МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМ.ІГОРЯ СІКОРСЬКОГО»

Кафедра обчислювальної техніки

О.Корочкін, О.Русанова

ПАРАЛЕЛЬНЕ ПРОГРАМУВАННЯ
ЛАБОРАТОРНИЙ ПРАКТИКУМ

Навчальний посібник для здобувачів ступеня бакалалавр
за освітньою програмою «Комп'ютерні системи та мережі»
спеціальності 123 «Комп'ютерна інженерія»

Електронне мережне навчальне видання

Затверджено
на засіданні кафедри
обчислювальної техніки
ФІОТ НТУУ «КПІ ім. І.Сікорського»
Протокол № 10 від 25.05.2022

Київ
КПІ ім. Ігоря Сікорського
2022

ЗМІСТ

ВСТУП

Мета виконання циклу лабораторних робіт з дисципліни “Паралельне програмування ” - закріплення теоретичних знань, вмінь та навичок розробки і аналізу паралельних алгоритмів, розробки та налагодження програм для паралельних комп’ютерних систем (ПКС), отримання практичних навичок по роботі з паралельними мовами (Java, C#, Ada) та бібліотеками паралельного програмування. (WinAPI, OpenMP, MPI)

Цикл охоплює три розділи:

- Розділ 1 «Потоки» (містить одну лабораторну робот ЛР0)
- Розділ 2 «Програмування для комп’ютерних систем зі спільною пам’яттю» (містить одну лабораторну роботи ЛР1-ЛР4)
- Розділ 3 «Програмування для комп’ютерних систем з локальною пам’яттю» (містить одну лабораторну робот ЛР5).

Таким чином, цикл лабораторних робіт складає ***шість*** робіт (ЛР0-ЛР5). Студент може обрати відповідний набір лабораторних робіт з ЛР0-ЛР5 (Таблиця 1), а також завдання (З1.Х-З5.Х) в для обраної ЛР. Обрання Завдання дозволяє використовувати для виконання ЛР різні мови (бібліотеки) паралельного програмування. Наприклад, вибір для ЛР5 завдання З5.1 пов’язаний з використанням мови Ada, а вибір завдання З5.2 – з використанням бібліотеки MPI.

Таблиця 1

Оцінка	Кількість ЛР	Розділ1	Розділ 2 ПКС СП				Розділ 3 ПКС ЛП
			Семафори, мютекси, події, критичні секції, атомік- змінні		Монітори (Java, Ada)	OpenMP	Повідомлення (MPI, Ada)
			Завдання				
		ЛР0	***ЛР1***	***ЛР2***	***ЛР3***	***ЛР4***	***ЛР5***
A	6	З0.1 … З0.4	З1.1	З1.2	З3.1 або З3.2	З4.1	З5.1 або З5.2
B/C	5		З1.1 або З1.2	-	З3.1 або З3.2	З4.1	З5.1 або З5.2
D/E	4		З1.1 або З.2	-	З3.3 або З3.4	-	З5.1 або З5.2

Лабораторні роботи пов’язані з практичним застосуванням засобів роботи з потоками в сучасних мовах паралельного програмування Java, Ada, C# та бібліотеках WinAPI, MPI, OpenMP.

В on-line режими студент *викладає ЛР в хмару* та *надсилає інформацію про це викладачу на вказану пошту* до проведення заняття. Викладач через Хмару інформує студента про результати перевірку ЛР

- *ЛР зараховано* з оцінкою A-E; (студент отримує завдання на наступну роботу)

- *ЛР потребує перероблення* (є зауваження, вказуються які). Перероблена ЛР надсилається викладачеві на наступному занятті.

Сумарні оцінки за лабораторні роботи і за модульну контрольну роботу визначають підсумкову залікову оцінку з кредитного модуля.

РОЗДІЛ 1. ПОТОКИ

1.1Теоретичні відомості

Потік – абстрактне поняття, що включає опис певних дій, пов'язаних з виконанням програми (частин програми) в комп'ютерній системі. При цьому потік оформлюється так, щоб система керування потоками в ОС могла ефективно перерозподіляти ресурси системи (процесори, пам'ять, прилади введення-виведення, файли і та ін.). Потік характеризується власним набором ресурсів і виділеною для нього ділянкою оперативної пам'яті.

Поняття потоку є поширенням поняття процесу, яке вперше з'явилось в багатозадачних операційних системах і сьогодні є фундаментальним для кожної сучасної ОС. В ОС процес пов'язаний з кожною множиною прикладних або системних програм, які виконуються в комп'ютерній системі. Це дозволяє системі керування процесами ОС, яка розміщується в ядрі ОС, ефективно маніпулювати процесами за допомогою спеціального блоку керування процесом (БКП або PCB – Process Control Block) . Блок керування процесом – динамічна структура даних, яка містить основну інформацію про процес:

- ім'я процесу;
- поточний стан процесу;
- пріоритет процесу;
- місце розміщення процесу в пам'яті;
- ресурси, що пов'язані з процесом та ін.

Поняття потоку з'явилось з появою систем реального часу, а також багатопроцесорних систем і, відповідно, паралельного програмування.

Стосовно паралельного програмування, потоки – це частини однієї програми користувача, які виконуються одночасно. Такі процеси отримали назву легкі процеси (lightweight processes). Для позначення легких процесів використовують також терміни потік (Java, C, C#) або задача (Ада, MPI). Тому традиційні процеси ОС називають важкими процесами (heavyweight processes).

