ПРАКТИЧНА РОБОТА 03
Алгоритми планування процесів
План

1. Планування процесів.
2. Алгоритми планування процесів.
3. Практичні завдання.

1. Планування процесів

Планування використання процесів виникає в мультипрограмних обчислювальних системах, де кілька процесів можуть одночасно знаходитись у стані готовності. Один з процесів повинен отримувати ядро процесора в користування, тобто перейде в стан виконання.

Планування використання процесору проявляється в короткостроковому плануванні процесів. Воно здійснюється, наприклад, при зверненні процесу, який виконується, до пристроїв вводу-виводу або після завершення деякого періоду часу. Короткострокове планування здійснюється досить часто, як правило, не рідше одного разу в 100мс. Вибір нового процесу для виконання впливає на функціонування системи до настання чергової аналогічної події, тобто протягом короткого проміжку часу.

У деяких системах може бути вигідним для підвищення продуктивності тимчасово видалити деякий частково виконаний процес з оперативної пам’яті на диск, а пізніше повернути його для подальшого виконання. Таку процедуру часто називають ***свопінгом (swapping)***.

Необхідність алгоритму планування залежить від завдань, для яких буде використовуватись операційна система. Завдання алгоритмів планування:

- Справедливість - кожному процесу справедливу частку процесорного часу
- Контроль над виконанням прийнятої політики
- Баланс – підтримка зайнятості всіх частин системи (наприклад, щоб були зайняті процесор і пристрої вводу-виводу)
- Пропускна здатність - кількість завдань на годину
- Оборотний час - мінімізація часу на очікування обслуговування та обробку задач.
- Використання процесу – щоб процесор завжди був зайнятий.
- Час відгуку – швидка реакція на запити
- Відповідність – виконання очікувань користувача (наприклад, користувач не готовий до довгого завантаження системи)
- Закінчення роботи до терміну - запобігання втрати даних

– Передбачуваність – запобігання деградації якості в мультимедійних системах (наприклад, погіршення якості звучання має бути нижче, ніж відео).

Для короткострокового планування вводяться динамічні параметри. Діяльність будь-якого процесу можна представити як послідовність циклів використання процесора і очікування завершення операцій вводу-виводу. Проміжок часу безперервного використання процесора називається ***CPU burst***, а проміжок часу безперервного очікування вводу-виводу – ***I/O burst***. Значення тривалості останніх і чергових CPU burst та I/O burst є важливими динамічними параметрами процесу.

Для оцінки ефективності функціонування алгоритму планування можуть бути застосовані кількісні показники. Позначимо через t процесорний час, необхідний процесу для виконання (***тривалість процесу***).

Нехай T – загальна тривалість перебування процесу в системі. Цю величину – інтервал між моментом введення процесу в систему і моментом отримання результатів – також називають іноді Для оцінки ефективності функціонування алгоритму планування можуть бути застосовані кількісні показники. Позначимо через t - процесорний час, необхідний процесу для виконання (будемо його називати тривалістю процесу).

Позначимо через T - загальний час перебування процесу в системі. Цю величину - інтервал між моментом введення процесу в систему і моментом отримання результатів - також називають іноді ***часом реакції процесу***. Поряд з часом реакції можуть бути корисні також і інші показники:

– Втрачений час $M = T - t$; визначає час, протягом якого процес знаходився в системі, але не виконувався.
– Відношення реактивності $R = t/T$; показує частку процесорного часу (часу виконання) або частку втраченого часу в загальній тривалості реакції.
– Штрафне відношення $P = T/t$; показує, у скільки разів загальний час виконання процесу перевищує необхідний процесорний час.

Середні значення величин T, M, R і P можуть служити кількісними показниками ефективності.

