I. Câu hỏi tự luận

Bài 1: So sánh Monolithic Kernel và Microkernel

- Đề bài:
- Trình bày sự khác biệt giữa Monolithic Kernel và Microkernel.
- So sánh ưu nhược điểm của hai mô hình này về hiệu suất, bảo trì, bảo mật.
- Giải thích tại sao Linux sử dụng Monolithic Kernel nhưng vẫn có tính linh hoạt cao.

ANSWER:

Sự khác biệt giữa Monolithic Kernel và Microkernel

Monolithic Kernel và **Microkernel** là hai kiến trúc hạt nhân hệ điều hành khác nhau:

1. **Monolithic Kernel:**
    - Tất cả các dịch vụ hệ điều hành chạy trong cùng một không gian địa chỉ hạt nhân.
    - Các dịch vụ như quản lý bộ nhớ, quản lý tiến trình, hệ thống tệp, và trình điều khiển thiết bị đều được tích hợp trực tiếp vào hạt nhân.
    - Ví dụ: Linux, Unix.
2. **Microkernel:**

Microkernel là một loại kiến trúc hạt nhân của hệ điều hành, trong đó chỉ các chức năng cơ bản nhất được chạy trong không gian địa chỉ hạt nhân. Những chức năng cơ bản này thường bao gồm:

- Quản lý bộ nhớ cơ bản.
- Quản lý tiến trình.
- Giao tiếp giữa các tiến trình.

Các dịch vụ hệ điều hành khác, như quản lý hệ thống tệp, trình điều khiển thiết bị, và giao diện người dùng, được chạy trong không gian người dùng. Điều này giúp giảm thiểu kích thước và độ phức tạp của hạt nhân, và cũng tăng tính bảo mật và độ tin cậy của hệ điều hành.

So sánh ưu nhược điểm của Monolithic Kernel và Microkernel

1. **Hiệu suất:**
    - **Monolithic Kernel:** Thường có hiệu suất cao hơn do các dịch vụ hệ điều hành chạy trong cùng một không gian địa chỉ, giảm thiểu thời gian chuyển ngữ cảnh.
    - **Microkernel:** Hiệu suất có thể thấp hơn do phải thực hiện nhiều giao tiếp giữa không gian người dùng và hạt nhân.
2. **Bảo trì:**
    - **Monolithic Kernel:** Khó bảo trì hơn do tất cả các dịch vụ hệ điều hành được tích hợp trực tiếp, việc thay đổi một phần có thể ảnh hưởng đến toàn bộ hệ thống.

- **Microkernel:** Dễ bảo trì hơn do các dịch vụ hệ điều hành được tách biệt, dễ dàng thay thế hoặc cập nhật một phần mà không ảnh hưởng đến các phần khác.

3. **Bảo mật:**
   - **Monolithic Kernel:** Dễ bị tấn công hơn do tất cả các dịch vụ hệ điều hành chạy trong cùng một không gian địa chỉ.
   - **Microkernel:** An toàn hơn do các dịch vụ chạy trong không gian người dùng, một lỗi trong một dịch vụ sẽ ít ảnh hưởng đến các dịch vụ khác.

Bài 2: Mô hình "Everything as a File" trong Linux
Đề bài:

- Giải thích mô hình "Everything as a File".
- Nêu các đối tượng trong Linux hoạt động như file (ví dụ: thiết bị, tiến trình).
- Chạy lệnh kiểm tra và phân tích đầu ra để chứng minh rằng Linux áp dụng mô hình này.

Hướng dẫn làm:
Giải thích mô hình:
Định nghĩa “Everything as a File” và lợi ích của nó.
Các đối tượng hoạt động như file:
Lấy ví dụ về file /dev, /proc, socket, process descriptor.
Thực hành:
Dùng các lệnh ls -l /dev, cat /proc/cpuinfo, echo "Test" > /dev/null để kiểm tra.

ANSWER:

Mô hình "Everything as a File"

"Everything as a File" là một nguyên tắc thiết kế trong hệ điều hành Unix và các hệ thống giống Unix, trong đó hầu hết mọi thứ, từ thiết bị phần cứng đến các tiến trình phần mềm, đều được biểu diễn dưới dạng tệp. Nguyên tắc này giúp đơn giản hóa việc tương tác với hệ điều hành, bởi vì các đối tượng khác nhau có thể được xử lý thông qua một giao diện thống nhất là hệ thống tệp.

