ЛЕКЦІЯ 6. МІЖПРОЦЕСОВА ВЗАЄМОДІЯ

План

6.1. Види міжпроцесової взаємодії
6.2. Базові механізми міжпроцесової взаємодії
6.3. Технології передавання повідомлень

Вступ

На попередній лекції ми розглядали взаємодію *потоків* одного процесу. Головною особливістю цієї взаємодії є простота технічної реалізації *обміну даними між ними* — усі потоки одного процесу використовують *один адресний простір*, а отже, можуть вільно отримувати доступ до спільно використовуваних даних, ніби вони є їх власними. Оскільки технічних труднощів із реалізацією обміну даними тут немає, основною проблемою, яку потрібно вирішувати в цьому випадку, є *синхронізація потоків*.

Для розв'язання задач синхронізації потоків одного процесу можна використовувати різні *синхронізаційні примітиви*. До найпростіших примітивів належать *м'ютекси, семафори, умовні змінні, блокування читання-записування і бар'єри.*

Механізмом більш високого рівня є *концепція монітора*, що поєднує м'ютекси та умовні змінні, а також задає деякі правила їхньої взаємодії для захисту спільно використовуваних даних.

З іншого боку, кожен потік виконується в рамках адресного простору деякого процесу, тому часто постає задача організації взаємодії між потоками різних процесів (рис. 6.1).

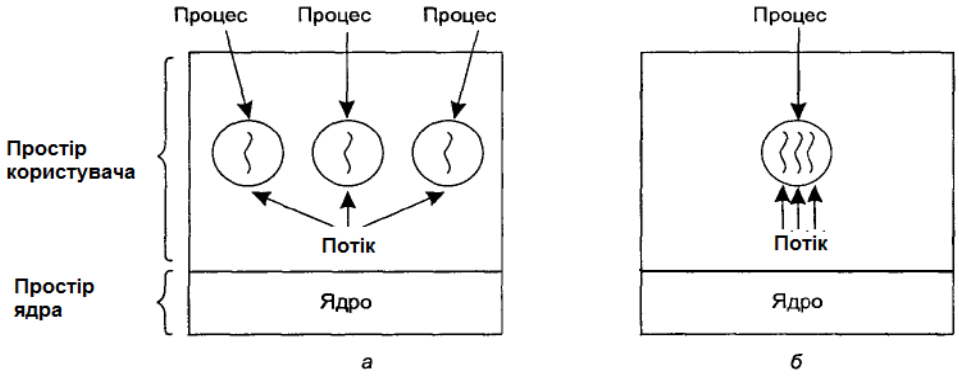

Рис. 6.1. а) три процеси з одиночними потоками б) один процес з трьома потоками.

Для потоків різних процесів питання забезпечення синхронізації теж є актуальними, але вони в більшості випадків не грунтуються на понятті спільно використовуваних даних (такі дані за замовчуванням для процесів відсутні). Крім того, додається досить складна задача забезпечення обміну даними між захищеними адресними просторами. Підходи до її розв'язання визначають різні види *міжпроцесової взаємодії*.

6.1. Види міжпроцесової взаємодії

Реалізація міжпроцесової взаємодії здійснюється трьома основними методами: *передавання повідомлень, розподілюваної пам'яті* та *відображуваної пам'яті.*

6.1.1. Методи розподілюваної пам'яті

Методи розподілюваної пам'яті (shared memory) дають змогу двом процесам обмінюватися даними через загальний буфер пам'яті. Перед обміном даними кожний із процесів має приєднати цей буфер до свого адресного простору з використанням спеціальних системних викликів (перед цим перевіряють права). Буфер доступний у системі за допомогою процедури іменування, термін його існування звичайно обмежений часом роботи всієї системи. Дані в ньому фактично є спільно використовуваними, як і для потоків. Жодних засобів синхронізації доступу до цих даних розподілювана пам'ять не забезпечує, програміст, так само, як і при розробці багатопотокових застосунків, має організовувати її сам.