2. ***Алгоритми планування процесів***

2.1. Алгоритм FCFS

FCFS (first come – first serve; перший прийшов – перший обслуговується). Найпростіший алгоритм, робота якого зрозуміла з назви. Це алгоритм без витіснення, тобто процес, обраний для виконання на ЦП, не переривається, поки не завершиться (або не перейде в стан очікування за

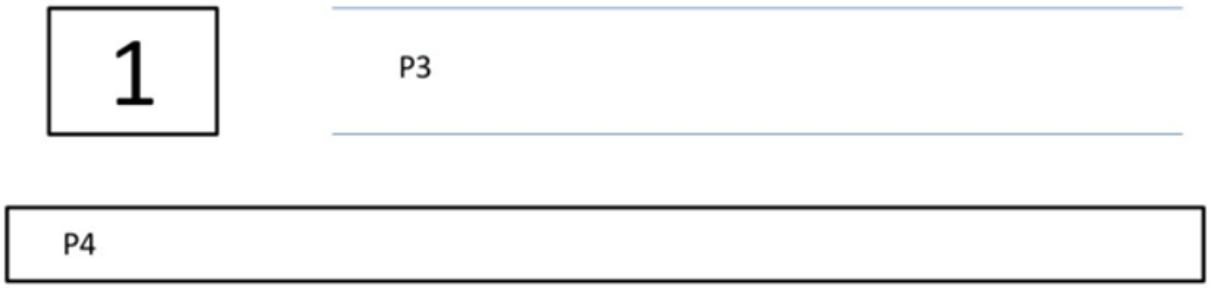
власною ініціативою). FCFS забезпечує мінімум накладних витрат. Середній втрачений час при застосуванні цього алгоритму не залежить від тривалості процесу, але штрафне відношення при однаковому втраченому часі буде великим для коротких процесів. Тому алгоритм FCFS вважається найкращим для довгих процесів. Істотним достоїнством цього алгоритму поряд з його простотою є та обставина, що FCFS гарантує відсутність нескінченного відкладання процесів: будь-який процес, який надійшов в систему, буде виконаний незалежно від ступеня її завантаженості.

Приклад виконання алгоритму FCFS для п’яти процесів:

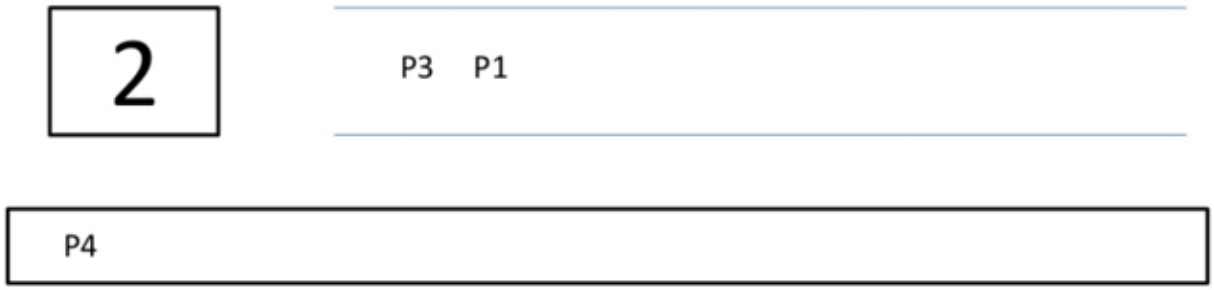
Процес	**Burst time**	**Час надходження (Arrival time)**
P1	6	2
P2	3	5
P3	8	1
P4	3	0
P5	4	4

Крок 0. Робота починається з процесу P4, чий час надходження дорівнює нулю.