Các đối tượng trong Linux hoạt động như file

Trong Linux, nhiều đối tượng khác nhau hoạt động như tệp, bao gồm:

1. **Thiết bị (Devices):**
    - Các thiết bị như ổ đĩa, bàn phím, chuột được biểu diễn dưới dạng tệp trong thư mục /dev.
    - Ví dụ: /dev/sda (ổ đĩa), /dev/tty (thiết bị đầu vào/đầu ra).
2. **Tiến trình (Processes):**
    - Các tiến trình đang chạy được biểu diễn dưới dạng tệp trong thư mục /proc.
    - Ví dụ: /proc/[PID] là một thư mục chứa thông tin về tiến trình với PID (Process ID) cụ thể.
3. **Tệp cài đặt và cấu hình (Configuration Files):**
    - Các tệp cấu hình hệ thống và ứng dụng được biểu diễn dưới dạng tệp văn bản trong các thư mục như /etc.

Bài 3: Cách Linux thực hiện Preemptive Multitasking

Đề bài:

- Giải thích Preemptive Multitasking là gì.
- Mô tả vai trò của Linux Scheduler trong việc quản lý tiến trình.

Hướng dẫn làm:

Giải thích:

- Phân biệt Preemptive Multitasking và Cooperative Multitasking.
- Vai trò của Scheduler:
- Mô tả thuật toán Completely Fair Scheduler (CFS) và các yếu tố quyết định scheduling.
- Thực hành kiểm chứng: Dùng ps -eo pid,pri,ni,cmd kiểm tra mức ưu tiên của tiến trình.

ANSWER:

Phân biệt Preemptive Multitasking và Cooperative Multitasking

1. **Preemptive Multitasking:**
    - **Nguyên tắc:** Hệ điều hành chủ động quản lý thời gian CPU dành cho mỗi tiến trình, ngắt tiến trình hiện tại để chuyển sang tiến trình khác theo lịch trình định sẵn.
    - **Ưu điểm:** Cải thiện khả năng phản hồi của hệ thống, đảm bảo rằng các tiến trình quan trọng không bị chờ đợi quá lâu.
    - **Nhược điểm:** Đòi hỏi hệ điều hành phức tạp hơn để quản lý ngắt và chuyển đổi ngữ cảnh.
    - **Ví dụ:** Các hệ điều hành hiện đại như Windows, Linux đều sử dụng Preemptive Multitasking.
2. **Cooperative Multitasking:**
    - **Nguyên tắc:** Các tiến trình tự nguyện nhường quyền điều khiển CPU cho các tiến trình khác khi không còn cần sử dụng.
    - **Ưu điểm:** Dễ dàng triển khai, không cần hệ điều hành phức tạp để quản lý chuyển đổi ngữ cảnh.
    - **Nhược điểm:** Nếu một tiến trình không nhường quyền điều khiển, toàn bộ hệ thống có thể bị đình trệ.
    - **Ví dụ:** Các hệ điều hành cũ như Windows 3.x và Mac OS 9.

Vai trò của Scheduler

Scheduler (Bộ lập lịch) là một thành phần quan trọng của hệ điều hành, có nhiệm vụ:

- **Quản lý thời gian CPU:** Quyết định tiến trình nào sẽ được cấp phát CPU và trong bao lâu.
- **Đảm bảo công bằng:** Đảm bảo rằng tất cả các tiến trình đều có cơ hội được thực thi.
- **Tối ưu hóa hiệu suất:** Tối ưu hóa việc sử dụng tài nguyên hệ thống để đạt hiệu suất cao nhất.
- **Đảm bảo thời gian thực:** Đảm bảo các tiến trình thời gian thực được thực thi đúng thời hạn yêu cầu.

Thuật toán Completely Fair Scheduler (CFS) và các yếu tố quyết định scheduling

Completely Fair Scheduler (CFS) là thuật toán lập lịch được sử dụng trong nhân Linux. Mục tiêu của CFS là đảm bảo tính công bằng tuyệt đối cho tất cả các tiến trình. Các yếu tố quyết định scheduling trong CFS bao gồm:

- **Vrun:** Thời gian tiến trình đã sử dụng CPU.
- **Fair share:** Tính toán thời gian công bằng mà mỗi tiến trình nên được sử dụng CPU.
- **Priority (Mức ưu tiên):** Các tiến trình có mức ưu tiên cao hơn sẽ được cấp phát CPU trước.