Під час виконання кількох програм у комп'ютерній системі або звичайному комп'ютері відбувається постійне перемикання з одного процесу на інший. Те ж саме відбувається і для паралельної програми, якщо вона включає кілька легких процесів (потоків). Але час, який витрачається на перемикання для легких процесів, менший, ніж для важких. Ще одна відмінність важких та легких процесів полягає в тому, що легкі процеси постійно взаємодіють, оскільки є частинами однієї програми (так звані *тісно зв'язані* процеси), в той час, як важкі процеси взаємодіють рідко (*слабко зв'язані* процеси).

Стани потоку. Керування процесом (потоком) включає виконання деяких операцій над ним, дозволяючи процесу пройти визначені стани (рис. 2.1). Види станів процесу визначаються конкретною ОС і системою керування процесами, але змістять здебільшого такі стани:

Породження – створення процесу і підготовка до першого виконання в системі;

Готовності – процес готовий до виконання і чекає звільнення процесора;

Виконання – процес виконується на процесорі;

Блокування – процес призупинений через очікування визначеної події: завершення введення даних, завершення заданого часу очікування, сигналу від іншого процесу, звільнення ресурсу та ін.;

Завершення – нормальне або аварійне завершення виконання процесу.

Причинами переходів є операції, що виконуються ОС над процесом: породження і завершення процесу, блокування і розблокування, завершення кванта часу роботи процесора, операції уведення-виведення та ін.

Процес також може мати особливий стан, який отримав назву *тупик* (*deadlock*). Процес у тупиковому стані блокований і очікує на подію, яка ніколи не відбудеться (сигналу від іншого процесу, звільнення ресурсу, введення даних та інше). Це зумовлює неможливість його продовження і, як наслідок, – зависання програми в цілому. Тупики – одна з головних проблем, що виникають під час виконання паралельних програм.

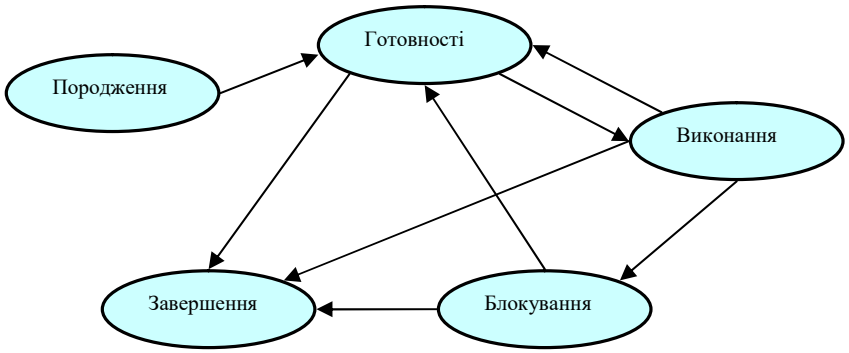


Рис. 1.1. Діаграма зміни станів процесу

Боротьба з тупиками зводиться до таких дій:

- відвернення тупиків;
- запобігання тупикам;
- визначення тупика;
- ліквідація тупика.

У цілому тупики є істотною загрозою і боротьбі з ними слід приділяти особливу увагу під час розроблення та налагодження паралельної програми..

Засоби роботи з потоками розподіляються на бібліотечні та мовні. Прикладами перших є засоби бібліотек WinAPI, PVM, MPI, OpenMP, Pthread, які реалізовані у вигляді набору ресурсів (функцій, змінних, типів). Вони дозволяють роботу з процесами в будь-яких мовах програмування, зокрема і тих, які не мають вбудованих засобів роботи з процесами. Пізніші мови програмування, такі як C#, Java або Ада, безпосередньо мають засоби роботи з

потоками, які вбудовані в мови, що забезпечує ефективне їх виконання та взаємодію.

Незалежно від реалізації (бібліотечної або мовної) засоби роботи з потоками мають забезпечити розробнику паралельного додатка такі можливості:

- об’явити потік (групу потоків);
- установити пріоритет потоку;
- запустити потік на виконання,
- призупинити потік на визначений час;
- блокувати та розблокувати потік;
- організувати взаємодію з іншими потоками (синхронізація або пересилання даних);
- завершити потік.

Існує декілька шляхів до створення потоку:

- через використання спеціальних модулів (класів), яки дозволяють описати потік (Java, Ada);
- через так звані потокові функції, коли дії майбутнього потоку описуються за допомогою функції, яка потім використовується для створення процесу (WinAPI, C#);
- через створення копій програмного модулю (MPI, PVM);
- через визначення в програмі ділянок, які можуть виконуватися паралельно (Occam, OpenMP).

1.2 Лабораторна робота ЛР0

Завдання. Лабораторна робота ЛР0 виконується кожним студентом. Для виконання робіт необхідно отримати варіант завдання для ЛР0, який включає номера трьох математичних функцій *F1, F2, F3* з Додатку Б (1.х, 2.х, 3.х). Функції пов’язані з виконанням векторна-матричних операцій.

Треба розробити програму за допомогою обраної мови або бібліотеки паралельного програмування, яка забезпечує паралельне виконання трьох математичних функцій ***F1, F2, F3*** згідно отриманого варіанту ЛР0.