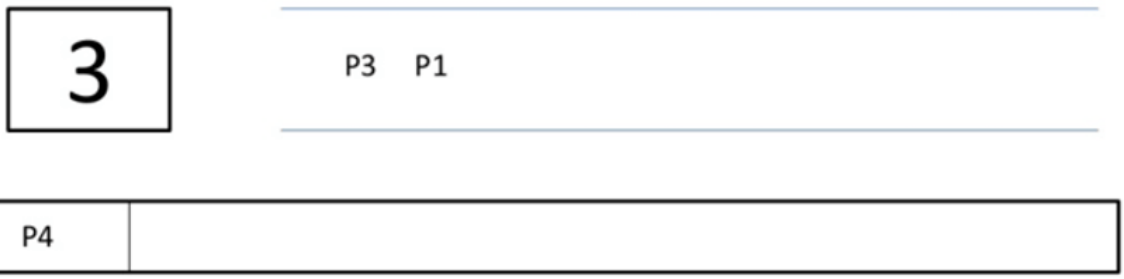
Крок 1. У момент часу time=1 надходить процес P3. Процес P4 ще виконується, тому P3 тримається в черзі.



Крок 2. У момент часу time= 2 надходить процес P1, який також зберігається в черзі.

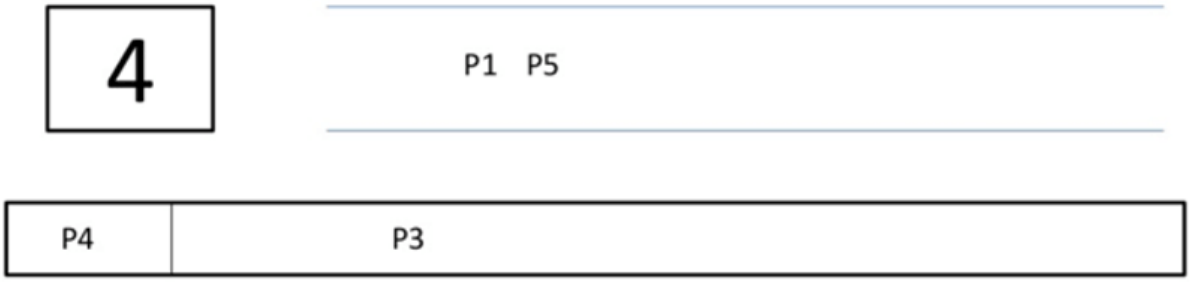


Крок 3. У момент часу time= 3 процес P4 завершує своє виконання.

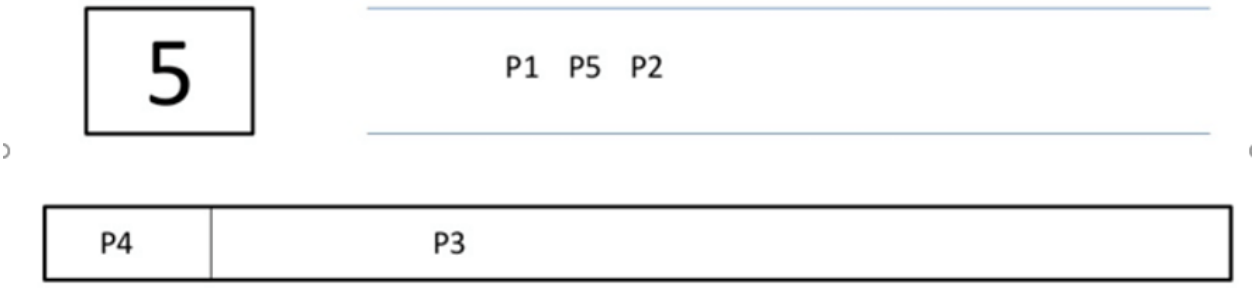


Крок 4. У момент часу time= 4 процес P3 (перший в черзі) починає виконуватись, а P5 надходить у чергу.

