9.1 ПРИНЦИПИ ВЗАЄМНОГО БЛОКУВАННЯ

У мультипрограмній системі процес знаходиться в стані тупика, дедлока, або клінчу, якщо він чекає деякої події, яка ніколи не станеться. Системна тупикова ситуація, або «зависання» системи – це ситуація, коли один або більше процесів виявляються в стані тупика. У мультипрограмних обчислювальних машинах однією з головних функцій операційної системи є розподіл ресурсів. Коли ресурси розподіляються між багатьма користувачами, кожному з яких надається право виняткового управління виділеними йому конкретними ресурсами, цілком можливо виникнення тупиків, із-за яких процеси деяких користувачів ніколи не зможуть завершитися.

У цьому розділі обговорюється проблема тупиків і наводяться основні результати наукових досліджень, присвячених питанням запобігання, обходу, виявленню тупиків і питанням відновлення після них. Розглядається також тісно пов'язана проблема нескінченного відкладання, коли процес, що навіть не знаходиться в стані тупика, чекає події, яка може ніколи не статися із-за «необ'єктивних» принципів, закладених в системі планування ресурсів.

Розглядаються можливі компромісні рішення з точки зору накладних витрат на включення засобів боротьби з тупиками і очікуваних від цього переваг. У деяких випадках ціна, яку доводиться платити за те, щоб зробити систему вільною від тупиків, занадто висока. В інших випадках, наприклад, в системах управління процесами реального часу, просто немає іншого вибору, оскільки виникнення тупиків може призвести до катастрофічних наслідків.

У комп'ютерних системах існує велика кількість ресурсів, кожен з яких в конкретний момент часу може використовуватися тільки одним процесом (неподільний ресурс). Поява двох процесів, які одночасно передають дані, наприклад, на принтер, призведе до друку безглуздого набору символів. Наявність двох процесів, що використовують один і той же елемент таблиці файлової системи, стане причиною руйнування файлової системи. Тому усі ОС мають здатність надавати процесу ексклюзивний доступ не до одного, а до декількох ресурсів.

У попередньому розділі були розглянуті способи синхронізації процесів, які дозволяють процесам успішно кооперуватися. Проте, якщо засобами синхронізації користуватися необережно, то можуть виникнути непередбачені утруднення – взаємні блокування, що називаються також клінчами (clinch), дедлоками (deadlocks) або тупиками.

Взаємному блокуванню можна дати таке формальне визначення:

Взаємне блокування в групі процесів виникає в тому випадку, якщо кожен процес з цієї групи чекає події, настання якої залежить виключно від іншого процесу з цієї ж групи.

Розглянемо приклад тупика. Нехай двом потокам (процесам), що виконуються в режимі мультипрограмування, для виконання їх роботи потрібно два ресурси, наприклад, порт і диск. На рис. 9.1, а показані фрагменти відповідних програм. І нехай після того, як потік А зайняв порт (встановив блокуючу змінну), він був перерваний. Управління отримав потік В, який спочатку зайняв диск, але при виконанні наступної команди був заблокований, оскільки порт виявився вже зайнятим потоком А. Управління знову отримав потік А, який відповідно до своєї програми зробив спробу зайняти диск і був заблокований: диск вже розподілений потоку В. У такому положенні потоки А і В можуть знаходитися скільки завгодно довго.

Залежно від співвідношення швидкостей потоків, вони можуть або абсолютно незалежно використати ресурси (г), що розділяються, або утворювати черги до ресурсів (в), що розділяються, або взаємно блокувати один одного (б).

У розглянутих прикладах тупик був утворений двома потоками, але взаємно блокувати один одного можуть і більше число потоків.

Тупикам можна запобігти на стадії написання програм, тобто програми мають бути написані так, щоб тупики не могла виникнути ні при якому співвідношенні взаємних швидкостей потоків. Так, якби в попередньому прикладі потік А і потік В просили ресурси в однаковій послідовності, то тупик був би в принципі неможливий. Другий підхід до запобігання тупиків називається динамічним і полягає у використанні певних правил при призначенні ресурсів процесам, наприклад, ресурси можуть виділятися в певній послідовності, загальній для усіх процесів.

