15.6 ДИСПЕТЧЕРИЗАЦІЯ ДИСКОВИХ ОПЕРАЦІЙ

Алгоритми диспетчеризації дискових операцій, які використовуються системою, залежать від призначення цієї системи, але більшість алгоритмів оцінюються за такими загальними критеріями [12]:

1. **Пропускна спроможність** – кількість запитів, що виконуються за одиницю часу.

2. **Середній час реагування** – середній час, що проходить між надходженням запиту і його виконанням.

3. **Розкид часу реагування** – рівень передбачуваності часу реагування на запит. Кожен запит повинен виконуватися впродовж певного періоду часу. Тобто алгоритм не повинен відкладати виконання запиту до безкінечності.

Розглянемо декілька широко застосованих алгоритмів диспетчеризації.

15.6.1 Алгоритм First Come First Served

Простим алгоритмом, до якого ми вже повинні були звикнути, є алгоритм **First Come First Served (FCFS)** – «першим прийшов, першим обслужений», або «першим увійшов – першим вийшов» (FIFO), що просто означає обробку запитів з черги в порядку їх надходження.

Алгоритм простий в реалізації, але може призводити до досить тривалого загального часу обслуговування запитів. Розглянемо приклад. Нехай у нас до диску з 40 циліндрів (від 0 до 39) є така черга запитів: 1, 36, 16, 34, 9, 12 і головки в початковий момент знаходяться на 11-му циліндрі. Тоді положення головок мінятиметься таким чином (рис. 15.9):

11→1→36→16→34→9→12

і всі головки переміститься на **111** циліндрів (10+35+20+18+25+3).

Рисунок 15.9 – Алгоритм планування FCFS

Неефективність алгоритму добре ілюструється двома останніми переміщеннями з 1 циліндра через увесь диск на 36 циліндр і потім знову з циліндра 9 на циліндр 34. При використанні FCFS сподіватися на високу продуктивність можна тільки при невеликій кількості процесів і запитах до близьких груп секторів. Із зростанням навантаження алгоритм FCFS швидко насичується, досягаючи граничної продуктивності, і час виконання запиту стає занадто великим (черга запитів швидко збільшується).

15.6.2 Алгоритм диспетчеризації SSTF

Алгоритм вибору найменшого часу обслуговування (Shortest Service Time First – **SSTF**) полягає у виборі того дискового запиту на введення-виведення, яке вимагає найменшого переміщення головок з поточної позиції. Отже, мінімізується час пошуку. Постійний вибір мінімального часу пошуку не дає гарантії, що середній час пошуку при усіх переміщеннях буде мінімальним, але, проте, ця стратегія забезпечує кращу в порівнянні з FIFO продуктивність. Оскільки головки можуть переміщатися в двох напрямах, то при рівних відстанях для ухвалення рішення може бути використаний випадковий вибір. Для попереднього прикладу (1, 36, 16, 34, 9, 12) цей алгоритм дасть таку послідовність положень головок (рис. 15.10):

11→12→9→16→1→34→36 (1+3+7+15+33+2)

і всі головки перемістяться на **61** циліндр (FCFS – 111).

Рисунок 15.10 – Алгоритм планування SSTF

Цей алгоритм зменшує загальні переміщення блоку головок в порівнянні з алгоритмом FCFS приблизно в два рази.

На жаль, алгоритм SSTF не обходиться без недоліків. Припустимо, що за час обробки запитів, показаних на рис. 15.9, продовжують надходити все нові і нові запити. Наприклад, якщо після переміщення до циліндра 16 є запит до циліндра 8, цей

запит буде мати пріоритет над запитом до циліндра 1. Якщо потім надійде запит до циліндра 13, то в наступну чергу блок перейде до нього, а не до циліндра 1.