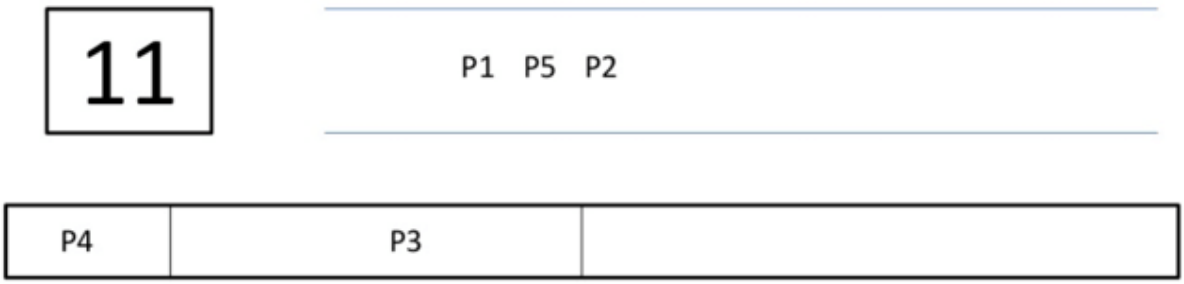
Процес	Тривалість (Burst time)	Час надходження (Arrival time)
P1	6	2
P2	3	5
P3	8	1
P4	3	0
P5	4	4



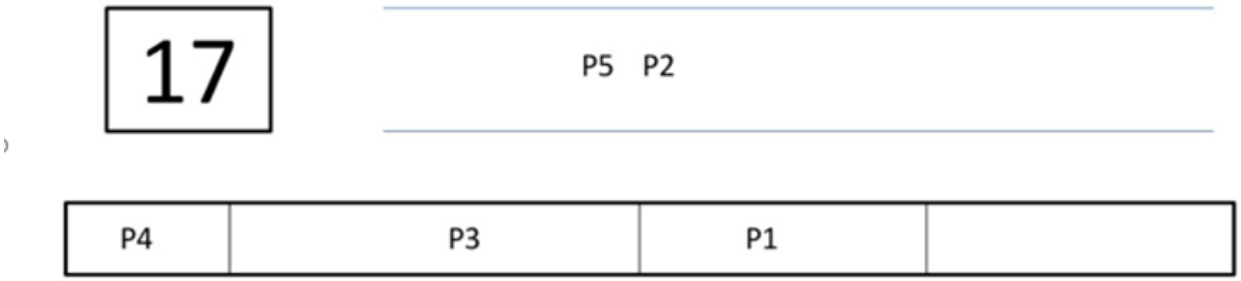
Крок 5. У момент часу time= 5 процес P2 надходить та поступає в чергу.



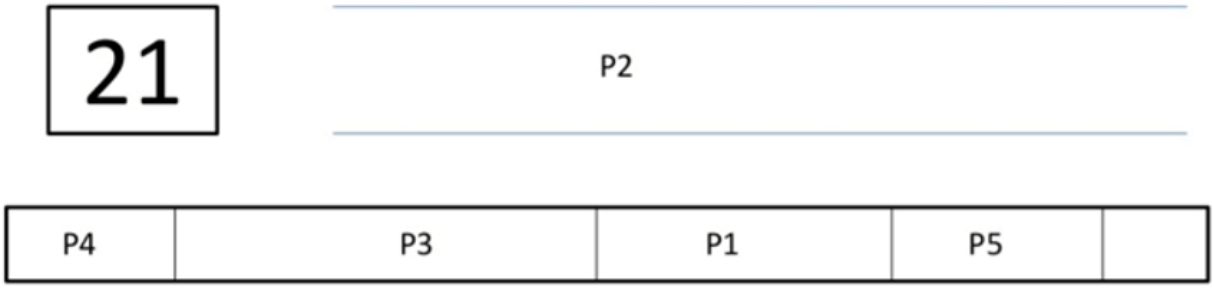
Крок 6. У момент часу time= 11 процес P3 завершує свою роботу.



