8.3 ПЕРЕДАЧА ПОВІДОМЛЕНЬ

Семафори і монітори взагалі-то були розроблені для розв'язання задач взаємного виключення в системі з одним або декількома процесорами, що мають доступ до загальної пам'яті. Ці примітиви непридатні в розподіленій системі, що складається з декількох процесорів з власною пам'яттю у кожного, оскільки всі вони базуються на використанні оперативної пам'яті, що розділяється.

Наприклад, якщо два процеси виконуються на одній і тій же машині, вони можуть мати спільний доступ до семафора, який зберігається, наприклад, в ядрі. Проте, якщо процеси виконуються на різних машинах, то цей метод не застосовний, для розподілених систем потрібні інші підходи.

Висновок з усього вищесказаного такий: семафори є примітивами занадто низького рівня, а монітори можуть використовуватися тільки в деяких мовах програмування. Ці ж примітиви не підходять для реалізації обміну інформацією між комп'ютерами – потрібне щось інше.

8.3.1 Примітиви передачі повідомлень

При взаємодії процесів між собою повинні задовольнятися дві фундаментальні вимоги: синхронізації і комунікації. Процеси мають бути синхронізовані, щоб забезпечити виконання взаємних виключень. Співпрацюючі процеси повинні мати можливість обмінюватися інформацією. Одним з підходів до забезпечення обох вказаних функцій є передача повідомлень. Важливою перевагою передачі повідомлень є її придатність для реалізації як в одно- так і багатопроцесорних системах з пам'яттю, що розділяється, так і в розподілених системах. Для зберігання відправленого, але ще не отриманого повідомлення потрібне місце. Воно називається **буфером повідомлень**, або **поштовою скринькою.**

З системами передачі повідомлень пов'язана велика кількість складних проблем і конструктивних питань, які не виникають у разі семафорів і моніторів. Особливо багато складнощів з'являється в разі взаємодії процесів, що відбуваються на різних комп'ютерах, сполучених мережею. Так, повідомлення може загубитися в мережі.

При розробці систем передачі повідомлень слід розв'язати низку запитань, які перераховані на рис. 8.3.

Системи передачі повідомлень можуть бути різних типів, ми ж розглянемо найзагальніші можливості і властивості таких систем. Функції передачі повідомлень представлені у вигляді пари примітивів

send(<одержувач >, <повідомлення>) і

receive(<відправник >,< повідомлення >),

які є швидше системними викликами, ніж структурними компонентами мови, що відрізняє їх від моніторів і робить схожими на семафори.

Синхронізація Відправлення Блокуюче Неблокуюче Отримання Блокуюче Неблокуюче Перевірка наявності	**Формат** Вміст Довжина Фіксована Мінлива
Адресація Пряма Відправлення Отримання Неявне Явне Непряма Статична Динамічна Володіння	**Принцип роботи черги** FIFO Пріоритетна

Рисунок 8.3 – Характеристики систем передачі повідомлень

Процес відправляє інформацію у виді <повідомлення> іншому процесу, визначеному як <одержувач>, викликом send. Отримує інформацію процес за допомогою виконання примітиву receive, якому вказує відправник повідомлення.

8.3.2 Синхронізація

Розглянемо, що відбувається після того, як процес викликає примітиви send або receive. При виконанні send є дві можливості: або процес, що відправляє повідомлення, блокується, або продовжує роботу. Аналогічно є дві можливості і в процесу, що виконує примітив receive.

1. Якщо повідомлення було заздалегідь відправлене, то процес отримує його і продовжує роботу.

2. Якщо повідомлення, яке очікує отримання, немає, то:

a) або процес блокується, поки повідомлення не буде отримано;

b) або процес продовжує виконання, відмовляючись від подальших спроб отримати повідомлення.

Таким чином, і відправник, і одержувач можуть бути такими, що блокуються або не блокуються. Зазвичай зустрічається три комбінації, наведені нижче.

1. Блокуюче відправлення, блокуюче отримання. І відправник, і одержувач блокуються до тих пір, поки повідомлення не буде доставлено за призначенням. Таку ситуацію іноді називають рандеву. Ця комбінація забезпечує тісну синхронізацію процесу.

2. Неблокуюче відправлення, блокуюче отримання. Хоча відправник і може продовжувати роботу, одержувач блокується до отримання повідомлення. Ця комбінація зустрічається найчастіше. Вона дозволяє процесу відправляти одне або декілька повідомлень різним одержувачам з максимальною швидкістю. Прикладом може служити серверний процес, існуючий для надання сервісів або ресурсів іншим процесам.

3. Неблокуюче відправлення, неблокуюче отримання.

Неблокуючий примітив send найприродніший для більшості задач з використанням паралельних обчислень. Але при такому підході на програміста покладається задача відстежування успішної доставки повідомлення адресатові. Процес-одержувач повинен, у свою чергу, відправити відповідь з підтвердженням отримання повідомлення. Інакше можлива ситуація, коли деяка помилка призведе до безперервної генерації повідомлень.

У разі використання примітиву receive для більшості задач природною видається блокуюча технологія, оскільки, найчастіше, процес, що запросив інформацію, потребує її для продовження роботи. Правда, якщо повідомлення втрачається, одержувач може виявитися назавжди заблокованим. Розв'язати цю проблему можна за допомогою неблокованого примітиву receive. Проте у цього варіанту є своє слабке місце: якщо повідомлення відправлене після того, як процес виконав операцію receive, то воно виявляється втраченим. Ще один можливий підхід для вирішення цієї проблеми полягає в тому, що перед тим, як виконати receive, необхідно перевірити, чи немає повідомлення, що очікує отримання.

8.3.3 Адресація

Різні схеми визначення процесів у примітивах send і receive розділяються на дві категорії: пряму і непряму адресацію.

При прямій адресації примітив send включає ідентифікатор процесу-одержувача. Коли застосовується примітив receive, можна піти двома шляхами. Перший шлях полягає у вимозі явної вказівки процесу-відправнику, тобто процес повинен знати заздалегідь, від якого саме процесу він чекає повідомлення. Такий шлях досить ефективний, якщо паралельні процеси співпрацюють.

Проте в багатьох випадках неможливо передбачити, який процес буде відправником очікуваного повідомлення. Як приклад можна навести процес сервера друку, який приймає повідомлення – запити на друк від будь-якого іншого процесу. Для таких додатків ефективнішим буде підхід з використанням неявної адресації. У цьому випадку параметр <відправник> набуває значення, яке повертається після виконання операції отримання повідомлення.

Ще одним поширеним підходом є непряма адресація. Вона припускає, що повідомлення відправляються не прямо від відправника одержувачеві, а направляється в спільно використовувану структуру даних, що складається з черг для тимчасового зберігання повідомлень. Такі черги іменують поштовими скриньками (mailbox). Таким чином, для зв'язку між двома процесами один з них посилає повідомлення у відповідну поштову скриньку, з якої його забирає другий процес.

Ефективність непрямої адресації полягає в гнучкості використання повідомлень. При такій схемі роботи з повідомленнями відношення між

відправником і одержувачем можуть бути будь-якими – «один до одного», «один до багатьох», «багато до одного» або «багато до багатьох».

Відношення «один до одного» забезпечує закритий зв'язок між двома процесами, ізолюючи їх від стороннього втручання. Відношення «багато до одного» корисно при взаємодії клієнт/сервер – один процес при цьому є сервером, обслуговуючим багато клієнтів. У такому разі про поштову скриньку говорять як про порт. Відношення «один до багатьох» забезпечує розсилку від одного процесу багатьом одержувачам, дозволяючи здійснити широкомовне повідомлення процесам.

Зв'язок процесів з поштовими скриньками може бути як статичним, так і динамічним. Порти найчастіше статистично пов'язані з певними процесами. Тобто, порт створюється і назначається процесу назавжди. Те ж спостерігається і у разі використання відношення «один до одного» – закриті канали зв'язку, як правило, також визначаються статично, раз і назавжди.

За наявності декількох відправників їх зв'язок з поштовою скринькою може здійснюватися динамічно з використанням для цієї мети додаткових примітивів connect і disconnect.

8.3.4 Формат повідомлення

Формат повідомлення залежить від переслідуваних цілей і від того, чи працює система передачі повідомлень на одному комп'ютері або в розподіленій системі. У ряді ОС розробники віддають перевагу коротким повідомленням фіксованої довжини, що дозволяє мінімізувати обробку і зменшити витрати пам'яті на їх зберігання. Проте гнучкіший підхід дозволяє використати повідомлення змінної довжини. На рис. 8.4 показаний формат типового повідомлення змінної довжини.

Тип повідомлення
Ідентифікатор одержувача
Ідентифікатор відправника
Довжина повідомлення
Управляюча інформація
Зміст повідомлення

Рисунок 8.4 – Узагальнений формат повідомлення

8.3.5. Взаємні виключення

У лістингу 8.10 показаний один із способів реалізації взаємних виключень з використанням системи передачі повідомлень. У цій програмі передбачається використання блокуючого receive і неблокуючого send. Певна кількість процесів, що паралельно виконуються, спільно використовують поштову скриньку mutex як для відправки повідомлень, так і для їх отримання. Поштова скринька після ініціалізації містить єдине повідомлення з порожнім змістом. Процес, що має намір увійти до критичного розділу, спочатку намагається отримати повідомлення. Якщо поштова скринька порожня, процес блокується. Як тільки процес отримує повідомлення, він тут же виконує критичний розділ і потім посилає повідомлення назад в поштову скриньку.

Лістинг 8.10 – Реалізація взаємних виключень з використанням повідомлень

```
const int n = /* Кількість процесів */;
void P(int n) {
message msg;
while(true) {
     receive(mutex, msg);
     /* Критичний розділ */;
     send(mutex, msg);
     /* Інший код */;
     }
}
void main() {
     create_mailbox(mutex);
     send(mutex, null);
     parbrgin(P(1),P(2),…,p(n));
}
```

У розглянутому рішенні передбачається, що якщо операція `receive` виконується паралельно більш ніж одним процесом, то:

– якщо є повідомлення, воно передається тільки одному з процесів, а інші процеси блокуються;

– якщо черга повідомлень порожня, блокуються усі процеси; при появі в черзі повідомлення, його отримує тільки один із заблокованих процесів.

Це припущення виконується практично для всіх засобів передачі повідомлень.

В якості іншого прикладу використання повідомлень в лістингу 8.11 наведено рішення задачі Виробник-Споживач з обмеженим буфером.

Лістинг 8.11. Рішення задачі Виробник-Споживач з обмеженим буфером і з використанням повідомлень

```
const int capacity = /* Ємність буфера */;
Null = /* Порожнє повідомлення */;
int I;
void producer() {
      message pmsg;
      while(true) {
           receive(myproduce, pmsg);
           pmsg = produce();
           send(mayconsume, pmsg);
           }
      }
void consumer() {
      message cmsg;
      while(true) {
           receive(myconsume, cmsg);
           consume(cmsg);
           send(mayproduce, null);
           }
      }
void main() {
       create_mailbox(mayproduce);
       create_mailbox(mayconsume);
       for (int I = 1; I <= capacity;
       I++) send(mayproduce, null);
       parbegin(producer, consumer);
       }
```

На відміну від семафорів тут передаються повідомлення, а не сигнали. У програмі використовуються дві поштові скриньки. Коли виробник генерує дані, він посилає їх в якості повідомлення в поштову скриньку `mayconsume`. Поки в цій поштовій скриньці є хоч би одне повідомлення, споживач може отримати дані. Отже, поштова скринька mayconsume служить буфером, дані в якому організовані у вигляді черги повідомлень. «Місткість» цього буфера визначається глобальною змінною capacity. Поштова скринька mayproduce спочатку заповнена порожніми повідомленнями в кількості, рівній місткості буфера. Кількість повідомлень в цій поштовій скриньці зменшується при кожному прибутті нових даних і збільшується при їх використанні.

Такий підхід досить гнучкий – він може працювати з будь-якою кількістю виробників і споживачів. Більш того, система виробників і споживачів може бути розподіленою, коли усі виробники і поштова скринька mayproduce знаходяться на одній машині, а споживачі і поштова скринька mayconsume – на іншій.