**Chương 1: Tổng quan về CAN Simulator**

1.1. Giới thiệu chương

1.2. Tổng quan về CAN Simulator

1.3. Các thành phần trong CAN Simulator

1.4. Kết luận chương

**Chương 2: Device Driver**

2.1. Giới thiệu chương

Chương này giới thiệu về Device Driver, giúp hiểu rõ hơn về cấu trúc, chức năng của Device Driver trong hệ thống Nhúng.

2.2. Tổng quan về Device Driver

Device Driver một chương trình máy tính điều khiển hoặc điều khiển một loại thiết bị cụ thể được gắn vào máy tính. Driver giúp phần mềm tương tác qua lại với phần cứng, cho phép các hệ điều hành và các chương trình máy tính khác truy cập các chức năng phần cứng mà không cần biết chi tiết chính xác về phần cứng đang được sử dụng.

2.3. Các lớp Device và Module

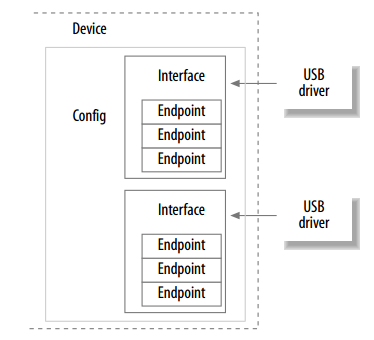
Có ba loại Device cơ bản, mỗi Module thực hiện một trong các loại này. Các Module được phân loại thành char module, block module, network module.

* Character devices: là loại device có thể được truy cập như một dòng byte( một tập tin), char driver chịu trách nhiện cho quá trình này. Một driver như vậy thường thực hiện ít nhất các lệnh gọi hệ thống như mở, đóng, đọc, ghi.
* Block devices: là loại device có thể lưu trữ một hệ thống tập tin. Trong hầu hết các hệ thống Unix, một Block device chỉ có thể xử lý các hoạt động I/O chuyển một hoặc nhiều khối tin, thường là 512 byte chiều dài
* Network devices: bất kỳ truyền nhận thông qua mạng cũng được thực hiện thông qua một tương tác mạng, tức là một thiết bị có thể trao đổi dữ liệu với các máy chủ khác. Một tương tác mạng chịu trách nhiệm gửi và nhận các gói dữ liệu, được điều khiển bởi hệ thống mạng con trong hạt nhân. Một network driver không biết gì về các kết nối cá nhân, nó chỉ xử lý gói tin.

2.4. USB Driver

2.4.1. Cơ bản về thiết bị USB

USB( universal serial bus) là chuẩn kết nối tuần tự đa dụng giữa máy chủ với những thiết bị ngoại vi. Hạt nhân Linux cung cấp một hệ thống con gọi là lõi USB để xử lý các tiến trình phức tạp.



Hình 2.1. USB

* Điểm cuối (endpoint) : Hình thức giao tiếp USB cơ bản nhất là thông qua điểm cuối, điểm cuối USB có thể mang dữ liệu theo một hướng, hoặc từ máy chủ đến thiết bị hoặc từ thiết bị tới máy chủ. Các điểm cuối có thể được coi là các đường ống đơn hướng. Điểm cuối USB có thể là một trong bốn loại khác nhau mô tả cách dữ liệu được truyền: Control, Interrupt, Bulk, Isochronous.
* Tương tác (interface) : bao gồm nhiều điểm cuối. Mỗi tương tác USB chỉ xử lý một loại kết nối USB, một số thiết bị USB có nhiều phần tương tác có thể thực hiện được nhiều chức năng
* Cấu hình (config) : Nhiều tương tác USB được xếp thành một cấu hình. Một thiết bị USB có thể có nhiều cấu hình và có thể chuyển đổi giữa chúng để thay đổi trạng thái của thiết bị. Mỗi cấu hình chỉ có thể được kích hoạt ở một thời điểm, Linux không thể xử lý tốt nhiều cấu hình cũng lúc.

2.4.2. USB và sysfs

2.4.3. USB urbs

Urb (USB request block) là mã USB trong giao tiếp hạt nhân Linux với tất cả các thiết bị USB. Một urb được sử dụng để gửi hoặc nhận dữ liệu từ một điểm cuối USB xác định trên một thiết bị USB cụ thể một cách không đồng bộ. Một trình điều khiển thiết bị USB có thể phân bổ nhiều urb cho một điểm cuối đơn hoặc có thể sử dụng lại một urb cho nhiều điểm cuối khác nhau tùy thuộc vào nhu cầu của trình điều khiển. Mỗi điểm cuối trong một thiết bị có thể xử lý một hàng đợi urb, nhiều urb có thể được gửi đến cùng một điểm cuối. Chu trình hoạt động điển hình của một urb như sau:

