РОЗДІЛ 2

ПРОЦЕСИ. ПОТОКИ. СИГНАЛИ[1]

2.1 Основні відомості про процеси. Управління процессами

Для розуміння структури операційних систем застосовують концепцію процесів. Цей термін уперше був застосований у 60-х роках минулого століття розробниками операційної системи Multics. Існує декілька визначень терміну «процес», наприклад

програма, що виконується;

екземпляр програми, що виконується на комп'ютері;

об'єкт, який можливо ідентифікувати і виконувати на процесорі;

одиниця активності, яка характеризується єдиним ланцюжком послідовних дій, поточним станом та зв'заним з ним набором системних ресурсів.

2.1.1 Властивості та класифікація процесів

Більшість сучасних операційних систем є мультипрограмними.[2] Переваги мультипрограмного режиму:

збільшення пропускної спроможності обчислювальної системи завдяки більш оптимальному завантаженню пристроїв;

можливість поєднувати тривалі операції (наприклад, вивід на принтер) з іншою корисною роботою;

нові засоби декомпозиції при розробці складних систем, що обробляють різні асинхронні події.

Оскільки мультипрограмність передбачає виконання декількох програм одночасно, то виникає необхідність керувати ними, розподіляючи між ними ресурси під час виконання і, як наслідок виникає таке поняття, як процес.

---

[1] При підготовці даного розділу були використані літературні джерела [1–4], [15–18], [20–22] та інші із списку літератури.

[2] Мультипрограмування – це такий спосіб організації обчислень, коли на однопроцесорній обчислювальній системі імітується одночасне виконання декількох програм.

Процес – це окрема програма в момент її виконання, а також виділені їй ресурси комп'ютера.

обчислювальних системах завжди виникають причини для створення нових процесів.

Ініціалізація системи. При запуску ОС відразу створюється кілька процесів, що забезпечують взаємодію з користувачем. Інші процеси є фоновими. Їх ще називають демонами (Disk And Execution MONitor).

Створення процесу іншим процесом, що працює. Створений процес дочірнім, а процес, що його створив, – батьківським.

Запит користувача на створення процесу. Процес завжди створюється однаково за допомогою особливого системного виклику для створення нового процесу.

В обчислювальних системах завжди є ситуації при яких процес може бути завершеним:

звичайне або успішне завершення процесу;

перевищення ліміту часу, що відведений для програми;

недостатній об'єм пам'яті;

зайве очікування (процес очікує настання певної події довше ніж задано в параметрах системи);

вихід із процесу при виникненні помилки, яку неможливо усунути;

поступила помилкова команда, або з недосяжними привілеями;

знищення процесу іншим процесом.

При виконанні програм процесором обчислювальної системи визначають наступні стани процесора:

Passive – процес завантаження в пам'ять, але його виконання ще неініційоване.

Ready – процес готовий до виконання (очікує ресурс центрального процесора).

Run – процес виконується на центральному процесорі.

Wait – процес очікує звільнення необхідного ресурсу.

В залежності від типу очікуваного ресурсу розрізняють системи:

з обмеженням по введення/виведення (в таких системах найбільший час очікування витрачається процесом при зверненні до підсистем введення/виведення);

з обмеженням по швидкодії (найбільший час витрачається на доступ до процесора).

Будь – який процес може знаходитись в одному з станів один раз в певний момент часу (рисунок 2.1).

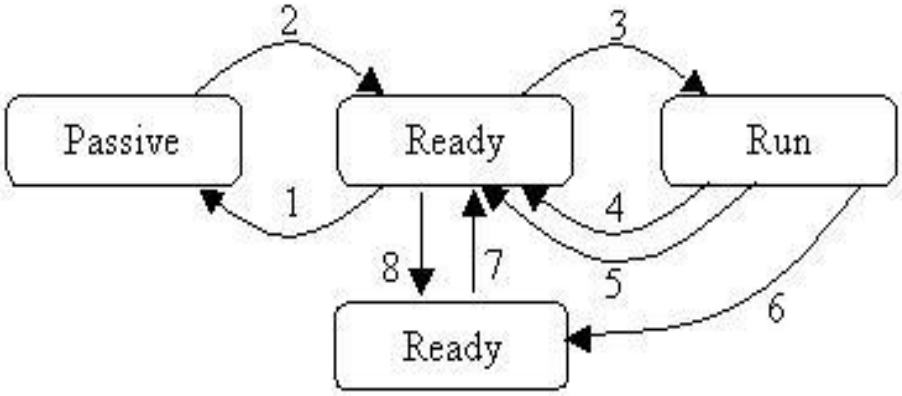


Рисунок 2.1 – Стани процесів

Визначають наступні умови зміни станів процесу:

2 – активізація процесу програмою – диспетчером або ОС;

3 – диспетчеризація[1] процесу;

4 – виникнення переривань;

5 – запит процесом певної системної функції в складі ОС (можливе створення певного процесу);

6 – запит процесом певної системної функції (вводу/виводу);

– завершення виконання системної функції;

– зупинка виконання процесу;

1 – завершення виконання процесу.