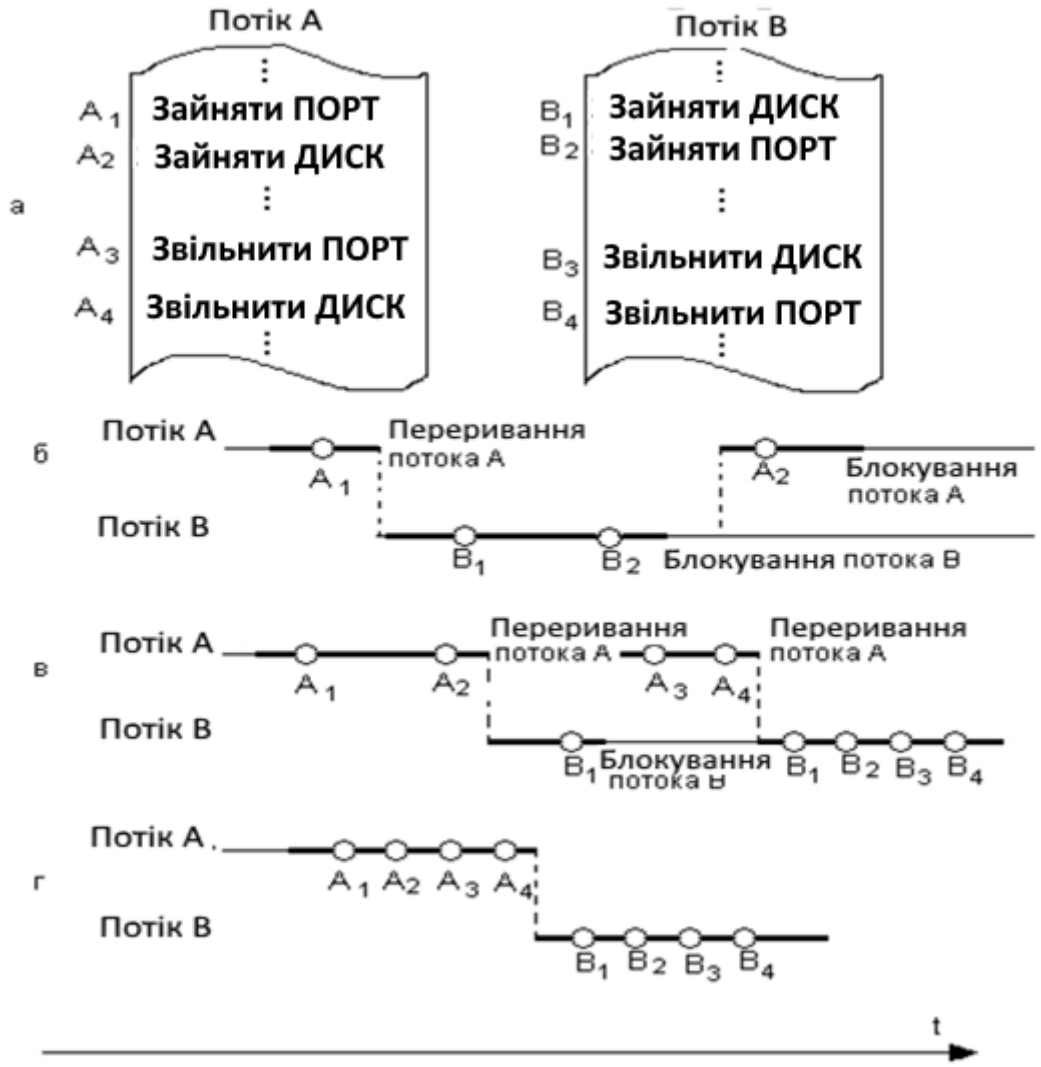

Рисунок 9.1 – Виникнення взаємних блокувань при виконанні програми:
(б) – взаємне блокування (клінч); (в) – черга до диска, що розділяється;
(г) – незалежне використання ресурсів

Існують формальні, програмно-реалізовані методи розпізнавання тупиків, засновані на веденні таблиць розподілу ресурсів і таблиць запитів до зайнятих ресурсів. Аналіз цих таблиць дозволяє виявити взаємні блокування.