Стандартна структура програми:

- модуль (клас) ***Data***, що містить ВСІ ресурси для створення потоків (типи, допоміжні процедури та функції та інше…);
- потоки ***T1, T2, T3***. Кожен потік здійснює дії, що необхідні для паралельного обчислення відповідної функції ***Fi***:
  - введення відповідних даних,
  - обчислення функції ***Fi,***
  - виведення результату виконання потоку Fi.

  Необхідні ресурси для побудови потоку беруться з модулю ***Data***;
- головну процедуру ***(Lab0).***

Треба виконати налагодження паралельної програми та дослідити її виконання для малих та великих значень N (N - розмір векторів та матриць, обов'язково позначаються в програмі через змінну N).

Введення вхідних даних. Для невеликих розмірів N=4 введення в потоці векторів та матриць здійсняються за допомогою клавіатуру. При цьому всі елементи векторів та матриць отримують для F1 значення 1, для F2 - значення 2, для F3 - значення 3. Треба дослідити проблеми введення з клавіатури і пояснити їх в протоколі ЛР1.

Для великих значень N >1000 введення даних вже передбачити через створювання в модулі Data ресурсів (методів), що дозволяють реалізувати введення шляхом:

- формування файлів з наступним зчитування даних з них
- встановлення всіх елементів даних заданому значенню (наприклад 1)
- використання генератору випадкових значень.

Під час виконання програми для N>1000 прослідити та проаналізувати процес завантаження програмою ядер багатоядерного процесора за допомогою Диспетчера задач ОС Windows. Зробити в протоколі висновки, що до завантаження процесора потоками.

Функції незалежні, спільних даних *не мають!*

Назва головні процедури (потоку) ***Lab0 ,*** пов'язана з номером ЛР, що виконується**,** назва потоку (задачі) ***T1-T3*** **-** з номером функції ***F1-F3*****.**

Лістинг програми повинен починатися з «шапки» - строк коментаріїв, де відображається наступна інформація: назва дисципліни, номер та назва ЛР, функції *F1, F2, F3,* ПІБ студента, група, дата.

Протокол виконання ЛР0 містить: титульний аркуш, завдання на ЛР, листінг програми з шапкою та коментарями.

Захист ЛР, якщо заняття відбувається в КПІ, здійснюється в два етапи. Спочатку студент відповідає на запитання, що пов'язані з теоретичною частиною завдання і програмою. За позитивної оцінки теоретичних знань студент показує виконання програми на комп'ютері. ЛРО оцінюється як «зарахована».

\

1.3 Варіанти завдань

Завдання 30.1. Потоки в мові Ada

Мета завдання: вивчення засобів мови Ада для роботи с потоками (задачами).

Необхідні теоретичні відомості: мова Ада забезпечує програмування паралельних процесів (потоків) за допомогою задачних модулів **(task**). Управління виконанням задач можна здійснювати через встановлення пріоритетів задач (прагма **priority**), а також через оператор **delay,** який блокує виконання задачі на заний період часу.

Задачі мають стандартну для мови структуру, тобто містять специфікацію і тіло. Специфікація задачі дозволяє описати ім'я задачі,

пріоритет, засоби взаємодії з іншими задачамі та інше. Тіло задачі визначає дії задачі.

Теоретичні відомості по програмуванню задач в мові Ада можна знайти в [1].

Завдання 30.2. Потоки в мові Java

Мета завдання: вивчення засобів мови Java для роботи с потоками.

Необхідні теоретичні відомості: мова Java забезпечує програмування паралельних процесів (потоків) за допомогою потоків **(threads**). Використається клас **Thread** або інтерфейс **Runnable.**

Потоки визначають паралельне виконання Java програми в ПКС. Управління виконанням потоків можна здійснити за допомогою пріоритетів потоків (метод **set_Priority()**), метода **sleep(),** який викликає блокування потоку на вказаний період часу

Метод **join()** використається для синхронізації основного метода з потоками, яки він запускає на виконання.

Метод **run()** визначає дії потоку при виконанні.

Теоретичні відомості по програмуванню потоків в мові Java можна знайти в [1,2, 28, 52, 56, 60].

Завдання 30.3. Потоки в мові C#

Мета роботи: вивчення засобів мови C# для роботи з потоками.

Необхідні теоретичні відомості: мова C# забезпечує можливість програмування паралельних процесів за допомогою потоків класу *Thread* з використанням концепції потокових функцій.

Теоретичні відомості по програмуванню потоків в мові C# можна знайти в [1].

Завдання 30.4.Потоки в бібліотеці WinAPI

Мета роботи: вивчення засобів бібліотеки WinAPI для роботи з потоками.

Необхідні теоретичні відомості: бібліотека WinAPI забезпечує можливість програмування паралельних процесів за допомогою функції *Create_Thread* з використанням концепції потокових функцій.

Теоретичні відомості по програмуванню потоків в бібліотеці WinAPI можна знайти в [1, 2].

РОЗДІЛ 2.
ПРОГРАМУВАННЯ ДЛЯ ПКС СП

2.1 Теоретичні відомості

Програмування для ПКС СП потребує організації взаємодії потоків, яка ґрунтується на моделі спільних змінних.

Взаємодія процесів

Взаємодія процесів пов'язана з двома основними завданнями:

- комунікацією процесів;
- синхронізацією процесів.

Комунікація процесів передбачає обмін даними між процесами. При цьому інформація передається від одного процесу до іншого в будь-якому напряму.

