Планування процесів

1. обмежена кількість ресурсів
2. декілька їхніх споживачів

ми змушені займатися розподілом наявних ресурсів між споживачами (плануванням використання ресурсів)

Таке планування повинно мати чітко поставлені цілі (чого ми хочемо домогтися за рахунок розподілу ресурсів) і алгоритми, які відповідають цілям і опираються на параметри споживачів.

Дана лекція присвячена плануванню виконання процесів у мультипрограмних обчислювальних системах або, інакше кажучи, плануванню процесів.

Рівні планування

Історично сформувалися під час еволюції два види планування в обчислювальних системах:

1. планування завдань

2. планування використання процесора.

Планування завдань з'явилося в пакетних системах після того, як для зберігання сформованих пакетів завдань почали використовуватися магнітні диски.

Планування використання процесора вперше виникає в мультипрограмних обчислювальних

системах, де в стані готовність можуть одночасно перебувати кілька процесів.

Обидва види планування ми будемо розглядати як **різні рівні планування процесів**.

Планування завдань використовується як **довготермінове планування процесів**. Воно відповідає за породження нових процесів у системі, визначаючи її *ступінь мультипрограмування*, тобто кількість процесів, що одночасно перебувають у ній.

Планування використання процесора застосовується як **короткотермінове планування процесів**. Воно здійснюється, наприклад, при зверненні виконуваного процесу до пристроїв вводу-виводу або просто після завершення певного інтервалу часу.

У деяких обчислювальних системах буває вигідно для підвищення продуктивності тимчасово вилучити будь-який процес, що частково виконався, з оперативної пам'яті на диск, а пізніше повернути його назад для подальшого виконання. Така процедура в англомовній літературі одержала назву **swapping.** Коли і який із процесів потрібно перекачати на диск і повернути назад, вирішується додатковим *проміжним рівнем планування процесів* -- **середньотерміновим**.

Критерії планування і вимоги до алгоритмів

Вибір конкретного алгоритму визначається класом завдань, що розв'язуються обчислювальною системою, і **цілями**, яких ми хочемо досягти, використовуючи планування:

- Справедливість
- Ефективність
- Скорочення повного часу виконання (turnaround time)
- Скорочення часу очікування (waiting time)
- Скорочення часу відгуку (response time)

Незалежно від поставлених цілей планування бажано також, щоб алгоритми мали наступні **властивості**.

- Були передбачуваними.
- Були пов'язані з мінімальними накладними витратами.
- Рівномірно завантажували ресурси обчислювальної системи.
- Мали масштабованість.

Багато з наведених вище цілей і властивостей є суперечливими.

Параметри планування

Для *здійснення поставлених цілей* розумні алгоритми планування повинні опиратися на

будь-які характеристики процесів у системі,
завдань у черзі на завантаження,
стани самої обчислювальної системи

Всі параметри планування можна розбити на дві більші групи: *статичні параметри* і *динамічні параметри*.

До статичних параметрів обчислювальної системи можна віднести *граничні значення її ресурсів* (розмір оперативної пам'яті, максимальна кількість пам'яті на диску для здійснення свопінгу, кількість підключених пристроїв вводу-виводу і т.д.).

Динамічні параметри системи описують к*ількість вільних ресурсів на даний момент*.

До статичних параметрів процесів відносяться характеристики, як правило властиві для завдань уже на етапі завантаження.

- Яким користувачем запущений процес
- пріоритет виконання.
- Скільки процесорного часу потрібно користувачеві для вирішення завдання.
- Яке співвідношення процесорного часу і часу, необхідного для здійснення операцій вводу-виводу.
- Які ресурси обчислювальної системи (оперативна пам'ять, пристрої вводу-виводу, спеціальні

бібліотеки і системні програми і т.д.) і в якій кількості необхідні завданню.

Алгоритми ***довготермінового*** планування використовують у своїй роботі *статичні і динамічні параметри обчислювальної системи і статичні параметри процесів* (динамічні параметри процесів на етапі завантаження завдань ще не відомі).

Алгоритми ***короткотермінового і середньотермінового*** планування додатково враховують і *динамічні характеристики процесів*.

Для *середньотермінового* планування як такі характеристики може використовуватися наступна інформація:

- скільки часу пройшло з моменту вивантаження процесу на диск або його завантаження в оперативну пам'ять;
- скільки оперативної пам'яті займає процес;
- скільки процесорного часу вже надано процесу.