Взаємні блокування можуть статися в більшості інших ситуацій окрім запитів виділених пристроїв введення-виведення. У системах баз даних програма може виявитися вимушеною заблокувати декілька записів, щоб уникнути стану конкуренції. Якщо процес А блокує запис R1, процес В блокує запис R2, а потім кожен процес намагається заблокувати чужий запис, ми також опинимося в тупику. Таким чином, взаємні блокування з'являються при роботі як з апаратними, так і з програмними ресурсами.

Як ми вже знаємо, ресурси бувають двох видів: вивантажувані і невивантажувані.

До вивантажуваних належать такі ресурси, які можуть бути безболісно відібрані в процесу, який їх має. Прикладом такого ресурсу може послужити пам'ять. Розглянемо систему, що має 256 Мб призначеної для користувача пам'яті, один принтер і два процеси по 256 Мб, кожен з яких хоче щось вивести на друк. Процес А просить і отримує принтер, а потім починає обчислювати значення, призначене для виведення на друк. Але до завершення обчислення закінчується виділений йому квант часу, і він вивантажується на диск.

Тепер запускається процес В, який безуспішно намагається оволодіти принтером. Потенційно виникає ситуація взаємного блокування, оскільки у процесу А є принтер, а у процесу В – пам'ять, і жоден з них не може продовжити свою роботу без ресурсу, що утримується іншим процесом.

На щастя, є можливість відібрати пам'ять у процесу В, вивантаживши цей процес на диск, і завантажити звідти процес А. Тепер А може відновити свою роботу, виконати друк і вивільнити принтер. І ніякого взаємного блокування не виникне.

А ось невивантажуваний ресурс не можна відібрати у його поточного власника, не викликавши збою в обчисленнях. Якщо у процесу, який вже приступив до запису на компакт-диск, несподівано відібрати пишучий привід і віддати його іншому процесу, то це призведе до псування компакт-диска. Тобто, приводи компакт-дисків не можна відібрати в довільний момент. Як правило, у взаємних блокуваннях фігурують невивантажувані ресурси. Взаємні блокування за участю вивантажуваних ресурсів можуть бути усунені шляхом перерозподілу ресурсів від одного процесу до іншого. Тому наша увага буде сконцентрована на невивантажуваних ресурсах.

Проблема тупиків включає такі задачі:

- запобігання тупиків;
- розпізнавання тупиків;
- відновлення системи після тупиків.

У деяких випадках, коли тупикова ситуація утворена багатьма процесами, що використовують багато ресурсів, розпізнавання тупиків є нетривіальною задачею.

Якщо ж тупикова ситуація виникла, то не обов'язково знімати з виконання усі заблоковані процеси. Можна зняти тільки частину з них, при цьому звільняються ресурси, очікувані іншими процесами. Можна повернути деякі процеси в область свопінгу, можна вчинити «відкат» деяких процесів до так званої контрольної точки, в якій запам'ятовується уся інформація, необхідна для відновлення виконання програми з цього місця.

9.1.1 Умови виникнення тупиків

Для того щоб взаємне блокування стала можливим, повинні виконуватися чотири необхідні умови (Коффман, Елфік і Шошані, 1971 р.) [12].