CFS sử dụng một cấu trúc cây đỏ-đen để quản lý các tiến trình, giúp tối ưu hóa việc tìm kiếm và cấp phát CPU cho các tiến trình.

Thực hành kiểm chứng

Để kiểm tra mức ưu tiên của tiến trình, bạn có thể sử dụng lệnh sau:

bash

ps -eo pid,pri,ni,cmd

Lệnh này sẽ liệt kê các tiến trình đang chạy cùng với thông tin về PID (Process ID), PRI (Priority), NI (Nice value), và CMD (Command).

- **PID:** Mã số nhận dạng của tiến trình.
- **PRI:** Mức ưu tiên của tiến trình, giá trị thấp hơn biểu thị ưu tiên cao hơn.
- **NI:** Giá trị Nice, ảnh hưởng đến mức ưu tiên của tiến trình (giá trị thấp hơn biểu thị ưu tiên cao hơn).
- **CMD:** Lệnh được sử dụng để khởi động tiến trình.

Phân tích đầu ra của lệnh này sẽ giúp bạn hiểu rõ hơn về cách hệ điều hành Linux quản lý và ưu tiên các tiến trình. Hy vọng bạn thấy thông tin này hữu ích!

II. Bài tập thực hành

Yêu cầu:

- Thực hiện trên môi trường Linux, kiểm tra đầu ra của từng bước.
- Làm việc với Kernel Modules

2.1 Bài tập: Giải thích kết quả của mỗi lệnh

2.1.1.Kiểm tra các module kernel đang chạy:

```
lsmod
```

Giải thích:

- `lsmod` là lệnh liệt kê các module kernel đang được nạp và chạy trong hệ thống.
- Đầu ra của lệnh này thường bao gồm ba cột thông tin:
    - **Module:** Tên của module.
    - **Size:** Kích thước của module (tính theo byte).
    - **Used by:** Số lượng các đối tượng (như tiến trình) đang sử dụng module này.

Ví dụ:

bash

```
Module                  Size  Used by
ext4                  536576  2
jbd2                  106496  1 ext4
mbcache                16384  2 ext4
```

Phân tích:

- `ext4` là tên của module (fourth extended filesystem)
- `536576` là kích thước của module `ext4`.
- `2` là số lượng đối tượng đang sử dụng module `ext4`

2.1.2.Xem thông tin về một module cụ thể:

```
modinfo ext4
```

Giải thích:

- `modinfo` là lệnh hiển thị thông tin chi tiết về một module kernel cụ thể.
- Đầu ra của lệnh này thường bao gồm các thông tin như:
    - **filename:** Đường dẫn đến tệp module.
    - **description:** Mô tả về module.
    - **author:** Tác giả của module.
    - **license:** Giấy phép sử dụng của module.
    - **alias:** Các tên khác của module.
    - **depends:** Các module mà module này phụ thuộc vào.

- **retpoline:** Nếu module hỗ trợ chống tấn công Spectre (Retpoline).
- **intree:** Nếu module là một phần của kernel chính.
- **vermagic:** Thông tin về phiên bản kernel tương thích với module này.

Ví dụ:

bash

```
filename:       /lib/modules/5.4.0-66-generic/kernel/fs/ext4/ext4.ko
description:    Fourth Extended Filesystem
author:         Remy Card, Stephen Tweedie, Andrew Morton, Andreas Dilger,
Theodore Ts'o and others
license:        GPL
alias:          ext3
depends:        jbd2,mbcache
retpoline:      Y
intree:         Y
name:           ext4
vermagic:       5.4.0-66-generic SMP mod_unload
```

Phân tích:

- **filename:** `/lib/modules/5.4.0-66-generic/kernel/fs/ext4/ext4.ko` là đường dẫn đến tệp module `ext4`.
- **description:** Mô tả rằng đây là "Fourth Extended Filesystem" (hệ thống tệp mở rộng thứ tư).
- **author:** Danh sách các tác giả đã đóng góp cho module này.
- **license:** Giấy phép sử dụng của module là GPL (General Public License).
- **alias:** Module `ext4` cũng có thể được gọi là `ext3`.
- **depends:** Module này phụ thuộc vào các module khác như `jbd2` và `mbcache`.
- **retpoline:** Module này hỗ trợ chống tấn công Spectre.
- **intree:** Module này là một phần của kernel chính.
- **vermagic:** Thông tin về phiên bản kernel tương thích.

2.2.Tìm hiểu hệ thống tập tin trong Linux

Bài tập: Giải thích kết quả của mỗi lệnh

2.2.1.Liệt kê các thiết bị trong /dev:

```
ls -l /dev
```

Giải thích:

- Lệnh `ls -l /dev` sẽ liệt kê các tệp và thư mục trong thư mục `/dev`, bao gồm các thiết bị hệ thống.
- Lệnh ls trong Linux là viết tắt của từ "list" (danh sách). Khi bạn thêm tùy chọn -l, nó chuyển đổi chế độ hiển thị sang danh sách dài (long listing format)
- Đầu ra sẽ hiển thị chi tiết các thiết bị phần cứng như ổ đĩa, bàn phím, chuột, và các thiết bị khác.
- Mỗi mục trong đầu ra bao gồm thông tin về : Quyền truy cập, Số liên kết, Chủ sở hữu, Nhóm chủ sở hữu, Kích thước, Thời gian sửa đổi cuối cùng và Tên thiết bị.

Ví dụ:

bash

```
crw-rw-rw- 1 root root 1, 3 Feb 15 20:45 null
brw-rw---- 1 root disk 8, 0 Feb 15 20:45 sda
drwxr-xr-x 2 root root 4096 Feb 15 20:45 pts
```

Phân tích:

- `c` hoặc `b` ở đầu dòng biểu thị loại thiết bị (`c` cho thiết bị ký tự, `b` cho thiết bị khối).
- `1, 3` và `8, 0` là các số chính và phụ của thiết bị, xác định cụ thể thiết bị phần cứng.

2.2.2.Kiểm tra thông tin CPU và bộ nhớ:

```
cat /proc/cpuinfo
```

Giải thích:

- Lệnh cat trong Linux là viết tắt của "concatenate". Được sử dụng để:

  1. **Hiển thị nội dung tệp**: Hiển thị nội dung của một hoặc nhiều tệp trên màn hình.
  2. **Nối tệp**: Ghép nhiều tệp lại với nhau và hiển thị hoặc lưu vào tệp mới.
  3. **Tạo tệp**: Tạo tệp mới và thêm nội dung vào tệp đó.

  Ví dụ: lệnh cat file.txt sẽ hiển thị nội dung của tệp file.txt trên màn hình. Lệnh cat file1.txt file2.txt > merged.txt sẽ ghép nội dung của file1.txt và file2.txt và lưu vào tệp mới có tên merged.txt

- Lệnh `cat /proc/cpuinfo` sẽ hiển thị thông tin chi tiết về CPU, bao gồm tên, số lõi, tốc độ xung nhịp, và các tính năng khác.

Ví dụ:

bash
```
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 158
model name      : Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz
stepping        : 10
cpu MHz         : 3200.000
cache size      : 12288 KB
```

Phân tích:

- `processor` xác định lõi CPU.
- `vendor_id` xác định nhà sản xuất.
- `model name` cho biết tên và tốc độ của CPU.
- `cpu MHz` cho biết tốc độ xung nhịp.
- `cache size` cho biết kích thước bộ nhớ đệm.

```
cat /proc/meminfo
```

Giải thích:

- Lệnh `cat /proc/meminfo` sẽ hiển thị thông tin về bộ nhớ hệ thống, bao gồm tổng dung lượng RAM, dung lượng RAM đang sử dụng, dung lượng RAM trống, và các thông số khác.

Ví dụ:

bash
```
MemTotal:       16384256 kB
MemFree:        12345678 kB
MemAvailable:   12345678 kB
Buffers:          123456 kB
Cached:         2345678 kB
SwapTotal:      4096000 kB
SwapFree:       4096000 kB
```

Phân tích:

- `MemTotal` là tổng dung lượng RAM.
- `MemFree` là dung lượng RAM trống.
- `MemAvailable` là dung lượng RAM có sẵn.
- `SwapTotal` là tổng dung lượng swap.
- `SwapFree` là dung lượng swap trống.