2.1.2 Моделі процесу

Модель з двома станами. Основним завданням ОС є управління виконанням процесів (визначення схеми чергування процесів та виділення їм ресурсів).

найпростішій моделі управління процесами в будь-який момент часу процес або виконується, або не виконується. На рисунку 2.2-а представлена

[1] Поняття диспетчеризації буде розглянуто в підрозділі «Планування і диспетчеризація процесів».

модель, процесу з двома станами, а на рисунку 2.2-б – діаграма використання черги.

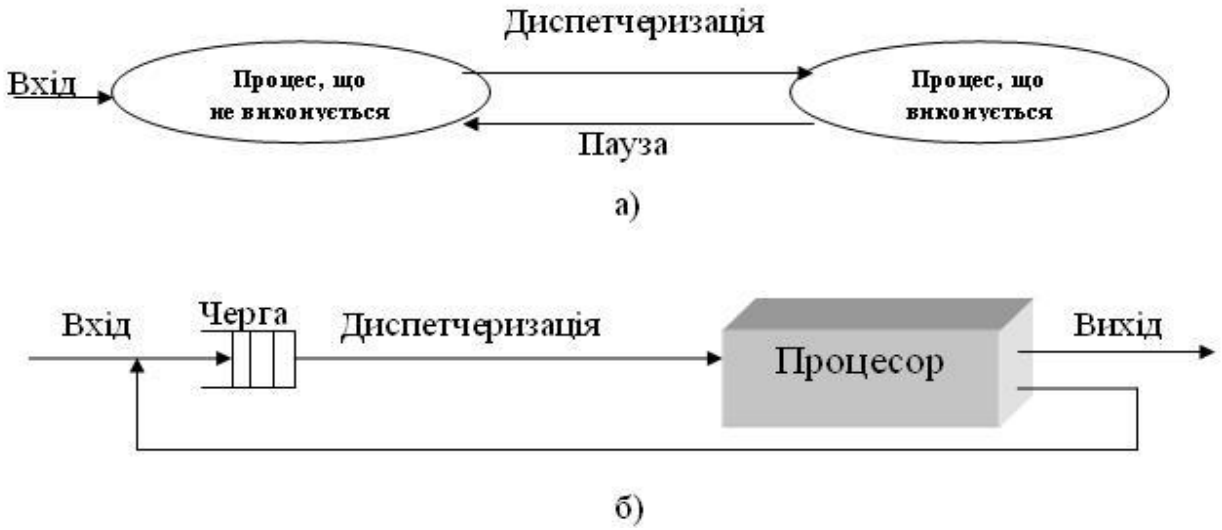


Рисунок 2.2 – Модель процесу з двома станами:
а) діаграма станів; б) використання черги

Створивши новий процес, ОС вводить його в систему в стані, що не виконується. Після цього створений процес чекає, коли він зможе бути запущений. Час від часу процеси, що виконуються будуть перериватися, а диспетчер буде вибирати інший процес для виконання .

Процеси, що не виконуються організовані в чергу (*пул*), де вони чекають свого виконання. Елементи такої черги - покажчики на процеси. Процес, робота якого була перервана, переходить в чергу процесів, що очікують виконання.

Якщо процес завершено, то він виводиться з системи. Далі для виконання диспетчер вибирає з черги наступний процес.

Модель з п'ятьма станами. В моделі з двома станами можливі дві некоректні ситуації, коли процес не виконується:

ОС вирішила на деякий час надати процесор іншому процесу;

процес блокований, так як з точки зору своєї внутрішньої логіки він не може виконуватися далі (наприклад, очікує вхідні дані).