* + Được tạo bởi trình điều khiển thiết bị USB
  + Được chỉ định cho một điểm cuối cụ thể của một thiết bị USB cụ thể
  + Được chuyển tới lõi USB bằng trình điều khiển thiết bị USB
  + Gửi tới trình điều khiển máy chủ USB của thiết bị xác định bởi lõi USB
  + Xử lý bởi trình điều khiển máy chủ USB…………
  + Khi urb hoàn thành, trình điều khiển máy chủ USB sẽ thông báo cho trình điều khiển thiết bị USB

2.4.3.1. Struct urb

Các trường cấu trúc của struct urb có liên quan với trình điều khiển thiết bị USB là:

* struct usb\_device \*dev
* usigned int pipe
* unsigned int transfer\_flags
* void \*transfer\_buffer
* dma\_addr\_t transfer\_dma
* int transfer\_buffer\_length
* unsigned char \*setup\_packet
* dma\_addr\_t setup\_dma
* usb\_complete\_t complete
* void \*context
* int actual\_length
* int status
* int start\_frame
* int onterval
* int number\_of\_packets
* int error\_count
* struct usb\_iso\_packet\_descriptor iso\_frame\_desc[0]

2.4.3.2. Creating and Destroying Urbs

2.4.3.3. Submitting Urbs

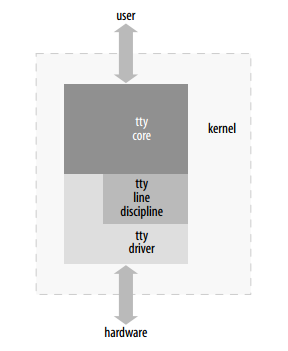
2.4.3.4. Completing Urbs

2.4.3.5. Canceling Urbs

2.5. TTY Driver

2.5.1. Cơ bản về TTY Driver

Tty driver trong Linux nằm ngay dưới mức Character driver tiêu chuẩn và cung cấp một loạt các tính năng tập trung vào việc cung cấp một sự tương tác cho các thiết bị đầu cuối sử dụng. Phần lõi chịu trách nhiêm kiểm soát cả luồng dữ liệu trên thiết bị tty lẫn định dạng dữ liệu, điều này cho phép các tty driver tập trung vào xử lý dữ liệu nhận từ phần cứng thay vì quan tâm tới cách kiểm soát tương tác với phạm vi của người dung một cách nhất quán. Hình 2.2 cho biết quá trình chuyển dữ liệu trong thiết bị tty .



Hình 2.2. Tổng quan tty core

Tty core lấy dữ liệu từ người dùng và dữ liệu sẽ được chuyển đến một thiết bị tty. Dữ liệu này sau đó chuyển đến tty line discipline driver và tới tty driver. Tty driver chuyển đổi dữ liệu sang một định dạng khác có thể gửi tới phần cứng. Đôi khi phần tty driver liên lạc trực tiếp đến tty core và tty core gửi dữ liệu trực tiếp đến tty driver, nhưng thường là tty line discipline sẽ sửa đổi dữ liệu được gửi giữa tty core và tty driver.

Có ba loại tty driver khác nhau: console, serial port, pty. Để xác định loại driver hiện đang được nạp trong kernel và các thiết bị tty nào hiện có thì sử dụng tập tin /proc/tty/drivers. Tập tin này bao gồm một danh sách các tty driver khác nhau, hiển thị tên của driver, tên node mạc định, số chính cho driver, phạm vi hoạt động của driver và loại tty driver.

2.5.2. TTY function pointers

- open and close ( open function, close function)

* open function được gọi bởi tty core khi người dung gọi open trên device node- tty driver đã gán. Tty core gọi hàm này với một con trỏ đến tty\_struct và một con trỏ tập tin. Trường open phải được thiết lập bởi một tty driver để nó hoạt động đúng, nếu không –ENODEV được trả lại cho người dung khi open được gọi. Khi hàm open được gọi, tty driver sẽ lưu dữ liệu trong biến tty\_struct

- flow of data ( write function)

- other buffering function ( char\_in\_buffer function)

2.5.3. TTY structure

- tty\_Driver structure

- tty\_Operation structure

- tty\_Struct structure

2.6. Kết luận chương

**Chương 3: Communication Protocol ( CANopen)**

3.1. Giới thiệu chương

3.2. Các thành phần trong Communication Protocol

- Common Format

+

+

+

- USB Frame

+ for simulator

+ for checkalive

+ for config

3.3. Tiến trình thực hiện ( sequence)

3.4. Kết luận chương

• Tài liệu tham khảo: