6.8 УПРАВЛІННЯ ПРОЦЕСАМИ І ПОТОКАМИ В LINUX

6.8.1 Процеси в LINUX

В ОС LINUX процес, або задача, подаються як структура даних. LINUX підтримує таблицю задач, що є списком покажчиків на кожну визначену в даний момент структуру даних. У цій структурі даних інформація розбита на такі категорії.

Стан. Стан виконання процесу (виконується, готовий до виконання, призупинений, зупинений, зомбі);

Інформація з планування. Інформація, яка потрібна ОС LINUX для планування процесів. Процес може бути звичайним або таким, що виконується в реальному часі. Крім того, він має деякий пріоритет. Процеси, що виконуються в реальному часі, плануються до звичайних процесів. Лічильник веде відлік часу, відведеного процесу.

Ідентифікатори. Кожен процес має свій власний ідентифікатор, а також ідентифікатори користувача і групи. Ідентифікатор групи використовується для того, щоб призначити групі користувачів права доступу до ресурсів.

Обмін інформацією між процесами. В ОС LINUX використовується такий же механізм міжпроцесної взаємодії, як і в ОС UNIX SVR4.

Зв'язки. Кожен процес містить у собі зв'язки з паралельними до нього процесами, із спорідненими йому процесами (з якими він має загальний батьківський процес) і зв'язки з усіма своїми дочірніми процесами.

Час і таймери. Сюди входить час створення процесу, а також кількість процесорного часу, витраченого на цей процес. З процесом також можуть бути пов'язані інтервальні таймери (кванти часу, один або декілька). Квант часу задається в процесі за допомогою системного виклику. Після закінчення кванта часу процесу подається відповідний сигнал. Таймер може бути створений для одноразового або періодичного використання.

Файлова система. Містить у собі покажчики на всі файли, які відкриті цим процесом.

Віртуальна пам'ять. Визначає відведену цьому процесу віртуальну пам'ять.

Контекст, залежний від процесора. Інформація про регістри і стек, що становить контекст цього процесу.

Створення процесу. Процес породжується за допомогою системного виклику fork(). При цьому виклику відбувається перевірка на наявність вільної пам'яті, доступної для розміщення нового процесу. Якщо необхідна пам'ять доступна, то створюється процес-нащадок поточного процесу, що є точною копією цього процесу (клонований або народжений процес). При цьому в таблиці процесів для нового процесу будується відповідна структура. Нова структура створюється також в таблиці користувача. При цьому всі її змінні ініціалізуються нулями. Цьому процесу привласнюється новий унікальний ідентифікатор, а ідентифікатор батьківського процесу запам'ятовується в блоці управління процесом.

Завершення процесу. Для завершення процесу використовується системний виклик exit(), при якому звільняються усі використовувані ресурси, такі як пам'ять і структури таблиць ядра. Крім того, завершуються і процеси- нащадки, породжені цим процесом.

Потім з пам'яті видаляються сегменти коду і даних, а сам процес переходить в стан зомбі (у полі Stat такі процеси позначаються буквою «Z»). Зомбі не займає процесорного часу, але рядок в таблиці процесів залишається, і відповідні структури ядра не звільняються. Після завершення батьківського процесу зомбі, що «осиротів», на короткий час стає нащадком init, після чого вже «остаточно помирає». І, нарешті, батьківський процес повинен очистити всі ресурси, займані дочірніми процесами.

Якщо батьківський процес з якоїсь причини завершиться раніше дочірнього, останній стає «сиротою» (orphaned process). Такі «сироти» також автоматично «усиновляються» програмою init, що виконується в процесі з номером 1, яка і приймає сигнал про їх завершення.

Також, процес може впасти в «сон», який не вдається перервати (у полі Stat це позначається буквою «D»). Процес, що знаходиться в такому стані, не реагує на системні запити і може бути знищений тільки перезавантаженням системи.

Взаємодія процесів. Найпоширенішим засобом взаємодії процесів є сокети (sockets). Програми підключаються до сокета і видають запит на прив'язку до потрібної адреси. Потім дані передаються від одного сокета до іншого відповідно до

вказаної адреси. Сигнал інформує інший процес про виникнення певних умов усередині поточного процесу, що вимагають реакції поточного процесу. Багато програм обробки сигналів для аналізу виниклої проблеми виводять дамп пам'яті.

Канали реалізовані в двох класах. Перший з них створюється за допомогою системного виклику pipe(). При цьому для обміну інформацією між процесами ініціалізувалася спеціальна структура в ядрі. Потім, коли процес породжує новий процес, між двома процесами відкривається комунікаційний канал. Іншим типом каналів є іменовані канали. При їх використанні із структурою, що управляє, в ядрі зв'язується спеціальний каталог, через який два автономні процеси можуть обмінюватися даними. При цьому, кожен процес повинен відкрити канал у вигляді звичайних файлів (один для читання, інший для запису). Потім операції введення-виведення виконуються звичайним способом.

