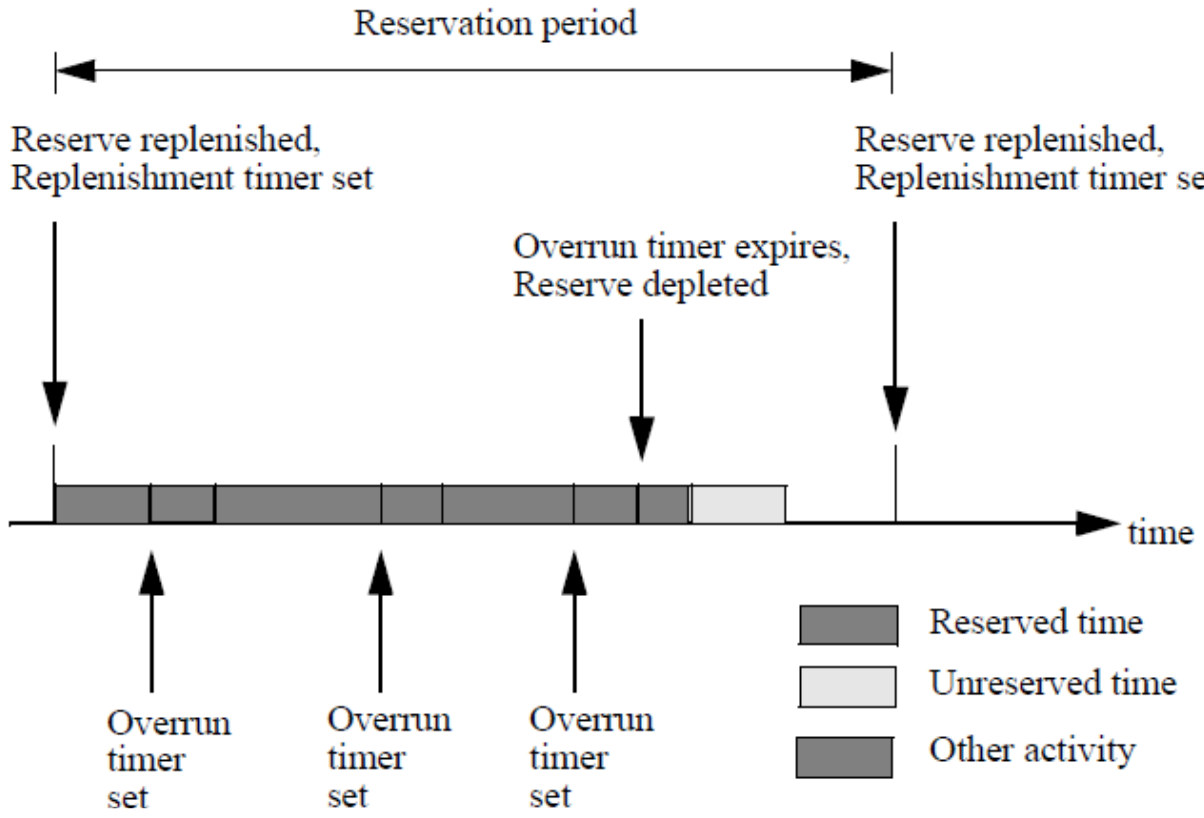

Рисунок 2.3 – Таймери виконання

Таймер перевитрати встановлюється під час перемикача резерву, щоб закінчити в кінці нового залишкового часу обчислення потоку або в кінці періоду резервування. Якщо новий потік буде попереджено до того, як закінчиться зарезервований час обчислення, таймер перевитрати буде скасовано. Якщо новий потік витрачає весь свій зарезервований час, таймер перевитрати закінчується, і планувальник викликається для здійснення певних дій на основі цієї події. На рисунку 2.3 показано, де таймер перевитрати встановлений для резерву; таймер перебігу також може бути встановлений для іншої діяльності, якщо вона зарезервована, але це не показано на рисунку. Таймер перевитрати на рисунку фактично закінчується до останнього часу, коли він встановлений.

Для поповнення дескриптора кожен резерв має таймер поповнення, який запускається під час запуску резерву і який періодично закінчується на кожному періоді бронювання. Таймер поповнення реєструє використання,

накопичене у відповідному резерві під час періоду бронювання, і реєструє це разом із поточним часом як «контрольна точка використання». Потім обробник таймера змінює стан резерву, щоб відобразити новий розподіл використання ресурсів, і скидає таймер періодичного поповнення. Ця модель поповнення відповідає схемі поповнення сервера, що описано в [22]; інші методи поповнення описані та проаналізовані Спрантом [23, 24]. Рисунок 2.3 ілюструє, де встановлений таймер поповнення щодо періоду бронювання.

2.5 Розширення резервування для забезпечення живучості

Однією з важливих особливостей моделі резервів є те, що резерви можуть передаватися від клієнтів до серверів, що дозволяє серверу скористатися ресурсами, які клієнт резервує для всіх своїх обчислень. Передача резервів також дозволяє серверу підтримувати живочость відповідних клієнтів, зберігаючи загальносистемний облік використання. Це називається розповсюдженням (розширенням) резерву.

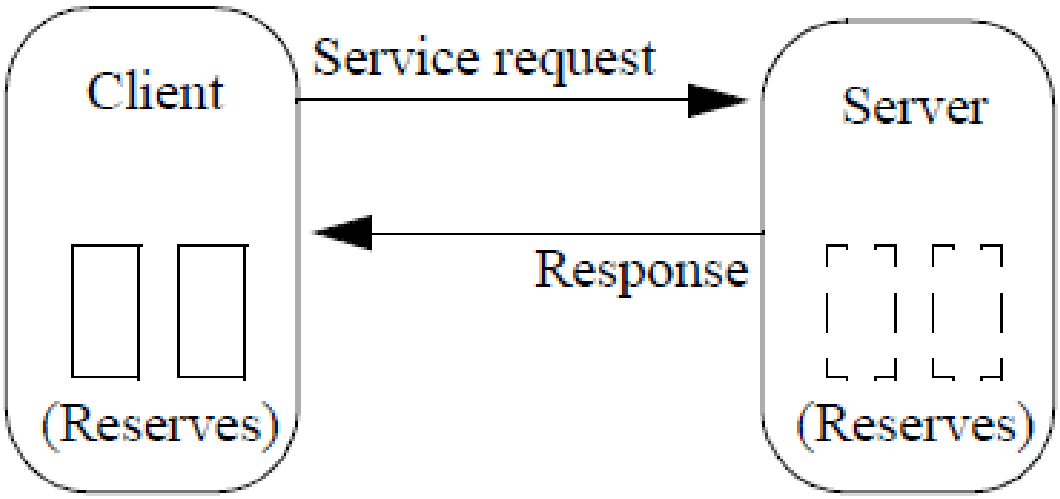

Рисунок 2.4 – Розмноження запитів на резервування

Рисунок 2.4 ілюструє взаємодію клієнт / сервер із розповсюдженням резервів. Припустимо, що клієнт отримує резервування ресурсів, достатнє для обчислень, які він буде виконувати локально, а також обчислень, які повинні виконувати сервери від імені клієнта. На малюнку ці резерви відображаються у клієнтів у вигляді двох маленьких прямокутників. Взаємодія - це прямий віддалений виклик процедури від клієнта до сервера. Для простоти припустимо, що клієнт надсилає на сервер запит RPC і чекає відповіді. Сервер обробляє запит і надсилає відповідь RPC, а потім клієнт отримує відповідь.