6.1.2. Методи передавання повідомлень

В основі методів передавання повідомлень (message passing) лежать різні технології, що дають змогу потокам різних процесів (які, можливо, виконуються на різних комп'ютерах) обмінюватися інформацією у вигляді фрагментів даних фіксованої чи змінної довжини, котрі називають *повідомленнями* (messages). Процеси можуть приймати і відсилати повідомлення, при цьому автоматично забезпечується їхнє пересилання між адресними просторами процесів одного комп'ютера або через мережу. Важливою особливістю технологій передавання повідомлень є те, що вони не спираються на спільно

використовувані дані — процеси можуть обмінюватися повідомленнями, навіть не знаючи один про одного.

6.1.3. Технологія відображуваної пам'яті

Ще однією категорією засобів міжпроцесової взаємодії є *відображувана пам'ять* (mapped memory). У ряді ОС відображувана пам'ять є базовим системним механізмом, на якому грунтуються інші види міжпроцесової взаємодії та системні рішення. Звичайно відображувану пам'ять використовують у поєднанні з інтерфейсом файлової системи, в такому разі говорять про *файли, відображувані у пам'ять* (memory-mapped files).

Ця технологія зводиться до того, що за допомогою спеціального *системного виклику* певну частину адресного простору процесу однозначно пов'язують із вмістом файла. Після цього будь-яка операція записування в таку пам'ять спричиняє зміну вмісту відображеного файла, яка відразу стає доступною усім застосункам, що мають доступ до цього файла. Інші застосунки теж можуть відобразити той самий файл у свій адресний простір і обмінюватися через нього даними один з одним.

6.1.4. Особливості міжпроцесової взаємодії

Тепер можна порівняти характеристики міжпроцесової взаємодії із характеристиками взаємодії *потоків одного процесу*.

- Проблема організації обміну даними є актуальною тільки для міжпроцесової взаємодії, оскільки потоки обмінюються даними через загальний адресний простір. Обмін даними між потоками схожий на використання розподілюваної пам'яті, але не потребує підготовчих дій.

- Проблема синхронізації доступу до спільно використовуваних даних є актуальною для взаємодії потоків і для міжпроцесової взаємодії із використанням розподілюваної пам'яті. Використання механізму передавання повідомлень не грунтується на спільно використовуваних даних.

6.2. Базові механізми міжпроцесової взаємодії

Розглянемо особливості організації взаємодії між потоками *різних процесів.* Основною характеристикою такої взаємодії є те, що у процесів немає спільного адресного простору, тому тут не можна безпосередньо працювати зі спільно використовуваними даними, як це було можливо для потоків. Тут ітиметься переважно про *процеси,* під якими розуміють потоки різних процесів.

6.2.1. Міжпроцесова взаємодія на базі спільної пам'яті

Для вирішення проблеми міжпроцесової синхронізації необхідно:

- по-перше, організувати спільну пам'ять між процесами (це може бути розподілювана пам'ять або файл, відображений у пам'ять);

- по-друге, розмістити в цій пам'яті стандартні синхронізаційні об'єкти (семафори, м'ютекси, умовні змінні);

- по-третє, використовуючи ці об'єкти, працювати зі спільно використовуваними даними, як це робилося у разі використання потоків.

Такий підхід широко застосовують на практиці. На жаль, досить складно запропонувати спосіб його реалізації для міжпроцесової синхронізації у більшості систем, оскільки різні системи пропонують різний набір засобів організації спільної пам'яті та засобів сигналізації, які можуть працювати в такій пам'яті.

За допомогою відображуваних файлів можна легко реалізувати розподіл пам'яті між процесами, якщо відобразити один і той самий файл на адресний простір кількох із них.

