Потоки

Визначення процесу, яке було описане вище, базується на двох незалежних концепціях: на групуванні ресурсів і на виконанні програм. Якщо ж їх розділити, то з'явиться поняття *потоку*. Ресурсами керувати простіше, об'єднавши їх у формі процесу, однак, з іншого боку, процес можна розглядати як потік виконуваних команд, або просто потік.

Потік – це незалежно планований контекст виконання, що розділяє єдиний адресний простір з іншими потоками свого процесу.

Елементи, що спільно використовуються потоками, – адресний простір, глобальні змінні, відкриті файли, дочірні процеси, сигнали і їх обробники, інформація про використання ресурсів.

Елементи, що окремо використовуються потоками, – лічильник команд, регістри, стек, стан. Лічильник команд при цьому відстежує порядок виконання дій, регістри зберігають поточні значення змінних, стек містить протокол виконання процесу і тимчасові змінні.

Хоча потік виконується всередині процесу, концепції процесу і потоку різні (табл. «Порівняння процесів і потоків»). Концепція потоків, зокрема, додає до моделі процесу можливість одночасного виконання в одному і тому ж середовищі процесу декількох достатньою мірою незалежних команд.

Порівняння процесів і потоків

Можливість	**Процес**	**Потік**
Час для створення	Велике	Мале
Час для перемикання	Велике	Мале
Механізм взаємодії	Складний	Простий
Колективні дані	Немає	Є
Захист	Є	Немає
Положення	Той же самий або на різних комп'ютерах	Той же комп'ютер

Потоки володіють деякими властивостями процесів, тому їх іноді називають «спрощеними процесами» (lightweightprocesses).

Багатопоточність - виконання декількох потоків в одному процесі. При запуску багатопотокового процесу в системі з одним процесором потоки працюють по черзі. Ілюзія ж паралельної роботи потоків створюється шляхом постійного перемикання системи між потоками. І лише в багатопроцесорній або багатоядерній системі може бути реалізовано реальне паралельне виконання команд різних потоків.

Різні потоки в одному процесі не настільки незалежні, як різні процеси. всіх потоків – один адресний простір, що означає спільне використання глобальних змінних. Оскільки будь-який потік має доступ до будь-якої адреси комірки пам'яті в адресному просторі процесу, то один потік може змінювати інформацію в стеку іншого потоку. Захисту від цього не існує, оскільки, по-перше, це неможливо, а по-друге, це й не потрібно, так як потоки зазвичай виконують спільну задачу.

Якщо програма паралельно виконує багато різних дій і деякі з них можуть час від часу блокуватися, то використання декількох потоків дає наступні переваги:

спрощується сам програмний додаток;

полегшується спільне використання даних;

підвищується продуктивність (за наявності операцій введення/виведення);

знижується навантаження на підсистему керування пам'яттю;

з'являється можливість для повноцінного використання багатопроцесорних систем.

2.6.1 Основні операції з потоками та способи реалізації потоків

Створення нового потоку відбувається за допомогою функції *pthread_create* – в UNIX, *CreateThread* – в Windows. Основним параметром цих функцій є ім'я процедури, яку необхідно запустити в новому потоці.

Windows також передбачена функція *CreateRemoteThread*, що дозволяє створити потік в іншому процесі.

Завершити потік можна зсередини самого цього потоку функцією *pthread_exit* в UNIX або *ExitThread* – в Windows. Після цього потік зникає і планувальник його не використовує. Завершити потік можна також з іншого потоку за допомогою функції *pthread_cancel* – в UNIX або *TerminateThread* – в Windows.

Очікування потоком завершення іншого певного потоку виконується за допомогою функції *pthread_join* – в UNIX або *WaitForSingleObject* і *WaitForMultipleObject* -в Windows.

Потік може добровільно надати свою чергу іншому потоку за допомогою функцій *pthread_yield* – в UNIX або *SwitchToThread* і *SuspendThread* – в Windows.

Існують такі способи реалізації потоків:

- *У просторі користувача.*
- *У просторі ядра.*
- *Змішана реалізація.*

випадку реалізації потоків у просторі користувача пакет підтримки потоків цілком розміщується в просторі користувача. Ядро ОС при цьому нічого не знає про потоки і керує звичайними однопоточними процесами. Кожен процес тут має власну таблицю потоків. Коли потік збирається виконати дію, яка може призвести до локального блокування, він викликає процедуру з пакету підтримки потоків. Ця процедура зберігає стан потоку в таблиці потоків, потім шукає в таблиці потік, готовий до запуску, і завантажує в регістри його стан.

Реалізація потоків у просторі користувача забезпечує наступні переваги:

можна реалізувати цей механізм в ОС, що не підтримує потоки;

переключення потоків відбувається на порядок швидше, ніж в режимі ядра;

кожен процес може мати свій власний алгоритм планування;

вище масштабованість додатків.

Проблеми, що виникають при реалізації потоків у просторі користувача:

блокуючий системний виклик в одному з потоків зупинить всі потоки процесу;

при запуску одного потоку жоден інший потік не буде запущений, поки перший потік добровільно не звільнить процесор.

випадку реалізації потоків у просторі ядра таблиця потоків розташовується в ядрі, яке також містить звичайну таблицю процесів. Всі запити, які можуть блокувати потік, тут реалізуються як системні виклики. Коли один потік блокується, ядро ОС запускає інший потік з цього ж процесу або потік з іншого процесу. Недоліком такого підходу є збільшення часу перемикання потоків, оскільки всі операції з потоками реалізуються через системні виклики, а не за допомогою бібліотечних функцій.

випадку змішаної реалізації потоків ядро ОС «знає» тільки про потоки свого рівня і управляє ними. Деякі потоки ядра при цьому можуть містити по декілька потоків користувацького рівня, які управляються так само, як потоки в системі, що не підтримує багатопоточність. У даному підході поєднуються переваги перших двох методів.

2.6.2 Організація процесів і потоків в Linux

UNIX-системах для кожного компонента образу процесу виділяють окрему ділянку пам'яті. Заначимо,що образ процесу UNIX-системах містить такі компоненти:

керуючий блок процесу;

код програми, яку виконує процес;

стек процесу, де зберігаються тимчасові дані (параметри процедур, повернені значення, локальні змінні тощо);

глобальні дані, спільні для всього процесу.

Керуючий блок процесу в Linux відображається структурою даних *task_struct,* основні поля якої містять наступну інформацію:

ідентифікаційні дані (зокрема pid — ідентифікатор процесу);

стан процесу (виконання, очікування тощо);

покажчики на структури предка і нащадків;

час створення процесу та загальний час виконання (так звані таймери процесу);

стан процесора (вміст регістрів і лічильник інструкцій);

атрибути безпеки процесу (uid, gid, euid, egid).

Також *task_struct* має кілька полів спеціалізованого призначення, що необхідні для різних підсистем Linux, а саме

відомості для обробки сигналів;

інформація для планування процесів;

інформація про файли і каталоги, пов'язані із процесом;

структури даних для підсистеми керування пам'яттю.

Структура процесу містить інформацію, необхідну протягом усього часу існування процесу (зокрема, коли він вивантажений на диск). Це такі дані, як pid, ефективний uid, пріоритет, покажчик на структуру користувача. Структури процесу поєднуються в черзі процесів у системі. Структура користувача містить інформацію, необхідну тільки під час виконання процесу (стан процесора, uid, gid, поточний каталог, інформацію про відкриті файли).

Необхідно зазначити, що в операційній системі Linux новий прцес створюється шляхом копіювання атрибутів поточного процесу.

Новий процес може бути клонованим (cloned) і, при цьому такі ресурси, як файли, обробники сигналів і віртуальна пам’ять, використовуються сумісно. І, як наслідок, якщо два процеси використовують одну і туж віртуальну пам’ять, то вони функціонують як потоки в рамках одного і тогож процесу. Але для потоків структури даних окремо не задаються, і таким чином в операційній системі Linux потоки і процеси не розрізняють.

На рисунку 2.13 представлено модель процесів і потоків Linux.

Виконуваний (*executing*). Це стан, при якому поточний процес або виконується, або готовий до виконання.

Підлягає перериванню (interruptible). Це стан блокування, в якому процес очікує настання події, наприклад, завершення операції введення/виведення, звільнення ресурсу або сигналу від іншого процесу.

Не підлягає перериванню (uninterruptible). Це стан блокування, при якому процес в неперервному стані безпосередньо очікує виконання певної апаратної умови, тому він не сприймає ні яких сигналів.

Зупинений (stopped). Процес був зупинений і може бути продовжений тільки при відповідному впливі іншого процесу.

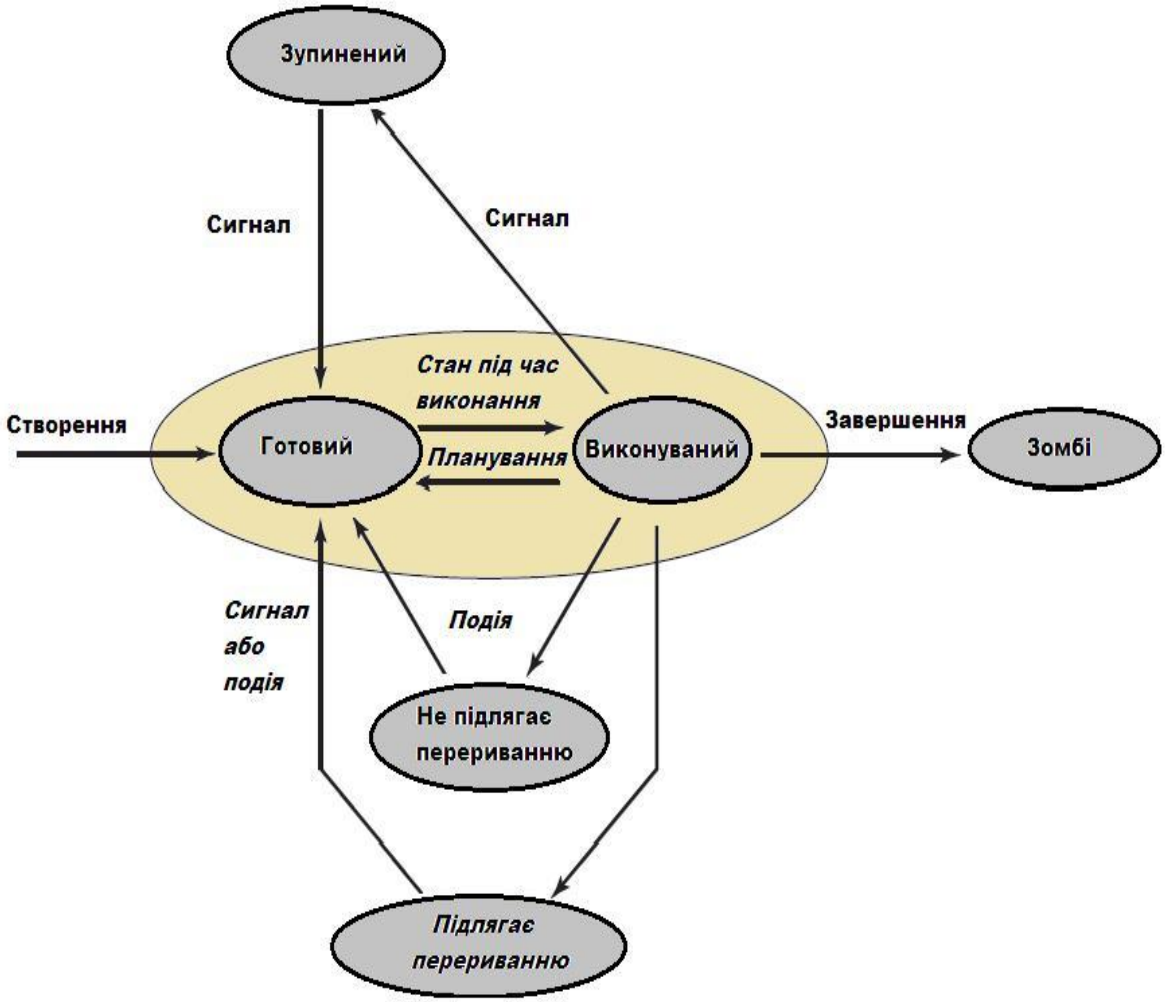

Рисунок 2.13 – Стани виконання процесу в Linux

Зомбі (*zombie*). Процес був завершений, але за якоюсь причиною його структура залишається в таблиці процесів.

2.7 Сигнали ОС UNIX

Сигнал в ОС UNIX - це асинхронне повідомлення процесу про будь-яку подію. Функції сигналів:

оповіщення процесів про виникнення системних подій;

забезпечення механізму для комунікації та синхронізації між прикладними процесами.

Сигнали дають можливість викликати спеціальну функцію – *обробник сигналу* при виникненні певної події. Після обробки сигналу таким обробником

основна програма продовжує роботу з того місця, в якому вона була перервана надходженням сигналу. Події позначаються цілими числами і представляються як символьні константи.