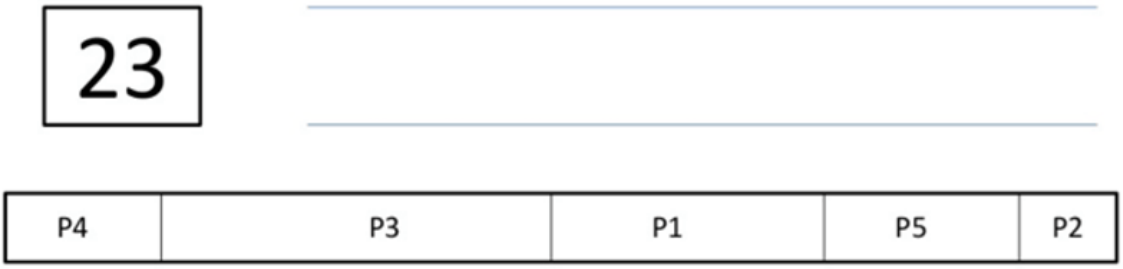
Крок 7. У момент часу time= 11 процес P1 починає виконання. Він має burst time = 6, тому завершує роботу в момент часу time=17.



Крок 8. У момент часу time= 17 процес P5 починає виконання. Тривалість його роботи дорівнює 4, тому процес завершується в момент часу time= 21.



Крок 9. У момент часу time= 21 процес P2 починає виконання та триває дві одиниці часу. Після цього всі процеси вважаються виконаними.



Тривалість очікування (waiting time) процесу є різницею між моментом запуску та моментом надходження процесу. Для представлених процесів вона обчислюється таким чином:

P4 = 0-0 = 0

P3 = 3-1 = 2

P1 = 11-2 = 9

P5= 17-4 = 13

P2= 21-5= 16

Середня тривалість очікування складає $\frac{0+2+9+13+16}{5} = 8$

2.2. Алгоритм RR

RR (round robin – карусель) – найпростіший алгоритм з витісненням. Процес отримує в своє розпорядження ЦП на деякий квант часу Q (в найпростішому випадку розмір кванта фіксований). Якщо за час Q процес не завершився, він витісняється з ЦП і направляється в кінець черги готових процесів, де чекає виділення йому наступного кванта, і т. д. Показники ефективності RR істотно залежать від вибору величини кванта Q. RR забезпечує найкращі показники, якщо тривалість більшості процесів наближається до розміру кванта, але не перевершує його. Тоді більшість процесів укладаються в один квант і не стають в чергу повторно. При $Q \to \infty$ RR вироджується в FCFS. При $Q \to 0$ накладні витрати на перемикання процесів зростають настільки, що поглинають весь ресурс ЦП. RR забезпечує найкращі показники справедливості: штрафне відношення P на великій ділянці тривалостей процесів t залишається практично сталим. Тільки на ділянці $t < Q$ штрафне відношення починає змінюватися і при зменшенні t від Q до 0 зростає експоненціально. Втрачений час M істотно зростає зі збільшенням тривалості процесу. Приклад роботи даного алгоритму можна знайти тут.

2.3. Алгоритм SJF

SJF (shortest job first - найкоротша робота – перша) – невитісняючий алгоритм, в якому найвищий пріоритет має найкоротший процес. Для того щоб застосовувати цей алгоритм, повинна бути відома тривалість процесу – задаватися користувачем або обчислюватися методом екстраполяції. Для коротких процесів SJN (shortest job next = SJF) забезпечує найкращі показники, ніж RR, як по втраченому часу, так і по штрафному відношенню. SJF забезпечує максимальну пропускну здатність системи – виконання максимальної кількості процесів за одиницю часу, але показники для довгих процесів значно гірші, а при високому ступені завантаження системи: активізація довгих процесів може відкладатися до нескінченності. Штрафне відношення слабко змінюється на основному інтервалі значень t, але значно зростає для найкоротших процесів: такий процес при вступі до систему має найвищий пріоритет, але змушений чекати, поки закінчиться поточний активний процес. Приклад роботи алгоритму наведено тут. За цим же посиланням присутній приклад для варіанту з наступного пункту 2.4.