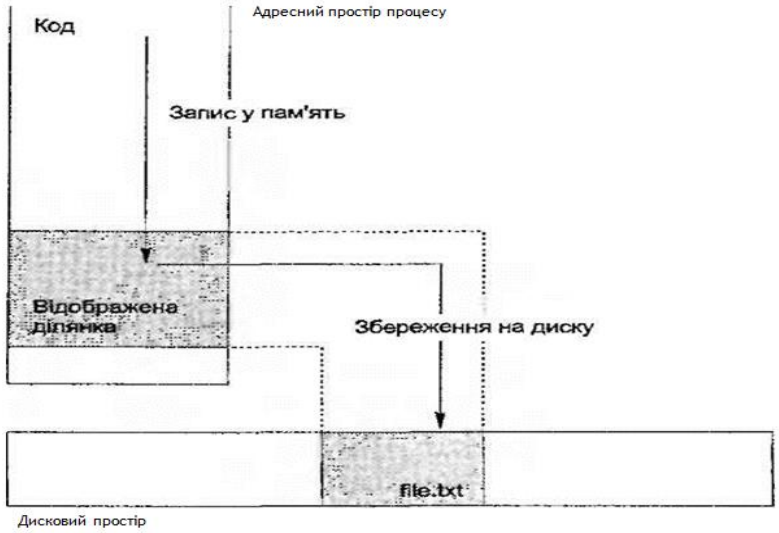

Рис. 11.2. Файл, що відображається у пам'ять

Рис. 6.2. Файл, що відображується у пам'ять процесу

Приклади використання відображуваної пам'яті на рівні операційної системи.

Відображення файла у пам'ять можна використати для завантаження виконуваних файлів у пам'ять (такі файли відображаються в адресний простір процесу і є простором підтримки для сторінок коду). Цей підхід використовується у Windows XP і в Linux.

На основі відображуваної пам'яті можна також реалізувати кешування диска.

За допомогою цієї технології можна виділяти пам'ять процесу. Для цього в його адресний простір відображають спеціальний файл, у разі доступу до якого завжди повертають нулі.

Можна забезпечити автоматичне збереження значень складних структур даних між викликами програми (такі структури створюють у ділянці пам'яті, що відповідає відображеному файлу; під час наступних викликів цей файл відображають у ту саму ділянку пам'яті знову, і всі структури поновлюються в цій пам'яті повністю).

6.2.2. Основи передавання повідомлень

Усі методи взаємодії, які було розглянуто дотепер, грунтуються на читанні й записуванні спільно використовуваних даних. На практиці така взаємодія не завжди можлива (наприклад, робота зі спільно використовуваними даними проблематична, якщо для процесів немає спільної фізичної пам'яті, а є тільки мережний зв'язок між комп'ютерами, на яких вони виконуються). У таких випадках можна використати засоби взаємодії, які не ґрунтуються на спільно використовуваних даних, передусім *засоби передавання повідомлень*.

Як було вже згадано, засоби передавання повідомлень ґрунтуються на обміні *повідомленнями* — фрагментами даних змінної довжини. Основою такого обміну є не спільна пам'ять, а *канал зв'язку* (communication channel). Він забезпечує взаємодію між процесами (для того, щоб спілкуватися, вони повинні створити канал зв'язку) і є абстрактним відображенням мережі зв'язку. Абстрактність каналу дає змогу реалізувати його не тільки на основі мережної взаємодії, але й спільної пам'яті (коли процеси перебувають на одному комп'ютері). При цьому такі зміни в реалізації будуть приховані від процесів, що взаємодіють.

Виокремлюють такі характеристики каналів зв'язку: спосіб задання; кількість процесів, які можуть бути з'єднані одним каналом; кількість каналів, які *можуть* бути створені між двома процесами; пропускна здатність каналу (кількість повідомлень, які можуть одночасно перебувати в системі й бути асоційованими з цим каналом); максимальний розмір повідомлення; спрямованість зв'язку через канал (двобічний або однобічний зв'язок).

Примітиви передавання повідомлень

Основна особливість передавання повідомлень полягає в тому, що процеси спільно використовують тільки канали. Немає необхідності забезпечувати взаємне виключення процесів під час доступу до спільно використовуваних даних, замість цього досить визначити *примітиви передавання повідомлень* — спеціальні операції обміну даними через канал, які забезпечують не лише обмін даними, але й синхронізацію.

