|  |
| --- |
| HỌC VIỆN KỸ THUẬT MẬT MÃ  **KHOA CÔNG NGHỆ THÔNG TIN**  Logo HvKTMM  BÁO CÁO MÔN HỌC  **Lập Trình Nhân Linux**  *Nhóm:* ***03*** *Sinh viên thực hiện:*  Nguyễn Mạnh Cao Anh – CT060102  Lê Công Bảo Ngọc – CT060129  Nguyễn Đăng Tú – CT060142  Giảng viên hướng dẫn:  **ThS. Phạm Văn Hưởng**  Hà Nội, 2025 |
| HỌC VIỆN KỸ THUẬT MẬT MÃ  **KHOA CÔNG NGHỆ THÔNG TIN**  Logo HvKTMM  BÁO CÁO MÔN HỌC  **Lập Trình Nhân Linux**  *Nhóm:* ***03*** *Sinh viên thực hiện:*  Nguyễn Mạnh Cao Anh – CT060102  Lê Công Bảo Ngọc – CT060129  Nguyễn Đăng Tú – CT060142  Giảng viên hướng dẫn:  **ThS. Phạm Văn Hưởng**  Hà Nội, 2025 |

# MỤC LỤC

[MỤC LỤC i](#_Toc195367774)

[DANH MỤC HÌNH VẼ iv](#_Toc195367775)

[DANH MỤC BẢNG vi](#_Toc195367776)

[DANH MỤC CODE vii](#_Toc195367777)

[LỜI NÓI ĐẦU viii](#_Toc195367778)

[CHƯƠNG 1. CƠ SỞ LÝ THUYẾT 1](#_Toc195367779)

[1.1 File Managerment 1](#_Toc195367780)

[1.1.1 Phân tích thiết kế 1](#_Toc195367781)

[1.1.2 Thực thi 3](#_Toc195367782)

[1.2 Auto install and uninstall 6](#_Toc195367783)

[1.2.1 Phân tích thiết kế 6](#_Toc195367784)

[1.2.2 Thực thi 7](#_Toc195367785)

[1.3 Task management 8](#_Toc195367786)

[1.3.1 Phân tích thiết kế 8](#_Toc195367787)

[1.3.2 Thực thi 9](#_Toc195367788)

[1.4 Time management 11](#_Toc195367789)

[1.4.1 Phân tích thiết kế 11](#_Toc195367790)

[1.4.2 Thực thi 13](#_Toc195367791)

[CHƯƠNG 2. LẬP TRÌNH C 15](#_Toc195367792)

[2.1 File management 15](#_Toc195367793)

[2.1.1 Phân tích thiết kế 15](#_Toc195367794)

[2.1.2 Thực thi 17](#_Toc195367795)

[2.2 Network management 20](#_Toc195367796)

[2.3 Process management 24](#_Toc195367797)

[2.3.1 Phân tích thiết kế 24](#_Toc195367798)

[2.3.2 Thực thi 25](#_Toc195367799)

[2.4 Socket 27](#_Toc195367800)

[2.4.1 Lập trình Socket 27](#_Toc195367801)

[2.4.2 Thực thi 27](#_Toc195367802)

[CHƯƠNG 3. LẬP TRÌNH VÀ TÍCH HỢP KERNEL 36](#_Toc195367803)

[3.1 Tổng quan 36](#_Toc195367804)

[3.1.1 Các tiện ích để điều khiển module nhân: 37](#_Toc195367805)

[3.2 Macro 38](#_Toc195367806)

[3.3 Thực thi 38](#_Toc195367807)

[CHƯƠNG 4. INTERRUPTS 49](#_Toc195367808)

[4.1 Tổng quan 49](#_Toc195367809)

[4.1.1 Khái niệm 49](#_Toc195367810)

[4.1.2 Ngoại lệ (Exceptions) 50](#_Toc195367811)

[4.2 Các khái niệm phần cứng 51](#_Toc195367812)

[4.2.1 Programmable Interrupt Controller 51](#_Toc195367813)

[4.2.2 Interrupt controllers in SMP systems 52](#_Toc195367814)

[4.2.3 Interrupt Control (Điều khiển ngắt) 53](#_Toc195367815)

[4.2.4 Interrupt Priorities (Độ ưu tiên ngắt) 54](#_Toc195367816)

[4.2.5 Xử lý interrupt trên kiến trúc x86 54](#_Toc195367817)

[4.2.6 Interrupt Handler Address (Địa chỉ của trình xử lý ngắt) 56](#_Toc195367818)

[4.2.7 Ngăn xếp của chương trình xử lí ngắt 57](#_Toc195367819)

[4.2.8 Xử lí yêu cầu ngắt 58](#_Toc195367820)

[4.3 Xử lí ngắt trong Linux 58](#_Toc195367821)

[4.3.1 Nested interrupts and exceptions 59](#_Toc195367822)

[4.3.2 Ngữ cảnh ngắt 60](#_Toc195367823)

[4.3.3 Các hoạt động có thể trì hoãn 60](#_Toc195367824)

[4.3.4 Soft IRQs (Interrupt Requests mềm) 61](#_Toc195367825)

[4.3.5 Tasklets (Công việc lệnh) 61](#_Toc195367826)

[4.3.6 Workqueues (Hàng đợi công việc) 62](#_Toc195367827)

[4.3.7 Timers (Bộ định thời) 62](#_Toc195367828)

[4.4 Thực thi 63](#_Toc195367829)

[KẾT LUẬN 69](#_Toc195367830)

[TÀI LIỆU THAM KHẢO 70](#_Toc195367831)

# DANH MỤC HÌNH VẼ

[Hình 1.1 Usecase chức năng trong file management 1](#_Toc195367832)

[Hình 1.2 Biểu đồ usecase chức năng auto install và install 6](#_Toc195367833)

[Hình 1.3 Sơ đồ usecase biểu thị task management 8](#_Toc195367834)

[Hình 1.4 Biểu đồ usecase chức năng task management 11](#_Toc195367835)

[Hình 2.1 Usecase file management trong lập trình C 15](#_Toc195367836)

[Hình 2.2 Usecase process management trong lập trình C 24](#_Toc195367837)

[Hình 3.1 Mô tả nhân Linux 36](#_Toc195367838)

[Hình 3.2 Lệnh lsmod 37](#_Toc195367839)

[Hình 3.3 Lệnh insmod 37](#_Toc195367840)

[Hình 3.4 Lệnh modinfo 37](#_Toc195367841)

[Hình 3.5 Lệnh rmmod 38](#_Toc195367842)

[Hình 3.6 Kết quả sau khi biên dịch nhân 47](#_Toc195367843)

[Hình 3.7 Choice = 1 (Tính giai thừa) 47](#_Toc195367844)

[Hình 3.8 Choice = 2 (Tổng ma trận) 47](#_Toc195367845)

[Hình 3.9 Choice = 3 (Tích ma trận) 48](#_Toc195367846)

[Hình 3.10 Choice = 4 (Liệt kê các số nguyên tố trong khoảng p-q) 48](#_Toc195367847)

[Hình 3.11 Choice = 5 (Có bao nhiêu số nhỏ hơn s trong ma trận a) 48](#_Toc195367848)

[Hình 3.12 Choice = 6 (Có bao nhiêu số chia hết cho s trong ma trận) 48](#_Toc195367849)

[Hình 4.1 Hardware Concepts 51](#_Toc195367850)

[Hình 4.2 Interrupt controllers in SMP systems 53](#_Toc195367851)

[Hình 4.3 Interrupt Priorities 54](#_Toc195367852)

[Hình 4.4 IDT in x86 achitecture 55](#_Toc195367853)

[Hình 4.5 Một số trường của mục nhập IDT 56](#_Toc195367854)

[Hình 4.6 Interrupt Handler Address 57](#_Toc195367855)

[Hình 4.7 Stack of interrupt handler 57](#_Toc195367856)

[Hình 4.8 Interrupt handling in Linux 59](#_Toc195367857)

[Hình 4.9 Nested interrupts and exceptions 59](#_Toc195367858)

[Hình 4.10 Kết quả thực thi 68](#_Toc195367859)

# DANH MỤC BẢNG

[Bảng 1.1 Mô tả chức năng list file 1](#_Toc195367861)

[Bảng 1.2 Mô tả chức năng create file 2](#_Toc195367862)

[Bảng 1.3 Mô tả chức năng remove file 2](#_Toc195367863)

[Bảng 1.4 Mô tả chức năng show file 2](#_Toc195367864)

[Bảng 1.5 Mô tả chức năng edit file 2](#_Toc195367865)

[Bảng 1.6 Mô tả chức năng đổi tên files 3](#_Toc195367866)

[Bảng 1.7 Mô tả chức năng Copy file 3](#_Toc195367867)

[Bảng 1.8 Mô tả chức năng auto install 6](#_Toc195367868)

[Bảng 1.9 Mô tả chức năng auto uninstall 6](#_Toc195367869)

[Bảng 1.10 Mô tả chức năng list của task management 8](#_Toc195367870)

[Bảng 1.11 Mô tả chức năng create của task management 8](#_Toc195367871)

[Bảng 1.12 Mô tả chức năng remove của task management 9](#_Toc195367872)

[Bảng 1.13 Mô tả chức năng edit của task management 9](#_Toc195367873)

[Bảng 1.14 Mô tả chức năng info 12](#_Toc195367874)

[Bảng 1.15 Mô tả chức năng thay đổi giờ hệ thống 12](#_Toc195367875)

[Bảng 1.16 Mô tả chức năng thay đổi ngày tháng trong hệ thống 12](#_Toc195367876)

[Bảng 1.17 Mô tả chức năng tự động cập nhật thời gian hệ thống 12](#_Toc195367877)

[Bảng 2.1 Mô tả chức năng list file trong lập trình C 15](#_Toc195367878)

[Bảng 2.2 Mô tả chức năng create file ở trong lập trình C 16](#_Toc195367879)

[Bảng 2.3 Mô tả chức năng remove file ở trong lập trình C 16](#_Toc195367880)

[Bảng 2.4 Mô tả chức năng show file trong lập trình C 16](#_Toc195367881)

[Bảng 2.5 Mô tả chức năng list process ở trong lập trình C 25](#_Toc195367882)

[Bảng 2.6 Mô tả chức năng kill process ở trong lập trình C 25](#_Toc195367883)

# DANH MỤC CODE

[Code 1.1 Code thực thi file management 5](#_Toc195367884)

[Code 1.2 Code thực thi auto install and uninstall 7](#_Toc195367885)

[Code 1.3 Code thực thi task management 11](#_Toc195367886)

[Code 1.4 Code thực thi time management 14](#_Toc195367887)

[Code 2.1 Code thực thi file management trong C 19](#_Toc195367888)

[Code 2.2 Code thực thi process management ở trong lập trình C 26](#_Toc195367889)

[Code 2.3 Code Server Socket 32](#_Toc195367890)

[Code 2.4 Code Client Socket 35](#_Toc195367891)

[Code 3.1 Code module nhân đơn giản 45](#_Toc195367892)

[Code 3.2 Code Makefile 46](#_Toc195367893)

[Code 3.3 File Kbuild tạo object module 46](#_Toc195367894)

[Code 4.1 Makefile 63](#_Toc195367895)

[Code 4.2 Code Kbd\_driver.c 67](#_Toc195367896)

# LỜI NÓI ĐẦU

Báo cáo này được thực hiện trong khuôn khổ bài tập lớn môn Lập trình Nhân Linux, nhằm mục đích nghiên cứu và triển khai các kỹ thuật lập trình ở cấp độ thấp trên hệ điều hành Linux. Trong suốt quá trình thực hiện, nhóm chúng em đã triển khai thành công nhiều chức năng thiết yếu phục vụ việc tương tác với hệ thống, từ việc quản lý tệp tin đến các hoạt động liên quan đến module kernel.