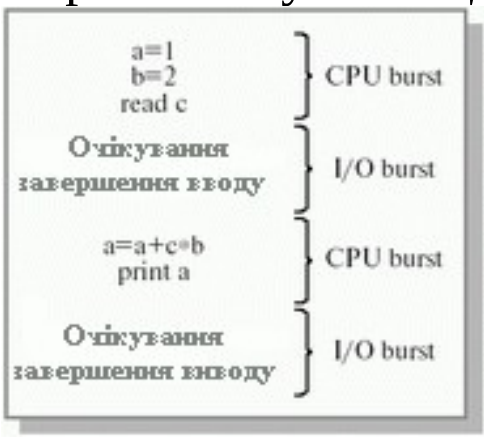

Рис. 3.1. *Фрагмент діяльності процесу з виділенням проміжків неперервного використання процесора і очікування вводу-виводу*

Для *короткотермінового* планування нам знадобиться ввести ще два динамічних параметри.

Діяльність будь-якого процесу можна уявити як послідовність циклів використання процесора і очікування завершення операцій вводу-виводу. Проміжок часу неперервного використання процесора називається **CPU burst**, а проміжок часу неперервного очікування вводу-виводу – **I/O burst**.

Витісняльне і невитісняльне планування

Процес планування здійснюється частиною операційної системи, яка називається **планувальником**. Планувальник може ухвалювати рішення щодо вибору для виконання нового процесу із процесів, що перебувають у стані готовність у таких чотирьох випадках.

1. **Виконання --> закінчив виконання**.
2. **Виконання --> очікування**.
3. **Виконання --> готовність**.
4. **Очікування --> готовність**.

У випадках 1 і 2 *процес*, що перебував у стані виконання, *не може далі виконуватися*, і операційна система змушена здійснювати планування вибираючи новий процес для виконання.

У випадках 3 і 4 *планування може як здійснюватися, так і не здійснюватися*, планувальник не обов'язково повинен ухвалювати рішення щодо вибору процесу для виконання, процес, що перебував у стані виконання може просто продовжити свою роботу.

Якщо в операційній системі планування здійснюється тільки у вимушених ситуаціях, говорять,

що має місце що **невитісняльне (nonpreemptive) планування**. Якщо планувальник приймає і вимушені, і невимушені рішення, говорять про **витісняльне (preemptive) планування**.

Термін "витісняльне планування" виник тому, що ***виконуваний процес поза своєю волею може бути витиснутий зі стану виконання іншим процесом***.

Невитісняльне планування використовується, наприклад, в MS Windows 3.1 і ОС Apple Macintosh. При такому режимі планування *процес займає стільки процесорного часу, скільки йому необхідно*.

Витісняльне планування зазвичай використовується ***в системах поділу часу***. У цьому режимі планування процес може бути припинений у будь-який момент виконання.

Алгоритми планування

First-Come, First-Served (FCFS)

першим прийшов, першим був обслужений

Уявімо собі, що процеси, що перебувають у стані готовність, вибудовані в чергу. Коли процес переходить у стан готовність, він, а точніше, посилання на його PCB міститься в кінці цієї черги. Вибір нового процесу для виконання здійснюється з початку черги з видаленням відтіля посилання на його PCB. Черга подібного типу має в програмуванні спеціальне найменування – FIFO[1], скорочення від First In, First Out (першим увійшов, першим вийшов).

Такий алгоритм вибору процесу здійснює *невитісняльне* планування. Процес, що одержав у своє розпорядження процесор, займає його до закінчення поточного CPU burst. Після цього для виконання вибирається новий процес із початку черги.

Таблиця 3.1.

Процес	p0	p1	p2
Тривалість наступного CPU burst	13	4	1

Перевагою алгоритму FCFS є *легкість його реалізації*, але в той же час він має і багато недоліків.

1. Варто відзначити, що абревіатура FCFS використовується для цього алгоритму планування замість стандартної абревіатури FIFO для механізмів подібного типу для того, щоб підкреслити, що організація готових процесів у чергу FIFO можлива і при інших алгоритмах планування (наприклад, для Round Robin – див. розділ "Round Robin (RR)").

Рис. 3.2. *Виконання процесів у порядку p0, p1, p2*

Якщо ті ж самі процеси розташовані в порядку p_2, p_1, p_0, то картина їхнього виконання буде відповідати рис. 3.3. Час очікування для процесу p_0 дорівнює 5 одиницям часу, для процесу p_1 – 1 одиниці, для процесу p_2 – 0 одиниць. Середній час очікування складе (5 + 1 + 0)/3 = 2 одиниці часу. Це в 5 (!) разів менше, ніж у попередньому випадку. Повний час виконання для процесу p_0 буде рівним 18 одиницям часу, для процесу p_1 – 5 одиницям, для процесу p_2 – 1 одиниці. Середній повний час виконання становить (18 + 5 + 1)/3 = 8 одиниць часу, що майже в 2 рази менше, ніж при першому розміщенні процесів.