Процес оповіщення про подію складається з двох етапів – *генерації сигналу і обробки сигналу.* При настанні події, пов'язаної з будь-яким сигналом, ядро ОС генерує цей сигнал шляхом встановлення відповідного біта в масці сигналів, що очікують в керуючому блоці процесу. При цьому процес може отримати сигнал лише в певні моменти часу:

при виборці процесу диспетчером на виконання з черги готових;

перед блокуванням або після блокування;

під час деяких блокувань (наприклад, під час очікування введення з клавіатури).

Незалежно від джерела сигналу всі сигнали процесу доставляє ядро ОС.

Обробка сигналів. Кожен сигнал володіє дією, що встановлена за замовчуванням, яка виконується ядром, якщо процес не встановив для цього сигналу іншу дію.

Всього таких дій може бути п'ять:

аварійне завершення зі створенням дампа пам'яті (*Dump*);

аварійне завершення без створення дампа пам'яті (*Abort*);

ігнорування сигналу (*Ignore*);

призупинення процесу (*Stop*);

відновлення роботи призупиненого процесу (*Continue*).

Процес може перевизначити дію за замовчуванням для будь-якого сигналу (крім сигналів *SIGKILL* і *SIGSTOP)* на ігнорування цього сигналу або на свій обробник сигналу. Процес також може в будь-який час вказати нову дію або скинути установки на дію за замовчуванням. Процес може тимчасово блокувати сигнал, і цей сигнал не буде оброблено, доки він не буде розблокований (функція *sigprogmask*).

Щоб вказати спосіб, яким необхідно обробити сигнал, слід виконати наступні дії:

написати функцію обробника сигналу, включивши її в програму, для якої вона призначена; 2) зареєструвати цю функцію, викликавши на початку програми функцію *sigaction*. Виклики *sigaction* можна роити багаторазово, задаючи різні реакції на сигнал.

Нижче наведено приклад обробки сигналів при натисканні комбінації клавіш *Ctrl + C* (дією за замовчуванням для цього сигналу зазвичай є аварійне завершення програми):

```
include <signal.h>
.....
void handler (int sig);
{printf ("Отримано сигнал переривання | n");
}
main ()
{Struct sigaction act;
        act. sa__handler = handler;
        if (sigaction (SIGINT, & act, 0)! = 0) {perror
        (...); exit (1); }
            / * Продовження роботи * /
}
```

таблиці 2.1 «Сигнали, визначені в стандарті POSIX» наведено 20 сигналів, що визначені в стандарті POSIX[1].

Таблиця 2.1 Сигнали, визначені в стандарті POSIX

№	Позначення сигналу	Дія за замовчуванням	Опис
1	*SIGHUP*	*Abort*	Зависання терміналу / аварійне зупинення керуючого процесу
2	*SIGINT*	*Dump*	Переривання від клавіатури (Ctrl + C)
3	*SIGQUIT*	*Dump*	Сигнал виходу з терміналу (Ctrl + \)
4	*SIGILL*	*Dump*	Невірна команда
5	*SIGTRAP*	*Dump*	Пастка трасування / точки зупину
6	*SIGABRT*	*Dump*	Сигнал *Abort* від клавіатури
7	*SIGBUS*	*Dump*	Помилка шини
8	*SIGFPE*	*Dump*	Виключення з плаваючою точкою

Стандарт POSIX розроблено Інститутом інженерів з електротехніки і електроніки (IEEE), для того що б надати можливість писати програми, що працюють в будь-якій *UNIX*-системі. Стандарт POSIX визначає мінімальний інтерфейс системних викликів, який має підтримувати всі системи, що сумісні з *UNIX*. *Деякі ОС,* що відрізняються від *UNIX*, тепер теж можуть підтримувати стандарт POSIX.

№	Позначення сигналу	Дія за замовчуванням	Опис
9	*SIGKILL*	*Abort*	Сигнал *Kill*
10	*SIGUSR1*	*Abort*	Сигнал № 1, визначений користувачем
11	*SIGSEGV*	*Dump*	Невірна адреса пам'яті
12	*SIGUSR2*	*Abort*	Сигнал № 2, визначений користувачем
13	*SIGPIPE*	*Abort*	Запис в канал без його читання
14	*SIGALRM*	*Abort*	Сигнал таймеру від функції *alarm*
15	*SIGTERM*	*Abort*	Сигнал завершення
16	*SIGSTKFLT*	*Abort*	Помилка стека сопроцесора
17	*SIGCHLD*	*Ignore*	Нащадок зупинений або завершений
18	*SIGCONT*	*Continue*	Продовжено зупинений процес
19	*SIGSTOP*	*Stop*	Процес зупинено
20	*SIGTSTP*	*Stop*	Сигнал зупинки терміналу (Ctrl +Z)

Під час обробки сигналу інші сигнали того ж самого типу блокуються. Тому обробники сигналів можуть не бути реєнтерабельними[1], так як кілька копій обробника не можуть виконуватися одночасно. Однак обробник сигналу одного типу може перервати роботу оброблювача сигналу іншого типу.

Звичайні сигнали ядро не запам'ятовує в черзі. Якщо якийсь сигнал вже обробляється, то другий сигнал того ж типу, що надійшов в цей момент, просто губиться. У системах реального часу втрата сигналів є критичною. Тому багато версій UNIX (включаючи Linux) підтримують сигнали реального часу. Такі сигнали ядро поміщає в чергу, і вони не губляться – гарантується їх доставка в порядку FIFO.

Посилка сигнала процесу. Послати сигнал процесу можна за допомогою системного виклику kill:

```
   include <sys/types.h>
   include <signal.h>
int kill (pid_t pid, intsig);
```

Ядро Linux є реентерабельним. Це означає, що одночасно в режимі ядра може виконуватися код кількох процесів. В однопроцесорних системах процесор у конкретний момент виконує код тільки одного процесу, інші перебувають у стані очікування. У багатопроцесорних системах код різних процесів може виконуватися паралельно.

Тут pid – це одержувач сигналу, а sig - номер сигналу. Якщо pid> 0, то сигнал sig надсилається процесові з *PID* = pid.

Якщо *pid* = 0, то сигнал посилається всім процесам у групі, до якої належить поточний процес. Якщо pid = -1, то сигнал посилається всім процесам в системі, крім першого (init), починаючи від найбільших PID в таблиці процесів. Якщо ж pid <-1, то сигнал посилається всім процесам у групі процесів-pid.

Контрольні питання

Що таке процес?

Що таке мультипрограмування?

Назвіть причини для створення процесу в обчислювальних системах?

Які ситуації призводять до завершення процесу?

Які можливі переходи між станами процесів в моделі процесу з п’ятьма станами?

З яких елементів складається образ процесу?

Що входить до управляючого блоку процесу?

Що включають атрибути процесу?

Які дії виконує ОС при створенні процесу?

Опишіть ОС на основі ядра в складі користувацьких процесів?

Що таке планування процесів? Які є типи планування процесів?

В чому відмінність між цілісним і розподіленим планувальником? Які основні функції планувальника процесів?

Що таке безпріорітетна організація процесів? Які недоліки такої організації процесів?

За яким принципом відбувається робота дисципліни диспетчеризації FIFO?

За яким принципом відбувається робота дисципліни диспетчеризації LIFO?

За яким принципом відбувається робота дисципліни диспетчеризації FCFS?

За яким принципом відбувається робота дисципліни диспетчеризації SJN?

За яким принципом відбувається робота дисципліни диспетчеризації SRT?

За яким принципом відбувається робота дисципліни диспетчеризації RR?

Які переваги багатозадачності з витісненням на відміну від багатозадачності без витіснення?

Як може бути досягнуто гарантоване обслуговування процесів?

Які є види взаємодії процесів?

Які можливі проблеми при взаємодії процесів?

Який механізм гарантовано запобігає виникненню взаємних блокувань при взаємодії процесів?

Які апаратні способи досягнення взаємного виключення?

Опишіть алгоритм Деккера? Який його недолік?

Для чого застосовуються примітиви синхронізації? Які існують примітиви синхронізації?

Що таке семафор? У якому випадку семафор називається сильним?

Наведіть приклад взаємного блокування з повторно використовуваними ресурсами?

Дайте визначення безпечних та небезпечних станів взаємодії процесів.

Назвіть і опишіть класичні проблеми міжпроцесної взаємодії.

Що таке потік? Якими елементами характеризується потік? Які переваги при використанні декількох потоків?

Які існують способи реалізації потоків?

Які є стани виконання процесів в Linux?

Що таке сигнал в ОС UNIX? Обробка сигналів UNIX?