Cụ thể, chúng em đã hoàn thiện hệ thống quản lý tệp tin cơ bản trong Shell, giúp thực hiện các thao tác như tạo, xóa, hiển thị, đổi tên và sao chép tệp tin. Bên cạnh đó, nhóm cũng đã xây dựng thành công chức năng tự động cài đặt và gỡ bỏ phần mềm (auto install và auto uninstall), mang lại sự thuận tiện cho người dùng khi làm việc với các module shell đã viết. Các chức năng quản lý tiến trình, bao gồm lập lịch, chỉnh sửa, liệt kê và xóa tác vụ qua crontab, cũng được hoàn thiện và đưa vào ứng dụng thực tế.

Ngoài ra, nhóm còn triển khai chức năng quản lý thời gian hệ thống, cho phép hiển thị thời gian, cài đặt thời gian thủ công và đồng bộ hóa thời gian với máy chủ mạng. Các bài toán tính toán trong module kernel cũng đã được thực hiện, bao gồm các phép toán như cộng ma trận, nhân ma trận, đếm số nguyên tố, và các phép toán đặc thù khác. Cuối cùng, nhóm đã hoàn thành việc chạy ví dụ về interrupts sử dụng workqueue trong môi trường kernel.

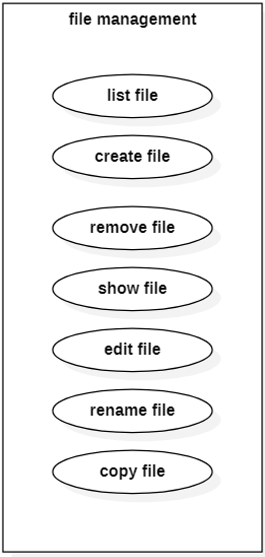
Mục tiêu của báo cáo là tìm hiểu quá trình thực hiện các chức năng nêu trên, cùng với kết quả đạt được, nhằm cung cấp cái nhìn toàn diện về lập trình nhân Linux và các ứng dụng thực tế của nó trong môi trường hệ thống.

# CƠ SỞ LÝ THUYẾT

## File Managerment

### Phân tích thiết kế

Đối với chức năng file management, chúng em đi tìm hiểu và phân tích ra bảy chức năng chính liên quan đến quản lý file bao gồm:



Hình 1.1 Usecase chức năng trong file management

|  |  |
| --- | --- |
| Chức năng | List file |
| Mô tả | Chức năng này sẽ tiến hành thực hiện hiển thị tất cả các file và thư mục có trong folder hiện tại. |
| Cách thực hiện | Thực hiện bằng cách sử dụng chương trình ls ở trong linux. |

Bảng 1.1 Mô tả chức năng list file

|  |  |
| --- | --- |
| Chức năng | Create file |
| Mô tả | Chức năng này sẽ tiến hành tạo ra file ở trong thư mục hiện tại và sử dụng tên được truyền vào làm tên file. |
| Cách thực hiện | Thực hiện bằng cách sử dụng chương trình touch ở trong linux |

Bảng 1.2 Mô tả chức năng create file

|  |  |
| --- | --- |
| Chức năng | Remove file |
| Mô tả | Chức năng này sẽ tiến hành xóa file ở trong thư mục hiện tại, tên file xóa sẽ được truyền vào như là một argument. |
| Cách thực hiện | Thực hiện bằng cách sử dụng chương trình rm ở trong linux |

Bảng 1.3 Mô tả chức năng remove file

|  |  |
| --- | --- |
| Chức năng | Show file |
| Mô tả | Chức năng này sẽ tiến hành hiển thị nội dung của file lên console. |
| Cách thực hiện | Thực hiện bằng cách sử dụng chương trình cat ở trong linux |

Bảng 1.4 Mô tả chức năng show file

|  |  |
| --- | --- |
| Chức năng | Edit file |
| Mô tả | Chức năng này sẽ tiến hành mở trình soạn thảo trên file, cho phép thực hiện chỉnh sửa nội dung file. |
| Cách thực hiện | Thực hiện bằng cách sử dụng chương trình vim ở trong linux. |

Bảng 1.5 Mô tả chức năng edit file

|  |  |
| --- | --- |
| Chức năng | Rename file |
| Mô tả | Chức năng này sẽ tiến hành đổi tên file. |
| Cách thực hiện | Thực hiện bằng cách sử dụng chương trình mv ở trong linux. |

Bảng 1.6 Mô tả chức năng đổi tên files

|  |  |
| --- | --- |
| Chức năng | Copy file |
| Mô tả | Chức năng này sẽ tiến hành copy file bất kỳ được chỉ định bằng tên file ở trong thư mục hiện tại và copy sang một thư mục khác. |
| Cách thực hiện | Thực hiện bằng cách sử dụng chương trình mv ở trong linux. |

Bảng 1.7 Mô tả chức năng Copy file

### Thực thi

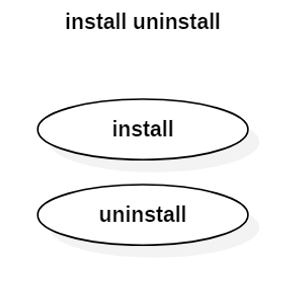
|  |
| --- |
| # Function to display the menu  show\_menu() {  echo "Menu:"  echo "-------------------------"  echo "1. List files"  echo "2. Create file"  echo "3. Remove file"  echo "4. Show file"  echo "5. Edit file"  echo "6. Rename file"  echo "7. Copy file"  echo "-------------------------"  echo "0. Exit"  echo "-------------------------"  }  # Function to get user input  get\_input() {  echo "Enter your choice:"  read choice}  # Loop to display the menu and get user input  while true  do  show\_menu  get\_input  case $choice in  1) command="list" ;;  2) command="create-file" ;;  3) command="remove-file" ;;  4) command="show-file" ;;  5) command="edit-file" ;;  6) command="rename-file" ;;  7) command="copy-file" ;;  0) exit ;;  \*) echo "Invalid choice" ;;  esac  # Execute the command based on user choice  if [ -n "$command" ]  then  if [ "$command" = "list" ]  then  ls -la  else  echo "Enter file name:"  read filename  if [ -z "$filename" ]  then  echo "Missing file name"  else  case "$command" in  "create-file") touch "$filename" ;;  "remove-file") rm -rf "$filename" ;;  "show-file") less "$filename" ;;  "edit-file") vim "$filename" ;;  "rename-file")  echo "Enter new file name:"  read new\_filename  if [ -z "$new\_filename" ]  then  echo "Missing new file name"  else  mv "$filename" "$new\_filename"  fi ;;  "copy-file")  echo "Enter destination file name:"  read destination\_filename  if [ -z "$destination\_filename" ]  then  echo "Missing destination file name"  else  cp "$filename" "$destination\_filename"  fi;;  esac  fi  command=""  fi  fi  done |

Code 1.1 Code thực thi file management

## Auto install and uninstall

### Phân tích thiết kế

Đối với chức năng auto install and uninstall, chúng em đi xây dựng hai chương trình đó là install và uninstall.



Hình 1.2 Biểu đồ usecase chức năng auto install và install

|  |  |
| --- | --- |
| Chức năng | Install |
| Mô tả | Chức năng install sẽ tiến hành đọc nội dung của file input.txt và cài đặt tất cả các package được liệt kê ở trong file. |
| Cách thực hiện | Đọc từng dòng tên các phần mềm được liệt kê ở trong file input.txt, sau đó dùng lệnh sudo apt-get với cờ -y để tiến hành cài đặt tất cả các gói. |

Bảng 1.8 Mô tả chức năng auto install

|  |  |
| --- | --- |
| Chức năng | Uninstall |
| Mô tả | Chức năng uninstall sẽ tiến hành đọc nội dung của file input.txt và gỡ cài đặt tất cả các chương trình được liệt kê |
| Cách thực hiện | Đọc từng dòng tên các phần mềm được liệt kê ở trong file input.txt, sau đó dùng lệnh sudo apt-get purge với cờ -y để tiến hành gỡ cài đặt tất cả các gói. |

Bảng 1.9 Mô tả chức năng auto uninstall

### Thực thi

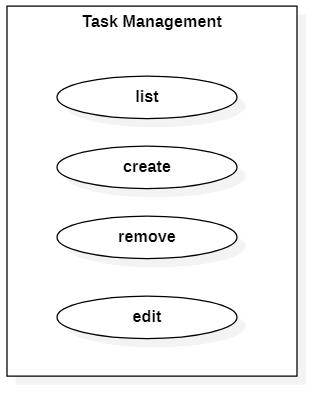
|  |
| --- |
| #!/bin/bash  function install\_packages {  while read package; do  brew install "$package"  done < "$1"  }  function uninstall\_packages {  while read package; do  brew uninstall "$package"  done < "$1"  }  if [ "$1" = "install" ]; then  install\_packages "test.txt"  echo "Packages have been installed successfully."  elif [ "$1" = "uninstall" ]; then  uninstall\_packages "test.txt"  echo "Packages have been uninstalled successfully."  else  echo "Invalid command. Please use 'install' or 'uninstall'."  exit 1  fi |

Code 1.2 Code thực thi auto install and uninstall

## Task management

### Phân tích thiết kế

Đối với task management chúng em tiến hành phân tích ra bốn chức năng lần lượt là liệt kê danh sách các task được lập lịch, tạo lập lịch tiến trình, xóa tất cả các lịch đã lập và cuối cùng là sửa đổi lịch tiến trình.