На рисунок 2.3-а, представлені три стани процесу: 1) як такий, що виконується (використовує процесор); 2) готовий до виконання (тимчасово призупинений, щоб дозволити виконуватися іншому процесу); 3) блокований (процес не може бути запущений, поки не відбудеться деяка зовнішня подія). Тут також добавлено ще два стани: новий (тільки що створений процес, який

ще не поміщений операційною системою в чергу процесів, що виконуються і ще не завантажений в ОП) та завершений (процес, вже вилучений з пулу виконаних процесів).

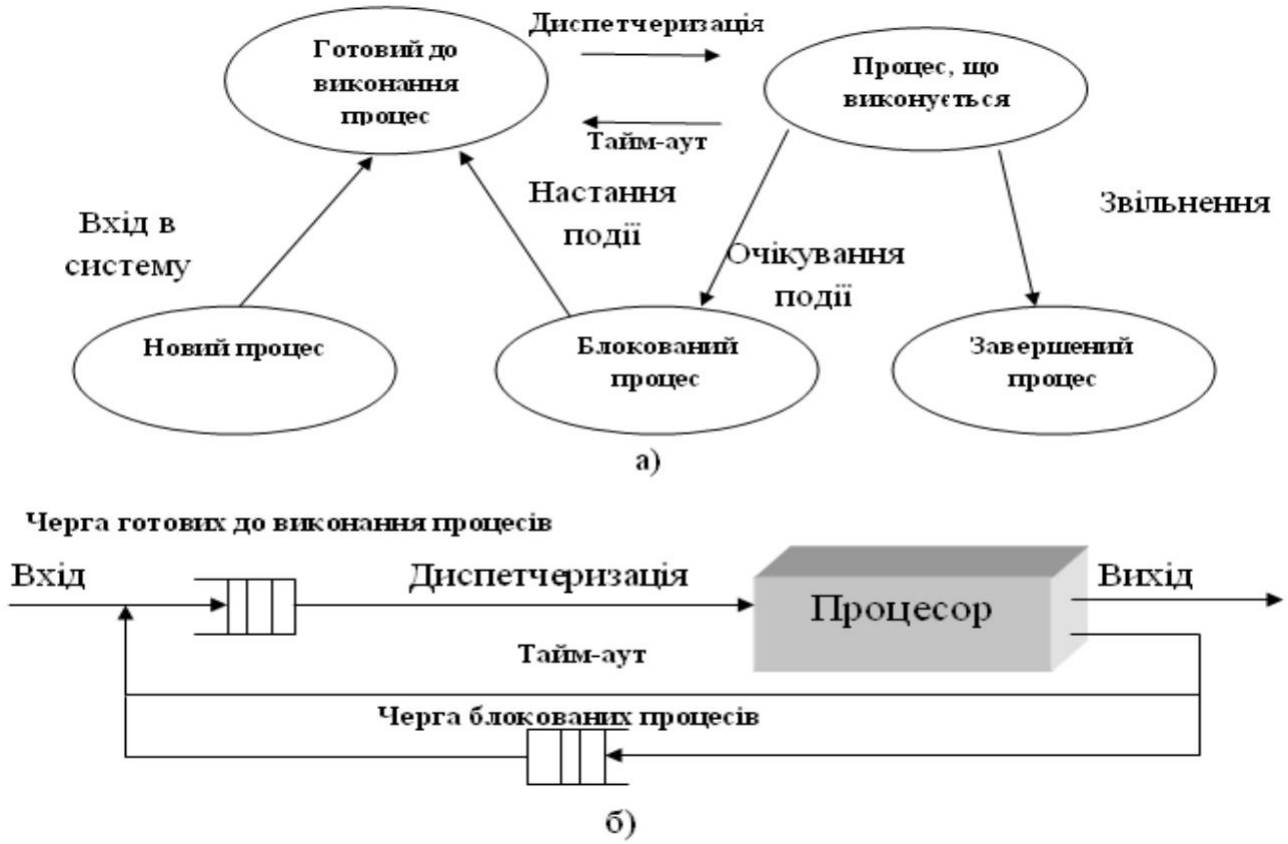


Рисунок 2.3 - Модель з п'ятьма станами:
а) діаграма станів; б) використання черги

Виходячи з діаграми станів, можливі наступні переходи між станами процесу:

нульовий стан → новий – причини переходу між цими станами збігаються з причинами створення процесів;

новий → готовий до виконання – цей перехід відбувається, коли ОС готова до обробки додаткового процесу, так як у більшості ОС є обмеження на кількість процесів або на обсяг виділеної для процесів віртуальної пам'яті;