2.4. Алгоритм PSJF

PSJF (preemptive SJF – SJF з витісненням) – поточний активний процес переривається, якщо його час, що залишився на виконання, більше, ніж у

новоприбулого процесу. Алгоритм забезпечує ще більше переважання коротких процесів над довгими. Зокрема, в ньому усувається те зростання штрафного відношення для найкоротших процесів, яке має місце в SJF.

2.5. Алгоритм RR SJF

Модифікація алгоритму RR з перевпорядкуванням процесів у черзі відповідно до часу виконання, що залишився.

2.6. Пріоритетне планування

Алгоритм SJF представляє собою окремий випадок пріоритетного планування. При пріоритетному плануванні кожному процесу присвоюється певне числове значення - пріоритет, відповідно до якого йому виділяється процесор. Процеси з однаковими пріоритетами плануються в порядку FCFS. Для алгоритму SJF в якості такого пріоритету виступає оцінка тривалості наступного CPU burst.

Чим менше значення цієї оцінки, тим вищий пріоритет має процес. Принципи призначення пріоритетів можуть спиратися як на внутрішні критерії обчислювальної системи, так і на зовнішні по відношенню до неї. Внутрішні використовують різні кількісні та якісні характеристики процесу для обчислення його пріоритету. Це можуть бути, наприклад, певні обмеження за часом використання процесора, вимоги до розміру пам'яті, число відкритих файлів і використовуваних пристроїв введення-виведення, ставлення середніх тривалостей I/O burst до CPU burst і т. д. Зовнішні критерії виходять з таких параметрів, як важливість процесу для досягнення будь-яких цілей, вартість сплаченого процесорного часу і інших політик.

Пріоритетне планування може бути як витісняючим, так і невитісняючим. При витісняє плануванні процес з більш високим пріоритетом, що з'явився в черзі готових процесів, витісняє що виконується процес з більш низьким пріоритетом. У разі невитісняючого планування він просто стає в початок черги готових процесів. Приклад невитісняючого пріоритетного планування можна знайти тут.

2.7. Алгоритм HPRN

HPRN (highest penalty ratio next – з найбільшим штрафним відношенням – наступний) - алгоритм без витіснення, що забезпечує найкращі показники справедливості. Це досягається за рахунок динамічного перевизначення пріоритетів. Всякий раз при вивільненні ЦП для всіх готових процесів обчислюється поточне штрафне відношення

$$p[i] = (w[i] + t[i]) / t[i]$$

де i – номер процесу; w[i] – час, витрачений процесом на очікування; t[i] – тривалість процесу (спрогнозована або заздалегідь визначена). Для щойно надійшовшого процесу p[i] = 1 ЦП надається процесу, що має найбільше значення p[i]. Для коротких процесів HPRN забезпечує приблизно ті ж показники справедливості, що й SJF, для довгих – ближчі до FCFS. На великому діапазоні середніх тривалостей процесів показники, що забезпечуються HPRN, представляють середнє між SJN і FCFS і слабо залежать від тривалості. Ще одна перевага HPRN в тому, що в часі очікування може враховуватися (з деякими ваговими коефіцієнтами) й очікування в інших чергах і, таким чином, виконується більш повний облік завантаження системи.

Істотним недоліком методу є необхідність переобчислення штрафного відношення для всіх процесів при кожному перемиканні, що погано узгоджується із загальною політикою мінімізації накладних витрат в дисциплінах без витіснення.

2.8. Алгоритм SRR

SRR (selfish RR – егоїстичний RR) – метод з витісненням, що дає додаткові переваги виконуваних процесів, що дозволяє підвищити пропускну здатність. Всі процеси поділяються на дві категорії: нові і вибрані. Новими вважаються ті процеси, які не отримали ще жодного кванта часу ЦП, всі інші процеси - вибрані. При надходженні в систему кожному процесу дається певний пріоритет P0, однаковий для всіх процесів, який в подальшому зростає. В кінці кожного кванта часу перераховуються пріоритети всіх процесів, причому пріоритети нових процесів зростають на величину dA, а обраних – на величину dB. ЦП віддається процесу з найвищим пріоритетом, а за однакової кількості пріоритетів – тому, який раніше поставлений в чергу. Показники алгоритму істотно залежать від обраного співвідношення між dA і dB. При dB/dA = 0 алгоритм вироджується в звичайний RR, при dB/dA>= 1 – в FCFS. Власне алгоритм SRR забезпечується в діапазоні значень 0<dB/dA<1. Приклад роботи такого алгоритму можна розглянути тут.