Hình 1.3 Sơ đồ usecase biểu thị task management

|  |  |
| --- | --- |
| Chức năng | List |
| Mô tả | Chức năng list sẽ tiến hành liệt kê những tiến trình đã được lập lịch bằng crontab. |
| Cách thực hiện | Sử dụng chương trình crontab để hiển thị danh sách các tiến trình được lập lịch. |

Bảng 1.10 Mô tả chức năng list của task management

|  |  |
| --- | --- |
| Chức năng | Create |
| Mô tả | Chức năng create sẽ tiến hành tạo lập lịch tiến trình tuỳ thuộc vào thông tin người dùng nhập vào. |
| Cách thực hiện | Sử dụng chương trình crontab để tạo lập lịch tiến trình |

Bảng 1.11 Mô tả chức năng create của task management

|  |  |
| --- | --- |
| Chức năng | Remove |
| Mô tả | Chức năng remove sẽ xoá tất cả những task đã được lập lịch. |
| Cách thực hiện | Sử dụng chương trình crontab để xóa tất cả những tiến trình đã được lập lịch. |

Bảng 1.12 Mô tả chức năng remove của task management

|  |  |
| --- | --- |
| Chức năng | Edit |
| Mô tả | Chức năng edit sẽ cho phép thực hiện chỉnh sửa những tiến trình đã được lập lịch. |
| Cách thực hiện | Sử dụng chương trình crontab và text editor để thực hiện chỉnh sửa nội dung lập lịch. |

Bảng 1.13 Mô tả chức năng edit của task management

### Thực thi

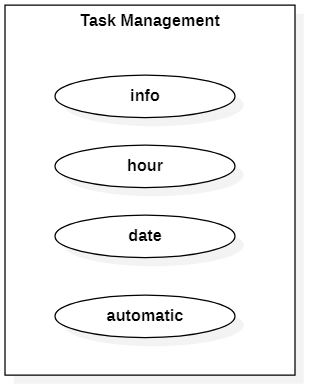
|  |
| --- |
| #!/bin/bash  # Liet ke tat ca cac tac vu  function list\_tasks {  crontab -l  }  # Tao mot tac vu moi  function create\_task {  read -p "Nhap vao cau lenh muon thuc hien: " command  read -p "Nhap vao thoi gian lich trinh (theo dinh dang \* \* \* \* \*): " schedule  (crontab -l ; echo "$schedule $command") | crontab -  }  # Sua mot tac vu  function edit\_task {  read -p "Nhap vao so thu tu cua tac vu muon sua: " task\_number  read -p "Nhap vao cau lenh muon thuc hien: " command  read -p "Nhap vao thoi gian lich trinh (theo dinh dang \* \* \* \* \*): " schedule  (crontab -l | sed -e "${task\_number}s/.\*/$schedule $command/") | crontab -  }  # Xoa mot tac vu  function delete\_task {  read -p "Nhap vao so thu tu cua tac vu muon xoa: " task\_number  (crontab -l | sed -e "${task\_number}d") | crontab -  }  # Menu chuc nang  while true  do  echo "1. Liet ke tat ca cac tac vu"  echo "2. Tao mot tac vu moi"  echo "3. Sua mot tac vu"  echo "4. Xoa mot tac vu"  echo "5. Thoat"  read -p "Nhap vao lua chon cua ban: " choice  case $choice in  1)  list\_tasks  ;;  2)  create\_task  ;;  3)  edit\_task  ;;  4)  delete\_task  ;;  5)  break  ;;  \*)  echo "Lua chon khong hop le"  ;;  esac  done |

Code 1.3 Code thực thi task management

## Time management

### Phân tích thiết kế

Chương trình time management được thiết kế thành bốn chức năng con bao gồm: hiện thông tin thời gian, cài đặt giờ, cài đặt ngày và tự động cập nhật ngày giờ.



Hình 1.4 Biểu đồ usecase chức năng task management

|  |  |
| --- | --- |
| Chức năng | Info |
| Mô tả | Hiển thị thời gian ngày tháng của hệ thống cũng như múi giờ. |
| Cách thực hiện | Sử dụng chương trình date để thực hiện xem giờ của hệ thống. |

Bảng 1.14 Mô tả chức năng info

|  |  |
| --- | --- |
| Chức năng | Hour |
| Mô tả | Chức năng hour được sử dụng để cập nhật lại thời gian giờ phút giây ở trong hệ thống. |
| Cách thực hiện | Sử dụng chương trình date để sửa giờ ở trong hệ thống. |

Bảng 1.15 Mô tả chức năng thay đổi giờ hệ thống

|  |  |
| --- | --- |
| Chức năng | Date |
| Mô tả | Chức năng hour được sử dụng để cập nhật lại thời gian ngày tháng năm ở trong hệ thống. |
| Cách thực hiện | Dùng chương trình date để sửa ngày tháng năm trong hệ thống. |

Bảng 1.16 Mô tả chức năng thay đổi ngày tháng trong hệ thống

|  |  |
| --- | --- |
| Chức năng | Automatic |
| Mô tả | Chức năng automatic được sử dụng để tự động cập nhật lại thời gian của hệ thống |
| Cách thực hiện | Sử dụng chương trình npdate để thực hiện tự động cập nhật lại thời gian của hệ thống. |

Bảng 1.17 Mô tả chức năng tự động cập nhật thời gian hệ thống

### Thực thi

|  |
| --- |
| #!/bin/bash  # Hiển thị thông tin thời gian hiện tại  function display\_time {  echo "Thời gian hiện tại là $(date +"%H:%M:%S")"  }  # Cài đặt giờ  function set\_time {  read -p "Nhập giờ mới (HH): " hour  read -p "Nhập phút mới (MM): " minute  read -p "Nhập ngày mới (dd): " day  read -p "Nhập tháng mới (mm): " month  read -p "Nhập năm mới (yyyy): " year  sudo systemsetup -setusingnetworktime off  sudo date $month$day$hour$minute$year  echo "Giờ mới đã được cài đặt."  }  # Tự động cập nhật ngày giờ  function auto\_update {  sudo systemsetup -setusingnetworktime on  echo "Ngày giờ đã được cập nhật tự động."  }  # Menu chức năng  while true  do  echo "1. Hiển thị thông tin thời gian"  echo "2. Cài đặt thời gian"  echo "3. Tự động cập nhật ngày giờ"  echo "4. Thoát"  read -p "Nhập lựa chọn của bạn: " choice  case $choice in  1)  display\_time  ;;  2)  set\_time  ;;  3)  auto\_update  ;;  4)  break  ;;  \*)  echo "Lựa chọn không hợp lệ."  ;;  esac  done |

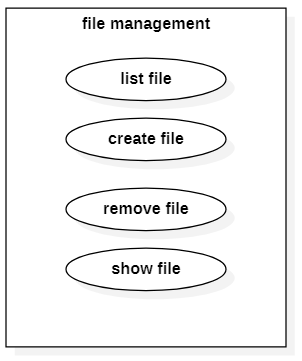
Code 1.4 Code thực thi time management

# LẬP TRÌNH C

## File management

### Phân tích thiết kế

Đối với chức năng file management, chúng em đi tìm hiểu và phân tích ra bảy chức năng chính liên quan đến quản lý file bao gồm:



Hình 2.1 Usecase file management trong lập trình C

|  |  |
| --- | --- |
| Chức năng | List file |
| Mô tả | Chức năng này sẽ tiến hành thực hiện hiển thị tất cả các file và thư mục có trong folder hiện tại. |
| Cách thực hiện | Sử dụng hàm diropen ở trong thư viện <dirent.h> để thực hiện đọc hết các file và folder trong thư mục được trỏ tới, tiếp theo sử dụng struct dirent để có thể lấy được tên file hoặc tên folder, khi đã có được tên rồi thì tiến hành in ra màn hình. |

Bảng 2.1 Mô tả chức năng list file trong lập trình C

|  |  |
| --- | --- |
| Chức năng | Create file |
| Mô tả | Chức năng này sẽ tiến hành tạo ra file ở trong thư mục hiện tại và sử dụng tên được truyền vào làm tên file. |
| Cách thực hiện | Thực hiện kiểm tra param xem đã truyền vào tên file chưa, nếu chưa sẽ báo lỗi. Khi đã có tên file, sử dụng hàm open ở trong thư viện <fcntl.h> với cờ là O\_CREAT kèm theo thuộc tính file là 0666. |

Bảng 2.2 Mô tả chức năng create file ở trong lập trình C

|  |  |
| --- | --- |
| Chức năng | Remove file |
| Mô tả | Chức năng này sẽ tiến hành xóa file ở trong thư mục hiện tại, tên file xóa sẽ được truyền vào như là một argument. |
| Cách thực hiện | Sử dụng hàm unlink ở trong thư viện <unistd.h> để thực hiện xóa file. |

Bảng 2.3 Mô tả chức năng remove file ở trong lập trình C

|  |  |
| --- | --- |
| Chức năng | Show file |
| Mô tả | Chức năng này sẽ tiến hành hiển thị nội dung của file lên console. |
| Cách thực hiện | Thực hiện bằng cách mở file với cờ O\_RDONLY và tiến hành đọc file và hiển thị nội dung file trên console đến hết file. |

Bảng 2.4 Mô tả chức năng show file trong lập trình C

### Thực thi

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <dirent.h>  void display\_files() {  struct dirent \*de;  DIR \*dr = opendir(".");  if (dr == NULL) {  printf("Could not open current directory");  return;  }  printf("List of files and directories:\n");  while ((de = readdir(dr)) != NULL) {  printf("%s\n", de->d\_name);  }  closedir(dr);  }  void create\_file(char\* filename) {  FILE\* fp = fopen(filename, "w");  if (fp == NULL) {  printf("Error creating file\n");  return;  }  printf("File created successfully\n");  fclose(fp);  }  void delete\_file(char\* filename) {  if (remove(filename) == 0) {  printf("File deleted successfully\n");  } else {  printf("Error deleting file\n");  }  }  void display\_file\_content(char\* filename) {  FILE\* fp = fopen(filename, "r");  if (fp == NULL) {  printf("Error opening file\n");  return;  }  char ch;  printf("Contents of file:\n");  while ((ch = fgetc(fp)) != EOF) {  printf("%c", ch);  }  fclose(fp);  }  int main() {  int choice;  char filename[100];  while(1) {  printf("Choose an option:\n");  printf("1. Display files and directories\n");  printf("2. Create a file\n");  printf("3. Delete a file\n");  printf("4. Display file content\n");  printf("5. Exit\n");  scanf("%d", &choice);  switch(choice) {  case 1:  display\_files();  break;  case 2:  printf("Enter file name: ");  scanf("%s", filename);  create\_file(filename);  break;  case 3:  printf("Enter file name: ");  scanf("%s", filename);  delete\_file(filename);  break;  case 4:  printf("Enter file name: ");  scanf("%s", filename);  display\_file\_content(filename);  break;  case 5:  exit(0);  default:  printf("Invalid choice\n");  }  }  return 0;  } |

Code 2.1 Code thực thi file management trong C

## Network management

Người dùng có thể chọn 4 chức năng để thao tác với network:

* List: hiển thị các giao diện mạng trong hệ thống
* Up: bật một giao diện mạng
* Down: tắt một giao diện mang
* Change: thay đổi địa chỉ ip giao diện mạng

Khai báo các thư viện cần thiết

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <unistd.h>  #include <errno.h>  #include <netdb.h>  #include <ifaddrs.h>  #include <sys/socket.h>  #include <net/if.h>  #include <sys/ioctl.h> |

**Liệt kê các giao diện mạng:**

Sử dụng struct ifaddr trong thư viện <ifaddrs.h> để có thể liệt kê được toàn bộ các network deveice trong máy. Thông tin được in ra bao gồm deveice name, protocol và address.

|  |
| --- |
| void list\_network\_interfaces() {  struct ifaddrs \*ifaddr; // giao dien mang  struct ifreq ifr; // ds mang  int family, s; // loai giao dien va ma loi  char host[NI\_MAXHOST];    if (getifaddrs(&ifaddr) == -1) {  perror("getifaddrs");  exit(EXIT\_FAILURE);  }  printf("Network Interface Information:\n");  for (struct ifaddrs \*ifa = ifaddr; ifa != NULL; ifa = ifa->ifa\_next) {  if (ifa->ifa\_addr == NULL)  continue;  family = ifa->ifa\_addr->sa\_family;  printf("%-8s %s (%d)\n",  ifa->ifa\_name,  (family == AF\_PACKET) ? "AF\_PACKET" :  (family == AF\_INET) ? "AF\_INET" :  (family == AF\_INET6) ? "AF\_INET6" : "???",  family);    if (family == AF\_INET || family == AF\_INET6) {  s = getnameinfo(ifa->ifa\_addr,  (family == AF\_INET) ? sizeof(struct sockaddr\_in) :  sizeof(struct sockaddr\_in6),  host, NI\_MAXHOST,  NULL, 0, NI\_NUMERICHOST);  if (s != 0) {  printf("getnameinfo() failed: %s\n", gai\_strerror(s));  exit(EXIT\_FAILURE);  }  printf("\t\taddress: <%s>\n", host);  } else if (family == AF\_PACKET && ifa->ifa\_data != NULL) {  printf("\t\tStatus: ");  strncpy(ifr.ifr\_name, ifa->ifa\_name, IFNAMSIZ);  int sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);  if (sockfd < 0) {  perror("socket");  exit(EXIT\_FAILURE);  }  if (ioctl(sockfd, SIOCGIFFLAGS, &ifr) == -1) {  perror("ioctl");  close(sockfd);  exit(EXIT\_FAILURE);  }  if (ifr.ifr\_flags & IFF\_UP) {  printf("UP\n");  } else {  printf("DOWN\n");  }  close(sockfd);  }  }  freeifaddrs(ifaddr);  } |

**Bật một giao diện mạng:**

Người dùng nhập tên giao diện mạng muốn bật, sau đó sử dụng lệnh “sudo ifconfig interface up” để bật giao diện mạng.

|  |
| --- |
| void enable\_interface(const char \*interface\_name) {  char command[100];  sprintf(command, "sudo ifconfig %s up", interface\_name);    if (system(command) == -1) {  perror("Error enabling interface");  exit(EXIT\_FAILURE);  }  printf("Interface %s has been enabled\n", interface\_name);  } |

**Tắt một giao diện mạng:**

Người dùng nhập tên giao diện mạng muốn tắt, sau đó sử dụng lệnh “sudo ifconfig interface down” để tắt giao diện mạng.

|  |
| --- |
| void disable\_interface(const char \*interface\_name) {  char command[100];  sprintf(command, "sudo ifconfig %s down", interface\_name);  if (system(command) == -1) {  perror("Error disabling interface");  exit(EXIT\_FAILURE);  }  printf("Interface %s has been disabled\n", interface\_name);  } |

**Thay đổi IP giao diện mạng:**

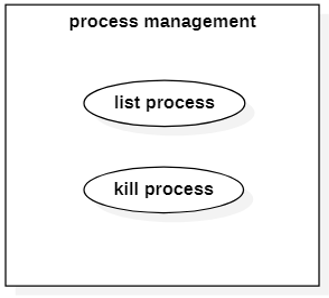
Người dùng nhập địa chỉ mới và tên interface. Trước tiên là tắt giao diện mạng, xóa bỏ tất cả ip động, cập nhật lại ip mới, bật lại giao diện mạng.

|  |
| --- |
| void change\_ip\_add(const char \*interface\_name){  char ip\_address[20]; // Địa chỉ IP mới  char command[100]; // Lệnh để thực thi  char commanddown[100];  char dhcp[100];  char deleteIp[100];  char up[100];  printf("Enter new IP address: ");  scanf("%s", ip\_address);  sprintf(command, "sudo ip addr add %s dev %s", ip\_address, interface\_name);  sprintf(commanddown, "sudo ifconfig %s down", interface\_name);  sprintf(dhcp, "sudo dhclient -r %s", interface\_name);  sprintf(deleteIp, "sudo ip addr flush dev %s", interface\_name);  sprintf(up, "sudo ifconfig %s up", interface\_name);  system(commanddown);  int i = 1;  int flag = 1;  while(i < 5){  if( system(dhcp)){  flag = 1;};  system(deleteIp);  system(command);  system(up);  i++;}  if(flag){  printf("success");  }else{  perror("Error executing command.\n");}  } |

## Process management

### Phân tích thiết kế

Chức năng process management bao gồm hai chức năng đó là liệt kê ra tất cả các process đang hoạt động và chức năng gửi signal tới từng process.



Hình 2.2 Usecase process management trong lập trình C

|  |  |
| --- | --- |
| Chức năng | List process |
| Mô tả | Chức năng này sẽ tiến hành hiển thị tất cả các process đang được thực thi ở trong máy. |
| Cách thực hiện | Sử dụng struct dirent và hàm opendir để mở thư mục /dir, những thư mục nào bắt đầu bằng số chính là các id biểu hiện tiến trình đang chạy, tiến hành đọc file status ở trong thư mục tiến trình để lấy thông tin tiến trình. |

Bảng 2.5 Mô tả chức năng list process ở trong lập trình C

|  |  |
| --- | --- |
| Chức năng | Kill process |
| Mô tả | Chức năng này sẽ tiến hành gửi signal tới process được chỉ định bằng id. |
| Cách thực hiện | Sử dụng hàm kill để gửi signal tới các process đang thực thi. |

Bảng 2.6 Mô tả chức năng kill process ở trong lập trình C

### Thực thi

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <signal.h>  #include <sys/types.h>  #include <sys/wait.h>  void list\_processes() {  printf("List of processes:\n");  system("ps -ax");  }  void signal\_process() {  pid\_t pid;  int sig;  printf("Enter process ID: ");  scanf("%d", &pid);  printf("Enter signal number: ");  scanf("%d", &sig);  if (kill(pid, sig) == -1) {  perror("Error sending signal");  } else { printf("Signal sent to process %d\n", pid); }  }  int main() {  int choice;  while (1) {  printf("\nProcess Management\n");  printf("1. List processes\n");  printf("2. Send signal to process\n");  printf("3. Exit\n");  printf("Enter your choice: ");  scanf("%d", &choice);  switch (choice) {  case 1:  list\_processes();  break;  case 2:  signal\_process();  break;  case 3:  exit(0);  default:  printf("Invalid choice\n");}}  return 0;} |

Code 2.2 Code thực thi process management ở trong lập trình C

## Socket

### Lập trình Socket

Tạo một ứng dụng nhắn tin đơn giản giữa client và server. Nhiều client và server trong cuộc trò chuyện. Server gửi tin nhắn dạng broadcast tới client, client sẽ gửi tin nhắn tới server, server có thể thấy tất cả tin nhắn của client gửi tới.

### Thực thi

**Code Server (WSL)**

|  |
| --- |
| #include <stdbool.h>  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <winsock2.h>  #include <ws2tcpip.h>  #pragma comment(lib, "ws2\_32.lib")  SOCKET createTCPIpv4Socket() {  return socket(AF\_INET, SOCK\_STREAM, 0);  }  struct sockaddr\_in\* createIPv4Address(char\* ip, int port) {  struct sockaddr\_in\* address = (struct sockaddr\_in\*)malloc(sizeof(struct sockaddr\_in));  address->sin\_family = AF\_INET;  address->sin\_port = htons(port);  if (strlen(ip) == 0)  address->sin\_addr.s\_addr = INADDR\_ANY;  else  inet\_pton(AF\_INET, ip, &(address->sin\_addr));  return address;  }  struct AcceptedSocket  {  int acceptedSocketFD;  struct sockaddr\_in address;  int error;  bool acceptedSuccessfully;  };  struct AcceptedSocket\* acceptIncomingConnection(int serverSocketFD);  void acceptNewConnectionAndReceiveAndPrintItsData(int serverSocketFD);  void receiveAndPrintIncomingData(int socketFD);  void startAcceptingIncomingConnections(int serverSocketFD);  DWORD WINAPI receiveAndPrintIncomingDataOnSeparateThread(LPVOID lpParam);  void sendReceivedMessageToTheOtherClients(char\* buffer, int socketFD);  struct AcceptedSocket acceptedSockets[10];  int acceptedSocketsCount = 0;  CRITICAL\_SECTION criticalSection;  void startAcceptingIncomingConnections(int serverSocketFD) {  while (true)  {  struct AcceptedSocket\* clientSocket = acceptIncomingConnection(serverSocketFD);  acceptedSockets[acceptedSocketsCount++] = \*clientSocket;  CreateThread(NULL, 0, receiveAndPrintIncomingDataOnSeparateThread, (LPVOID)clientSocket->acceptedSocketFD, 0, NULL);  }  }  DWORD WINAPI receiveAndPrintIncomingDataOnSeparateThread(LPVOID lpParam) {  int socketFD = (int)lpParam;  char buffer[1024];  while (true)  {  int amountReceived = recv(socketFD, buffer, 1024, 0);  if (amountReceived > 0)  {  buffer[amountReceived] = 0;  printf("%s\n", buffer);  sendReceivedMessageToTheOtherClients(buffer, socketFD);  }  if (amountReceived == 0)  break;  }  closesocket(socketFD);  ExitThread(0);  }  void receiveAndPrintIncomingData(int socketFD) {  char buffer[1024];  while (true)  {  int amountReceived = recv(socketFD, buffer, 1024, 0);  if (amountReceived > 0)  {  buffer[amountReceived] = 0;  printf("%s\n", buffer);  sendReceivedMessageToTheOtherClients(buffer, socketFD);  }  if (amountReceived == 0)  break;  }  closesocket(socketFD);  }  void sendReceivedMessageToTheOtherClients(char\* buffer, int socketFD) {  EnterCriticalSection(&criticalSection);  for (int i = 0; i < acceptedSocketsCount; i++)  {  if (acceptedSockets[i].acceptedSocketFD != socketFD)  {  send(acceptedSockets[i].acceptedSocketFD, buffer, strlen(buffer), 0);  }  }  LeaveCriticalSection(&criticalSection);  }  struct AcceptedSocket\* acceptIncomingConnection(int serverSocketFD) {  struct sockaddr\_in clientAddress;  int clientAddressSize = sizeof(struct sockaddr\_in);  int clientSocketFD = accept(serverSocketFD, (struct sockaddr\*)&clientAddress, &clientAddressSize);  struct AcceptedSocket\* acceptedSocket = (struct AcceptedSocket\*)malloc(sizeof(struct AcceptedSocket));  acceptedSocket->address = clientAddress;  acceptedSocket->acceptedSocketFD = clientSocketFD;  acceptedSocket->acceptedSuccessfully = clientSocketFD > 0;  if (!acceptedSocket->acceptedSuccessfully)  acceptedSocket->error = clientSocketFD;  return acceptedSocket;  }  int main() {  WSADATA wsaData;  int result = WSAStartup(MAKEWORD(2, 2), &wsaData);  if (result != 0)  {  printf("Failed to initialize Winsock. Error code: %d\n", result);  return 1;  }  int serverSocketFD = createTCPIpv4Socket();  struct sockaddr\_in\* serverAddress = createIPv4Address("", 2000);  int bindResult = bind(serverSocketFD, (struct sockaddr\*)serverAddress, sizeof(\*serverAddress));  if (bindResult == 0)  printf("Socket was bound successfully\n");  int listenResult = listen(serverSocketFD, 10);  InitializeCriticalSection(&criticalSection);  startAcceptingIncomingConnections(serverSocketFD);  shutdown(serverSocketFD, SD\_BOTH);  closesocket(serverSocketFD);  WSACleanup();  return 0;  } |

Code 2.3 Code Server Socket

**Code Client (Linux)**

|  |
| --- |
| #include <sys/types.h>  #include <sys/socket.h>  #include <netinet/in.h>  #include <stdio.h>  #include <string.h>  #include <stdlib.h>  #include <unistd.h>  #include <arpa/inet.h>  #include <pthread.h>  // Receives a message from a client.  void\* recvMsg(void\* sock)  {  int their\_sock = \*((int\*)sock);  char msg[500];  int len;  while ((len = recv(their\_sock, msg, 500, 0)) > 0)  {  msg[len] = '\0';  fputs(msg, stdout);  memset(msg, '\0', sizeof(msg));  }  }  int main(int argc, char\* argv[])  {  struct sockaddr\_in their\_addr;  pthread\_t sendt, recvt;  int my\_sock;  int their\_sock;  int their\_addr\_size;  int portno;  int len;  char msg[500];  char username[100];  char res[600];  char ip[INET\_ADDRSTRLEN];  if (argc > 3)  {  printf("too many arguments...");  exit(1);  }  portno = atoi(argv[2]);  strcpy(username, argv[1]);  my\_sock = socket(AF\_INET, SOCK\_STREAM, 0); // Creates client socket.  if (my\_sock < 0) // Validates the socket.  {  perror("opening channel unsuccessful...");  exit(1);  }  memset(their\_addr.sin\_zero, '\0', sizeof(their\_addr.sin\_zero));  their\_addr.sin\_family = AF\_INET; // Sets IPv4.  their\_addr.sin\_port = htons(portno); // Sets port number.  their\_addr.sin\_addr.s\_addr = inet\_addr("192.168.1.2"); // Sets IP address.  if (connect(my\_sock, (struct sockaddr\*) & their\_addr, sizeof(their\_addr)) < 0) // Connects to server and Validates the connection.  {  perror("connection not esatablished...");  exit(1);  }  inet\_ntop(AF\_INET, (struct sockaddr\*) & their\_addr, ip, INET\_ADDRSTRLEN);  printf("connected to %s, start chatting\n", ip);  pthread\_create(&recvt, NULL, recvMsg, &my\_sock);  while (fgets(msg, 500, stdin) > 0)  {  strcpy(res, username);  strcat(res, ": ");  strcat(res, msg);    len = write(my\_sock, res, strlen(res));    if (len < 0)  {  perror("message not sent...");  exit(1);  }  memset(msg, '\0', sizeof(msg));  memset(res, '\0', sizeof(res));  }  pthread\_join(recvt, NULL);  close(my\_sock);  } |

Code 2.4 Code Client Socket

# LẬP TRÌNH VÀ TÍCH HỢP KERNEL

## Tổng quan

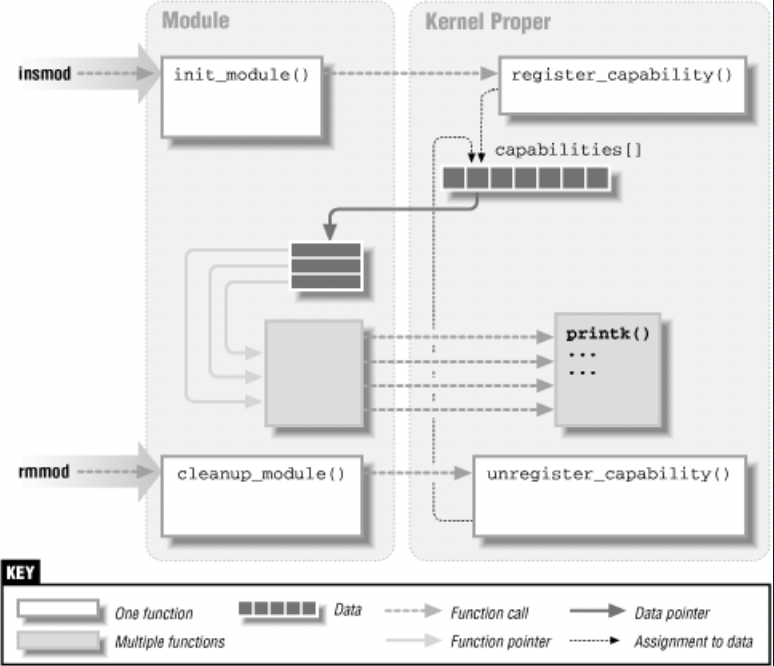
Nhân Linux là nền tảng của hệ điều hành giống Unix. Kernel chịu trách nhiệm giao tiếp giữa phần cứng và phần mềm và phân bổ các tài nguyên có sẵn.

Tất cả các bản phân phối Linux đều dựa trên một nhân được xác định trước. Tuy nhiên, nếu muốn tắt một số tùy chọn và trình điều khiển hoặc thử các bản vá thử nghiệm, chúng ta cần phải xây dựng một nhân Linux.

Mô-đun nhân là đoạn mã, có thể được tải và dỡ tải từ nhân theo yêu cầu.

Mô-đun nhân cung cấp một cách dễ dàng để mở rộng chức năng của hạt nhân cơ sở mà không cần phải xây dựng lại hoặc biên dịch lại hạt nhân. Hầu hết các trình điều khiển được triển khai dưới dạng mô-đun nhân Linux. Khi những trình điều khiển đó không cần thiết, chúng tôi chỉ có thể dỡ bỏ trình điều khiển cụ thể đó, điều này sẽ làm giảm kích thước hình ảnh hạt nhân.

Các mô-đun nhân sẽ có phần mở rộng .ko. Trên một hệ thống linux bình thường, các module nhân sẽ nằm bên trong thư mục **/lib/modules/<kernel\_version>/kernel/.**

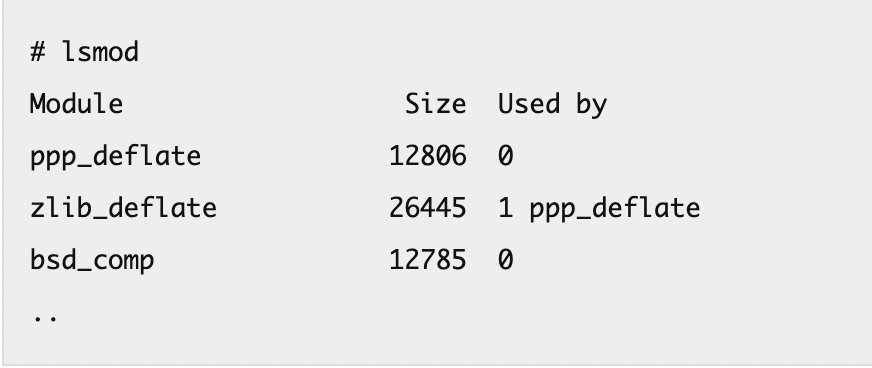


Hình 3.1 Mô tả nhân Linux

### Các tiện ích để điều khiển module nhân:

#### Lsmod

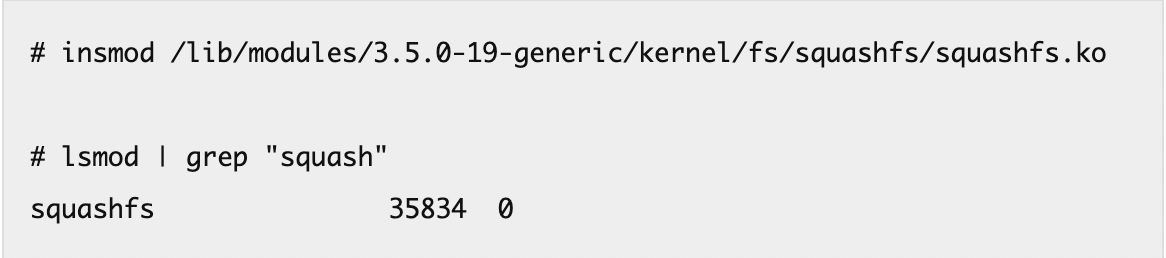
Lệnh lsmod sẽ liệt kê các mô-đun đã được tải trong nhân



Hình 3.2 Lệnh lsmod

#### Insmod

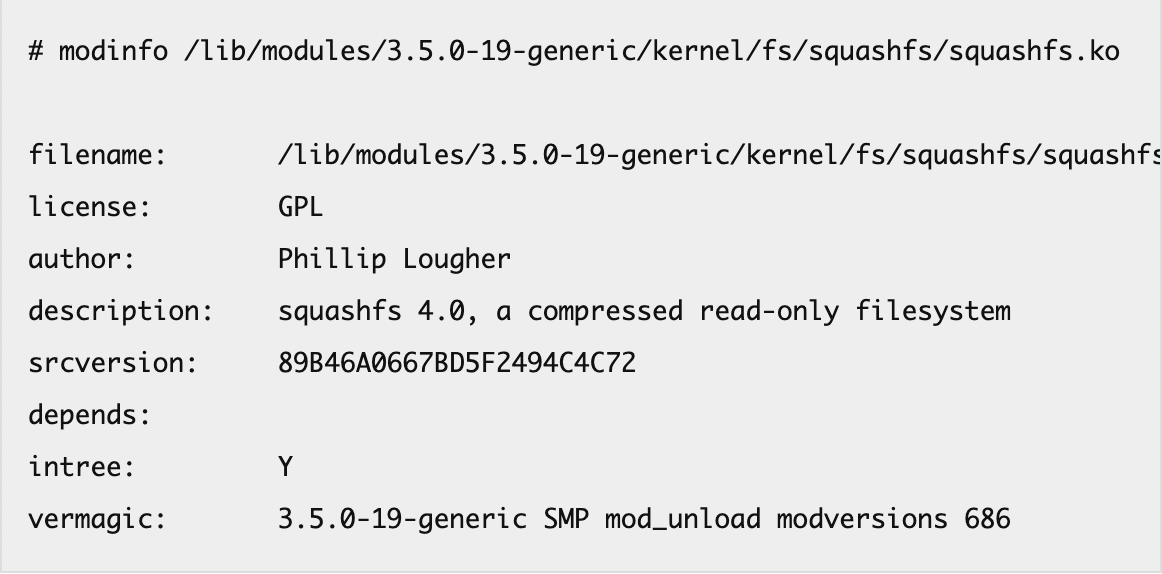
Lệnh insmode sẽ thêm một module mới vào nhân



Hình 3.3 Lệnh insmod

#### Modinfo

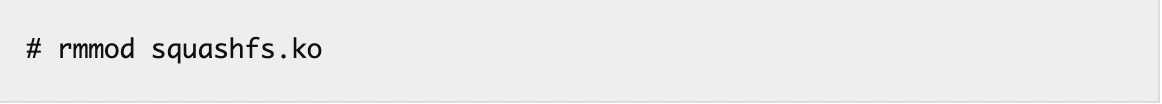
Lệnh modinfo sẽ hiển thị thông tin về module nhân



Hình 3.4 Lệnh modinfo

#### Rmmod

Lệnh rmmod sẽ xóa một module khỏi nhân. Bạn không thể xóa một module đã được sử dụng bởi bất kỳ chương trình nào.