Є два примітиви передавання повідомлень: **send** (для відсилання повідомлення каналом) і **receive** (для отримання повідомлення з каналу). Розглянемо, як особливості реалізації send і receive дають змогу виділити різні класи методів передавання повідомлень.

Зазначені примітиви передавання повідомлень можуть задавати *прямий* і *непрямий* обмін даними. При прямому обміні даними необхідно явно вказувати процес, з яким необхідно обмінюватись інформацією. Непрямий обмін здійснюють через спеціальний об'єкт (поштову скриньку, порт); процеси можуть поміщати повідомлення в поштову скриньку і отримувати їх звідти.

Синхронне й асинхронне передавання повідомлень

Зупинимося на основних питаннях синхронізації під час передавання повідомлень, можна виокремити різні групи методів передавання повідомлень залежно від того як вони дають можливість відповісти на два запитання. Чи може потік бути призупинений під час виконання операції send, якщо повідомлення не було отримане?

Чи може потік бути призупинений під час виконання операції receive, якщо повідомлення не було відіслане?

У реальних системах відповідь на друге запитання практично завжди буде позитивною — неблокувальне приймання повідомлень призводить до того, що вони губляться. Варіанти відповідей на перше запитання визначають два класи передавання повідомлень — *синхронне* і *асинхронне.*

Під час *синхронного* передавання повідомлень операція send призупиняє процес до отримання повідомлення, а під час *асинхронного* передавання повідомлень - не призупиняє процес (тобто є *неблокувальною);* після відсилання повідомлення процес продовжує своє виконання, не чекаючи отримання результату.

Реалізація синхронного й асинхронного передавання повідомлень залежить від низки характеристик каналу й обміну повідомленнями, насамперед від пропускної здатності каналу.

Якщо пропускна здатність дорівнює нулю (повідомлення не можуть очікувати в системі), відправник завжди має очікувати, поки одержувачу не надійде повідомлення, а одержувач має очікувати, поки повідомлення йому не буде відіслано. Два процеси мають явно домовлятися про майбутній обмін.

Якщо пропускна здатність обмежена (у системі можуть перебувати максимум *n* повідомлень для цього каналу), відправник має очікувати тільки тоді, коли черга повідомлень для цього каналу переповнена (у ній перебуває рівно *n* повідомлень), одержувач має очікувати, якщо ця черга порожня.

Якщо пропускна здатність необмежена, очікування можливе тільки для одержувача за порожньої черги.

Під час обміну повідомленнями необхідне підтвердження їх отримання. Деякі методи обміну повідомленнями не вимагають підтвердження зовсім, в інших випадках можлива ситуація, коли відправника після виконання операції send блокують доти, поки одержувач не надішле йому інше повідомлення із підтвердженням отримання; таку технологію називають *обміном повідомленнями із підтвердженням отримання.*

6.2.3. Технології передавання повідомлень

Розглянемо методи передавання повідомлень, які застосовують на практиці.

Канали

Канал — це найпростіший засіб передавання повідомлень. Він є циклічним буфером, записування у який виконують за допомогою одного процесу, а читання — за допомогою іншого. У конкретний момент часу до каналу має доступ тільки один процес. Операційна система забезпечує синхронізацію згідно правилу: якщо процес намагається записувати в канал, у якому немає місця, або намагається зчитати більше даних, ніж поміщено в канал, він переходить у стан очікування.

Розрізняють безіменні та поіменовані канали.

До *безіменних каналів* немає доступу за допомогою засобів іменування, тому процес не може відкрити вже наявний безіменний канал без його дескриптора. Це означає, що такий процес має отримати дескриптор каналу від процесу, що його створив, а це можливо тільки для зв'язаних процесів.

До *поіменованих каналів* (named pipes) є доступ за іменем. Такому каналу може відповідати, наприклад, файл у файловій системі, при цьому будь-який процес, який має доступ до цього файла, може обмінюватися даними через відповідний канал. Поіменовані канали реалізують непрямий обмін даними.

Обмін даними через канал може бути однобічним і двобічним.