готовий → виконується – цей перехід відбувається в момент вибору нового процесу для запуску;

виконується → завершений – причини переходу між цими станами збігаються з причинами завершення процесів;

виконується → готовий до виконання – цей перехід відбувається найчастіше після закінчення кванта часу, виділеного процесу. У деяких ОС може відбутися витіснення процесу і до закінчення цього кванта часу (наприклад, при появі процесу з більш високим пріоритетом). Витіснення (preemtion) - це ситуація, коли процес позбавляється деякого ресурсу до того, як він завершить роботу з ним;

виконується → блокований – цей перехід виконується, якщо для продовження виконання процесу потрібно дочекатися настання деякої події (звільнення занятого ресурсу, закінчення операції ввід-вивід, повідомлення від іншого процесу і т.д.);

блокований → готовий до виконання – цей перехід відбувається, коли настає очікувана процесом подія.

На рисунку 2.3-б показаний один із способів реалізації почергового виконання процесів. Маємо дві черги: 1 – для готових до виконання і 2 – для заблокованих процесів.

Кожен процес, який надходить в систему для обробки, поміщується в чергу готових до виконання процесів, і ОС вибирає з неї для виконання черговий процес. Якщо схема пріоритетів відсутня, то така черга має тип FIFO («First Input First Output» – «першим увійшов-першим вийшов» – «класичний» варіант черги, послідовність обробки елементів якої на виході та ж, що і послідовність поміщення цих елементів в чергу). Коли виконання процесу переривається, він, в залежності від обставин, може або завершитися, або потрапити в одну з двох згаданих вище черг. Якщо ж станеться якась подія, то всі процеси, що очікують його переміщаються з черги блокованих в чергу готових процесів.

Недоліком такої схеми є те, що після будь-якої події ОС повинна переглядати всю чергу блокованих процесів, відшукуючи серед них ті, які очікують саме цієї події. Тому ефективніше організувати кілька черг – свою для кожної події. Тоді при настанні якої-небудь події всі процеси з відповідної черги можуть бути відразу переведені в чергу готових до виконання процесів.

Якщо ж диспетчеризація здійснюється з використанням пріоритетів[1], то можна організувати кілька черг готових процесів, по одній черзі на кожен рівень пріоритету. У цьому випадку потрібно також передбачити механізм контролю виконання процесів з відповідних черг відповідно до їх пріоритету.

---

[1] Поняття пріоритету процесу буде викладене в підрозділі 2.1.5.

2.1.3 Опис процесів

Для керування процесами ОС створює і підтримує таблиці процесів, які містять покажчики на образ кожного процесу.

Образ процесу містить програму (код), дані, стек і атрибути процесу. Структура даних разом з атрибутами процесу часто називається його ***керуючим блоком*** (рисунку 2.4)**.**

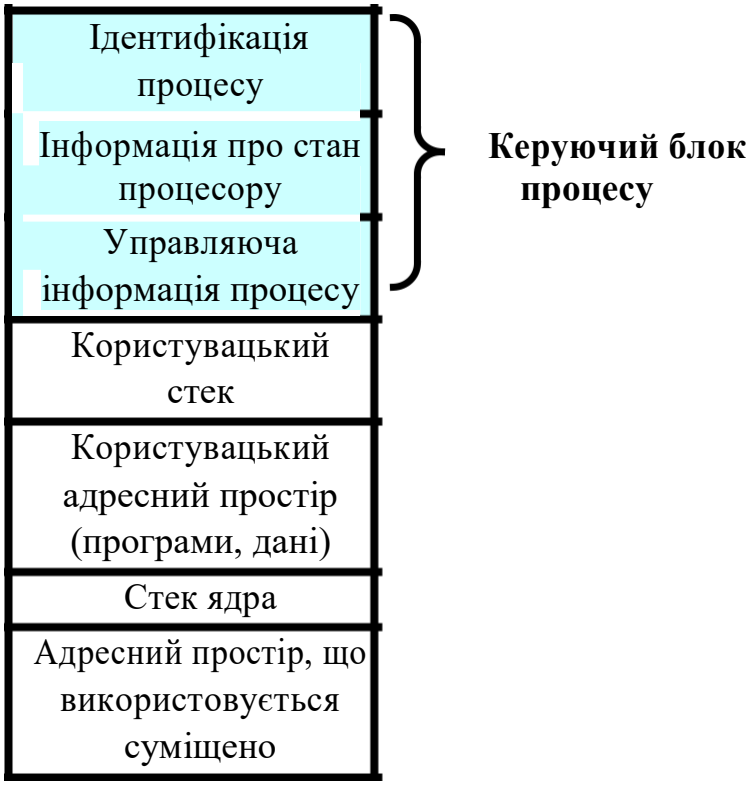


Рисунок 2.4 – Образ процесу

Атрибути процесу включають в себе наступне:

