5.8 ВЗАЄМОДІЯ ПРОЦЕСІВ

У багатозадачному і мережевому середовищі процеси повинні якось спілкуватися один з одним. У більшості ОС передбачені механізми взаємодії процесів, які зобов'язані координувати (синхронізувати) свої зусилля для досягнення загального результату. Ситуації, коли процесам доводиться взаємодіяти:

1. Передача інформації від одного процесу іншому.
2. Контроль над діяльністю процесів. Наприклад, коли вони борються за один ресурс.
3. Узгодження дій процесів. Наприклад, коли один процес поставляє дані, а інший їх виводить на друк. Якщо узгодженості не буде, то другий процес може почати друк раніше, ніж поступлять дані.

Другий і третій випадок стосуються і потоків, які ми розглядатимемо пізніше. У першому випадку в потоків немає ніяких труднощів, оскільки вони використовують загальний адресний простір.

Передача може здійснюватися декількома способами: пам'ять, що розділяється, сигнали, повідомлення, виклик віддаленої процедури, сокети.

5.8.1 Пам'ять, що розділяється

Традиційно вважається, що основним способом міжпроцесного обміну є ресурс, що розділяється, такий як пам'ять, яка грунтується на відповідних об'єктах ядра. При цьому виникають задачі створення, іменування і захисту таких ресурсів. Один з процесів створює ресурс, що розділяється, наділяє його атрибутами захисту і ім'ям, за яким цей ресурс може бути доступний іншим процесам (навіть у разі завершення роботи процесу-творця). Як приклад розглянемо спілкування через пам'ять, що розділяється (рис. 5.9).

Наприклад, в ОС Windows сегмент пам'яті, що розділяється, створюється за допомогою Win32-функції CreateFileMapping. Це фрагмент пам'яті, доступний за іменем (параметр lpname), який базується на відповідному об'єкті ядра. Процесу-творцеві повертається описувач (handle) ресурсу. Інші процеси, що бажають мати доступ до ресурсу, також повинні отримати його описувач. В даному випадку це можна

зробити за допомогою функції OpenFileMapping, вказавши ім'я ресурсу в якості одного з параметрів [28].

Рисунок 5.9 – Адресні простори процесів, взаємодіючих через сегмент пам'яті

5.8.2 Сигнали

Сигналами називають програмні переривання, що повідомляють процес про настання певної події. На відміну від інших механізмів взаємодії процесів, сигнали не дозволяють процесам обмінюватися один з одним якою-небудь інформацією. Системні сигнали залежать від операційної системи і типів програмних переривань, підтримуваних певним процесором. При надходженні сигналу ОС спочатку визначає, кому призначений цей сигнал, а потім – як процес повинен на нього відреагувати. Процеси можуть перехоплювати сигнали, ігнорувати їх або маскувати.

Процес перехоплює сигнал і визначає процедуру, що викликається ОС у разі надходження сигналу.

Процеси можуть проігнорувати сигнал, тобто перекласти відповідальність за виконання дії по обробці сигналу за умовчанням на операційну систему.

Найчастіше за умовчанням задається аварійне завершення (abort) процесу. Іншим варіантом дії за умовчанням є ігнорування сигналу.

Процеси можуть заблокувати обробку сигналу шляхом його маскування. Коли процес маскує сигнал певного типу (наприклад, сигнал призупинення), операційна система блокує сигнали цього типу, поки маскування сигналу не буде відключено.

5.8.3 Передача повідомлень

З розвитком розподілених систем зріс інтерес до взаємодії процесів за допомогою повідомлень. Повідомлення можуть передаватися в одному напрямі, тоді для будь-якого повідомлення один процес є відправником, а інший – одержувачем. Передача повідомлень може також бути двонаправленою, тобто кожен процес під час взаємодії одночасно може бути відправником і одержувачем.

Прийом і відправка повідомлень реалізується у вигляді виклику системних функцій, доступних у більшості мов програмування. У разі блокуючої передачі процес змушений чекати до тих пір, поки повідомлення не буде доставлено одержувачеві, вимагаючи підтвердження прийому.

При неблокуючій передачі процес-відправник може продовжувати виконання інших операцій, навіть якщо повідомлення ще не було доставлене одержувачеві (або він не повідомив про це відправника). Щоб реалізувати неблокуючу передачу, необхідно використати механізм буферизації повідомлень для зберігання повідомлень до моменту їх доставки одержувачеві.

Блокуюча передача є прикладом синхронного зв'язку, тоді як неблокуюча передача – асинхронного. Під час відправки повідомлення можна вказати процес-одержувач або опустити ім'я процесу, у такому разі буде зроблена широкомовна передача повідомлень усім процесам системи (поштові скриньки в Windows).

Асинхронний зв'язок у поєднанні з неблокуючою передачею сприяє збільшенню швидкодії системи за рахунок зменшення часу очікування різних подій процесами. Наприклад, якщо процес відправить інформацію на зайнятий сервер друку, система зберігатиме цю інформацію до тих пір, поки сервер друку не буде готовий її прийняти, тоді як процес-відправник зможе продовжити виконання інших завдань, не чекаючи звільнення сервера друку.

Популярною реалізацією механізму передачі повідомлень є канал (труба, pipe) – захищена ОС область пам'яті, яка виступає буфером псевдофайлу, що дозволяє декільком процесам обмінюватися між собою даними. Операційна система синхронізує доступ до буфера. Після того, як записуючий процес закінчить вести запис у буфер (ймовірно заповнивши його), система призупинить роботу записуючого процесу, дозволивши процесу-читачу почати читання даних з буфера. У міру

зчитування даних з буфера (ймовірно спустошивши його), операційна система, у свою чергу, призупинить його виконання, дозволивши записуючому процесу знову почати запис інформації в буфер.