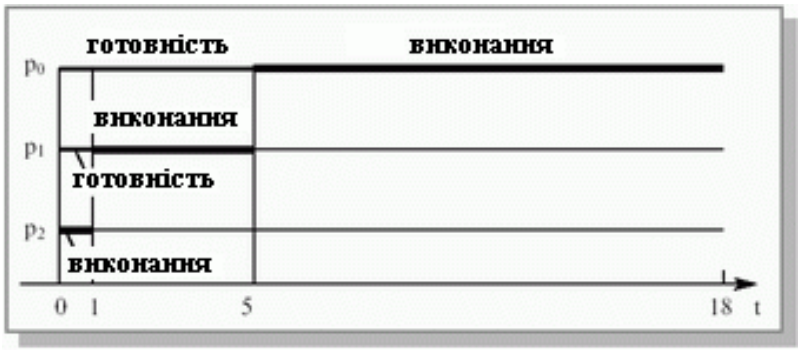

Рис. 3.3. *Виконання процесів у порядку p2, p1, p0*

Як ми бачимо, ***середній час очікування і середній повний час виконання для цього алгоритму істотно залежать від порядку розташування процесів у черзі***. Якщо в нас є процес із тривалим CPU burst, то короткі процеси, що перейшли в стан готовність після тривалого процесу, будуть дуже довго чекати початку виконання. Тому алгоритм ***FCFS практично не застосовується для систем поділу часу*** -- занадто великим виходить середній час відгуку в інтерактивних процесах.

Round Robin (RR)

Модифікацією алгоритму FCFS є алгоритм, що одержав назву Round Robin (Round Robin – це вид дитячої каруселі в США) або скорочено RR. Насправді, це той же самий алгоритм, тільки ***реалізований у режимі витісняльного планування***. Можна уявити собі безліч готових процесів організованих циклічно – процеси сидять на каруселі. Карусель обертається так, що кожний процес перебуває біля процесора невеликий фіксований квант часу, зазвичай 10 – 100 мілісекунд (див. рис. 3.4). Поки процес перебуває поруч із процесором, він одержує процесор у своє розпорядження і може виконуватися.

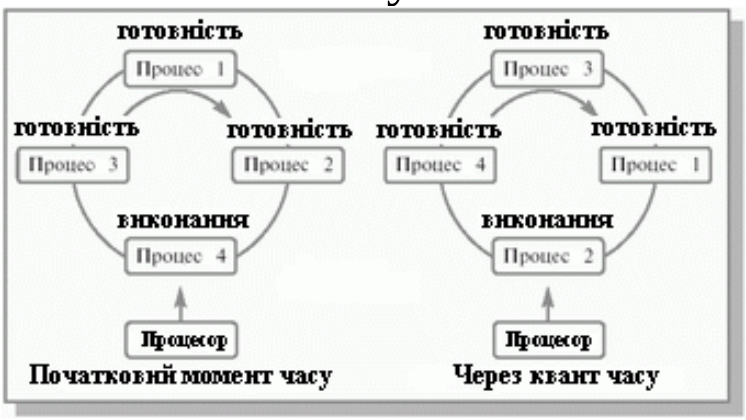

Рис. 3.4. *Процеси на каруселі*

Реалізується такий алгоритм так само, як і попередній, за допомогою організації процесів, що перебувають у стані готовність, у чергу FIFO. Планувальник вибирає для наступного виконання процес, розташований на початку черги, і встановлює таймер для генерації переривання після закінчення певного кванту часу. Під час виконання процесу можливі два варіанти.

- Час неперервного використання процесора, необхідний процесу (залишок поточного CPU burst), менший або дорівнює тривалості кванту часу. Тоді процес за своїм бажанням звільняє процесор до завершення кванта часу, на виконання надходить новий процес із початку черги, і таймер починає відлік кванта заново.
- Тривалість залишку поточного CPU burst процесу більше, ніж квант часу. Тоді після закінчення цього кванта процес переривається таймером і розміщується в кінець черги процесів, готових до виконання, а процесор виділяється для використання процесу, що перебуває на її початку.

Таблиця 3.2.