ідентифікатори (ідентифікатор даного процесу, ідентифікатор батьківського процесу, ідентифікатор користувача);

інформацію про стан процесу (вміст регістрів загального призначення, керуючих регістрів і регістрів стану, покажчики на стек);

керуючу інформацію про процес (стан процесу – виконується, готовий до виконання, блокований і т.д.; пріоритет; інформацію, пов'язану з плануванням; інформацію про події; інформацію про зв'язки з іншими процесами (в чергах, родинні зв'язки); відомості про обмін інформацією між процесами (різні прапори, сигнали, повідомлення); привілеї процесу; інформацію про володіння ресурсами та їх використання).

Процеси можна класифікувати наступним чином:

За приналежністю до центрального процесору (ЦП):

внутрішні (в операційних системах прийнято розрізняти не тільки час існування процесу, але й час його народження. Такою точкою відліку прийнято вважати ЦП. Процеси виконані на ньому називаються програмним або внутрішніми);

зовнішні (це процеси, розвиток яких проходить під контролем не ЦП, а інших).

За приналежністю до ОС:

системні;

користувацькі.

За генеалогічним типом:

породжуючі (це процеси, які можуть породжувати існування інших процесів);

породжені (процеси, які починають існувати в результаті існування іншого процесу).

За результативністю:

еквівалентні (два процеси, які мають один і той же результат обробки даних в одній і тій же програмі на одному і тому ж, або на різних процесорах);

тотожні (якщо в кожному з еквівалентних процесів обробка даних проходить в одній і тій же програмі, але траси не співпадають);

рівні (при співпаданні слідів тотожних процесів).

За динамічним принципом:

- *послідовні* (якщо інтервали двох процесів не пересікаються в часі);
- *паралельні* (якщо на певному інтервалі часу існують одночасно два процеси);

За зв’язаністю:

взаємозв’язані (якщо між ними створюються зв’язки за допомогою системи управління процесів);

ізольовані;

інформаційно-незалежні (якщо два взаємозв’язані процеси використовують одні і ті ж ресурси, але не обмінюються між собою інформацією);

взаємодіючі (якщо між двома процесами є інформаційні зв’язки).

2.1.4 Управління процесами

Створення процесів. При створенні нового процесу ОС повинна виконати наступні дії:

присвоїти новому процесу унікальний ідентифікатор, при цьому в таблицю процесів вноситься новий запис;

виділити простір для процесу – для образу процесу і для його керуючого блоку;

ініціалізувати керуючий блок процесу;

встановити необхідні зв'язки – наприклад, включити новий процес в чергу готових процесів;

створити або розширити інші структури даних.

Перемикання процесів може відбутися в будь-який момент, коли управління передається операційній системі: у разі апаратного переривання, в тому числі переривання від таймера, спрацьовування пастки (виключення), при системному виклику. Всі перераховані причини реалізуються *через механізм переривань.*

Дії ОС при перемиканні процесів:

збереження контексту процесора (вмісту регістрів загального призначення і керуючих регістрів) в керуючому блоці процесу;

оновлення керуючого блоку виконується в даний момент процесу – потрібно змінити стан цього процесу і зберегти інформацію з обліку використання ним ресурсів;

переміщення керуючого блоку даного процесу у відповідну чергу (готових, заблокованих процесів та ін);

вибір необхідного процесу для виконання;

оновлення керуючого блоку обраного процесу - необхідно встановити для нього стан виконання;

оновлення структур даних по управлінню пам'яттю;

відновлення контексту процесора для вибраного процесу.

Вважається, що ОС теж є програмою, яка виконується процесором, тобто виникає питання: чи є ОС процесом? Якщо так, то має здійснюватися певним чином управління таким процесом. При розв’язанні цієї проблеми виникли різні підходи до розробки ОС:

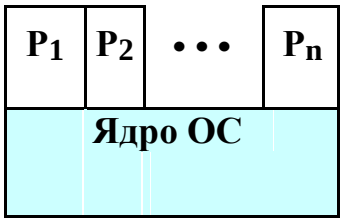
автономне ядро;

ядро в складі користувацьких процесів;

ОС на основі процесів.