2.2.3.Ghi dữ liệu vào /dev/null và quan sát kết quả:

```
echo "Test" > /dev/null
```

Giải thích:

- Lệnh `echo "Test" > /dev/null` ghi dữ liệu "Test" vào tệp đặc biệt `/dev/null`.
- `/dev/null` là một thiết bị đặc biệt được gọi là "bit bucket" hoặc "black hole", nơi mọi dữ liệu ghi vào sẽ bị hủy mà không gây lỗi.

Ví dụ:

bash
```
echo "Test" > /dev/null
```

Phân tích:

- Dữ liệu "Test" sẽ không được lưu trữ ở bất kỳ đâu và không có đầu ra nào được hiển thị. Đây là cách phổ biến để hủy dữ liệu không cần thiết.

2.3.Quản lý tiến trình trong Linux

Hướng dẫn làm

1. **Kiểm tra PID bằng `ps aux` trước khi dùng `kill`:**
   - Sử dụng lệnh `ps aux` để liệt kê tất cả các tiến trình đang chạy và tìm PID của tiến trình bạn muốn kết thúc.
2. **Quan sát thay đổi trước và sau khi `kill` tiến trình:**
   - Trước khi dùng `kill`, hãy chú ý trạng thái và thông tin về tiến trình bạn muốn kết thúc.
   - Sau khi dùng `kill -9 <PID>`, sử dụng lại lệnh `ps aux` hoặc `top` để kiểm tra xem tiến trình đã bị kết thúc chưa. Nếu tiến trình đã bị kết thúc, nó sẽ không còn xuất hiện trong danh sách các tiến trình đang chạy.

Bài tập: Giải thích kết quả của mỗi lệnh
2.3.1.Liệt kê tất cả tiến trình đang chạy:

```
ps aux
```

Giải thích:

- `ps aux` là lệnh hiển thị tất cả các tiến trình đang chạy trên hệ thống, bao gồm cả các tiến trình của người dùng khác và tiến trình không liên kết với terminal.
- Đầu ra bao gồm nhiều cột thông tin như:
  - **USER:** Tên người dùng sở hữu tiến trình.
  - **PID:** Mã số nhận dạng tiến trình.
  - **%CPU:** Phần trăm CPU mà tiến trình sử dụng.
  - **%MEM:** Phần trăm bộ nhớ mà tiến trình sử dụng.
  - **VSZ:** Kích thước bộ nhớ ảo của tiến trình.
  - **RSS:** Kích thước bộ nhớ vật lý mà tiến trình đang sử dụng.
  - **TTY:** Terminal liên kết với tiến trình (nếu có).
  - **STAT:** Trạng thái của tiến trình.
  - **START:** Thời điểm bắt đầu tiến trình.
  - **TIME:** Tổng thời gian CPU mà tiến trình đã sử dụng.
  - **COMMAND:** Lệnh khởi động tiến trình.
- Lệnh ps aux trong Linux là một kết hợp của các tùy chọn lệnh ps. Nó hiển thị tất cả các tiến trình đang chạy trên hệ thống, bao gồm cả tiến trình của các người dùng khác và tiến trình không liên kết với terminal

  ps: Lệnh này viết tắt của "process status" (trạng thái tiến trình). Nó được sử dụng để hiển thị thông tin về các tiến trình đang chạy.

  a: Tùy chọn này hiển thị thông tin về tất cả các tiến trình có liên kết với một terminal (TTY), bao gồm các tiến trình của người dùng khác.

  u: Tùy chọn này hiển thị thông tin tiến trình theo định dạng người dùng thân thiện hơn, bao gồm tên người dùng, thời gian CPU sử dụng, và các thông số khác.

x: Tùy chọn này hiển thị thông tin về các tiến trình không liên kết với một terminal cụ thể.

```
Bash                                                    Copy

USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.1 104588  8480 ?        Ss   10:00   0:01 /sbin/init
user      1234  0.3  2.0 246788 83460 pts/0    Sl+  10:05   0:20 /usr/bin/py
```