Hình 3.5 Lệnh rmmod

#### Modeprobe

Lệnh modprobe là một lệnh thông minh sẽ tải / dỡ các mô-đun dựa trên sự phụ thuộc giữa các mô-đun.

## Macro

Macro chủ yếu được sử dụng để xác định mô-đun. Một macro quan trọng nhất là MODULE\_LICENSE(“GPL”). Nếu không xác định macro này, sẽ không thể sử dụng bất kỳ thư viện hệ thống nào khác đang chạy theo giấy phép GPL. Đối với mẫu đơn giản nhất này, đây sẽ không phải là vấn đề lớn, nhưng khi phát triển trình điều khiển hạt nhân, v.v., đây có thể là một vấn đề lớn. Vì vậy, hãy luôn nhớ giữ giấy phép GPL nếu cần lấy  các thư viện GPL được cấp phép.

Ngoài ra, nếu phát hành mô-đun, thì phải chọn cấp phép thích hợp cho mô-đun và trong trường hợp đó, sẽ phải sử dụng liên kết tĩnh đến GPL các thư viện được cấp phép.

## Thực thi

Một driver không thể tự nó thực thi mà hoạt động tương tự như một thư viện được nạp và đăng ký các hàm bởi một ứng dụng đang chạy. Driver được viết bằng C, nhưng không có hàm main(). Hơn nữa, bởi vì driver được nạp và liên kết với hệ điều hành, nên nó cần được biên dịch giống cách biên dịch nhân hệ điều hành, và các header files được sử dụng trong mã nguồn driver chỉ là những cái mà nhân hệ điều hành cung cấp, không bao giờ có các hàm của thư viện lập trình C (mà thường để trong thư mục /usr/include).

Một điểm thú vị khác là mã nguồn nhân được lập trình theo kiểu hướng đối tượng trong C, mã nguồn driver cũng tương tự như vậy. Bất kỳ một driver nào trên Linux đều có một hàm tạo (constructor) và một hàm hủy (destructor).

Hàm tạo của driver được gọi khi driver được nạp vào nhân hệ thống, và hàm hủy được gọi khi gỡ driver khỏi hệ thống (dùng lệnh rmmod). Hai hàm này cũng giống như các hàm thông thường, ngoại trừ việc chúng cần có các chỉ thị \_init và \_exit tương ứng và phải được đăng ký sử dụng các macros module\_init() và module\_exit(), (được định nghĩa trong tệp tiêu đề module.h)

Mô tả chương trình: Xây dựng một module nhân giúp tính giải các bài toán (tính giai thừa, cộng và nhân ma trận,…)

**Code file test.c**

|  |
| --- |
| #include <linux/module.h>  #include <linux/moduleparam.h>  #include <linux/kernel.h>  #include <linux/init.h>  #include <linux/stat.h>  MODULE\_LICENSE("GPL");  MODULE\_AUTHOR("Nhom3");  MODULE\_DESCRIPTION("A Linux Kernel Module to perform operations on matrices");  int factorial(int n);  int matmul(int p, int q, int s, int a[p\*q], int b[q\*s], int c[p\*s]);  int matadd(int p, int q, int a[p\*q], int b[p\*q]);  int primeBetween(int m, int n);  int smallernuminmatrix(int p, int q, int a[p\*q], int s);  int numdivisibleinmatrix(int p, int q, int a[p\*q], int s);  int primeofmat(int p, int q, int a[p\*q]);  int prime(int n);  static int choice = 0;  static int p, q, s, d, l;  static int a[2500], b[2500], c[2500];  module\_param(choice, int, S\_IRUGO);  module\_param(d, int, S\_IRUGO);  module\_param(p, int, S\_IRUGO);  module\_param(q, int, S\_IRUGO);  module\_param(s, int, S\_IRUGO);  module\_param\_array(a, int, NULL, S\_IRUGO);  module\_param\_array(b, int, NULL, S\_IRUGO);  module\_param\_array(c, int, NULL, S\_IRUGO);  static int p02\_init(void)  {  switch (choice)  {  case 1:  {  printk(KERN\_ALERT "\n%d! = %d", d, factorial(d));  break;  }  case 2:  {  matadd(p, q, a, b);  break;  }  case 3:  {  matmul(p, q, s, a, b, c);  break;  }  case 4:  {  primeBetween(p, s);  break;  }  case 5:  {  smallernuminmatrix(p, q, a, s);  break;  }  case 6:  {  numdivisibleinmatrix(p, q, a, s);  break;  }  case 7:  {  primeofmat(p, q, a);  break;  }  }  return 0;  }  int factorial(int n)  {  int gt = 2;  int i = 3;  for(i=3; i<=n; i++)  {  gt \*= i;  }  return gt;  }  int matadd(int p, int q, int a[p\*q], int b[p\*q]) {  int i, j;  for (i = 0; i < p; i++) {  for (j = 0; j < q; j++) {  \*(c + j + i \* q) = \*(a + j + i \* q) + \*(b + j + i \* q);  //c[i][j] = a[i][j] + b[i][j];  // \*(a + j + i \* q) += \*(b + j + i \* q);  }  }  printk(KERN\_ALERT "\n------------------------------\nThe sum of 2 matrices is a matrix with height %d and width %d\n",p,q);  for (i=0;i<p;i++){  for (j=0;j<q;j++){  printk(KERN\_ALERT "%-7d", \*(c + (q \* i) + j));  }  printk(KERN\_ALERT "\n");  }  return 0;  }    int matmul(int p, int q, int s, int a[p\*q], int b[q\*s], int c[p\*s]) {  int i, j;  for (i=0;i<p;i++){  for (j=0;j<s;j++){  for (l=0;l<q;l++){  //c[i][j] = c[i][j] + a[i][l] \* b[l][j];  \*(c + (s \* i) + j) = \*(c + (s \* i) + j) + \*(a + (q \* i) + l) \* \*(b + (l \* s) + j);  }  }  }  printk(KERN\_ALERT "\n------------------------------\nThe product of 2 matrices is a matrix with height %d and width %d\n",p,s);  for (i=0;i<p;i++){  for (j=0;j<s;j++){  printk(KERN\_ALERT "%-7d", \*(c + (s \* i) + j));  }  printk(KERN\_ALERT "\n");  }  return 0;  }  int primeBetween(int num1, int num2) {  int flag\_var, i, j;  for (i = num1 + 1; i < num2; ++i) {  flag\_var = 0;  for (j = 2; j <= i / 2; ++j) {  if (i % j == 0) {  flag\_var = 1;  break;  }  }  if (flag\_var == 0) printk(KERN\_ALERT "%d\n",i);  }  return 0;  }  int smallernuminmatrix(int p, int q, int a[p\*q], int s) {  int i, j, count = 0;  for (i = 0; i < p; i++) {  for (j = 0; j < q; j++) {  if (a[j + q \* i] < s) {  count++;  }  }  }  printk(KERN\_ALERT "There are %d numbers smaller than %d in this matrix", count, s);  return 0;  }  int numdivisibleinmatrix(int p, int q, int a[p\*q], int s) {  int i, j, count = 0;  for (i = 0; i < p; i++) {  for (j = 0; j < q; j++) {  if (a[j + q \* i] % s == 0) {  count++;  }  }  }  printk(KERN\_ALERT "There are %d numbers divisible by %d in this matrix", count, s);  return 0;  }  int primeofmat(int p, int q, int a[p\*q]) {  int i, j, count = 0;  for (i = 0; i < p; i++) {  for (j = 0; j < q; j++) {  count += prime(a[j + q \* i]);  }  }  printk(KERN\_ALERT "There are %d prime numbers in this matrix", count);  return 0;  }  int prime(int n) {  if (n <= 1) {  return 0;  }  int i;  for (i = 2; i\*i <= n; i++) {  if (n % i == 0) {  return 0;  }  }  return 1;  }  void display(int p, int q, int a[p][q]) {  int i, j;  for (i = 0; i < p; i++) {  for (j = 0; j < q; j++) {  printk(KERN\_ALERT "%d ", a[i][j]);  }  printk("\n");  }  }  static void p02\_exit(void)  {  printk(KERN\_ALERT "Goodbye\n");  }  module\_init(p02\_init);  module\_exit(p02\_exit); |

Code 3.1 Code module nhân đơn giản

Mã nguồn driver được lưu thành file test.c. Chú ý rằng trong mã nguồn sẽ không cho phép các thư viện ở tầng ứng dụng như stdio.h, thay vì đó sử dụng các thư viện tầng nhân như kernel.h. Hàm printk() tương đương như printf() tuy nhiên dữ liệu không xuất ra thiết bị ra chuẩn mà xuất ra log file của kernel và phải sử dụng các công cụ đặc biệt để mở. (Ví dụ lệnh dmesg hoặc công cụ Log File Viewer trên Ubuntu). Tệp tiêu đề version.h bao gồm thông tin phiên bản module tương thích với nhân mà module sẽ nạp vào. Các macros MODULE\_\* cung cấp các thông tin liên quan đến module, đóng vai trò như chữ ký cho module đó.

**Makefile** và **Kbuild** để tạo module nhân tương ứng.

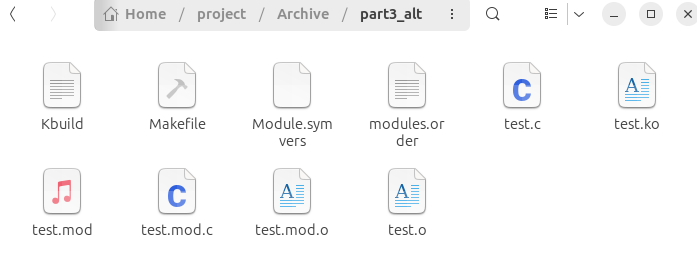
|  |
| --- |
| obj-m += test.o  KDIR := /lib/modules/$(shell uname -r)/build  PWD := $(shell pwd)  default:   $(MAKE) -C $(KDIR) M=$(PWD) modules  clean:  $(MAKE) -C $(KDIR) M=$(PWD) clean |

Code 3.2 Code Makefile

|  |
| --- |
| *EXTRA\_CFLAGS = -Wall*  *obj-m = test.o* |

Code 3.3 File Kbuild tạo object module

Sử dụng lệnh **make** để biên dịch module nhân test.