Синхронізація включає узгодження поведінки процесів і залежність виконання одного процесу від *подій*, що можуть бути в іншому процесі.

Механізм реалізації взаємодії процесів залежить від виду використовуваної комп'ютерної системи і в загальному випадку ґрунтується або на використанні моделі *спільних змінних* (системи зі спільною пам'яттю), або на використанні моделі *посилання повідомлень* (системи з локальною пам'яттю).

У системах зі спільною пам'яттю взаємодія процесів виконується за допомогою спільної пам'яті. Комунікація процесів і синхронізація процесів виконується через *спільні змінні.* Для передавання даних процес записує ці дані в спільні змінні, звідки їх зчитує інший процес. Синхронізація виконується за допомогою відповідних спільних змінних, які змінюються процесом, у якому видбувається подія, і зчитуються (аналізуються) процесом, що чекає на подію.

Використання спільних змінних, що доступні будь-якому процесу (це обов'язково глобальні змінні), приводить до того, що в програмі може відбуватися одночасне звертання процесів до спільних змінних, що призводить до конфлікту процесів або некоректної роботи програми. Тому програмування

з використанням моделі спільних змінних пов'язано з необхідністю вирішення специфічних проблем, які загалом зводяться до розв'язання двох основних завдань – *взаємного виключення* і *синхронізації процесів.*

Завдання взаємного виключення

Завдання взаємного виключення полягає в тому, що під час виконання двох і більше паралельних процесів може виникнути одночасне звернення процесів до одного і того ж *спільного ресурсу* (СР). Таке звернення зазвичай призводить до конфлікту процесів, що полягає або в різкому уповільненні роботи програми, або в некоректній її роботі, якщо, наприклад, один процес зчитає дані, а другий їх змінює в цей же час, або (в крайньому випадку) – до аварійного завершення програми.

Загальна схема вирішення завдання взаємного виключення ґрунтується на тому, що потрібно *призупинити* (*блокувати*) процес, який звертається до спільного ресурсу, який вже використовується в цей момент іншим процесом. *Розблокування* процесу має бути виконано одразу після звільнення спільного ресурсу.

Існують *два підходи* до вирішення завдання взаємного виключення. Перший підхід ґрунтується на *контролі процесів* і пов'язаний з виявленням у процесах ділянок, у яких вони звертаються до спільного ресурсу. Такі ділянки процесів отримали назву *критичних ділянок (КД).* Для вирішення завдання взаємного виключення необхідно *не допустити* одночасного входження процесів у свої критичні ділянки. Якщо один процес уже знаходиться в критичній ділянці, то будь-який інший процес за намагання входження в свою критичну ділянку має бути блокований доти, доки перший процес не вийде зі своєї критичної ділянки.

Класична схема організації такого контролю потребує використання операцій (примітивів) `ВХІДКД` і `ВИХІДКД`, що розміщуються відповідно перед і після критичної ділянки, тобто обрамляють критичну ділянку, створюючи "мур" навколо неї.

Алгоритм виконання операції `ВХІДКД`:

1. Перевірити, чи знаходиться будь-який інший процес у своїй критичний ділянці ?
2. Якщо знаходиться, то блокувати процес, що виконує операцію *ВХІДКД*.
3. Якщо не знаходиться, то встановити заборону на входження всіх процесів у свої критичної ділянки і дозволити цьому процесу увійти в його критичну ділянку.

Алгоритм виконання операції *ВИХІДКД*:

1 Зняти заборону на входи процесів в їх критичні ділянки.

Конкретна реалізація примітивів *ВХІДКД* і *ВИХІКД* у мовах та бібліотеках програмування виконана у вигляді механізмів семафорів, мютексів, критичних секцій.

Другий підхід до вирішення завдання взаємного виключення передбачає контроль безпосередньо за спільним ресурсом. Такий контроль може бути здійснений за допомогою оголошення належних змінних спільними ресурсами, а також за допомогою моніторів. З цією метою створюється програмний модуль (монітор), що містить в собі спільний ресурс, а також набір процедур для доступу до спільного ресурсу. Тепер доступ до спільного ресурсу з процесів можливий тільки через виклик потрібної процедури монітора.

Процедури монітора мають важливу властивість – вони *взаємно виключають* один одного. Тобто, якщо процес викликав і виконує будь-яку процедуру монітора, то виклик іншим процесом будь- якої процедури цього монітора приведе до блокування цього процесу до того часу, поки не закінчиться виконання вже розпочатої процедури монітора. Таким чином, монітор дозволяє виконання тільки однієї процедури. Якщо ввести термін «процес знаходиться в моніторі», під яким розуміти, що процес виконує будь-яку процедуру монітора, то можна стверджувати, що в моніторі може знаходитись тільки один процес. Механізм моніторів реалізується різними способами: у мові Ада – за допомогою захищених модулів, в мові Java – за допомогою синхронізованих методів.

Завдання синхронізації процесів

Завдання синхронізації двох процесів полягає в тому, що в одному процесі, наприклад *B,* у визначеній точці (*точці події*) виконується подія (обчислення даних, уведення або виведення даних і т.ін.), а другий процес *A* у визначеній точці (*точці очікування події*) блокується доти, доки ця подія не відбудеться і він зможе продовжити своє виконання. Точка події і точка очікування – це *точки синхронізації.* Якщо процес *A* вийшов на точку синхронізації, коли подія вже відбулася, то він не блокується і продовжує виконуватись.

