**1. Bộ Lập Lịch (Scheduler) trong Linux Kernel**

Bộ lập lịch của Linux Kernel chịu trách nhiệm quản lý thời gian CPU cho các tiến trình và luồng. Bộ lập lịch chính của Linux là:

* **Completely Fair Scheduler (CFS)**: CFS sử dụng cây đỏ-đen (red-black tree) để đảm bảo phân phối công bằng thời gian CPU giữa các tiến trình. Mỗi tiến trình có một giá trị vruntime (virtual runtime) để theo dõi thời gian thực thi của nó, với mục tiêu đảm bảo rằng các tiến trình đều được cấp thời gian CPU một cách công bằng.
* **Real-Time Scheduling**: Linux hỗ trợ các thuật toán lập lịch thời gian thực như FIFO và Round Robin, đảm bảo các yêu cầu thời gian thực được đáp ứng.

Bổ xung priority

**2. Quản Lý Bộ Nhớ (Memory Management - MM) trong Linux Kernel**

Quản lý bộ nhớ trong Linux gồm nhiều khía cạnh như phân phối bộ nhớ, quản lý bộ nhớ ảo, và tối ưu hóa sử dụng bộ nhớ:

* **Virtual Memory (Bộ Nhớ Ảo)**: Linux sử dụng bộ nhớ ảo để cung cấp không gian địa chỉ liên tục cho các tiến trình, tách biệt bộ nhớ vật lý và bộ nhớ ứng dụng.
* **Paging**: Sử dụng để quản lý không gian địa chỉ ảo, chia bộ nhớ thành các trang (pages) và khung trang (frames).
* **Slab Allocator**: Một hệ thống cấp phát bộ nhớ để quản lý các đối tượng nhỏ, giảm thiểu phân mảnh bộ nhớ.
* **Swapping**: Khi bộ nhớ vật lý đầy, Linux có thể chuyển đổi các trang ít sử dụng sang bộ nhớ phụ (swap space).

**3. Thread (Luồng) và Process (Tiến Trình) trong Linux**

* **Process**: Là một phiên bản của chương trình đang chạy, có không gian địa chỉ riêng. Cấu trúc dữ liệu chính của một tiến trình là task\_struct.
  + **Memory layout**
  + **Size of each segment + chứng minh**
  + **task\_struct: thông tin của 1 processs**
  + **cấp phát động + memory leak**
* **Thread**: Là đơn vị nhỏ hơn của tiến trình, chia sẻ không gian địa chỉ và tài nguyên với các thread khác trong cùng tiến trình. Các thread trong Linux được quản lý bởi task\_struct tương tự như tiến trình.
  + **Where memory stay in process’s memory**
  + **Stack size**
  + Stack overflow + vì sao nên sử dụng heap và cấp phát động

**4. Các Cách Đồng Bộ Giữa Các Thread trong Linux**

* **Mutex (Mutual Exclusion)**: Đảm bảo chỉ một thread có thể truy cập tài nguyên tại một thời điểm.
* **Condition Variables**: Sử dụng cùng với mutex để cho phép thread chờ đợi hoặc báo hiệu khi một điều kiện thay đổi.
* **Semaphore**: Quản lý số lượng thread truy cập vào tài nguyên đồng thời.
* **Barrier:**
* **Spinlocks**: Thường được sử dụng trong kernel để bảo vệ tài nguyên dùng chung. Hiệu quả trong các hệ thống SMP.
* **Read-Copy-Update (RCU)**: Cho phép các thread đọc tài nguyên mà không bị khóa, cải thiện hiệu suất cho các hệ thống đa luồng.

**5. Giao Tiếp Giữa Các Process trong Linux**

* **Message Queues**: Cho phép các tiến trình trao đổi thông điệp thông qua hàng đợi tin nhắn.
* **Shared Memory**: Cung cấp một vùng nhớ chung mà các tiến trình có thể cùng truy cập và sử dụng.
* **Sockets**: Sử dụng để giao tiếp giữa các tiến trình trên cùng hoặc khác máy, hỗ trợ giao tiếp qua mạng.

**6. Các Vấn Đề Liên Quan trong Linux**

* **Deadlock**: Xảy ra khi hai hay nhiều tiến trình hoặc thread chờ đợi nhau để giải phóng tài nguyên, dẫn đến tình trạng chờ đợi vô tận. Để tránh deadlock, có thể sử dụng các chiến lược như thứ tự khóa cố định và timeout.
* **Starvation**: Xảy ra khi một tiến trình hoặc thread không bao giờ có cơ hội thực thi vì các tiến trình hoặc thread khác liên tục chiếm giữ tài nguyên. CFS được thiết kế để giảm thiểu vấn đề này bằng cách đảm bảo phân phối công bằng thời gian CPU.
* **Race Condition**: Xảy ra khi kết quả của chương trình phụ thuộc vào thứ tự thực thi của các thread hoặc tiến trình. Sử dụng các cơ chế đồng bộ hóa như mutex và semaphore là cách để tránh race condition.