Черги повідомлень

Іншою технологією асинхронного непрямого обміну даними є застосування *черг повідомлень* (message queues). Для таких черг виділяють спеціальне місце в системній ділянці пам'яті ОС, доступне для застосувань користувача. Процеси можуть створювати нові черги, відсилати повідомлення в конкретну чергу й отримувати їх звідти. Із чергою одночасно може працювати кілька процесів. Повідомлення — це структури даних змінної довжини. Для того щоб процеси могли розрізняти

адресовані їм повідомлення, кожному з них присвоюють тип. Відіслане повідомлення залишається в черзі доти, поки не буде зчитане. Синхронізація під час роботи з чергами схожа на синхронізацію для каналів.

Сокети

Найрозповсюдженішим методом обміну повідомленнями є використання *сокетів* (sockets). Ця технологія насамперед призначена для організації мережного обміну даними, але може бути використана й для взаємодії між процесами на одному комп'ютері.

Сокет — це абстрактна кінцева точка з'єднання, через яку процес може відсилати або отримувати повідомлення. Обмін даними між двома процесами здійснюють через пару сокетів, по одному на кожен процес. Абстрактність сокету полягає в тому, що він приховує особливості реалізації передавання повідомлень — після того як сокет створений, робота з ним не залежить від технології передавання даних, тому один і той самий код можна без великих змін використовувати для роботи із різними протоколами зв'язку.

Особливості протоколу передавання даних і формування адреси сокету визначає *комунікаційний домен;* його потрібно зазначати під час створення кожного сокету. Прикладами доменів можуть бути *домен Інтернету* (який задає протокол зв'язку на базі TCP/IP) і *локальний домен* або *домен UNIX,* що реалізує зв'язок із використанням імені файла (подібно до поіменованого каналу). Сокет можна використовувати у поєднанні тільки з одним комунікаційним доменом. *Адреса сокету* залежить від домену (наприклад, для сокетів домену UNIX такою адресою буде ім'я файла).

Способи передавання даних через сокет визначаються його *типом.* У конкретному домені можуть підтримуватися або не підтримуватися різні типи сокетів

Наприклад, і для домену Інтернет, і для домену UNIX підтримуються сокети таких типів:

- *потокові* (stream sockets) — задають надійний двобічний обмін даними суцільним потоком без виділення меж (операція читання даних повертає стільки даних, скільки запитано або скільки було на цей момент передано);
- *дейтаграмні* (datagram sockets) — задають ненадійний двобічний обмін повідомленнями із виділенням меж (операція читання даних повертає розмір того повідомлення, яке було відіслано).

Під час обміну даними із використанням сокетів зазвичай застосовується технологія клієнт-сервер, коли один процес (сервер) очікує з'єднання, а інший (клієнт) з'єднують із ним.

Перед тим як почати працювати з сокетами, будь-який процес (і клієнт, і сервер) має створити сокет за допомогою системного виклику socket ( ). Параметрами цього виклику задають комунікаційний домен і тип сокету. Цей виклик повертає *дескриптор сокету* — унікальне значення, за яким можна буде звертатися до цього сокету.

Подальші дії відрізняються для сервера і клієнта. Спочатку розглянемо послідовність кроків, яку потрібно виконати для сервера.

1. Сокет пов'язують з адресою за допомогою системного виклику bind( )). Для сокетів домену UNIX як адресу задають ім'я файла, для сокетів домену Інтернету — необхідні характеристики мережного з'єднання. Далі клієнт для встановлення з'єднання й обміну повідомленнями має буде вказати цю адресу.

2. Сервер дає змогу клієнтам встановлювати з'єднання, виконавши системний виклик listen( ) для дескриптора сокету, створеного раніше.

3. Після виходу із системного виклику listen( ) сервер готовий приймати від клієнтів запити на з'єднання. Ці запити вишиковуються в чергу. Для отримання запиту із цієї черги і створення з'єднання використовують системний виклик accept( ). Внаслідок його виконання в застосування повертають новий сокет для обміну даними із клієнтом. Старий сокет можна використовувати далі для приймання нових запитів на з'єднання. Якщо під час виклику accept( ) запити на з'єднання в черзі відсутні, сервер переходить у стан очікування.