Існує декілька схем синхронізації процесів :

- один процес очікує на подію в одному процесі;
- один процес очікує на події в кількох процесах;
- декілька процесів очікують на подію в одному процесі.

Точки *S* і *W* на рис. 3.3 – це точки синхронізації процесів; *S* означає точку посилання сигналу про подію, *W* –точку очікування сигналу про подію.

Для вирішення завдання синхронізації, яке іноді називають *синхронізацією за подіями (event synchronization)*, можна використовувати різні механізми синхронізації процесів, таки як семафори, події, монітори.

Семафори

Механізм семафорів запропонував математик Е.Дейкстра. У класичній інтерпретації механізм семафорів – це спеціальний захищений тип `Semaphore` та дві неподільні операції над змінною *S* цього типу: *P*(*S*) і *V*(*S*). Неподільність операції означає, що її не можна переривати, поки не завершиться її виконання. Бінарні семафори набувають значень 0 і 1, багатозначні (множні) семафори – значень 0, 1, . . , M.

Алгоритми операцій `P(S)` і `V(S)`:

<u>Операція P(S)</u>: (*ВХІДКД*)

1. Перевірити значення `S`.
2. Якщо `S = 0`, то блокувати процес, що виконує цю операцію.

3. Інакше `S:= S – 1` і дозволити входження процесу в критичну ділянку.

Операція V(S): (*ВИХІДКД*)

1. `S := S – 1`

Механізм семафорів є універсальним, який може бути використаний як для вирішення завдання взаємного виключення (рис. 3.4), так і для синхронізації процесів (рис. 3.5).

Монітори

Ідея монітора, яку запропонував Б. Хансен і розвинув С. Хоар, ґрунтується на об'єднанні змінних, що описують спільний ресурс, і дій, які визначають засоби доступу до спільного ресурсу. *Монітор – програмний модуль,* що містить змінні та процедури для роботи з ними, причому доступ до змінних можливий *тільки через процедури* монітора.

Монітор – засіб розподілу ресурсів і взаємодії процесів. Це призначення монітора реалізується за допомогою властивостей, якими наділені процедури монітора. Характерна особливість процедур монітора – *взаємне виключення* ними одне одного. У будь-який момент часу може виконуватися *тільки одна* процедура монітора. Якщо будь-який процес викликав і виконує процедуру монітора, то жоден процес не може виконувати будь-які процедури цього монітора. За спроби виклику іншим процесом процедури, що виконується, або іншої процедури монітора цей процес блокується і розміщується в черзі блокованих процесів доти, доки активний процес не закінчить виконання процедури монітора. Тобто в моніторі не може "знаходитись" більше одного процесу. Така властивість процедур монітора забезпечує взаємне виключення процесів, які працюють з монітором.

Загальна структура монітора:

```
monitor  Ім'я_Монітора;
        -- Опис локальних даних
        -- Опис процедур для доступу до даних
begin
        -- Ініціалізація локальних даних
```

```
end  Ім'я_Монітора;
```

У моніторі декларуються локальні змінні (спільні змінні), які захищені монітором, і процедури монітора. Значення локальних змінних можуть бути встановлені під час створення монітора. Далі значення цих змінних можуть бути прочитані або змінені процесами тільки за допомогою процедур, визначених у моніторі.

Приклад монітора:

```
monitor  Склад;
     Товар: Data;        -- спільний ресурс
     procedure На_Склад (T: in Data);
     procedure Зі_Складу(T: out Data);
begin
     Товар:= 0.0; -- ініціалізація спільного
                  -- ресурсу
end  Склад;
```

Властивості процедур монітора забезпечують вирішення завдання взаємного виключення за доступу до спільних ресурсів, об'явленими в моніторі. При цьому монітор формує чергу процесів, які викликали процедури монітора і є блокованими через зайнятість монітора (тобто спільного ресурсу).

2.2 Завдання для ЛР1-ЛР4

Завдання на лабораторну роботуЛР1-ЛР4 включає:

- структуру ПКС СП
- математичну задачу
- мову (або бібліотеку) програмування
- засоби організації взаємодії потоків.

Виконання лабораторній роботи включає розробку і аналіз алгоритму рішення вхідної задачі, створення і налагодження програми.

Звіт по лабораторної робот містить опис всіх етапів створення програми: розробки паралельного математичного алгоритму, алгоритмів всіх

паралельних задач (потоків), структурної схеми взаємодії потоків, тексту програми.

Згідно РСО, для отримання заліку студент повинен набрати більше ніж 60 балів, виконав і захистів необхідну кількість робіт (від 3-х до 5). Кожна лабораторна робота оцінюється максимально в 20 балів. Під час захисту ЛР студент відповідає на питання, які пов'язані з теоретичною і практичною частинами ЛР. Це визначає кількість балів за ЛР.

Лабораторні роботи ЛР1-ЛР4 пов'язані с програмуванням для ПКС зі спільною пам'яттю (ПКС СП). Кількість процесорів (Р) – від 4 до 6. На рис 2.1 наведений приклад структури ПКС СП.