2.3.2.Xem thông tin tiến trình theo thời gian thực:

```
top
```

Giải thích:

- `top` là lệnh hiển thị thông tin về các tiến trình đang chạy theo thời gian thực.
- Đầu ra bao gồm các thông tin như:
    - **PID:** Mã số nhận dạng tiến trình.
    - **USER:** Tên người dùng sở hữu tiến trình.
    - **PR:** Mức ưu tiên của tiến trình.
    - **NI:** Giá trị Nice của tiến trình.
    - **VIRT:** Kích thước bộ nhớ ảo.
    - **RES:** Kích thước bộ nhớ vật lý mà tiến trình đang sử dụng.
    - **SHR:** Kích thước bộ nhớ dùng chung.
    - **S:** Trạng thái của tiến trình.
    - **%CPU:** Phần trăm CPU mà tiến trình sử dụng.
    - **%MEM:** Phần trăm bộ nhớ mà tiến trình sử dụng.
    - **TIME+:** Tổng thời gian CPU mà tiến trình đã sử dụng.
    - **COMMAND:** Lệnh khởi động tiến trình.

```
Bash                                                    Copy

top - 10:15:00 up 2 days,  5:32,  3 users,  load average: 0.00, 0.01, 0.05
Tasks: 112 total,   1 running, 111 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.3 us,  0.1 sy,  0.0 ni, 99.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0
KiB Mem :  12345678 total,   2345676 free,   3456784 used,   4567890 buff/ca
KiB Swap:   4096000 total,   4096000 free,         0 used.   6789100 avail M

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 1234 user      20   0  246788  83460   6544 S   0.3  2.0   0:20.32 python3
 5678 root      20   0   45678  12345   4567 S   0.1  0.1   0:02.45 top
```

2.3.3.Kết thúc một tiến trình cụ thể:

```
kill -9 <PID>
```

Giải thích:

- `kill -9 <PID>` là lệnh để kết thúc một tiến trình cụ thể bằng cách gửi tín hiệu `SIGKILL` đến tiến trình có mã số nhận dạng là `<PID>`.
- Lệnh kill -9 là lệnh để gửi tín hiệu SIGKILL (tín hiệu số 9) đến một tiến trình cụ thể, yêu cầu tiến trình đó phải kết thúc ngay lập tức.
- kill: Lệnh dùng để gửi tín hiệu đến tiến trình.
- -9: Tham số đại diện cho tín hiệu SIGKILL (tín hiệu số 9), buộc tiến trình phải dừng ngay lập tức mà không có cơ hội lưu trạng thái hiện tại.

Ví dụ:

Nếu bạn muốn kết thúc tiến trình có PID là `1234`, bạn sẽ chạy lệnh:

bash
```
kill -9 1234
```

2.4.Tạo và quản lý thread trong Linux bằng C

Bài tập:

- Viết chương trình tạo 3 thread in ra thông điệp.
- Sử dụng pthread_create(), pthread_join().
- Kiểm tra ID của từng thread.

Hướng dẫn làm:

- Cài đặt gcc nếu chưa có (sudo apt install gcc).
- Biên dịch chương trình với -lpthread.
- Quan sát thứ tự thực thi của các thread.

ANSWER:

Bước 1: Cài đặt GCC

Sử dụng lệnh sau để cài đặt GCC:

bash

```
sudo apt install gcc
```

Bước 2: Viết chương trình C tạo 3 thread

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

// Hàm được thực thi bởi mỗi thread
void* print_message(void* threadid) {
    long tid = (long)threadid;
    printf("Thread ID: %ld - Hello, Linux!\n", tid);
    pthread_exit(NULL);
}

int main() {
    pthread_t threads[3];
    int rc;
    long t;

    for(t = 0; t < 3; t++) {
        printf("In main: creating thread %ld\n", t);
        rc = pthread_create(&threads[t], NULL, print_message, (void*)t);
        if (rc) {
            printf("ERROR; return code from pthread_create() is %d\n", rc);
            exit(-1);
        }
    }

    // Join các thread để đảm bảo tất cả các thread kết thúc trước khi main
kết thúc
```