Коли клієнт надсилає запит на послугу, він також посилає посилання на резерви, які він виділив. Ці резерви повинні використовуватися сервером під час обробки запиту клієнта. Таким чином, сервер повинен почати стягувати плату за використання ресурсів із резервів клієнта, коли він починає обробляти цей запит, і він повинен припиняти стягувати плату за ці резерви, коли закінчує із запитом і відправляє відповідь назад.

В ідеалі сервер планував би виконувати запити на обслуговування в тому ж порядку, що і обчислення, якщо клієнти могли робити їх замість сервера. Наприклад, планувальник процесора замовляє готові потоки на основі параметрів резерву процесора. Це впорядкування можна розглядати як своєрідне пріоритетне впорядкування серед цих видів діяльності. Якщо потік робить запит сервера, сервер повинен приймати пріоритет у впорядкуванні планувальника, поки він обслуговує цей запит. Тоді той факт, що клієнт покладається на сервер для певних обчислень, не впливає на хід його обчислень щодо інших потоків.

Щоб допомогти серверу досягти цього ідеалу, механізм RPC повинен поширювати на сервер пріоритет резерву, як представлений резервом та його параметрами резервування. Черга сервісних запитів для сервера повинна підтримуватися з резервним пріоритетом, і для запобігання інверсії пріоритетів у доступі до сервера необхідно використовувати своєрідне «пріоритетне успадкування».

При отриманні нового запиту на обслуговування, механізм «резервування пріоритетного успадкування» ставить запит у чергу пріоритетів, а потім перевіряє, чи потік, який обслуговуватиме запити, чекає нових запитів чи обслуговує попередній запит. Якщо потік зайнятий, і «пріоритет» нового запиту перевищує «пріоритет» поточного оброблюваного запиту, механізм RPC встановлює пріоритет потоку, а таким чином, й пріоритет нового запиту. Однак це не змінює резерв, від якого потік виконується. Коли потік сервера закінчує попередній запит і отримує новий запит, він зберігає пріоритет цього нового запиту (який він успадкував раніше), а також починає стягувати резерви, пов'язані з новим запитом.

Описаний тут механізм успадкування «пріоритету», який називається «розповсюдження резерву», відрізняється від традиційного розширення пріоритету двома особливостями:

- розповсюдження резервіву визначає, як сервер повинен змінювати резерви, за які він бере плату, на основі клієнта, який він обслуговує, тоді як традиційне пріоритетне успадкування не має поняття стягнення плати за резерв або рахунок;
- «пріоритет» сервера може змінюватися під час обробки запиту, якщо зарезервовані ресурси вичерпуються протягом цього часу, тоді як при традиційному успадкуванні пріоритетів пріоритет фіксований.

Той факт, що пріоритет сервера може знизитися під час обробки запиту, ускладнює управління резервами та успадкування пріоритету. Коли резерв вичерпується, планувальник повинен визначити, чи потік, який стягується з резервування, успадкував резерв пріоритет чи ні. Якщо ні, порядок потоку в черзі готових може змінитися. Якщо це так, планувальник повинен визначити з черги запитів, що очікує на розгляд, яким повинен бути відповідний пріоритет резерву для потоку, враховуючи зміну стану резерву, який вичерпався.

Розповсюдження резервів з клієнта на сервер не є обов'язковим. Є різні моделі програмування, де це корисно, а де ні. Коротше кажучи, розповсюдження резервів корисно, коли система організована таким чином, що програма розподіляє резерви ресурсів, необхідних для всієї її діяльності, і передає ці резерви навколо серверів, які вона викликає для виконання роботи від її імені. У цій моделі додатки не повинні чітко узгоджувати параметри якості обслуговування з цими серверами. Сценарій, коли розповсюдження резервів не є настільки корисним, полягає в тому, що система організована таким чином, що додатки чітко узгоджують якість обслуговування з усіма своїми серверами.

3 РЕАЛІЗАЦІЯ МЕТОДУ РЕЗЕРВУВАННЯ ОБЧИСЛЮВАЛЬНИХ РЕСУРСІВ ДЛЯ ЗАБЕЗПЕЧЕННЯ ЖИВУЧОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Основні особливості програмування резервування обчислювальних ресурсів для забезпечення живучості програмного забезпечення

К основним проблеми, що пов'язані з програмуванням із резервами, включаючи дизайнерські рішення та компроміси, які повинен зробити програміст можливо віднести:

- як структурувати програми, щоб скористатися резервами;
- як відобразити резерви на структурі програми;
- як слід розмірювати параметри бронювання;
- як адаптивні програми повинні регулювати рівні бронювання.

Можна уявити програму як граф обчислювальних вузлів, і кожен обчислювальний вузол має з ним пов'язаний резерв. Визначення, які саме резерви слід розподіляти, якими мають бути їх параметри резервування та як резерви повинні бути пов'язані з цими обчисленнями, включає проектні рішення, які впливають на структуру програми.

Наприклад, програміст повинен вирішити, чи додатки, які залежать один від одного, явно узгоджуватимуть між собою вимоги щодо часу щодо конкретних послуг, які вони надають один одному. Альтернативою є виділення резервів ресурсів для їх сумісної діяльності, а потім передача цих резервів у міру переходу абстрактної діяльності одне до іншого. У першому випадку розподіл вимог та чітка специфікація вимог щодо часу для кожного обчислення в кожній програмі створює велику кількість обчислень, яку потрібно зробити. В останньому випадку вимоги узагальнюють всю діяльність, не вказуючи кожну деталь на цьому шляху. Поки кожна фаза діяльності дотримується кількох правил, таких як не введення непотрібних

затримок у загальну діяльність, однакові вимоги щодо термінів на високому рівні можуть бути задоволені без надмірного розбирання програм.

Інше дизайнерське рішення, яке розглядається тут, стосується гнучкості програм, що використовують резерви. Жорсткі програми реального часу, як правило, визначають фіксовані параметри бронювання. Адаптивні програми можуть мати можливість контролювати використання ресурсів та коригувати параметри бронювання відповідно до їхньої поведінки з часом. Вони можуть навіть мати можливість вибрати різні алгоритми з різною семантикою та різними характеристиками продуктивності, щоб налаштувати час обчислень.

Нарешті, у цій главі розглядається питання програмування з кількома ресурсами. Це вимагає розбиття додатків на підрахування в тих точках, де потрібні різні ресурси. Координація резервування ресурсів на декількох ресурсах для задоволення наскрізних обмежень часу вимагає ретельного проектування. Описано два підходи з використанням резервів.

3.2 Врахування резервів при розробці додатків

Зазвичай програми включають одну або кілька одночасних функцій. З кожною функцією може бути пов'язаний потік, і кожна функція має графік викликів, що описує підпрограми, які викликаються кожною програмою. Графік викликів розширений, включаючи запити на зовнішні сервери або системні служби, здійснені кожною підпрограмою.