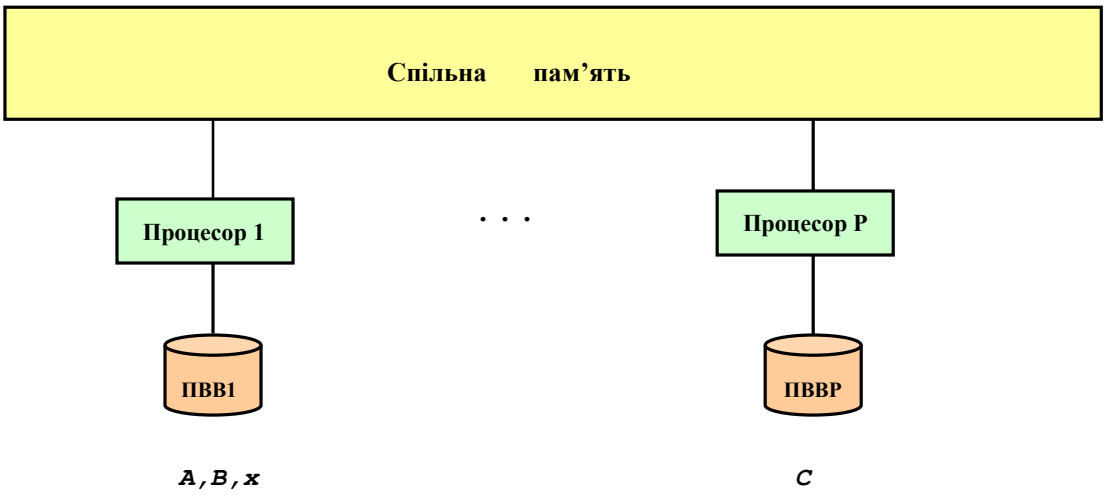


Рис. 2.1 Структура ПКС зі спільною пам'яттю

Для лабораторних робіт ЛР1-ЛР4 необхідно виконати:

Етап 1. Розробку паралельного математичного алгоритму рішення математичної задачі, де виявити спільні ресурси;

Етап 2. Розробку алгоритм кожного потоку (Т1 – Тр) з визначенням критичних ділянок (КД) і точок синхронізації (Wij , Sij);

Етап 3. Розробку структурної схеми взаємодії потоків (Див. **Додаток А)**, де застосувати ВСІ вказані засоби взаємодії потоків

Етап 4. Розробку програму (обов'язкові "шапка", коментарі до основних частин програми пов'язаних з синхронізацією потоків, обчисленнями, рішенням завдання взаємного виключення).

Для готової програми:

- виконати налагодження програми;
- отримати *правильні* результати обчислень.
- за допомогою Диспетчеру задач Windows проконтролювати завантаження ядер процесору.

2.3 Варіанти завдань

Завдання до ЛР1 та ЛР2. Виконання ЛР1 та ЛР2 пов'язано з використанням нізкорівневих засобів організації взаємодії потоків. Це семафори, мютекси, події, критичні секції, атомік –змінні. Використаються мови C# і бібліотека WinAPI.

Завдання З1.1 Мова C#

Мета завдання: розробка програми для ПКС СП

Мова програмування: C#

Засоби організації взаємодії процесів: семафори, мютекси, події, критичні секції бібліотеки, бар'єри , атомік змінні (типи)

Література: [1, Розділ 6, с. 162-165.],

Завдання З1.2 Бібіліотека WinAPI

Мета завдання: розробка програми для ПКС зі СП

Мова програмування: WinAPI

Засоби організації взаємодії процесів: семафори, мютекси, події, критичні секції, барє атомік змінні (типи) ри

Література: [1]

Завдання до ЛР3. Виконання ЛР3 пов'язано з використанням механізму моніторів для організації взаємодії потоків. Використаються мови Ада і Java.

Завдання З3.1. Мова Ада. Захищені модулі

Мета завдання: розробка програми для ПКС зі СП

Мова програмування: Ада

Засоби організації взаємодії процесів: монітори (захищені модулі мови Ада).

Література: [1, Розділ 6, с. 165-170.], [2, Цикл 5, с. 104 -107]

Завдання З3.2 Java-монітори

Мета завдання: розробка програми для ПКС зі СП

Мова програмування: Java

Засоби організації взаємодії процесів: монітори мови Java,

Література: [1, Розділ 6, с. 170-174.]

Завдання до ЛР4.Виконання ЛР4 пов'язано з використанням бібліотеки OpenMP. Використаються мова за вибором.

Завдання З4.1 Бібліотека OpenMP.

Мета зпадання: розробка програми для ПКС зі СП

Мова програмування: за вибором (C, Java, C++,)

Засоби організації взаємодії процесів: бар'єри, критичні секції OpenMP

Література: [1, с. 28 -32]

==========

РОЗДІЛ 3.
ПРОГРАМУВАННЯ ДЛЯ ПКС ЛП

3.1 Теоретичні відомості

Цей розділ присвячено питанням організації обчислень в комп'ютерних системах з розподіленою пам'яттю на підставі посилання повідомлень. Розглянуто реалізації механізму повідомлень в мовах Оккам і Ада, бібліотеках PVM і MPI .

Загальна схема передачі повідомлень між процесами ґрунтується на використанні операцій `Send()` і `Receive()`. За допомогою операції `Send()` процес відправляє дані іншому процесу, який отримує ці дані за допомогою операції `Receive()`:

```
Процес1                    Процес2
  . . .                      . . .
Send(Дані);                Receive(Дані);
```

Існують різноманітні схеми використання операцій `Send()` і `Receive()`: іменовані, синхронні, асинхронні, з блокуванням або без блокування та ін.