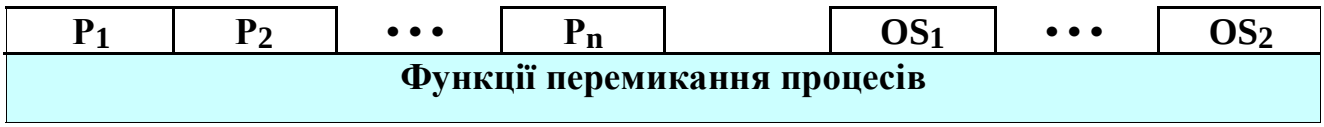
Ядро поза процесами (автономне ядро) – використовується в багатьох ранніх ОС. Концепція процесу тут застосовується тільки до користувацьких програмам, тоді як код ОС виконується як окремий об'єкт, що працює в привілейованому режимі (рисунок 2.5-а).

Ядро в складі користувацьких процесів – використовується в невеликих ЕОМ (в ПК і робочих станціях). Майже всі програми такої ОС виконуються в контексті користувача процесу (рисунок 2.5-б).



а) Автономне ядро

б) Функції операційної системи, що виконуються в користувацьких системах



в) Функції операційної системи, що виконуються як окремі процеси.

Рисунок 2.5 – Виконання коду ОС по відношенню до процесів: а) ядро поза процесами; б) ядро в складі користувацьких процесів; в) ОС на основі процесів

При виклику системних процедур, що працюють в режимі ядра, і поверненні з них використовується окремий стек ядра. Код та дані ОС знаходяться в спільно використовуваному адресному просторі і доступні для використання всіма користувацькими процесами. Тому при перериванні, включаючи системні виклики, перемикання процесів не відбувається –

перемикається тільки режим роботи процесора (користувацький ←→ ядро) в рамках одного і того ж процесу. При цьому в рамках одного і того ж процесу можуть виконуватися і користувацькі програми, і програми ОС. Програми ОС, виконувані в різних користувацьких процесах, є ідентичними.

ОС на основі процесів. На рисунку 2.5-в показана структура ОС на основі процесів.

Переваги такого підходу:

він сприяє розробці модульних ОС з простими міжмодульними інтерфейсами;

деякі другорядні функції ОС зручно реалізувати у вигляді окремих процесів; ці функції може викликати тільки ОС, і такі процеси можуть чергуватися з користувацькими;

в багатопроцесорних і багатокомп'ютерних обчислювальних системах окремі служби ОС можуть виконуватися на різних процесорах, що, відповідно, підвищує продуктивність системи.

2.1.5 Стратегії планування і диспетчеризація процесів

Планування – це системний процес, який здійснює установку користувацьких процесів в чергу та визначення атрибутів їх виконання в рамках використовуваної обчислювальної системи.

загальному випадку стратегія планування визначає які процеси повинні бути заплановані на виконання з метою досягнення результатів вирішуваної задачі.

Стратегії можуть бути наступними:

• закінчувати обчислення в тому самому порядку, в якому вони були розпочаті;

• перевагу надавати більш коротким процесам;

• надавати всім процесам однакові послуги, в тому числі і однаковий час очікування.

Розрізняють три типи планування:

– *короткострокове планування* – **диспетчеризація –** рішення про додавання процесу в пул процесів, що виконуються;

середньострокове планування – рішення про додавання процесу до числа процесів, що повністю або частково розміщені в оперативній пам'яті;

довгострокове планування – рішення про те, який з доступних процесів буде виконуватися процесором (полягає у виборі таких обчислювальних процесів, які менше всього б конкурували між собою в процесі досягнення мети обчислень).

Системний процес який здійснює планування може бути реалізований двома способами:

цілісний планувальник – це програмний модуль, який є частиною операційної системи, його робота ініціюється перериваннями і він виконується як окремий системний процес;

розподілений планувальник – програмний модуль планувальника або його частина записується в кожний процес при його завантаженні в оперативну пам'ять.

У загальному випадку планувальник здійснює наступні функції:

1) запис копії процесу в пам'ять обчислювальної системи;

2) аналіз атрибутів при виконанні обчислювального процесу та формування набору робочих атрибутів в залежності від характеристик обчислювальної системи;

3) запис процесу в чергу процесів у випадку, якщо є вільне місце в черзі, або перевід процесу в пасивний стан у випадку, якщо всі елементи черги зайняті; 4) періодичний перегляд черги процесів, запис або знищення процесів в черзі у відповідності з часом функціонування процесів.

Відома велика кількість правил (дисциплін диспетчеризації), у відповідності до яких формується список (черга) готових до виконання задач.

Розрізняють два великих класи дисциплін диспетчеризації: безпріоритетні і пріоритетні.

При безпріоритетній організації задачі або процеси вибираються згідно з певним, раніше установленим, порядком без врахування важливості цих процесів та необхідного часу обслуговування.

При реалізації пріоритетних дисциплін певним окремим задачам чи процесам надаються виключне право або набір прав для установки такого процесу в чергу.

На рисунку 2.6 представлена структура дисциплін диспетчеризації.

Далі опишемо принципи роботи деяких дисциплін диспетчеризації:

Дисципліна FIFO (first input first output) – є реалізацією безпріоритетної черги. Якщо в черзі звільняється елемент і він не останній, то всі після нього посуваються вперед і в останній вільний елемент черги записується ім'я нового процесу. Першим з черги вибирається елемент, який стоїть на початку черги.

Дисципліна LIFO (last input first output) – першим з черги вибирається елемент, який прийшов останнім.

Рисунок 2.6 – Структура диспетчеризації

Дисципліна FCFS (first come first server) – є подібна до FIFO. Ця дисципліна враховує перебування процесів в станах блокування, наприклад при операціях вводу/виводу. Ті, які були заблоковані в процесі роботи після переходу в стан готовності поступають в чергу готовності перед новими задачами, які ще не обслуговувались. При такій дисципліні організовуються дві черги: черга готових до обслуговування процесів і черга нових процесів (рисунок 2.7).

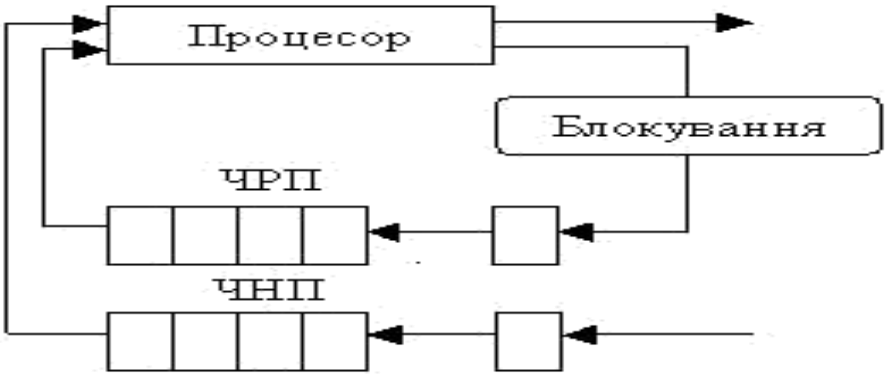


Рисунок 2.7 – Дисципліна FCFS (ЧНП – черга нових процесів; ЧРП – черга робочих процесів)

Ця дисципліна не потребує втручання ззовні в хід обчислювального процесу. Не відбувається розподіл процесорного часу.

Переваги: простота реалізації та невеликий розхід системних ресурсів для організації черги задачі. Однак при збільшенні завантаженості ОС, збільшується середній час обслуговування задачі, коли «короткі» завдання (які потребують невеликих затрат машинного часу) повинні очікувати такий самий час, що й довгі завдання.

Дисципліна SJN (short jump next) – згідно з цією дисципліною ОС вимагає, щоб для кожного процесу була відома оцінка необхідних обчислювальних ресурсів. Для постановки задачі в чергу диспетчер оцінює необхідний час виконання задачі і ставить задачу перед довшою задачею. При цій дисципліні існує одна черга процесів і завдання, що були заблоковані знову поступають в кінець черги як і нові завдання. Це приводить до того, що завдання які потребують мало часу очікують процесу як і довгі процеси.

Дисципліна SRT (short remain time) – розроблена з метою забезпечення більш якісного обслуговування коротких завдань. Виконує спочатку ті процеси, час завершення виконання яких найменший.

Дисципліна RR (round robin) – кожна задача отримує порцію часу (квант часу). Після закінчення цього кванту часу задача знімається з виконання на процесорі і він передається наступній задачі. Задача, яка була знята з черги записується в кінець черги задач, які готові до виконання. Величина кванту часу, як правило, вибирається, як середнє значення між достатнім часом реакції системи на запити користувачів та процесорним часом, який необхідний для перемикання між задачами.