1. **Умова взаємного виключення (Mutual exclusion).** Кожен ресурс або виділений в даний момент тільки одному процесу, або доступний.
2. **Умова утримання і очікування ресурсів.** Процес може утримувати виділені ресурси під час запиту (очікування) інших ресурсів.
3. **Умова невивантажуваності (відсутності перерозподілу).** У процесу не можна примусово забрати раніше отримані ресурси. Процес, що володіє ними, повинен сам звільнити ресурси.
4. **Умова циклічного очікування.** Існує замкнутий ланцюг процесів, кожен з яких утримує як мінімум один ресурс, необхідний процесу, наступному в ланцюзі після даного (рис. 9.2).

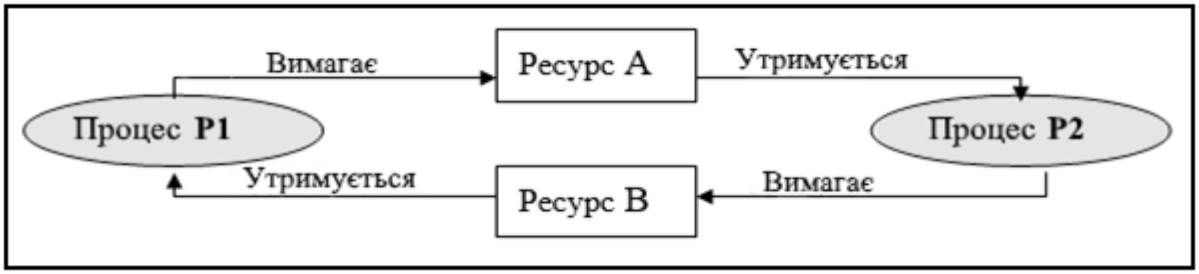

Рисунок 9.2 – Циклічне очікування ресурсів

Для того щоб сталася взаємне блокування, повинні виконуватися усі ці чотири умови. Якщо хоч би одна з них відсутня, тупикова ситуація неможлива.

9.1.2 Моделювання взаємного блокування

Холт (Holt, 1972 р.) показав, як ці чотири умови можуть бути змодельовані з використанням спрямованих графів. У графів є два види вузлів: процеси, показані колами, і ресурси, показані квадратами. Спрямоване ребро (дуга), яке слідує від вузла ресурсу (квадрата) до вузла процесу (кола), означає, що цей ресурс був раніше запрошений, отриманий і на даний момент утримується цим процесом [10].

Розглянемо, наприклад, два процеси – P1 і P2, і два ресурси – R1 і R2. Припустимо, що кожному процесу для виконання частини своїх функцій потрібен доступ до загальних ресурсів. Тоді можливе виникнення такої ситуації: ОС виділяє ресурс R1 процесу P2, а ресурс R2 – процесу P1 (рис. 9.3, а). В результаті кожен процес чекає отримання одного з двох ресурсів. При цьому жоден з них не звільняє вже наявний ресурс, чекаючи отримання другого ресурсу для виконання функцій, що вимагають наявності двох ресурсів. У результаті процеси виявляються взаємно заблокованими як показано на рис. 9.3, б, або в.

У зв'язку з проблемою тупиків було виконано багато цікавих досліджень в області інформатики і операційних систем.