Іменована схема потребує застосування імен взаємодіючих процесів в операціях `Send()` і `Receive()`:

```
Send   (Ім'я_Процесу_Одержувача,  Дані)
Receive(Ім'я_Процесу_Відправника, Дані)
```

У разі застосування непрямої схеми взаємодії процесів із використанням поштової скриньки, буфера, каналу, лінка або труби (pipe) треба вказувати ім'я засобу, що застосовується:

```
Send   (Буфер, Дані)
```

```
Receive(Буфер, Дані)
```

Схема, у якій вказані імена обох процесів – відправника і одержувача, має назву *симетричної. Асиметрична* схема потребує вказати ім'я лише одного процесу, до якого звертаються. Асиметрична схема вигідна в моделі *клієнт–сервер*, коли процес–сервер не знає і не повинен знати, з яким процесом–клієнтом він буде взаємодіяти.

Існує кілька варіантів непрямої взаємодії процесів:

- один процес з кількома процесами;
- кілька процесів з кількома процесами;
- кілька процесів з одним процесом;
- один процес з одним процесом.

Передача повідомлень може бути також синхронною або асинхронною. Синхронна схема ґрунтується на тому, що процес, який виконує операцію `Send()` або `Receive()`, блокується, поки другий процес не вийде на приймання повідомлення (операція `Receive())` або отримання повідомлення (операція `Receive()).`

Оптимальною з погляду розробника є можливість використання різних видів операцій передавання-приймання повідомлень, як це зроблено в бібліотеці PVM, яка містить набір з кількох видів операцій `Send()` і `Receive().`

3.2 Лабораторна робота ЛР5

Лабораторна робота ЛР5 розділу (завдання 35.1 і 35.2) пов'язані з програмуванням обчислень для паралельних комп'ютерних систем з локальною (розподіленою) пам'яттю (ПКС ЛП). Приклад структури ПКС ЛП (лінійна структура) наведений на рис. 3.1) **.**

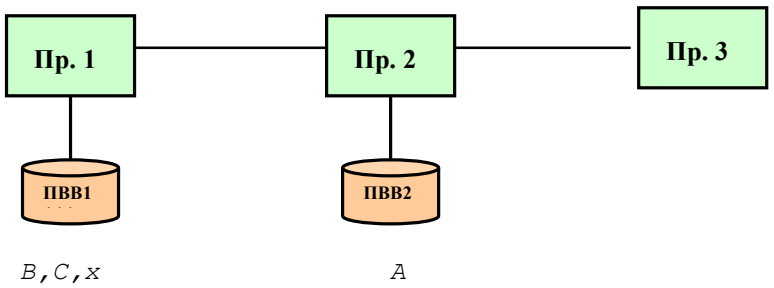


Рис. 3.1. Структура ПКС з локальною пам'яттю

Для лабораторної роботи ЛР5 (обирається одне завдання З5.1 або З5.2) необхідне:

- розробити паралельний алгоритм рішення математичної задачі;
- описати алгоритм кожного процесу (Т1 - Тр);
- розробити структурну схему взаємодії задач (через механізм рандеву для З5.1, через конструкції `send /receive` MPI для З5.2);
- розробити програму
- виконати налагодження програми;
- отримати *правильні* результати обчислень.
- за допомогою Диспетчеру задач Windows проконтролювати завантаження ядер процесору.

3.3 Варіанти завдань до ЛР5

Для виконання лабораторної роботи ЛР5 обирається одне завдання: З5.1 або З5.2

Завдання З5.1 Мова Ада. Рандеву

Мета роботи: розробка програми для ПКС з ЛП

Мова програмування: Ада

Засоби організації взаємодії процесів: механізм рандеву

Література: [1, Розділ 6, с. 175-181.],

Завдання 35.2 Бібліотека MPI

Мета роботи: розробка програми для ПКС з ЛП

Мова програмування: за вибором

Засоби організації взаємодії процесів: посилання повідомлень (Send/Receive, колективні операції Bcast, Gather, Reduce, Scatter та граф зв'язку).

Література: [1. Розділ 6],

==========

ОФОРМЛЕННЯ ЗВІТУ

Звіт по ЛР повинен містить наступні розділи:

1. Титульний лист (номер роботи, тема роботи, ПІБ виконавця, групу)

2. Лист технічного завдання на роботу (ТЗ)

(варіант структура ПКС, математична задача, засоби програмування)

3. Виконання роботи

- Етап 1.Розробка паралельного математичного алгоритму рішення задачі

(наприклад, Ан = Вн + х* Сн СР: х)

- Етап 2. Розробка алгоритму для кожного потоків (Т1 , Т2 … Т4-Т6)
- Етап 3. Розробка структурної схеми взаємодії потоків (Додаток А)
- Етап 4. Лістинг програми (з «шапкою» і коментарями)

4 Висновки

==========

Л І Т Е Р А Т У Р А

1. Жуков І., Корочкін О. Паралельні та розподілені обчислення. Навч. посібник – К.: Корнійчук, 2005. - с. 240.

2. Жуков І., Корочкін О. Паралельні та розподілені обчислення. Навч. посібник. - Друге видання – К.: Корнійчук, 2014. - с. 284.