При високій завантаженості диска блок головок буде більшу частину часу залишатися в його середній частині, тому запитам до крайніх циліндрів доведеться чекати, поки статистичні відхилення в завантаженості диска не приведуть до відсутності запитів до середніх циліндрів. Запити, віддалені від середньої частини, можуть погано обслуговуватися. Тут цілі досягнення рівнодоступності і мінімізації часу відгуку вступають в конфлікт.

Алгоритм SSTF погано підходить для інтерактивних систем, які повинні надавати кожному користувачеві рівні, передбачувані часи реагування.

15.6.3 Алгоритм диспетчеризації SCAN

Усі стратегії, розглянуті раніше (окрім FIFO), можуть залишити деякий запит до тих пір, поки не звільнитися вся черга. Тобто при роботі завжди можуть бути нові запити, які будуть вибрані до вже наявного запиту в черзі. Уникнути такого роду «голодування» можна при використанні іншої стратегії.

У простому з алгоритмів сканування – **SCAN** – переміщення головки відбувається тільки в одному напрямі, задовольняючи ті запити, які відповідають вибраному напряму. Після досягнення останньої доріжки у вибраному напрямі (чи коли вичерпуються можливі запити), напрям змінюється на протилежний.

Нехай в попередньому прикладі в початковий момент часу головки рухаються у напрямі зменшення номерів циліндрів. Послідовність переміщення головок для запитів (12, 16, 34, 36, 39, 9, 1) виглядає таким чином:

11→12→16→34→36→39→9→1 (1+4+18+2+3+30+8)

і всі головки перемістяться на **66** циліндрів (FCFS – 111, SSTF – 61). В даному випадку алгоритм SCAN виявився дещо гіршим, ніж SSF.

Але, якщо ми знаємо, що обслужили останній попутний запит у напрямі руху головок, то ми можемо не доходити до краю диска, а відразу змінити напрям руху на зворотній (рис. 15.11):

11→12→16→34→36→9→1 (1+4+18+2+27+8)

і всі головки перемістяться на **60** циліндри**.** Отримана модифікація алгоритму SCAN дістала назву **LOOK** (SCAN-LOOK).

Рисунок 15.11 – Алгоритм планування SCAN-LOOK

Неважко побачити, що стратегія SCAN надає перевагу тим завданням, чиї запити стосуються доріжок, які знаходяться щонайближче до центру або найвіддаленіші від нього. Вона також може віддавати перевагу запитам, які поступили останніми.

15.6.4 Алгоритм диспетчеризації C-SCAN

Допустимо, що до моменту зміни безпосередньо руху головки в алгоритмі SCAN, тобто коли головка досягла одного з країв диска, у цього краю накопичилася велика кількість нових запитів, на обслуговування яких буде витрачено досить багато часу (не забуваємо, що треба не лише переміщати головку, але ще і передавати прочитані дані). Тоді запити, що належать до іншого краю диска і поступили раніше, чекатимуть обслуговування несправедливо довго. Для скорочення часу очікування запитів застосовується інша модифікація алгоритму SCAN – **C-SCAN** (circular SCAN – цикличне сканування) [12]. Коли головка досягає одного з країв диска, вона без читання попутних запитів (іноді істотно швидше, ніж при виконанні звичайного пошуку циліндра) переміщається на інший край, звідки знову починає рух у тому ж напрямі. Для цього алгоритму послідовність переміщень виглядатиме так (12, 16, 34, 36, 39, 9, 1):

11→12→16→34→36→39; 0→1→9 (1+4+18+2+3+1+8)

і всі головки перемістяться на **37** циліндрів.

За аналогією з алгоритмом LOOK для алгоритму SCAN також можна запропонувати метод **C-LOOK** для алгоритму C-SCAN (C-SCAN-LOOK):

11→12→16→34→36; 0→1→9 (1+4+18+2+1+8=**34**) (FCFS – 111, SSTF – 61, SCAN – 66, SCAN – LOOK – 60)

Існують і зовсім інші алгоритми, але на цьому закінчимо наш огляд.