2.9. Лотерейне планування

Процесам роздають "лотерейні квитки" на доступ до ресурсів. Планувальник може вибрати будь-який квиток, випадковим чином. Чим більше квитків у процесу, тим вищий у нього пріоритет. Розподіл квантів часу зазвичай здійснюють за алгоритмом RR з пріоритетами.

Розглянемо приклад для випадку з двома процесами A (60 квитків, номери від 1 до 60) та B (40 квитків, номери від 61 до 100). Планувальник випадковим чином обирає число від 1 до 100. Якщо обрано номер від 1 до 60,

тоді виконується процес А, інакше – В. Наприклад, для 10 квитків, обраних планувальником:

```
Номер квитка: 73 82 23 45 32 87 49 39 12 09.
Обраний потік: B  B  A  A  A  B  A  A  A  A.
```

Зауважте, що потік А в прикладі виконався 7 з 10 разів, а потік В – 3, що не зовсім відповідає початковому розподілу квитків (60 на 40). Для досягнення відповідно потрібно більше разів обирати квитки.

2.10. Алгоритм HLRR

HLRR ("half-life round-robin"). Алгоритм напіврозпаду є модифікацією алгоритму RR. З кожним і-м процесом пов'язано деякий пріоритетне число P[i]. Чим воно менше, тим вище пріоритет процесу.

Кожен новий процес отримує деяке початкове значення пріоритетного числа P0, однакове для всіх процесів. Крім того, з кожним процесом пов'язаний лічильник процесорного часу U[i] з початковим значенням 0. Процес з найменшим значенням P[i] отримує квант часу Q (за однакової кількості пріоритетних чисел ЦП віддається процесу, який ще довше). За час кванта інтервальний таймер видає кілька сигналів-переривань з інтервалом dT. За кожним таким перериванням лічильник U[i] активного (тільки активного!) процесу збільшується на 1.

Використання ЦП процесом закінчується при закінченні кванта або при переході процесу в очікування. При цьому модифікуються лічильники процесорного часу всіх (в тому числі і неактивних) процесів:

$$U[i] = U[i]/2$$

і для всіх процесів переобчислюють пріоритетні числа:

$$P[i] = P0 + U[i]/2.$$

і модифікується черга виконання процесів.

2.11. Багаторівневі черги. Алгоритм FB

Для систем, у яких процеси можуть бути легко розсортованими в різні групи, було розроблено інший клас алгоритмів планування. Для кожної групи процесів створюється своя черга готовності процесів. Для цих черг приписуються фіксовані пріоритети. Наприклад, пріоритет черги системних процесів встановлюється вищим за пріоритет черги користувацьких процесів. Це значить, що жодний користувацький процес не буде обрано для виконання, поки є хоча б один готовий до виконання системний процес. Всередині цих черг для планування можуть застосовуватись найрізноманітніші алгоритми. Наприклад, для великих обчислювальних процесів, які не потребують взаємодії з користувачем (фонових процесів), може використовуватись

алгоритм FCFS, а для інтерактивних процесів – алгоритм RR. Такий підхід отримав назву багаторівневих черг (Multilevel Queue). Він підвищує гнучкість планування: для процесів з різними характеристиками застосовується найбільш доречний алгоритм.