Час	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
p_0	В	В	В	В	Г	Г	Г	Г	Г	В	В	В	В	В	В	В	В	В
p_1	Г	Г	Г	Г	В	В	В	В										
p_2	Г	Г	Г	Г	Г	Г	Г	Г	В									

На продуктивність алгоритму RR сильно впливає величина кванта часу.

Таблиця 3.3.

Час	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
p_0	В	Г	Г	В	Г	В	Г	В	Г	В	В	В	В	В	В	В	В	В
p_1	Г	В	Г	Г	В	Г	В	Г	В	Г	В							
p_2	Г	Г	В															

При дуже великих значеннях кванта часу, коли кожний процес устигає завершити свій CPU burst до виникнення переривання за часом, алгоритм RR вироджується в алгоритм FCFS. *При дуже малих величинах* створюється ілюзія того, що кожний з n процесів працює на власному віртуальному процесорі із продуктивністю ~ 1/n від продуктивності реального процесора.

Shortest-Job-First (SJF)

Якби ми знали ***час наступних CPU burst для процесів***, що перебувають у стані готовність, то могли б вибрати для виконання не процес із початку черги, а процес із ***мінімальною тривалістю CPU burst***. Якщо ж таких процесів два або більше, то для вибору одного з них можна використати вже відомий нам алгоритм FCFS. Квантування часу при цьому не застосовується. Описаний алгоритм одержав назву "найкоротша робота першого" або Shortest Job First (SJF).

SJF-алгоритм короткотермінового планування може бути як **витісняльним**, так і **невитісняльним**.

Основною складністю при реалізації алгоритму SJF є **неможливість точного знання тривалості наступного CPU burst для процесів, що виконуються**.

Гарантоване планування

При інтерактивній роботі N користувачів в обчислювальній системі можна застосувати алгоритм планування, який гарантує, що кожний з користувачів буде мати у своєму розпорядженні ~1/N частину процесорного часу.

Пронумеруємо всіх користувачів від 1 до N.

Для кожного користувача з номером i введемо дві величини:

T_i – час знаходження користувача в системі або, інакше кажучи, тривалість сеансу його спілкування з машиною

τ_i – сумарний процесорний час уже виділений всім його процесам протягом сеансу. Справедливим для користувача було б одержання T_i/N процесорного часу.

Якщо

$\tau_i << T_i/N,$

то і-й користувач несправедливо обділений процесорним часом. Якщо ж

$\tau_i >> T_i/N$

то система лояльно ставиться до користувача з номером i. Обчислимо для процесів кожного користувача значення коефіцієнта справедливості

τ_i/T_i

і будемо надавати черговий квант часу готовому процесу з найменшою величиною цього відношення. Запропонований алгоритм називають алгоритмом гарантованого планування. До недоліків цього

алгоритму можна віднести неможливість вгадати поведінку користувачів. Якщо деякий користувач відправиться на кілька годин пообідати і поспати, не перериваючи сеансу роботи, то після повернення його процеси будуть одержувати невиправдано багато процесорного часу.

Пріоритетне планування

Алгоритми SJF і гарантованого планування є окремими випадками пріоритетного планування.

При пріоритетному плануванні кожному процесу привласнюється певне числове значення -- **пріоритет**, відповідно до якого йому виділяється процесор. Процеси з однаковими пріоритетами плануються в порядку FCFS.

Для алгоритму SJF як такий пріоритет виступає *оцінка тривалості наступного CPU burst*. Чим менше значення цієї оцінки, тим вищий пріоритет має процес.

Для алгоритму гарантованого планування пріоритетом служить обчислений *коефіцієнт справедливості*. Чим він менше, тим більше в процесу пріоритет.

Алгоритми призначення пріоритетів процесів можуть опиратися як на внутрішні параметри, пов'язані із тим, що відбувається всередині обчислювальної системи, так і на зовнішні стосовно до неї.

До внутрішніх параметрів відносяться різні кількісні і якісні характеристики процесу такі як: обмеження за часом використання процесора, вимоги до розміру пам'яті, кількість відкритих файлів і використовуваних пристроїв вводу-виводу, відношення середніх тривалостей I/O burst до CPU burst і т.д.

Алгоритми SJF і гарантованого планування використовують внутрішні параметри.

Як зовнішні параметри можуть виступати важливість процесу для досягнення яких-небудь цілей, вартість оплаченого процесорного часу і інші політичні фактори.

Планування з використанням пріоритетів може бути як витісняльним, так і невитісняльним.