Hình 3.6 Kết quả sau khi biên dịch nhân

Ta sẽ tạo được tệp test.ko, đó chính là module Kernel mẫu chúng ta vừa tạo ra

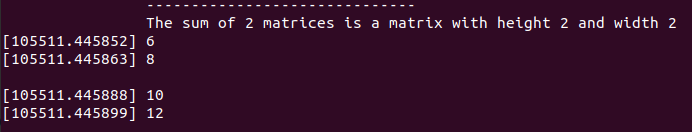
**Các option được đưa ra:**

* sudo insmod test.ko choice=1 d=5



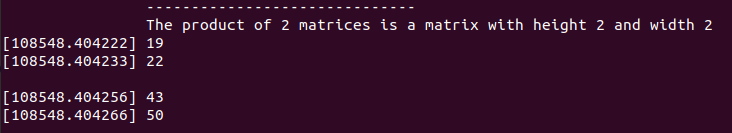
Hình 3.7 Choice = 1 (Tính giai thừa)

* sudo insmod test.ko choice=2 p=2 q=2 a=1,2,3,4 b=5,6,7,8



Hình 3.8 Choice = 2 (Tổng ma trận)

* sudo insmod test.ko choice=3 p=2 q=2 s=2 a=1,2,3,4 b=5,6,7,8



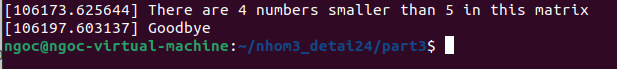
Hình 3.9 Choice = 3 (Tích ma trận)

* sudo insmod test.ko choice=4 p=10 s=30



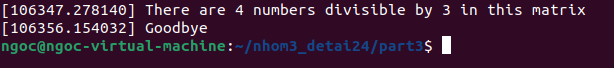
Hình 3.10 Choice = 4 (Liệt kê các số nguyên tố trong khoảng p-q)

* sudo insmod test.ko choice=5 p=2 q=3 s=5 a=1,2,3,4,5,6



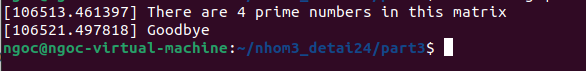
Hình 3.11 Choice = 5 (Có bao nhiêu số nhỏ hơn s trong ma trận a)

* sudo insmod test.ko choice=6 p=2 q=3 s=3 a=3,6,7,9,10,12



Hình 3.12 Choice = 6 (Có bao nhiêu số chia hết cho s trong ma trận)

* sudo insmod test.ko choice=7 p=2 q=3 a=2,3,4,5,6,7

  
Hình 3.15: Choice = 7 (Có bao nhiêu số nguyên tố trong ma trận)

# INTERRUPTS

## Tổng quan

### Khái niệm

Interrupt là một sự kiện làm thay đổi luồng thực thi bình thường của một chương trình và có thể được tạo ra bởi các thiết bị phần cứng hoặc bởi chính CPU. Khi một interrupt xảy ra, luồng thực thi hiện tại được tạm dừng và trình xử lý interrupt chạy. Sau khi trình xử lý interrupt chạy xong, luồng thực thi trước đó được tiếp tục.

Các interrupt có thể được nhóm thành hai loại dựa trên nguồn gốc của interrupt:

* Synchronous Interrupt (Ngắt đồng bộ): được tạo ra bằng cách thực thi một lệnh.
* Synchronous Interrupt (Ngắt không đồng bộ): được tạo ra bởi một sự kiện bên ngoài.

Các interrupt có thể được phân thành hai loại khác nhau dựa trên khả năng hoãn lại hoặc vô hiệu hóa tạm thời:

* Maskable Interrupt (Có thể vô hiệu hóa): có thể bị bỏ qua và được tạo ra thông qua chân INT.
* Non-maskable Interrupt (không thể vô hiệu hóa): không thể bị bỏ qua và được tạo ra thông qua chân NMI.

Synchronous Interrupt (Interrupt đồng bộ), thường được gọi là Exception (ngoại lệ), xử lý các điều kiện phát hiện bởi bộ xử lý trong quá trình thực thi lệnh. Ví dụ về ngoại lệ bao gồm chia cho không (Divide by zero) hoặc lời gọi hệ thống (System calls).

Synchronous Interrupt (Interrupt không đồng bộ), thường được gọi là interrupt, là các sự kiện bên ngoài được tạo ra bởi các thiết bị I/O. Ví dụ, một card mạng tạo ra interrupt để báo hiệu rằng một gói tin đã đến.

Hầu hết các interrupt có thể bị vô hiệu hóa, điều này có nghĩa là chúng có thể tạm thời hoãn việc chạy trình xử lý interrupt khi interrupt bị vô hiệu hóa đến khi interrupt được kích hoạt lại. Nhưng có một số interrupt quan trọng không thể bị vô hiệu hóa hoặc hoãn lại.

### Ngoại lệ (Exceptions)

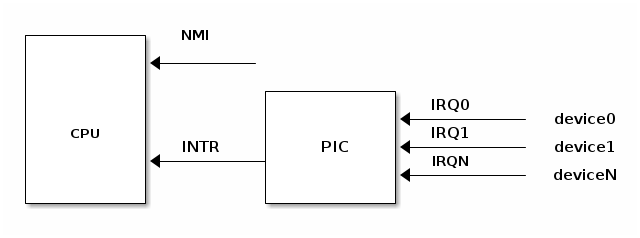
Có hai nguồn gốc cho các ngoại lệ:

* Processor detected (do bộ xử lý phát hiện):
* Faults (Lỗi): Đây là loại ngoại lệ được báo cáo trước khi thực thi lệnh và thường có thể được sửa chữa. EIP (Instruction Pointer) được lưu trữ là địa chỉ của lệnh gây ra lỗi, vì vậy sau khi lỗi được sửa chữa, chương trình có thể thực thi lại lệnh gây lỗi. Ví dụ về lỗi là page fault (sự cố trang).
* Traps (Cạm bẫy): Đây là loại ngoại lệ được báo cáo sau khi thực thi lệnh trong đó ngoại lệ đã được phát hiện. EIP được lưu trữ là địa chỉ của lệnh sau lệnh gây ra cạm bẫy. Ví dụ về cạm bẫy là debug trap (cạm bẫy gỡ lỗi).
* Aborts (Hủy bỏ): Đây là loại ngoại lệ được sinh ra khi gặp phải một lỗi nghiêm trọng khiến hệ thống không thể tiếp tục thực thi một cách an toàn. Không giống như faults hay traps, aborts thường không thể được xử lý để tiếp tục chương trình, và do đó chương trình bị dừng ngay lập tức. EIP (Instruction Pointer) không được đảm bảo là chính xác, vì hệ thống không thể xác định chính xác lệnh nào đã gây ra lỗi. Một ví dụ điển hình là machine check abort (hủy bỏ do kiểm tra máy) hoặc parity error trong bộ nhớ, thường xảy ra do lỗi phần cứng như hư RAM hoặc lỗi bus hệ thống.
* Programmed (nguồn gốc từ lập trình):
* INT n: Đây là loại ngoại lệ được tạo ra bởi lệnh INT trong mã chương trình. Ngoại lệ này xảy ra khi một lệnh INT được thực thi và nó gọi một trình xử lý interrupt được định nghĩa trước đó để xử lý ngoại lệ cụ thể. Giá trị n xác định loại interrupt cụ thể được gọi.

## Các khái niệm phần cứng

### Programmable Interrupt Controller

(Bộ điều khiển Interrupt có thể lập trình)



Hình 4.1 Hardware Concepts

Một thiết bị hỗ trợ các interrupt có một chân đầu ra được sử dụng để thông báo về yêu cầu Interrupt (IRQ). Các chân IRQ được kết nối đến một thiết bị được gọi là Bộ điều khiển Interrupt có thể Lập trình (PIC), mà nó kết nối đến chân INTR trên CPU.

Một PIC thông thường có một tập hợp các cổng được sử dụng để trao đổi thông tin với CPU. Khi một thiết bị kết nối với một trong các dòng IRQ của PIC cần sự chú ý của CPU, quá trình sau xảy ra:

1. Thiết bị tạo ra một interrupt trên chân IRQn tương ứng.
2. PIC ánh xạ IRQ thành một vector và ghi nó vào một cổng để CPU đọc.
3. PIC tạo ra một interrupt trên chân INTR của CPU.
4. PIC chờ đợi CPU xác nhận một interrupt trước khi tạo ra một interrupt khác.
5. CPU xác nhận interrupt, sau đó nó bắt đầu xử lý interrupt.

Sau khi interrupt được CPU xác nhận, bộ điều khiển interrupt có thể yêu cầu một interrupt khác, bất kể CPU đã hoàn thành xử lý interrupt trước đó hay chưa. Do đó, tùy thuộc vào cách hệ điều hành điều khiển CPU, có thể xảy ra nested interrupts (interrupt lồng nhau).

Bộ điều khiển interrupt cho phép tắt từng dòng IRQ một cách riêng lẻ. Điều này giúp đơn giản hóa thiết kế bằng cách đảm bảo rằng các trình xử lý interrupt luôn được thực thi theo chuỗi.

### Interrupt controllers in SMP systems

(Bộ điều khiển ngắt trong hệ thống đa xử lý (SMP))

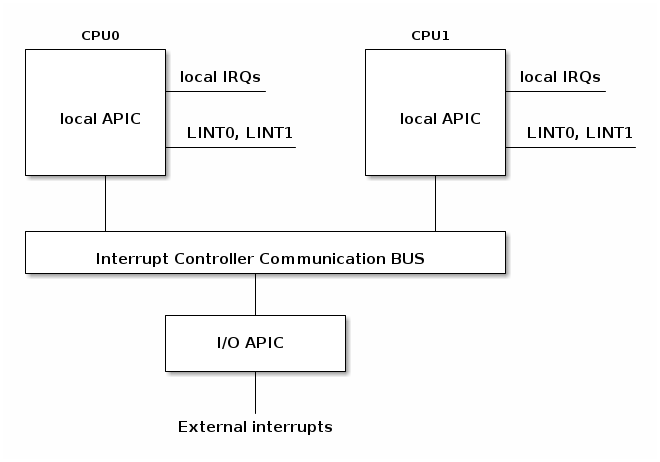
Các bộ điều khiển Interrupt trong các hệ thống SMP (Symmetric Multiprocessing - đa xử lý đối xứng) là các thành phần quan trọng để quản lý và phân phối các interrupt trong các hệ thống máy tính có nhiều bộ xử lý.

Trong hệ thống SMP, có nhiều bộ xử lý hoạt động đồng thời và có thể nhận và xử lý các interrupt độc lập. Để đảm bảo sự đồng bộ và hiệu suất cao, các bộ điều khiển Interrupt trong hệ thống SMP phải được thiết kế để phân phối các interrupt một cách công bằng và hiệu quả giữa các bộ xử lý.

Các bộ điều khiển Interrupt trong hệ thống SMP thường sử dụng các thuật toán và giao thức phân phối interrupt phức tạp để đảm bảo rằng các interrupt được gửi tới bộ xử lý phù hợp và tránh tình trạng xung đột interrupt (interrupt collision).

Một số hệ thống SMP có thể sử dụng mô hình bộ điều khiển Interrupt tập trung (centralized interrupt controller), trong đó có một bộ điều khiển Interrupt chính quản lý và phân phối các interrupt cho các bộ xử lý khác nhau. Trong khi đó, một số hệ thống SMP có thể sử dụng mô hình bộ điều khiển Interrupt phân tán (distributed interrupt controller), trong đó các bộ điều khiển Interrupt riêng biệt được phân phối trên các bộ xử lý để quản lý các interrupt cục bộ.

Quá trình quản lý interrupt và phân phối trong hệ thống SMP đòi hỏi sự đồng bộ và điều phối chặt chẽ giữa các bộ xử lý để đảm bảo tính nhất quán và hiệu suất hệ thống. Các bộ điều khiển Interrupt trong hệ thống SMP đóng vai trò quan trọng trong việc đảm bảo sự hoạt động ổn định và hiệu quả của hệ thống đa xử lý.