Черга повідомлень є механізмом, коли один процес надає блок даних зі встановленими прапорами, а інший процес розшукує блок даних, прапори якого встановлені в необхідних значеннях.

Семафори є засобом передачі прапорів від одного процесу до іншого. «Піднявши» семафор, процес може повідомити, що він знаходиться в певному стані. Будь-який інший процес в системі може відшукати цей прапор і виконати необхідні дії.

Спільно використовувана пам'ять дозволяє процесам отримати доступ до однієї і тієї ж області фізичної пам'яті.

6.8.2 Потоки в LINUX

Поняття процесу і потоку в LINUX дуже тісно пов'язані і тому їх важко відрізнити, потоки навіть часто називають легковаговими процесами.

Основні відмінності процесу від потоку полягають у тому, що кожному процесу відповідає своя незалежна від інших область пам'яті, таблиця відкритих файлів, поточна директорія і інша інформація рівня ядра. Потоки ж не пов'язані безпосередньо з цими сутностями. В усіх потоків, тих, що належать цьому процесу, все вище перераховане загальне, оскільки належить цьому процесу.

Для управління потоками використовуються відповідні засоби, які доступні програмістові через мовні конструкції, системні виклики ОС або спеціально

розроблені бібліотеки. Наприклад, бібліотека потоків Pthread визначає об'єкт атрибутів потоку, що інкапсулює властивості потоку, до яких творець об'єкту може отримати доступ і модифікувати їх.

Створення потоку і ідеологія POSIX API. При вибраному для вивчення низькорівневому підході до підтримки потоків у мові усі операції, пов'язані з ними, виражаються явно через виклики функцій мови С і інтерфейси підтримки потоків відповідному стандарту POSIX API. Згідно з ним потік (нитка) створюється за допомогою такого виклику:

int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void* (*start)(void *), void *arg)

Спрощено виклик **pthread_create(&thr, NULL, start, NULL)** створить потік який почне виконувати функцію start і запише в змінну thr ідентифікатор створеного потоку. На прикладі цього виклику детально розглянемо декілька допоміжних концепцій POSIX API з тим, щоб не зупинятися на них далі.

Перший аргумент цієї функції **thread** – це покажчик на змінну типу pthread_t, в яку буде записаний ідентифікатор створеного потоку, який потім можна буде передавати іншим викликам, коли неоюхідно зробити що-небудь з цим потоком.

Другий аргумент цієї функції **attr** – це покажчик на змінну типу **pthread_attr_t,** яка задає набір деяких властивостей створюваного потоку.

Третій аргумент виклику **pthread_create** – це покажчик на функцію типу void*(). Саме цю функцію і починає виконувати знову створений потік. При цьому, в якості параметра цієї функції передається четвертий аргумент виклику pthread_create.

Функція pthread_create повертає нульове значення в разі успіху і ненульовий код помилки в разі невдачі. Це також одна з особливостей POSIX API. Замість стандартного для Unix підходу, коли функція повертає лише деякий індикатор помилки, а код помилки встановлює в змінній errno, функції Pthreads API повертають код помилки в результаті свого аргументу.

Життєвий цикл потоку. Розглянемо тепер життєвий цикл потоку, а саме послідовність станів, в яких знаходиться потік за час свого існування.

Потік може перебувати в одному з чотирьох станів: готовності, виконання, останову-очікування, блокування (рис.6.9).

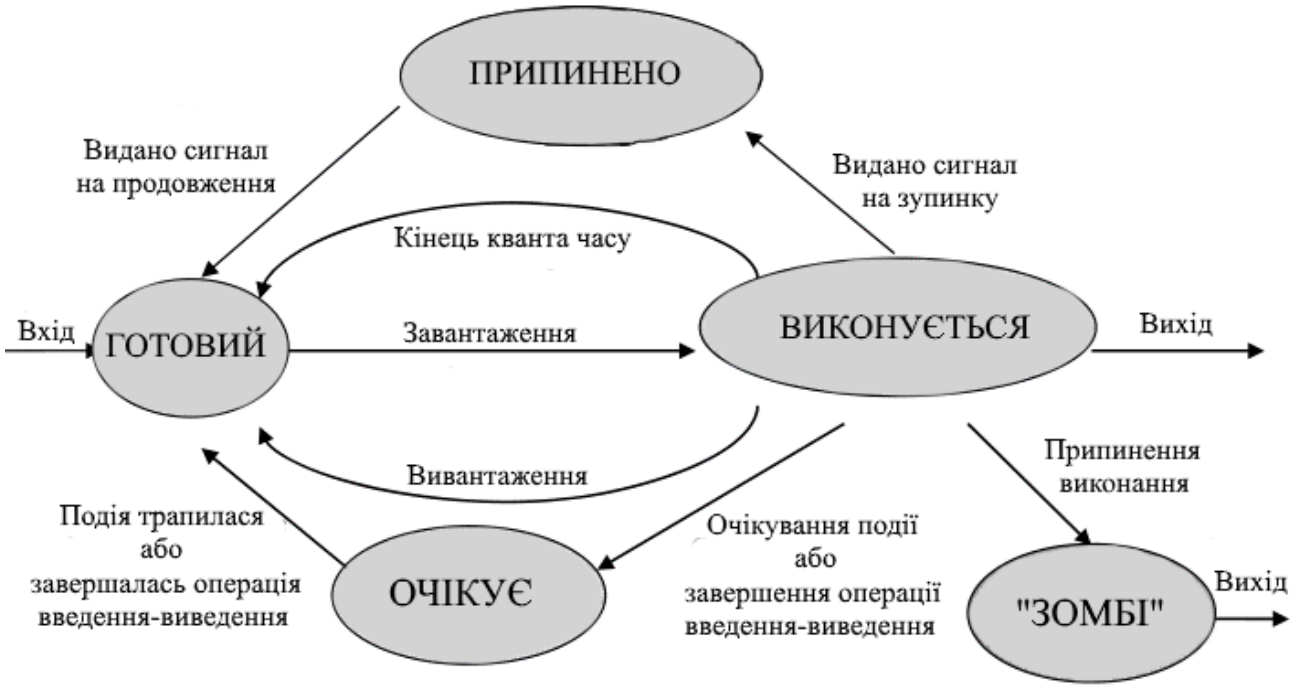

Рисунок 6.9 – Стани потоків в средовищах Unix/Linux

Готовий. Потік знаходиться в стані готовності, коли він готовий до виконання. Можливо, він тільки що був створений, або був витіснений з процесора іншим потоком, або тільки що був розблокований (вийшов з відповідного стану).

Усі готові до роботи потоки поміщаються в черги готовності згідно зі своїми пріоритетами. Потік переходить в стан виконання, коли він вибирається з черги і назначається процесору. Потік знімається з процесора і поміщається в чергу готових потоків, якщо його квант часу закінчився або якщо він перейшов в стан готовності з великим пріоритетом. Потік готовий до виконання, але чекає процесора.

Виконується. Потік, що виконується, може отримати сигнал і перейти в стан останову, який принципово відрізняється від стану очікування. Потік отримує сигнал зупинитися, якщо він знаходиться в стані відладки або із-за виникнення особливої ситуації в системі. Пізніше потік може бути розбуджений або ліквідований. Якщо потік не є відкріпленим, то після завершення виконання він переходить в спеціальний стан «зомбі». В цьому стані він вже не здатний продовжувати виконання, він також не використовує системних ресурсів, але не може покинути систему, поки його батьківський потік не поміняє його статус на завершення.

Чекає. Потік, що чекає, може бути в двох підстанах:

1. Такий, що переривається. Це стан блокування, в якому процес чекає на подію (наприклад, завершення операції введення-виведення, звільнення ресурсу або сигналу від іншого процесу).

2. Такий, що не переривається. Це стан блокування іншого роду. Його відмінність від попереднього полягає в тому, що в цьому стані процес безпосередньо чекає виконання якоїсь апаратної умови, тому він не сприймає ніяких сигналів.

Зупинений. Процес (потік) був зупинений і може бути продовжений тільки при відповідній дії іншого процесу. Наприклад, процес знаходиться в стані відладки, може перейти в стан зупинки.

Завершення потоку, особливості головного потоку. Потік завершується, коли відбувається повернення з функції start. При цьому якщо ми хочемо отримати повернене значення функції, то ми повинні скористатися функцією:

int pthread_join(pthread_t thread, void value_ptr)**

Ця функція чекає завершення потоку з ідентифікатором thread, і записує її повернене значення в змінну, на яку вказує value_ptr. При цьому, звільняються всі ресурси, пов'язані з потоком, і, отже, ця функція може бути викликана для цього потоку тільки один раз.

Окрім повернення з функції потоку, існує ще один спосіб завершити потік, а саме – викликати exit(), аналогічний викличу exit() для процесів:

int pthread_exit(void *value_ptr)

Цей виклик завершує виконуваний потік, повертаючи в якості результату його виконання value_ptr. Реально, що при виклику цієї функції, потік з неї просто не повертається. Потрібно звернути також увагу на той факт, що функція exit() як і раніше завершує процес, тобто, у тому числі, знищує всі потоки.