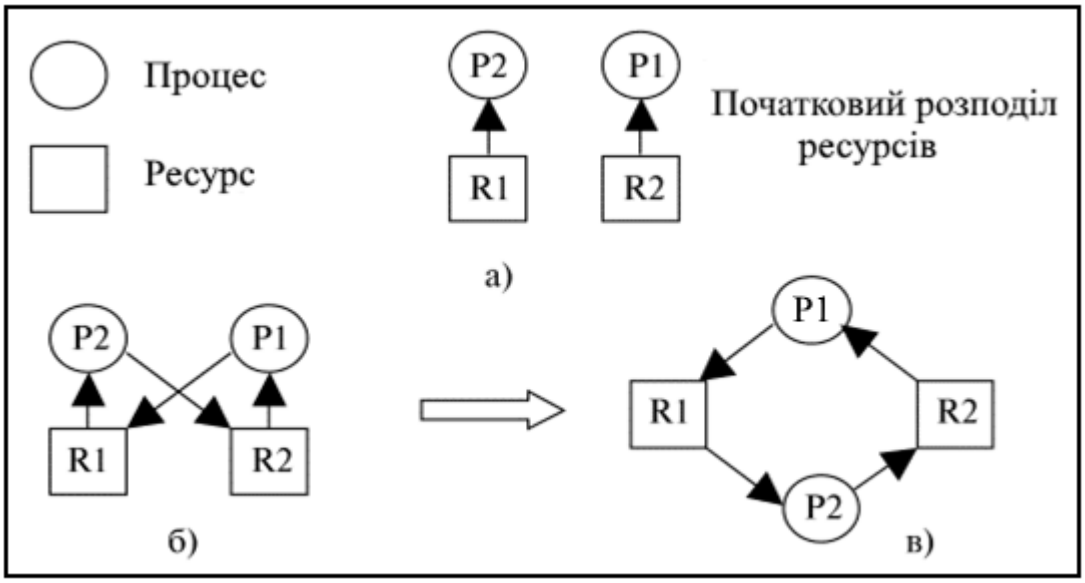

Рисунок 9.3 – Графи розподілу ресурсів

Основні напрямки боротьби з тупиками:

1. Ігнорувати цю проблему.

2. Виявлення і відновлення. Дайте взаємним блокуванням проявити себе, виявіть їх і зробіть необхідні дії.

3. Динамічне ухилення від них за рахунок ретельного розподілу ресурсів.

4. Запобігання за рахунок структурного придушення однієї з чотирьох необхідних умов для їх виникнення.

Стратегії запобігання взаємних блокувань дуже консервативні, оскільки вирішують проблему взаємних блокувань шляхом обмеження доступу процесів до ресурсів і накладення обмежень на процеси. Їх протилежність – стратегії виявлення і усунення взаємних блокувань, які не обмежують доступу процесів до ресурсів і не накладають ніяких обмежень на дії процесів.

Сучасні ОС періодично виконують алгоритми, за допомогою яких виявляється наявність взаємних блокувань у процесах. Перевірка наявності взаємних блокувань може виконуватися як при кожному запиті ресурсу, так і рідше, залежно від того, наскільки ймовірне виникнення взаємних блокувань. З одного боку, перевірка при кожному запиті має дві основні переваги: раніше виявлення і спрощення алгоритму. З іншого боку, така часта перевірка призводить до помітного споживання часу процесора. Узагальнені алгоритми виявлення і усунення взаємних блокувань можна знайти в літературі [9; 11].

9.1.3 Ігнорування проблеми взаємних блокувань

Простий підхід – не помічати проблему тупиків (алгоритм страуса). Для того щоб прийняти таке рішення, необхідно оцінити ймовірність виникнення взаємних блокувань і порівняти її з імовірністю збитку від інших відмов апаратного і програмного забезпечення. Проектувальники зазвичай не бажають жертвувати продуктивністю системи або зручністю користувачів для впровадження складних і дорогих засобів боротьби з тупиками.

Будь-яка ОС, що має в ядрі ряд масивів фіксованої розмірності, потенційно страждає від тупиків, навіть якщо вони не виявлені. Таблиця відкритих файлів, таблиця процесів, фактично кожна таблиця є обмеженим ресурсом. Заповнення усіх записів таблиці процесів може призвести до того, що черговий запит на створення процесу може бути відхилений. При несприятливому збігу обставин декілька процесів можуть видати такий запит одночасно і опинитися в тупику. Чи слід відмовлятися від виклику CreateProcess, щоб розв'язати цю проблему?

Підхід більшості популярних ОС (Unix, Windows та інших) полягає в тому, щоб ігнорувати цю проблему в припущенні, що малоймовірний випадковий тупик прийнятніший, ніж безглузді правила, що примушують користувачів обмежувати число процесів, відкритих файлів тощо.