Обговорюючи взаємодію процесів, що виконуються на одному комп'ютері, ми припускаємо, що обмін інформацією відбувається без помилок. У розподілених системах при передачі даних можуть виникати помилки, що в ряді випадків призводять до втрати інформації. Тому відправники і одержувачі часто взаємодіють між собою за допомогою протоколу квитування (установка перемикача в положення, що відповідає отриманому сигналу), використовуваного для підтвердження факту прийому інформації. Механізм тайм-ауту застосовується для обмеження часу очікування повідомлення про доставку. Якщо сигнал про доставку повідомлення не поступить після закінчення заданого інтервалу, повідомлення буде відправлене повторно.

Система передачі повідомлень з функцією повторної передачі даних дозволяє ідентифікувати нові повідомлення за їх порядковим номером. Одержувач повинен перевіряти ці номери, щоб знати, чи всі повідомлення були доставлені, і при необхідності розставити їх у правильному порядку. Якщо підтвердження про приймання повідомлення загубиться, і відправник вирішить передати повідомлення повторно, новому повідомленню привласнюється той же самий порядковий номер, який належав утраченому повідомленню.

5.8.4 Віддалений виклик процедур

Віддалений виклик процедур ґрунтується на тому, що процес, який виконується на одному комп'ютері, запускає процес на віддаленому комп'ютері. Фактично здійснюється виклик процедури, яка реально знаходиться і підтримується на іншому комп'ютері. Віддалений виклик зовні дуже схожий на локальний. Дії з передачі параметрів і отримання результату істотно відрізняються – все виконується за допомогою передач інформаційних пакетів мережею. Детальніше ці дії виглядають так, як описано нижче.

1. Програма-клієнт робить локальний виклик процедури, що називається «заглушкою», при цьому клієнтові здається, що, викликаючи «заглушку», він насправді викликає процедуру сервера. Насправді, задача «заглушки» – прийняти

аргументи, що адресуються процедурі, перетворити їх в деякий формат і сформувати мережевий запит.

2. Мережевий запит пересилається мережою на віддалену систему, при цьому, наприклад, використовується стек протоколів TCP/IP.

3. На сервері все відбувається в зворотному порядку: «заглушка» сервера чекає на запит і при його отриманні витягає з нього параметри.

4. «Заглушка»сервера виконує виклик справжньої процедури (якій адресований запит клієнта), передаючи їй потрібні параметри.

5. Після виконання процедури при передачі результатів її роботи управління знову повертається в «заглушку» сервера, яка формує повідомлення- відгук.

6. Відгук передається клієнтові.

7. Відгук приймається «заглушкою» клієнта, яка витягає необхідні дані і передає їх програмі. Процес завершений.

5.10.5 Сокети

Сокети (англ. socket – поглиблення, гніздо, роз'єм) – підтримуваний ядром механізм програмного інтерфейсу для забезпечення обміну даними між процесами, що приховує особливості середовища і дозволяє однаково взаємодіяти процесам як на одному комп'ютері, так і в мережі (рис. 5.10).

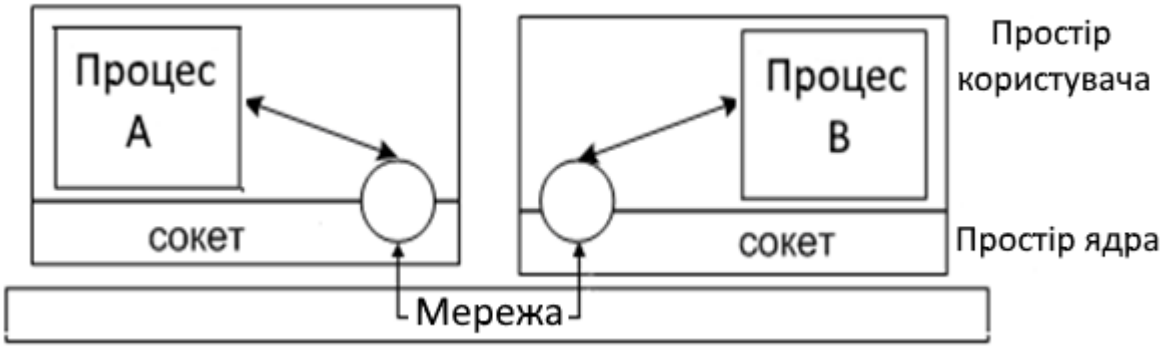

Рисунок 5.10 – Використання сокетів для роботи мережі

Слід відрізняти клієнтські і серверні сокети. Клієнтські сокети грубо можна порівняти з апаратами телефонної мережі, а серверні – з комутаторами. Клієнтській додаток (наприклад, браузер) використовує тільки клієнтські сокети, а серверний (наприклад, веб-сервер, якому браузер посилає запити) – як клієнтські, так і серверні сокети.

Сокети можуть динамічно створюватися і знищуватися. При створенні сокета процесу повертається дескриптор файлу для встановлення з'єднання, читання і запису даних, а також розриву з'єднання.

Інтерфейс сокетів уперше з'явився в BSD Unix. Програмний інтерфейс сокетів описаний в стандарті POSIX.1 і в тій чи іншій мірі підтримується всіма сучасними операційними системами.