Для клієнта послідовність дій після створення сокету зовсім інша. Замість трьох кроків досить виконати один - встановити з'єднання із використанням системного виклику connect( ). Параметрами цього виклику задають дескриптор створеного раніше сокету, а також адресу, подібну до вказаної на сервері для виклику bind( ).

Після встановлення з'єднання (і на клієнті, і на сервері) з'явиться можливість передавати і приймати дані з використанням цього з'єднання. Для передавання даних застосовують системний виклик send(), а для приймання — recv().

Зазначену послідовність кроків використовують для встановлення надійного з'єднання.

Віддалений виклик процедур

Технологія *віддаленого виклику процедур* (Remote Procedure Call, RPC) є прикладом синхронного обміну повідомленнями із підтвердженням отримання. Розглянемо послідовність кроків, необхідних для обміну даними в цьому разі.

1. Операцію send оформляють як виклик процедури із параметрами.

2. Після виклику такої процедури відправник переходить у стан очікування, а дані (ім'я процедури і параметри) доставляються одержувачеві. Одержувач може перебувати на тому самому комп'ютері, чи на віддаленій машині; технологія RPC приховує це. Класичний віддалений виклик процедур передбачає, що процес-одержувач створено внаслідок запиту.

3. Одержувач виконує операцію receive і на підставі даних, що надійшли, виконує відповідні дії (викликає локальну процедуру за іменем, передає їй параметри і обчислює результат).

4. Обчислений результат повертають відправникові як окреме повідомлення.

5. Після отримання цього повідомлення відправник продовжує своє виконання, розглядаючи обчислений результат як наслідок виклику процедури.

Висновки

1. Потоки різних процесів, що взаємодіють, мають використовувати засоби міжпроцесової взаємодії, завданнями якої є забезпечення обміну даними між захищеними адресними просторами, а також їхня синхронізація. До основних видів міжпроцесової взаємодії належать передавання повідомлень, розподілювана і відображувана пам'ять.

2. Основним видом міжпроцесової взаємодії у розподілених системах є передавання повідомлень. Головною особливістю передавання повідомлень є те, що ця технологія не вимагає наявності спільно використовуваних даних. Процеси обмінюються повідомленнями змінної довжини за допомогою примітивів send і receive. Ця технологія може бути застосована для організації взаємодії між процесами, виконуваними на віддалених комп'ютерах.

Контрольні запитання та завдання

1. Припустімо, що до комунікаційного каналу, наданого ОС, можуть «підключатися» два процеси. Які синхронізаційні примітиви на рівні ядра ОС можуть бути використані для обміну даними цим каналом відповідно до технології програмних каналів (pipes)? Як приклад ОС візьміть систему Linux.

2. Перелічіть можливі відмінності реалізації черги повідомлень і програмного каналу на рівні ядра ОС.

3. Система обміну повідомленнями надає примітиви send і receive. Примітив receive призупиняє процес, якщо немає повідомлень, призначених для нього. Чи можливе взаємне блокування процесів, якщо не враховувати інших повідомлень, а спільних даних у процесів немає?

4. Перелічіть спільні риси і відмінності поіменованих каналів і сокетів.

5. Чи потрібно для потоків одного процесу організовувати обмін даними методом розподілюваної пам'яті?

6. Яка різниця між безіменними та поіменованими каналами? Які канали реалізують прямий обмін даними?

7. У чому полягає основна відміна між передаванням повідомлень і розподілюваної пам'яті при організації взаємодії процесів?

8. Перелічіть спільні риси і відмінності поіменованих каналів і сокетів.

9. Які є типи сокетів для домену Інтернет? Різниця між дейтаграмними і потоковими сонетами.

10. Прямий і непрямий обмін даними при передачі повідомлень та його характеристика.

Література до лекції 6.

1. Шеховцов В. А. Операційні системи. Підручник для ВНЗ. – К.: ВНV, 2008. –576 с.
2. Таненбаум Э. Операционные системы– СПб: Питер. – 2002 г. – 1040 с.