```
    for(t = 0; t < 3; t++) {
        pthread_join(threads[t], NULL);
    }

    printf("Main program completed.\n");
    return 0;
}
```

Bước 3: Biên dịch chương trình với -lpthread

Sử dụng lệnh sau để biên dịch chương trình:

bash
```
gcc -o thread_name thread_name.c -lpthread
```

Bước 4: Chạy chương trình

Chạy chương trình sau khi biên dịch:

bash
```
./thread_name
```

Quan sát thứ tự thực thi của các thread

Kết quả đầu ra sẽ bao gồm thông tin về ID của từng thread và thông điệp từ mỗi thread. Bạn sẽ thấy rằng thứ tự thực thi của các thread có thể không cố định, điều này minh chứng cho việc các thread chạy song song và có thể bị thay đổi thứ tự bởi hệ điều hành.

Dưới đây là một ví dụ về kết quả đầu ra:

bash
```
In main: creating thread 0
In main: creating thread 1
In main: creating thread 2
Thread ID: 0 - Hello, Linux!
Thread ID: 1 - Hello, Linux!
Thread ID: 2 - Hello, Linux!
Main program completed.
```

2.5.Lập trình với Preemptive Scheduling

Bài tập:

- Viết chương trình tạo 2 tiến trình con.
- Dùng nice để điều chỉnh mức độ ưu tiên.
- Kiểm tra mức ưu tiên bằng ps -eo pid,pri,ni,cmd.

Hướng dẫn làm:

- Thử chạy tiến trình với nice -n 10 ./program.
- Quan sát xem tiến trình nào chạy nhanh hơn.

ANSWER

Chương trình C

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

void child_process(int id) {
    printf("Child process %d with PID %d\n", id, getpid());
    for (int i = 0; i < 100000000; i++) { // Vòng lặp để tiêu tốn thời gian
CPU
        if (i % 10000000 == 0) {
            printf("Child process %d is working (i = %d)\n", id, i);
        }
    }
    printf("Child process %d completed\n", id);
}

int main() {
    pid_t pid1, pid2;

    // Tạo tiến trình con thứ nhất
    pid1 = fork();
    if (pid1 < 0) {
        perror("Fork failed");
        exit(1);
    } else if (pid1 == 0) {
        // Tiến trình con thứ nhất
        child_process(1);
        exit(0);
    }

    // Tạo tiến trình con thứ hai
    pid2 = fork();
    if (pid2 < 0) {
        perror("Fork failed");
        exit(1);
```

```
    } else if (pid2 == 0) {
        // Tiến trình con thứ hai
        child_process(2);
        exit(0);
    }

    // Chờ các tiến trình con kết thúc
    wait(NULL);
    wait(NULL);

    printf("Main process completed\n");
    return 0;
}
```

Hướng dẫn biên dịch và chạy chương trình

1. **Cài đặt GCC (nếu chưa có):**

bash
```
sudo apt install gcc
```

2. **Biên dịch chương trình:**

bash
```
gcc -o process_example process_example.c
```

3. **Chạy các tiến trình với mức độ ưu tiên khác nhau:**

- Chạy tiến trình với mức độ ưu tiên bình thường:

bash
```
./process_example
```

- Chạy tiến trình với mức độ ưu tiên thay đổi bằng `nice`:

bash
```
nice -n 10 ./process_example
```

Kiểm tra mức độ ưu tiên của các tiến trình

Sử dụng lệnh `ps` để kiểm tra mức độ ưu tiên của các tiến trình:

bash
```
ps -eo pid,pri,ni,cmd
```

Quan sát thứ tự thực thi của các tiến trình

Sau khi chạy các tiến trình với mức độ ưu tiên khác nhau, bạn có thể quan sát thứ tự thực thi của chúng. Tiến trình nào có mức độ ưu tiên cao hơn (giá trị `nice` nhỏ hơn) sẽ có khả năng chiếm nhiều tài nguyên CPU hơn và hoàn thành nhanh hơn.

Ví dụ về kết quả lệnh `ps`

bash
```
  PID PRI  NI CMD
 1234  30  10 ./process_example
 1235  20   0 ./process_example
```

Trong ví dụ trên, bạn có thể thấy mức độ ưu tiên (PRI) và giá trị `nice` (NI) của các tiến trình. Tiến trình có giá trị `nice` lớn hơn sẽ có mức độ ưu tiên thấp hơn, và ngược lại.