3. Жуков І, Корочкін О. Паралельні та розподілені обчислення обчисленняя. Лабораторний практикум. Нав. посібник – К.: Корнійчук, 2008. - 224 с.

4. Багатопотоковість в Java [Електронний ресурс] https://uk.myservername.com/multithreading-java-tutorial-with-examples

5. Maurice Herlihy , Nir Shavit .The Art of Multiprocessor Programming, Second Edition,

6. Scott Oaks, Henry Wong Java Threads, 3rd Edition, O'Reilly Media, Inc. 2004.

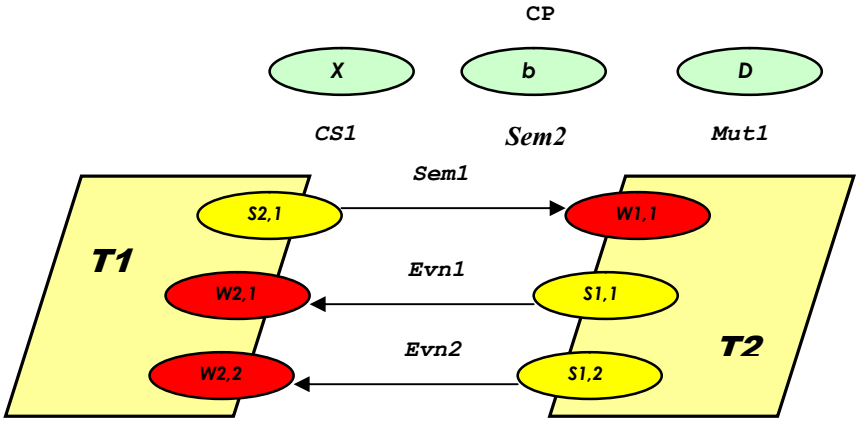
ДОДАТКИ

Додаток А

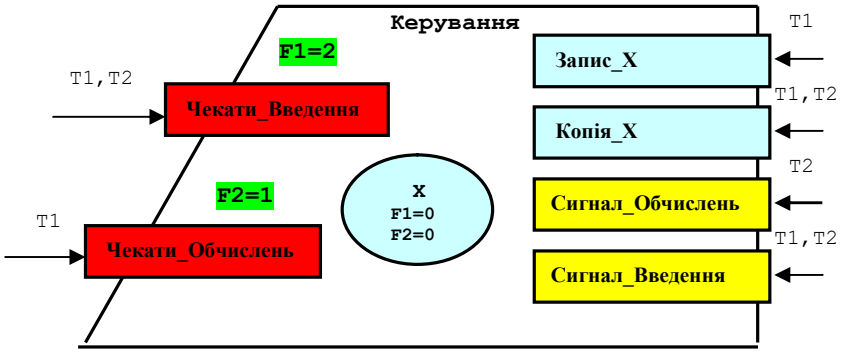
Схеми взаємодії процесів

Для ПКС СП

Семафори, мютекси, події, кр.секції

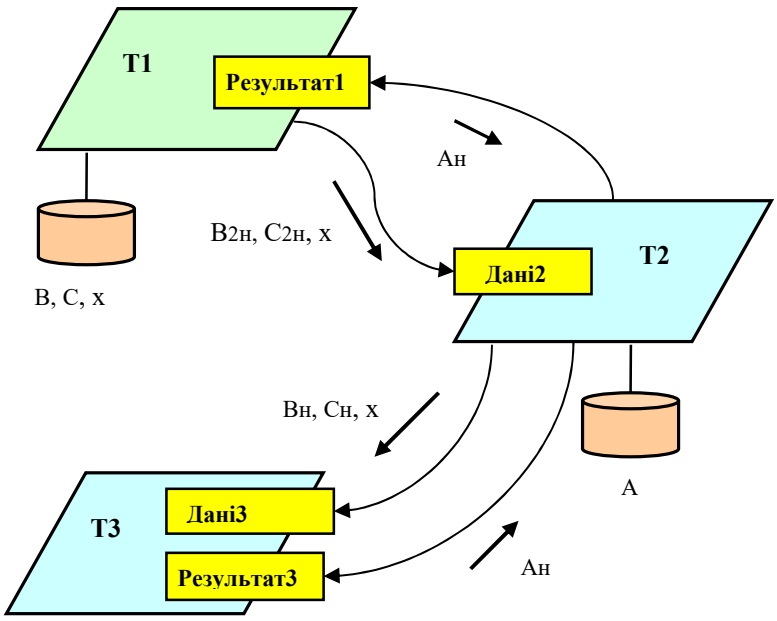


Монітори



Для ПКС з ЛП

Ада. Рандеву



Додаток Б

ПЕРЕЛІК ЛАБОРАТОРНИХ РОБІТ

ПКС СП. ЛР1-ЛР4

31.1. WinAPI Семафори, мютекси, події,
критичні секції, атомік змінні
ЛР1 (Мова за вибором)

---

32. 1. C# Семафори, мютекси, події, критичні секції,
атомік змінні, бар'єри
ЛР2

33.1. Ada Монітори (захищені типи)
33.2. Java Монітори
ЛР3

34.1 OpenMP Критичні секції, бар'єри (Мова за вибором)
ЛР4

==========

ПКС ЛП. ЛР5

35.1. Ada Рандеву
35.2. MPI Повідомлення (Мова за вибором)
ЛР5