Алгоритм FB (foreground-background – передній-задній плани) передбачає, що черга готових процесів розбивається на дві підчерги – передньопланову та фонову. Черги обслуговуються за алгоритмом RR, проте передньопланова черга має абсолютний пріоритет: поки в ній є процеси, фонова черга не обслуговується. Новий процес спрямовується в передньопланову чергу. Якщо процес використав встановлену кількість N квантів у передньоплановій черзі, проте не завершився, він переходить у фонову чергу.

2.12. Багаторівневі черги зі зворотним зв'язком (Multilevel Feedback Queue)

Подальшим розвитком алгоритму багаторівневих черг є додавання до нього механізму зворотного зв'язку. Так процес не постійно приписаний до певної черги, а може мігрувати від черги до черги залежно від своєї поведінки.

Розглянемо спрощену ситуацію, коли процеси в стані готовності організовані в 4 черги. Планування процесів між чергами здійснюється на основі витісняючого пріоритетного механізму. Чим вище розташована черга, тим вище її пріоритет. Процеси в черзі 1 не можуть виконуватись, якщо в черзі 0 є принаймні один процес і т. д. Якщо під час роботи процесу з'являється інший процес у якій-небудь більш пріоритетній черзі, тоді процес, який виконується, витісняється цим новим процесом. Планування процесів всередині черг 0-2 здійснюється за допомогою алгоритму RR, а планування процесів у черзі 3 базується на алгоритмі FCFS.

Новий процес надходить у чергу 0. При виборі для виконання він отримує квант часу Q одиниць. Якщо тривалість його CPU burst менша за цей квант часу, процес залишається в черзі 0. Інакше він переходить у чергу 1. Для процесів з черги 1 квант часу має величину 2Q. Якщо процес не вкладається у відведений час, він переходить у чергу 2, інакше – залишається в черзі 1. Аналогічно черга 2 має квант часу 4Q, а для третьої черги квантування часу не застосовується. За умови відсутності готових процесів у інших чергах процес з черги 3 може виконуватись до завершення його CPU burst. Чим триваліший процес, тим у менш пріоритетну чергу він надходить, проте може розраховувати на більший період виділеного процесорного часу. Таким чином, через деякий час всі процеси з малою тривалістю роботи будуть

розміщеними у високопріоритетних чергах, а всі процеси з великою кількістю обчислень та низькою інтерактивністю – в низькопріоритетних.

3. Практичні завдання

Зімітуйте планування процесів відповідно до свого варіанту (остача від ділення свого номеру в списку підгрупи + 1):

№	Алгоритм планування
1.	SJF з пріоритетами, лотерейне планування, FCFS
2.	RR з пріоритетами, FB, PSJF
3.	RR з пріоритетами , HLRR, FCFS
4.	HPRN, HLRR, PSJF
5.	HPRN, невитісняюче пріоритетне планування, SJF
6.	HPRN, FB, RR
7.	SRR, FB, FCFS
8.	SRR, MFQ, SJF
9.	SRR, MFQ, RR
10.	Лотерейне планування, MFQ, FB

Усі параметри планування задаються користувачем. Реалізуйте імітацію появи процесів у системі випадковим чином. Для кожного процесу генерується час його появи в системі, кількість квантів, необхідних для роботи процесу та, за потреби, пріоритет.

Користувач задає параметри генератора процесів: кількість працюючих процесів, обмеження на максимальний час виконання кожного процесу та, за потреби, кількість допустимих пріоритетів. Для отриманого графіку появи процесів у системі застосувати алгоритм планування.

Для кожного процесу з тривалістю виконання t обчислювати:

- T – загальну тривалість перебування процесу в системі;
- Втрачений час $M = T - t$;
- Відношення реактивності $R = t/T$;
- Штрафне відношення $P = T/t$;

Обчисліть середні значення наведених вище параметрів при використанні алгоритму планування процесів. Відобразіть на екрані результати планування і виконання процесів у вигляді схеми, графіка або таблиці. Виконайте тестування роботи програми для ***трьох*** різних наборів вхідних параметрів і ***трьох*** різних графіків появи процесів в системі.