Багаторівневі черги (Multilevel Queue)

Для систем, у яких процеси можуть бути легко розсортовані по різних групах, був розроблений інший клас алгоритмів планування. Для кожної групи процесів створюється своя черга процесів, що перебувають у стані готовність (див. рис. 3.5). Цим чергам приписуються фіксовані пріоритети.

Рис. 3.5. *Кілька черг планування*

Багаторівневі черги зі зворотнім зв'язком (Multilevel Feedback Queue)

Подальшим розвитком алгоритму багаторівневих черг є додавання до нього механізму зворотного зв'язку.

Тут процес не постійно приписаний до певної черги, а може мігрувати з однієї черги в іншу залежно від свєї поведінки.

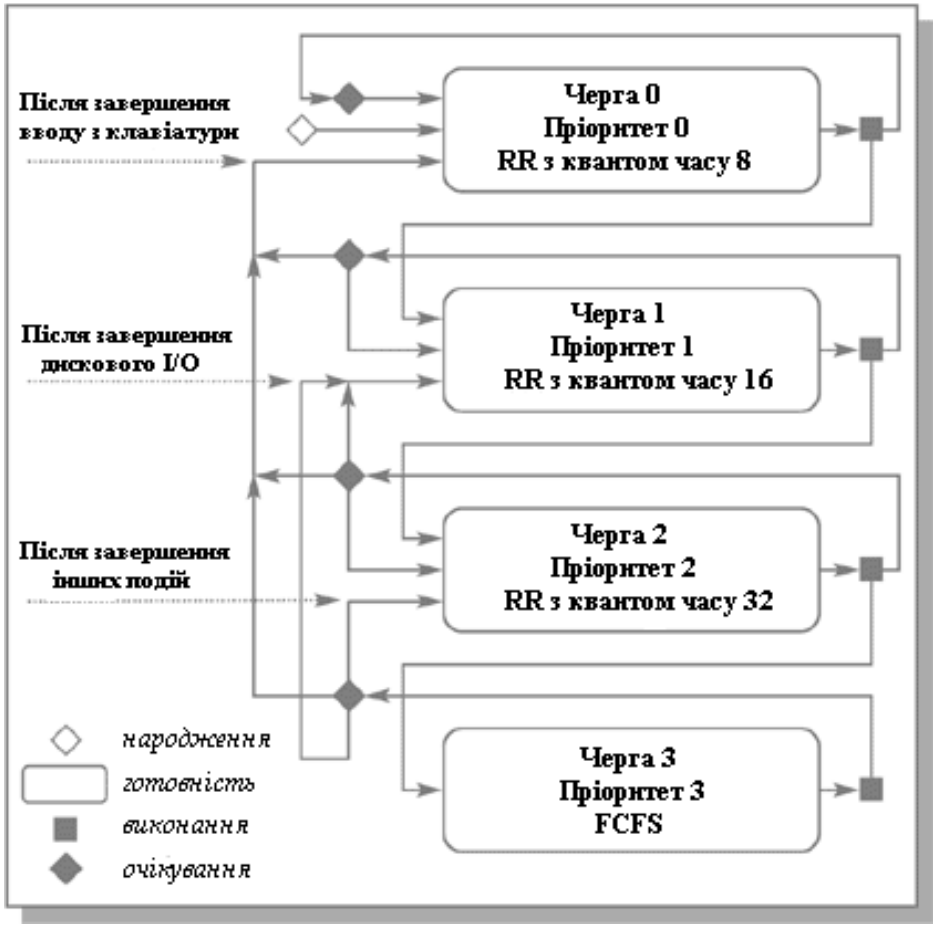

Рис. 3.6. *Схема міграції процесів у багаторівневих чергах планування зі зворотнім зв'язком. Витиснення процесів пріоритетнішими процесами і завершення процесів на схемі не показано*

Багаторівневі черги зі зворотним зв'язком є найбільш загальним підходом до планування процесів із числа підходів, розглянутих нами. Вони найважчі в реалізації, але в той же час мають найбільшу гнучкість. Зрозуміло, що існує багато інших різновидів такого способу планування, крім варіанта, наведеного вище. Для повного опису їхнього конкретного втілення необхідно вказати:

- Кількість черг для процесів, що перебувають у стані готовність.
- Алгоритм планування, що діє між чергами.
- Алгоритми планування, що діють усередині черг.
- Правила переміщення народженого процесу в одну із черг.
- Правила переміщення процесів з однієї черги в іншу.

Змінюючи будь-який із перерахованих пунктів, ми можемо істотно змінювати поведінку обчислювальної системи.