2.1.6 Алгоритми в диспетчеризації з витісненням та без

Диспетчеризація без розподілу процесорного часу (*багатозадачність без витіснення*) – це такий спосіб диспетчеризації, при якому активний процес виконується до тих пір, поки він за власною ініціативою не передасть управління диспетчеру задач для вибору іншого, готового до виконання процесу. До них відносяться дисципліни: FCFS, SJN, SRT.

Диспетчеризація з розподілом процесорного часу (*багатозадачність з витісненням*) – це такий спосіб, де рішення про перемикання з однієї задачі на іншу приймається диспетчером задач, а не власною активною задачею. Процес, що виконується може бути перерваним і переведений операційною системою в стан готовності до виконання. Рішення про витіснення може прийматися при запуску нового процесу про перериванню або періодично, згідно переривань від таймеру. До таких дисциплін відносяться RR та інші, реалізовані на її основі.

Будь-які процеси мають бути гарантовано певним чином обслужені операційною системою. Гарантоване обслуговування може бути досягнуто трьома способами:

ОС виділяє мінімальну частину процесорного часу деякому класу процесів, у випадку, коли хоча б один з них готовий до виконання;

ОС виділяє мінімальну долю процесорного часу певному конкретному процесу якщо він готовий до виконання;

ОС виділяє стільки часу певному процесу, щоб він міг виконати своє обчислення в певний термін.

Для порівняння алгоритмів диспетчеризації можуть бути використані наступні критерії:

використання (завантаження) процесору – відсоток часу протягом якого процесор зайнятий;

пропускна здатність процесора – це кількість процесів, яка виконується процесором за одиницю часу;

час обороту – інтервал часу від моменту появи процесу у вхідній черзі до моменту завершення процесу. Цей час обороту включає в себе час очікування у вхідній черзі, час очікування у черзі готовності, час готовності у чергах до периферійних пристроїв, час виконання на процесорі та час вводу/виводу;

час очікування – це сумарний час знаходження процесу в черзі очікування готових процесів;

час відповіді – це час від моменту поступлення процесу на вхідну чергу до моменту першого звернення процесу на ввід/вивід даних.

Як відомо, при виконанні операційною системою багатьох процесів може відбуватися зниження продуктивності обчислювальної системи.

Існують певні методи підвищення продуктивності системи:

сумісне планування, при якому всі потоки однієї задачі одночасно вибираються для виконання процесором (у випадку наявності мультипроцесорної системи) і одночасно знімається з виконання. В цьому випадку зменшується час перемикання між задачами.

планування, при якому задачі, що знаходяться в критичній області не перериваються, а закінчують виконання критичної секції. Задачі, які чекають на виконання критичної секції коду не виконують її поки не завершиться переривання.

При виконанні програм, які реалізують функції контролю чи управління в системах реального часу може виникнути ситуація, коли одна або декілька задач не можуть бути вирішені протягом довгого проміжку часу через значну завантаженість обчислювальної системи. В цьому випадку втрати, які пов’язані невиконанням цих задач можуть бути більшими ніж втрати від невиконання задач з більш високим пріоритетом. Тому іноді виникає необхідність динамічної зміни пріоритету задачі в процесі її виконання. Це дозволяє реалізувати більш швидку реакцію на запити користувачів і гарантувати виконання будь-яких запитів.

Операційна система може змінити пріоритет задачі за допомогою:

Підвищення пріоритету активної задачі: при завантаженні задачі на виконання її пріоритет автоматично збільшується і при цьому знижується час реакції цієї активної задачі на дії користувачів в порівнянні з іншими фоновими задачами;

Підвищення пріоритету операції вводу/виводу: після завершення операції вводу/виводу задача отримує найвищий пріоритет в певному класі задач, що дозволяє більш швидко закінчити всі незавершені операції вводу/виводу;

Підвищення пріоритету «забутих» задач: якщо задача не отримує часу процесора протягом певного відрізку часу, то диспетчер задач тимчасово присвоює їй більш високий пріоритет, який не перевищує певної заданої межі, і таким чином можливо перемикатися на такі «забуті» задачі більш швидко. Після виконання такої задачі за певний квант часу її пріоритет знижується до попереднього значення.

Питання взаємоблокуваня винесено в окремий підрозділ 2.4. Взаємне блокування.