Програму можна розділити на окремі функції. Кожна з них періодично виконує обчислення, які можуть бути розбиті на підобчислення шляхом розділення обчислень при викликах процедур, віддалених викликах процедур (RPC) та системних перехопленнях. Результат схожий на граф викликів, що включає запити до серверів, баз даних та операційної системи.

Для цілей цього аналізу програма визначається як абстрактний потік управління, який запускається в процесі і переміщується до серверів рівня

користувача та операційної системи, коли здійснюються виклики цим серверам та системі. Це схоже на потоки, що обходять об'єкти в хмарах [26]. Така абстрактна модель потоку відповідає стилю синхронного програмування, який на відміну від асинхронного стилю програмування, де дії по суті розгалужуються, роблячи асинхронні запити на обслуговування до зовнішніх серверів або примітивів ядра. Наприклад, розглянемо анімаційний додаток, який синтетично генерує анімаційні кадри в режимі реального часу. Додаток складається з двох видів діяльності: одного - для створення та відображення анімаційних кадрів, а другого - для обробки подій користувальницького інтерфейсу, таких як запити щодо зміни розміру вікна анімації.

Кожна з цих операцій може викликати модулі в одному адресному просторі, здійснювати RPC до серверів або робити системні дзвінки. Згідно з цим визначенням, коли (синхронний) RPC робиться на сервері, дія «переміщується» на сервер на час обчислення сервера, а потім дія повертається клієнту, коли запит повертається. Якщо сервер повинен викликати інший сервер синхронно, дія перейде на другий сервер на час реалізації запита. Те саме стосується системного виклику. Коли здійснюється системний запит, дія переходить у діючу систему і повертається, коли запит повертається.

Робота програми може бути періодичною. Наприклад, розглянемо діяльність з генерації кадру програми анімації. Припустимо, ця діяльність бере початок у підпрограмі (яка називається process_frame), яка періодично викликається кожні 33 мс для обробки та відображення кадрів. Тепер припустимо, що process_frame викликає generator_frame і display_frame, які в кінцевому підсумку виконують RPC для сервера віконної системи, який отримує доступ до буфера кадру. На рисунку 3.1 показаний приклад графіка викликів, коріння якого знаходиться у функції process_frame.

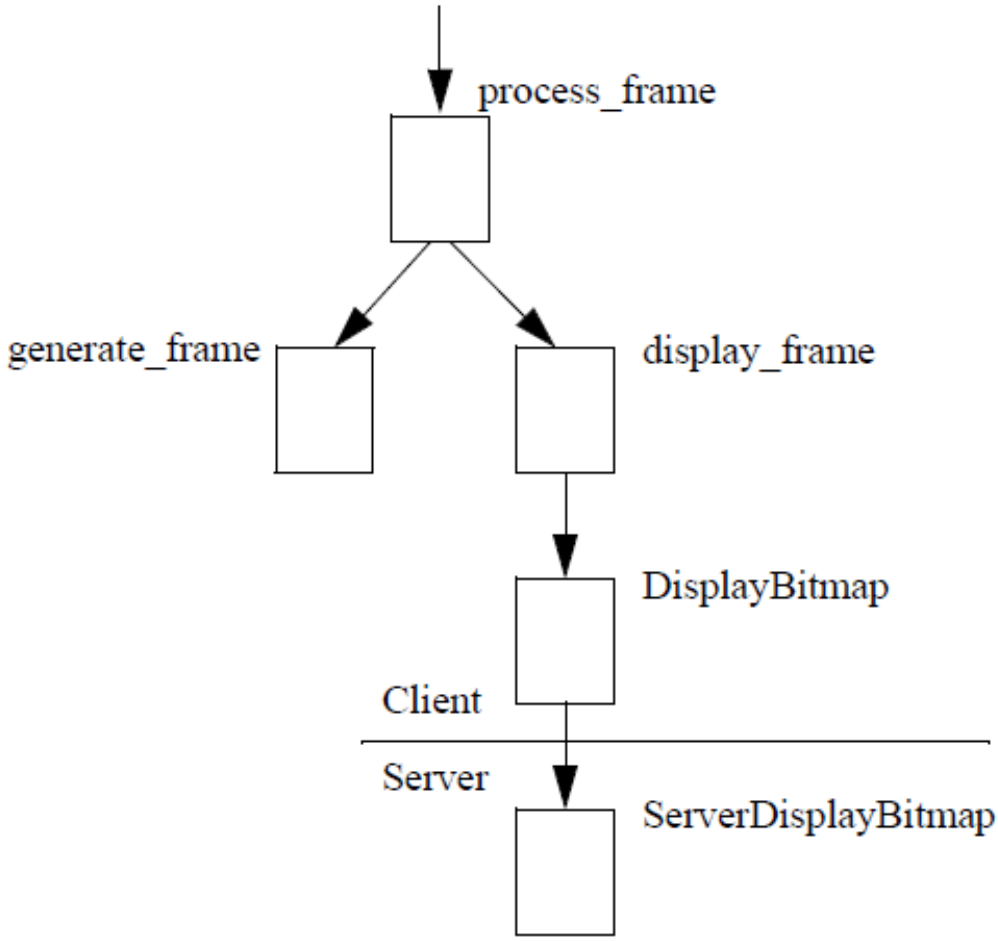

Рисунок 3.1 – Графік викликів для генерації та відображення кадру

Цей графік викликів включає RPC від клієнта до сервера. Підпрограма DisplayBitmap робить RPC в клієнті, а підпрограма ServerDisplayBitmap на сервері продовжує виконання функції. Таким чином, цей графік фіксує всі етапи анімаційної діяльності.

3.3 Реалізація резервування для забезпечення живучості

Резерви процесора ефективно виконувати за допомогою операційної системи Mach Real-Time. Обговоримо програми, які були модифіковані для використання резервів процесорів, програмне забезпечення для обробки мережевих протоколів, модифіковане для використання резервів, менеджер

QOS для узгодження розподілу ресурсів із програмами та інструменти для моніторингу резервів.

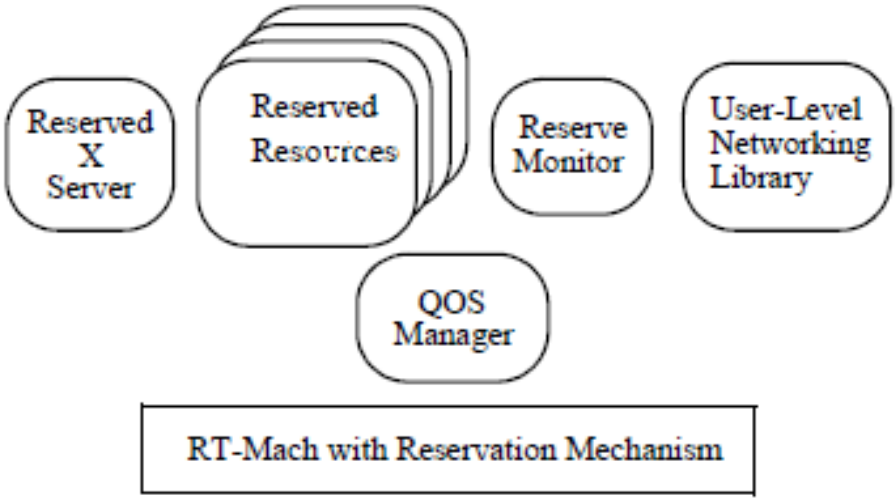

Рисунок 3.2 – Системні компоненти

По-перше, це реалізація резервів процесора в ядрі RT-Mach, що є основою для решти робіт із програми. Резерви були реалізовані як нова абстракція ядра з операціями для створення / завершення, запиту параметрів резервування, прив'язки потоків до резервів та вилучення інформації про використання про резерви.

Було модифіковано декілька реальних додатків для використання резервів процесорів, зокрема: відеопрогравач Quick-Time, розроблений в CMU, під назвою QTPlay, декодер MPEG під назвою mpeg_play [93] та версія сервера X [34].

Версія бібліотеки сокетів на рівні користувача [70] була модифікована для використання також резервів. Ця реалізація сокета підтримує передбачувану продуктивність програм, які надсилають та отримують мережеві пакети.

Менеджер QOS був реалізований, щоб забезпечити більш складні узгодження параметрів резерву, ніж ті, що передбачені самим механізмом

ядра. Менеджер QOS взаємодіє з програмами, намагаючись збалансувати використання ресурсів та домовлятися з програмами, коли виникають конфлікти в запитах на резервування ресурсів.

Були впроваджені інструменти, що допомагають управляти запасами та контролювати розподіл ресурсів та вимірювати використання. Додаток rmon - це резервний монітор, який надає графічний інтерфейс користувача для резервів. Він відображає рівні бронювання та використання в недавній історії, а також дозволяє користувачеві змінювати параметри бронювання з графічного інтерфейсу.

Реалізація резервів процесора в RT-Mach передбачала додавання нової абстракції резервів, реалізацію операцій над резервами, створення нового планувальника та додавання коду для точного вимірювання використання. На додаток до нового планувальника, реалізація резерву також потребувала змін у механізмах пріоритетного успадкування RT-Mach для підтримки успадкування резервів та розповсюдження резервів.

Резервною абстракцією в RT-Mach керується подібно до інших абстракцій, таких як хости, набори процесорів, завдання, потоки тощо, що виникли в Mach 3.0 [11]. На ці типи ресурсів у Mach посилаються порти, які використовуються як можливості.

У RT-Mach резерви процесорів розподіляються з наборів процесорів. У уніпроцесорній версії існує лише один набір процесорів, тому всі резерви походять від цього набору процесорів.

Абстракції, такі як завдання та потоки, пропонують основні операції, такі як створення, знищення, отримання атрибутів та встановлення атрибутів. Основні операції з резервами такі:

- reserve_create (поза резервом) Створює новий резерв процесора і повертає його як вихідний параметр;

- reserve_terminate (резерв) Припиняє дію даного резерву, роблячи його зарезервовану потужність доступною для інших запитів;

- reserve_set_attribute(reserve, attr_name, attr_value, attr_value_size).

Встановіть значення атрибута резерву;

- reserve_get_attribute(reserve, attr_name, out attr_value, out attr_value_size) Отримати значення атрибута резерву;

- processor_set_reserves(processor_set, out reserve_list) Повертає список резервів, пов'язаних із заданим набором процесорів.

Операції get attribute і set attribute надають програмісту доступ до деяких атрибутів резервів. Зовні видимі атрибути, які є резерви, з’являються у наступному списку. Типи даних для атрибутів наводяться в дужках після назв атрибутів; Int - ціле число, timespec_t вказує значення часу, mach_reserve_name_t - рядок фіксованої довжини, а boolean_t - логічний прапор.

Контрольний пункт відбувається на кожному періоді для кожного резерву. У той час накопичене використання реєструється разом із абсолютним часом границі періоду. Ця інформація може бути використана пізніше програмами або інструментами моніторингу, які потребують інформації про те, скільки використання було зараховано на резерв у певний період бронювання.

Є кілька інших атрибутів резерву, які використовуються лише внутрішньо. Вони складають частину стану планування для резерву і недоступні через операцію get attribute. Ці внутрішні атрибути:

- reserved(boolean_t) Біт, який вказує, перебуває резерв у зарезервованому режимі чи безрезервному режимі;

- wait_replenish (boolean_t) Внутрішній біт, що вказує на те, що програмна функція вичерпалася за поточний період. Черга програм чекає поповнення;

- start(timespec_t) Абсолютний час, коли починався перший період для резерву.

Спочатку створений резерв не має пов'язаного резервування ресурсів. Отримання резервування ресурсів для резерву вимагає додаткового дзвінка. Наступна операція дозволяє програмісту вказувати параметри резервування

та запитувати бронювання. Цей дзвінок використовується, коли резервування не пов'язане з резервом, або якщо програміст бажає запросити бронювання з параметрами, які відрізняються від резервування, пов'язаного з резервом.

reserve_request(reserve, reservation_parameters) Запит на за резервування ресурсу, який повинен бути виділений в чергу. Абонент надає параметри бронювання. Параметри бронювання включають бажаний зарезервований час на період, сам період та час початку, щоб бронювання набрало чинності.

Ця операція використовується для запиту бронювання з певними параметрами. Якщо раніше не було бронювання, пов'язаного з резервом, і запит на бронювання вдалося виконати, тоді операція повертає успіх, і бронювання надається для цього резерву. Якщо запит на бронювання не вдається, операція повертає помилку, і резерв залишається без бронювання.

Якщо програма вже мала бронювання під час здійснення запиту, поведінка полягає в наступному. Новий запит на бронювання буде задоволений, нові параметри бронювання будуть пов'язані з резервом, і старе бронювання буде звільнено в цьому процесі. Якщо новий запит на бронювання не вдається, старі параметри бронювання залишаються в силі; тобто старе бронювання не буде звільнено, якщо новий запит на бронювання не може бути задоволений.

Операція запиту викликає політику контролю за допуском, щоб визначити, чи можна прийняти новий запит на бронювання з урахуванням колекції інших застережень, які вже були прийняті для ресурсу. Реалізація RT-Mach використовує тест допуску, заснований на монотонному аналізі ставки, але рішення є дещо оптимістичним, оскільки воно використовує коефіцієнт використання 90% для тестування на планування. Це базується на аналізі середньої запланованої межі [23], яка говорить, що для випадково сформованого набору завдань графічна межа складає в середньому 88%.

3.4 Управління процесом резервування

Було впроваджено менеджер з управління якістю для забезпечення прийняття рішень про розподіл ресурсів. Він експортує інтерфейс, який дозволяє програмісту створювати та припиняти резерви, запитувати бронювання на певному бажаному рівні та встановлювати переваги для мінімального рівня бронювання. Якщо зарезервоване навантаження стає високим, і сервер з труднощами надає мінімальні рівні резервування для нових запитів, сервер починає знижувати деякі раніше надані бронювання до їх мінімальних рівнів, щоб прийняти новий запит бронювання на мінімальному рівні.

Загалом, інформація про вимоги щодо розподілу ресурсів може надходити з різних джерел і з часом може змінюватися. Інформація про розподіл ресурсів може надходити від самих додатків, які можуть запитувати ресурс і вести переговори, якщо запит не може бути задоволений негайно. Це може походити від статичних вподобань користувачів щодо того, які програми повинні мати більше ресурсів за яких обставин. І він може походити з різних елементів інтерфейсу користувача, призначених для прийняття рішень щодо управління ресурсами для користувача консолі.

Менеджер ЯП використовує зібрану інформацію для прийняття політичних рішень щодо розподілу ресурсів для різних видів діяльності. Інформація може надходити від уподобань користувача файли, самі програми та графічні засоби управління ресурсами. Рисунок 3.3 узагальнює інформаційний потік, пов'язаний з менеджером QOS.

Статичні налаштування користувача, які використовує менеджер QOS, можуть походити з конфігураційного файлу, що знаходиться в домашньому каталозі користувача або в системному каталозі за замовчуванням. Такий файл може містити довільно складні правила, які менеджер QOS повинен використовувати для прийняття політичних рішень щодо розподілу. Наприклад, файл може містити правила, що вказують, як фокус користувача

повинен впливати на розподіл ресурсів. У ньому можуть бути правила, які визначають, які програми є більш важливими (наприклад, вказуючи, що аудіо / відео програми важливіші, ніж передача файлів). Можуть існувати правила про те, як тимчасові властивості вказують, які програми є більш важливими (наприклад, надання нещодавно створеним програмам переваги перед старими програмами). І можуть бути правила щодо того, як минуле використання повинно вплинути на майбутнє бронювання.

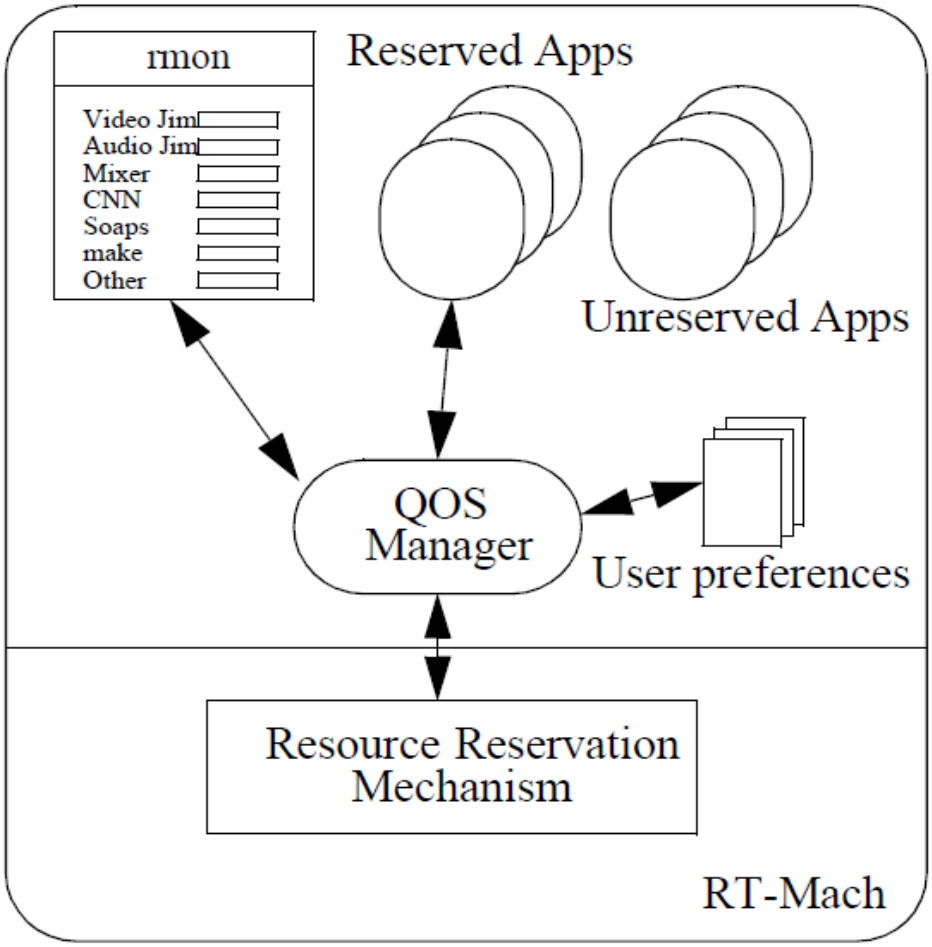

Рисунок 3.3 – Схема управління ресурсами

Динамічні вподобання користувачів можуть надходити від самих програм, від окремого інструменту або від якогось механізму, пов'язаного з диспетчером вікон. У будь-якому випадку підказки, надані користувачем, які можна взяти в користувальницькому інтерфейсі, дуже важливі для

політичних рішень, які необхідно прийняти щодо того, де розподілити ресурсний потенціал. Ці підказки можуть бути явними, коли користувач робить певні дії, щоб змінити розподіл ресурсної потужності різних видів діяльності. Або запити можуть бути неявними, як у випадку з менеджером вікон, який помічає, на якому вікні фокусується користувач (на основі положення вказівника миші), і передає цю інформацію менеджеру QOS.

Інформація про нещодавнє використання ресурсів різними підпрограмами може бути використана для визначення того, якими мають бути майбутні рівні резервування ресурсів для цих видів діяльності. Наприклад, аудіоплеєр, який отримує передачі по мережі, може стати тихим через тривале затишшя відправника. Коли це трапляється, може бути доречним помітити відсутність використання ресурсів у пов'язаному ресурсі та тимчасово зменшити рівень резервування, щоб звільнити більше можливостей, які можна зарезервувати для інших видів діяльності. Менеджер ЯП з такою функцією, безсумнівно, також забезпечить механізм для таких неактивних програм, щоб вони ожили на своєму початковому рівні резервування, як тільки вони знову стануть активними.

Політика контролю за допуском менеджера ЯП повинна узгоджуватися з контролем допуску до системи. Система бронювання має політику контролю за допуском, яка дозволяє їй застосовувати бронювання та підтримувати свою внутрішню узгодженість щодо розподілу ресурсів та забезпечення їх виконання. Менеджер QOS повинен мати версію тієї самої політики контролю прийому, щоб він міг оцінювати отримані запити на бронювання та розглядати більш складні питання, такі як те, як різні запити, які він отримує, можна поєднувати або змінювати, щоб краще поєднувати.

Ця конструкція була обрана, оскільки вона забезпечує простий і швидкий тест контроль за допуском ядра, дозволяючи довільно складним рішенням щодо контролю за допуском та переговорам проводитись у менеджерах QOS на рівні користувача. Можна поєднати дві політики, але у цього підходу є недоліки. Якби в ядрі була реалізована складна політика з

переговорами, система стала б більш складною, повільнішою та менш гнучкою. Якби ядро залежало від контролю за допуском на рівні користувача для власної потреби, воно було б уразливим до помилок у менеджерах ЯП на рівні користувача.

Менеджер QOS реагує на нові запити на бронювання, що напружують наявний ресурсний потенціал, намагаючись звільнити ресурсний ресурс із числа раніше зарезервованих видів діяльності, з урахуванням обмежень, які дозволяють ці види діяльності, виражених їх мінімальним рівнем резервування. Цю політику можна розширити, щоб врахувати інформацію про те, які види діяльності слід знизити в першу чергу, чи можна узгоджувати нові мінімуми з діяльністю, щоб звільнити ще більше потенціалу, або ж тим діям, що вимагають застережень, слід відмовити, щоб зберегти раніше зарезервовані заходи поточні рівні бронювання [79]. Інше розширення може оновити бронювання до старих бажаних значень, як тільки резервується ресурсна потужність стане достатньою.

3.5 Розробка монітору резервування ресурсів

Резервний монітор, званий rmon, надає користувачеві на консолі графічний інтерфейс користувача для моніторингу та контролю резервів процесора. Двома важливими аспектами цього інструменту є подання інформації про використання та підтримка контролю за резервуванням ресурсів.

Первинний вигляд rmon відображає основну інформацію про всі резерви процесора в системі. Ця інформація складається з назви заповідника, графічного представлення недавньої інформації про використання, нормалізованої до періоду бронювання, та самого періоду бронювання. Рисунок 3.4 показує дамп екрану цього основного вигляду.

Як показано на рисунку, кожен резерв займає один рядок на дисплеї. Кожен рядок містить такі елементи:

- кнопка деталізації - натискання кнопки деталізації відкриває інше вікно, яке відображає більш детальну інформацію, пов’язану з резервом;

- назва ресурсу – кожен ресурс може мати пов’язану з ним назву для полегшення ідентифікації;

- графічний дисплей використання - цей графік відображає смужку з довжиною, що відповідає відсотку використання ресурсів за останні кілька періодів бронювання. Використання нормується до періоду бронювання, а на графіці є маркування для позначення масштабу використання;

- період резервування - період резервування вказує інтервал усереднення вимірювання використання.

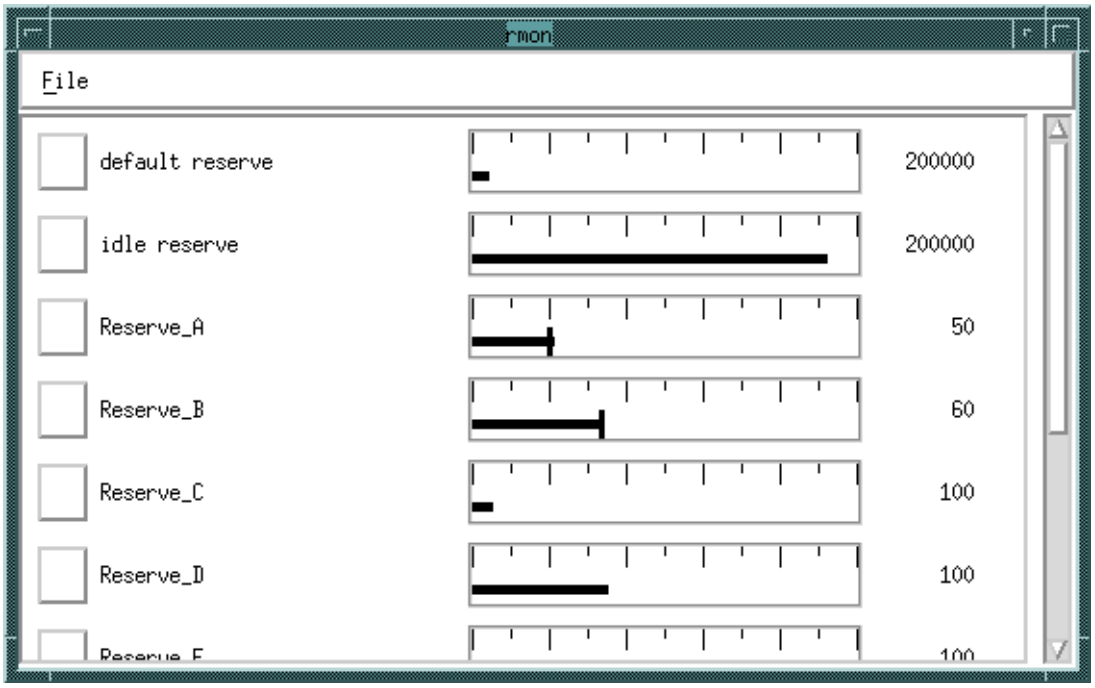

Рисунок 3.4 – Основний вигляд rmon

Коли резерви створюються та припиняються в системі, відповідні рядки створюються та знищуються на первинному поданні. Два системні резерви (так звані резерв за замовчуванням і резерв в режимі очікування) завжди існують. Тому вони завжди з’являються у вигляді. Резерв за замовчуванням - це плата за все використання програм, які не мають власних

приватних резервів. Він не має жодного фактичного резервування ресурсу, пов'язаного з цим; він просто накопичує використання незарезервованих програм. Резерв простою накопичує використання простою потоку; йому також бракує фактичного резервування ресурсів.

3.6 Експериментальна оцінка

Експериментальна оцінка, представлена в цьому розділі, відповідає на два запитання: Чи можуть зарезервовані програми досягти передбачуваної поведінки, і яка вартість для передбачуваності? Ці питання вирішуються за допомогою синтетичних орієнтирів, реальних застосувань та вимірювань окремих механізмів.

Експеремент стасовно передбачуваність показує, що для широкого кола наборів завдань, завдання в реальному часі демонструють передбачувану поведінку та відповідають своїм часовим обмеженням:

- Незалежні синтетичні вимірювання навантаження показують, що для чистих обчислень, які не взаємодіють з іншими завданнями, механізм резервування успішно гарантує обмеження часу.
- Експерименти «клієнт/сервер» показують, що механізм розповсюдження резервів допомагає гарантувати обмеження часу клієнта, навіть коли є багато клієнтів із застереженнями та без них.
- Результати експериментів з відеопрогравачем і сервером показують, як ця пара клієнт / сервер координується для задоволення часових обмежень відеопрогравача, навіть коли є беззастережні клієнти, які змагаються за увагу серверу.
- Досвід роботи з декодером та сервером демонструє, як програма може запуститись з неточною оцінкою необхідної часу, а потім відрегулює параметри резервування, щоб збалансувати потреби у використанні та доступність ресурсів.
- Досвід роботи зі структурою програмного забезпечення мережевого

протоколу на основі бібліотеки показує, що обробка протоколів для додатків у реальному часі може бути гарантовано за допомогою резервів процесора.

Використовувалися дві методики вимірювання:

- Порівняння витрат на планування системи для періодичних програм у режимі реального часу, що використовують резерви, та періодичних програм, які не використовують резерви, показує, що вартість варіюється, як очікувалось, залежно від періоду програми.

- Вимірювання різних внутрішніх операцій, таких як резервний час перемикання, обробка таймера перевитрати, обробка таймера поповнення та операції контрольних пунктів використання, забезпечують засіб оцінки вартості планування для зарезервованих наборів завдань.

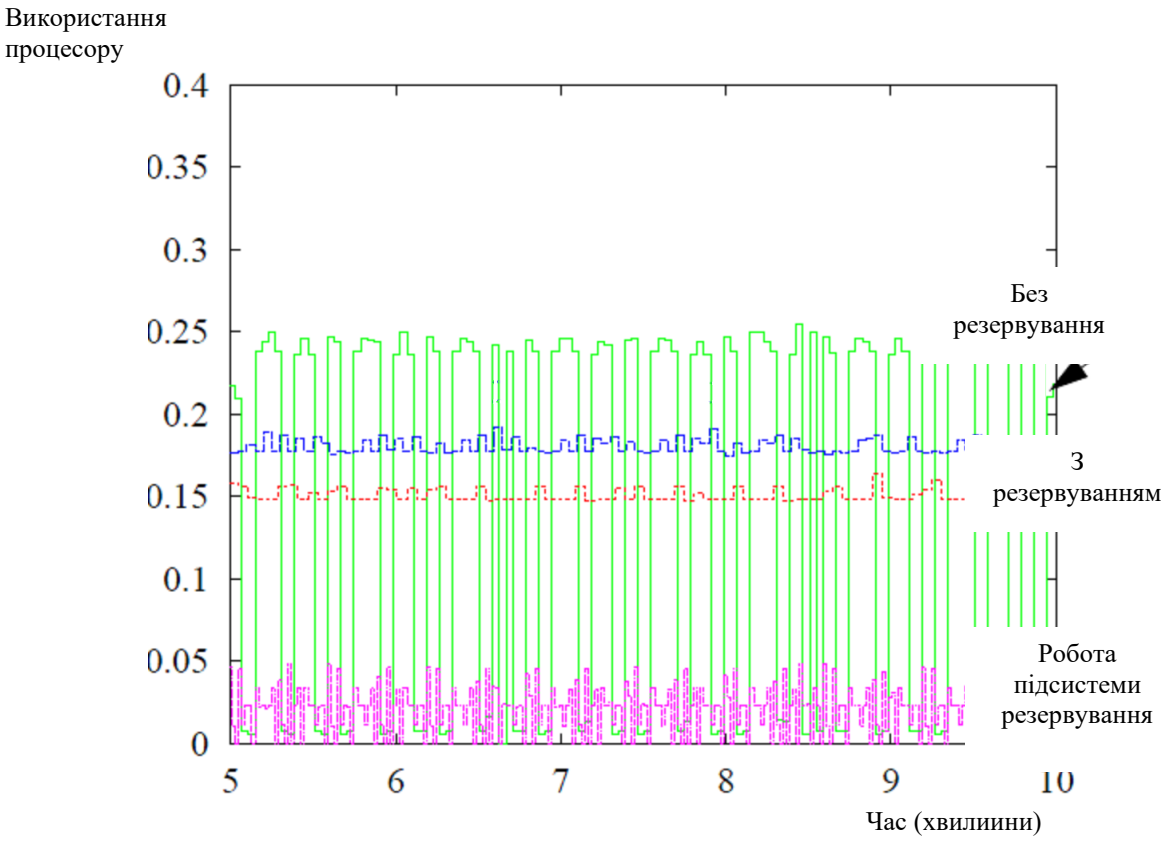

Рисунок 3.5 – Використання процесора під час функціонуваня системи

Результати експерименту представлені на рисунку 3.5. Ці результати показують, що у випадку, коли зарезервовані клієнти конкурують з

безрезервним клієнтом за один сервер, зарезервовані клієнти все ще можуть задовольнити свої обмеження часу.

Зарезервований клієнт із обчисленням 8 мс / 50 мс має середнє використання 0,181. Він має 5-процентиль 0,176 і 95-процентиль 0,187. Зарезервований клієнт з обчисленням 8 мс / 60 мс має середнє використання 0,151 з 5-процентилем 0,148 і 95-процентилем 0,158. Таким чином, ці зарезервовані програми можуть отримати час процесора, який вони зарезервували. Як видно з графіка, клієнту, що не зарезервований, вдається завершити обчислення протягом одних періодів, але не в інших. Таким чином, функція використання рухається туди-сюди між тим, як отримувати приблизно 0,22 використання в періоди, коли обчислення завершено, і отримувати 0 використання у періоди, коли вона не може завершити обчислення. Середнє використання для цього безрезервного клієнта становить 0,131; 5-процентний становить 0,0059, а 95-процентний - 0,222.

ВИСНОВКИ

У магістерській атестаційній роботі представлена вичерпна модель, що описує запаси ресурсів, операції, які вони підтримують, алгоритм планування та механізм забезпечення, а також те, як резервування для різних типів ресурсів можна інкапсулювати в єдину структуру з метою підтримки живучості програмної системи. Впровадження та експериментальна оцінка демонструють, що реальні програми з нетривіальними взаємодіями клієнт / сервер можуть досягти передбачуваних показників у реальному часі, використовуючи резерви ресурсів. Резерви забезпечують цю передбачуваність, навіть коли існує багато додатків у режимі реального часу та не в режимі реального часу, які конкурують за однакові ресурси.

Ця робота показує, що системні механізми забезпечення живучості, які розглядають всю діяльність, важливі для управління ресурсами в режимі реального часу. Крім того, це показує, що дотримання резервування є суттєвим, інакше ефективність системи падає. Для досягнення передбачуваної поведінки корисні такі методи програмування, як архітектура конвеєрів програмного забезпечення із синхронізованими періодами та термінами.

Абстракція резервів ресурсів забезпечує модель того, як алгоритми планування та аналіз у режимі реального часу можуть бути інтегровані в структуру операційної системи. Це більш ефективно на відміну від специфікації алгоритмів планування та аналізу в контексті спрощеної моделі завдань, де багато питань практичних систем та проблем програмування ігноруються. Дві ключові особливості абстракції - це гнучке прив'язування резервів до потоків та забезпечення параметрів резервування.

Оскільки резерви є первинними об'єктами в системі, а не тісно та постійно прив'язані до потоків (або процесів), управління ресурсами набагато простіше. Наприклад, параметри резервування можуть виділятися для

резерву потокам, а потім посилання на резерв може передаватися постачальникам системних послуг, що викликаються потоком. Дозволяючи прив'язування резервів до потоків, резерви можна передавати, а використання ресурсів для абстрактної діяльності відстежується та гарантується протягом усіх викликів сервера.

Механізм забезпечення спрощує розробку та налагодження програм для програмістів жорстких та програмних додатків реального часу. Програмісти жорстких програм реального часу можуть використовувати механізм накопичення для вимірювання вимог свого коду під час розробки. Під час виконання механізм забезпечення та вимірювання можуть бути використаний для ізоляції помилок синхронізації. Програмісти програм у режимі реального часу можуть використовувати механізм забезпечення, щоб забезпечити ізоляцію між програмами та надати інформацію про вимоги щодо використання ресурсів для адаптивних програм. Це позбавляє програміста від вичерпних вимірювань та аналізу, зазвичай необхідних для досягнення передбачуваності в режимі реального часу.

Робота, описана в цьому документі, відкриває багато шляхів для подальших досліджень. Забезпечуючи загальну структуру та тестову платформу для резервування ресурсів операційної системи, ця робота забезпечує конкретний контекст для багатьох дослідницьких тем, таких як, алгоритми розподілу ресурсів та адаптивні методи програмування програм.

Ця робота щодо резервування ресурсів також забезпечує пункт проектування для порівняння з іншими підходами до проектування системи та для інших парадигм планування. Ідея примусового резервування ресурсів також може бути використана як основний матеріал для вивчення проблем вищого рівня, таких як роль користувача в управлінні ресурсами.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Волк М.О., Колюжний В.Д., Демчук В.Г. Методи резервування обчислювальних ресурсів для забезпечення живучості програмного забезпечення. Проблеми інформатизації: Матеріали восьмої міжнародної науково-технічної конференції. Черкаси – Баку – Бельсько – Бяла – Харків, 26 – 27 листопада 2020 року. с. 50

2. John Strassner and Jerey O. Kephart, \Autonomic Systems and Networks: Theory and Practice, Network Operations and Management Symposium (NOMS), 2016.

3. Brownsword, Lisa; Obemdorf, Tricia; & Sledge, Carol A. Developing New Processes for COTS-based Systems. IEEE Software 17,4 (July/August 2014): 48-55.

4. Carrol, John M. Five Reasons for Scenario-Based Design. Proceedings of the Thirty-Second Annual Hawaii International Conference on Systems Sciences. Maui, Hawaii, Jan. 5-8, 1999. Los Alamitos, CA: IEEE Computer Society Press, 2009.

5. Department of Defense. Department of Defense Trusted Computer System Evaluation Criteria. DoD 5200.28-STD. National Computer Security Center, Department of Defense Computer Security Center, 2015.

6. Ebert, Christof. Dealing with Nonfunctional Requirements in Large Software Systems. Annals of Software Engineering 3 (September 1997): 367-395.

7. Ellison, Robert; Fisher, David; Linger, Richard; Lipson, Howard; Longstaff, Thomas; & Mead, Nancy. A Survivable Network Analysis Method. Proceedings ofthe 2018 IEEE Information Survivability Workshop. Orlando, Florida, Oct. 28-30, 2018. Los Alamitos, CA: IEEE Computer Society, 2018.

8. A. K. Agrawala, K. D. Gordon, and P. Hwang, editors. Mission Critical Operating Systems. IOS Press, Amsterdam, 2012.

9. D. Anastassiou. Digital Television. Proceedings of the IEEE, 82(4):510–

519, April 2014.

10. D. P. Anderson, S. Tzou, R. Wahbe, R. Govindan, and M. Andrews. Support for Continuous Media in the DASH System. In Proceedings of the 20th International Conference on Distributed Computing Systems, pages 54–61, May 2020.

11. D. P. Anderson, R. G. Herrtwich, and C. Schaefer. SRP: A Resource Reservation Protocol for Guaranteed-Performance Communication in the Internet. Technical Report TR-90-006, International Computer Science Institute, February 2016.

12. D. P. Anderson and R. Kuivila. A System for Computer Music Performance. ACM Transactions on Computer Systems, 8(1), February 2011.

13. N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. Hard Real-Time Scheduling: The Deadline-Monotonic Approach. In Eighth IEEE Workshop on Real-Time Operating Systems and Software, pages 133–137, May 2017.

14. T. P. Baker. Stack-based Scheduling of Real Time Processes. The Journal of Real-Time Systems, 3(1):67–99, March 2011.

15. P. Barham, M. Hayter, D. McAuley, and I. Pratt. Devices on the Desk Area Network. IEEE Journal on Selected Areas in Communications, 13(4):722–732, May 2015.

16. [10] S. Baron and W. Robin Wilson. Disributed Reserveition Overview. SMPTE Journal, 6(103):391–394, June 2014.

17. J. Boykin, D. Kirschen, A. Langerman, and S. LoVerso. Programming under Mach. Addison-Wesley. 2003.

18. J. F. K. Buford, editor. Multimedia Systems. ACM Press and Addison-Wesley, 2014.

19. Frei, R., McWilliam, R., Derrick, B., Purvis, A., Tiwari, A. & Di Marzo Serugendo, G. (2013). Self-healing and self-repairing technologies. International Journal of Advanced Manufacturing Technology, 69(5), pages 1033–1061.

20. Perino, N. (2013). A framework for self-healing software systems. In Proceedings of the 2013 International Conference on Software Engineering (ICSE '13). IEEE Press, Piscataway, NJ, USA, pages 1397-1400.

21. Yaloveha V., Zavou, A., Portokalidis, G. & Keromytis. A.D. (2012). Self-healing multitier architectures using cascading rescue points. In Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC '12). ACM, New York, NY, USA, pages 379-388.

22. Bucchiarone, A., Pelliccione, P., Vattani, C. & Runge, O. (2009). SelfRepairing systems modeling and verification using AGG. JointWorking IEEE/IFIP Conference on Software Architecture &European Conference on Software Architecture. Pages 181-190.

23. Dai, Y., Xiang, Y., Li, Y., Xing, L. & Zhang, G. (2011). ConsequenceOriented Self-Healing and Autonomous Diagnosis for Highly ReliableSystems and Software. IEEE Transactions on Reliability, vol.60, no.2,pages 369 - 380.

24. Duan, S., Franklin, P., Thummala, V., Dongdong, Z. & Babu, S.(2009). Shaman: A Self-Healing Database System. IEEE 25thInternational Conference on Data Engineering (ICDE '09). Pages 1539-1542.

25. Eckardt, T., Heinzemann, C., Henkler, S., Hirsch, M., Priesterjahn, C.,& Schäfer W. (2013). Modeling and verifying dynamic communicationstructures based on graph transformations. Computer Science -Research and Development. Volume 28, issue 1, pages 3-22.

26. Frei, R., McWilliam, R., Derrick, B., Purvis, A., Tiwari, A. & DiMarzo Serugendo, G. (2013). Self-healing and self-repairingtechnologies. International Journal of Advanced ManufacturingTechnology, 69(5), pages 1033–1061.

27. Gaudin, B. & Hinchey, M. (2011). FastFIX: An approach to selfhealing. Federated Conference on Computer Science and InformationSystems (FedCSIS), pages 957-964.

28. Perino, N. (2013). A framework for self-healing software systems. InProceedings of the 2013 International Conference on SoftwareEngineering (ICSE '13). IEEE Press, Piscataway, NJ, USA, pages1397-1400.

29. Portokalidis, G. & Keromytis, A.D. (2011). REASSURE: A Selfcontained Mechanism for Healing Software Using Rescue Points.Proceedings of the 6th International Workshop on Security (IWSEC),pages 16-32, Tokyo, Japan.

30. Yuan, E., Esfahani, N. & Malek, S. (2010). A Systematic Survey ofSelf-Protecting Software Systems. ACM Transactions on Autonomousand Adaptive Systems.

31. Fuad, M.M., Deb, D. & Baek J. (2011). Self-Healing by Means ofRuntime Execution Profiling. Proceedings of 2011 InternationalConference on Computer and Information Technology (ICCIT 2011),pages 202–207, Dhaka, Bangladesh.