Hình 4.2 Interrupt controllers in SMP systems

### Interrupt Control (Điều khiển ngắt)

Để đồng bộ hóa quyền truy cập vào dữ liệu chia sẻ giữa trình xử lý ngắt và các hoạt động đồng thời khác như khởi tạo trình điều khiển hoặc xử lý dữ liệu của trình điều khiển, thường cần kích hoạt và vô hiệu hóa ngắt theo một cách điều khiển.

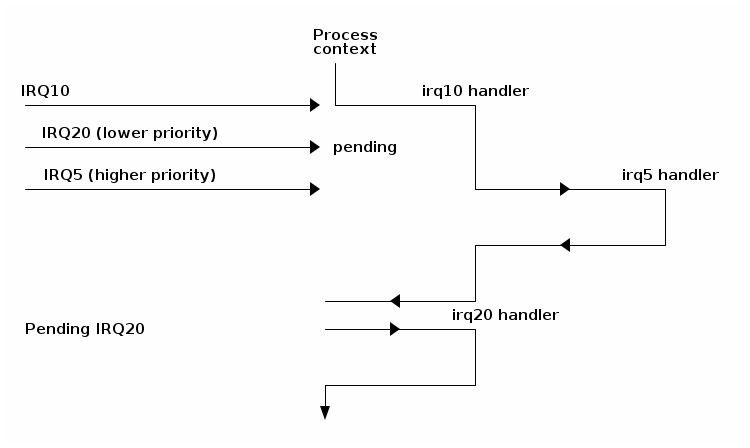
Điều này có thể được thực hiện ở một số mức:

* Mức thiết bị: bằng cách lập trình các thanh ghi điều khiển thiết bị
* Mức PIC: Bộ điều khiển ngắt có thể được lập trình để vô hiệu hóa một dòng IRQ cụ thể.
* Mức CPU: Ví dụ trên kiến trúc x86, có thể sử dụng các lệnh sau:
* sti (SeT Interrupt flag): Kích hoạt cờ ngắt, cho phép CPU nhận các ngắt.
* cli (CLear Interrupt flag): Vô hiệu hóa cờ ngắt, ngăn CPU nhận các ngắt mới.

Bằng cách điều khiển việc kích hoạt và vô hiệu hóa ngắt theo cách điều khiển, ta có thể quản lý quyền truy cập đồng bộ vào dữ liệu chia sẻ và đảm bảo tính nhất quán và an toàn trong hệ thống.

### Interrupt Priorities (Độ ưu tiên ngắt)

Đa số kiến trúc cũng hỗ trợ mức ưu tiên của ngắt (interrupt priorities). Khi chức năng này được kích hoạt, nó cho phép lồng ngắt chỉ cho những ngắt có mức ưu tiên cao hơn mức ưu tiên hiện tại.



Hình 4.3 Interrupt Priorities

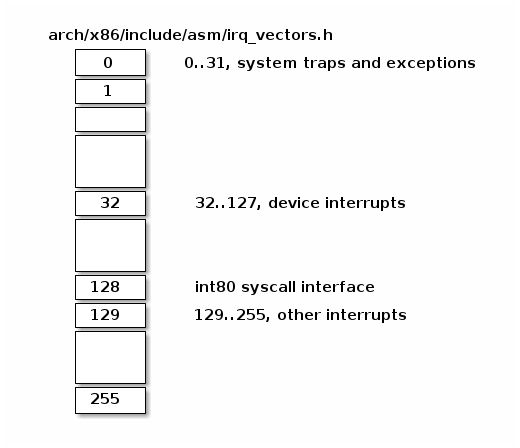
### Xử lý interrupt trên kiến trúc x86

**Bảng mô tả ngắt (IDT)**

Bảng mô tả ngắt (Interrupt Descriptor Table - IDT) là bảng ánh xạ mỗi vector ngắt hoặc ngoại lệ đến một trình xử lý tương ứng. Chúng ta sẽ gọi mã ngắt hoặc ngoại lệ là số vector và các chỉ thị xử lý tương ứng là trình xử lý ngắt/ngoại lệ.

IDT có các đặc điểm sau:

* Nó được sử dụng như một bảng nhảy bởi CPU khi một vector cụ thể được kích hoạt.
* Nó là một mảng gồm 256 mục x 8 byte.
* IDT có thể nằm ở bất kỳ vị trí nào trong bộ nhớ vật lý.
* Bộ xử lý xác định vị trí của IDT thông qua IDTR (Interrupt Descriptor Table Register).



Hình 4.4 IDT in x86 achitecture

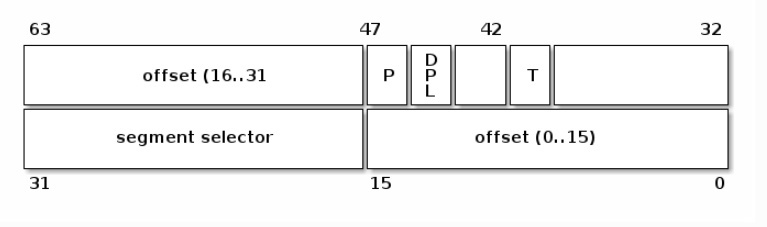
Trên x86, một mục trong IDT có kích thước 8 byte và được gọi là gate (cổng).   
Có 3 loại cổng:

* Interrupt Gate (Cổng ngắt): Lưu trữ địa chỉ của trình xử lý ngắt hoặc ngoại lệ. Khi nhảy tới trình xử lý, cổng này vô hiệu hóa các ngắt có thể vô hiệu hóa (cờ IF được xóa).
* Trap Gate (Cổng trap): Tương tự như cổng ngắt, nhưng không vô hiệu hóa các ngắt có thể vô hiệu hóa khi nhảy tới trình xử lý ngắt/ ngoại lệ.
* Task Gate (Cổng tác vụ): Không được sử dụng trong Linux.

Các loại cổng này được sử dụng để xác định phản ứng của hệ thống khi có ngắt hoặc ngoại lệ xảy ra. Chúng lưu trữ địa chỉ của trình xử lý tương ứng để xử lý sự kiện liên quan. Tùy vào loại cổng, có thể vô hiệu hóa ngắt có thể vô hiệu hóa hoặc giữ nguyên trạng thái của chúng trong quá trình xử lý.

Chúng ta hãy xem một số trường của mục nhập IDT:

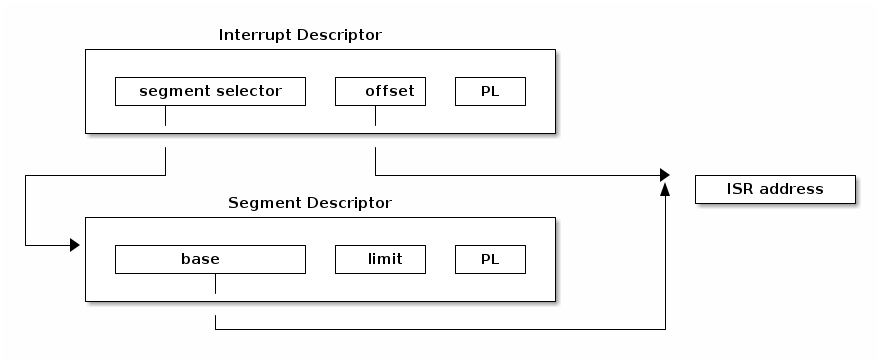
* Segment Selector (Bộ chọn đoạn): Segment selector là một chỉ số được sử dụng để xác định mô tả đoạn (segment descriptor) thích hợp trong Bảng mô tả đoạn toàn cục (Global Descriptor Table - GDT) hoặc Bảng mô tả đoạn cục bộ (Local Descriptor Table - LDT). Nó cung cấp thông tin về đoạn bộ nhớ nơi các trình xử lý ngắt đặt, chẳng hạn như đoạn mã (code segment).
* Offset (Độ lệch): Offset đại diện cho vị trí cụ thể bên trong đoạn mã (code segment) mà trình xử lý ngắt hoặc ngoại lệ nằm. Nó chỉ định địa chỉ bộ nhớ mà việc thực thi sẽ chuyển đến khi cổng được kích hoạt.
* T (Type): Trường T trong bản mô tả của cổng (gate descriptor) đại diện cho loại cổng. Nó xác định liệu cổng có phải là cổng ngắt (interrupt gate), cổng trap (trap gate) hay cổng tác vụ (task gate), như đã đề cập trước đó.
* DPL (Descriptor Privilege Level - Cấp đặc trưng đặc quyền): Trường DPL chỉ định cấp đặc trưng đặc quyền tối thiểu yêu cầu để truy cập vào nội dung của các đoạn liên quan đến cổng. Nó xác định mức đặc quyền mà một tiến trình hoặc mã phải có để sử dụng cổng và truy cập vào nội dung của các đoạn đó.



Hình 4.5 Một số trường của mục nhập IDT

### Interrupt Handler Address (Địa chỉ của trình xử lý ngắt)

Để tìm địa chỉ của trình xử lý ngắt, trước tiên chúng ta cần tìm địa chỉ bắt đầu của đoạn mã (code segment) mà trình xử lý ngắt đặt. Để làm điều này, chúng ta sử dụng bộ chọn đoạn (segment selector) để truy cập vào Bảng mô tả đoạn toàn cục (Global Descriptor Table - GDT) hoặc Bảng mô tả đoạn cục bộ (Local Descriptor Table - LDT), nơi chúng ta có thể tìm thấy mô tả đoạn tương ứng. Điều này sẽ cung cấp địa chỉ bắt đầu được lưu trong trường 'base'.

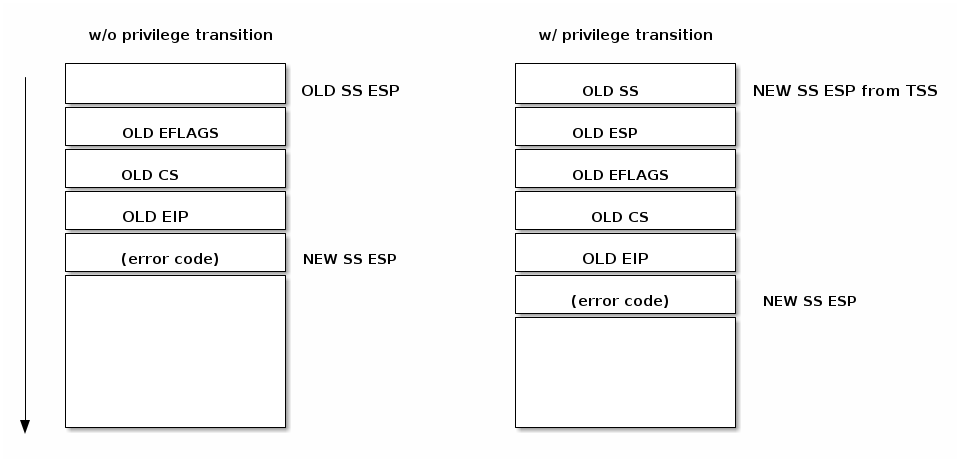


Hình 4.6 Interrupt Handler Address

### Ngăn xếp của chương trình xử lí ngắt

Tương tự như việc chuyển quyền điều khiển tới một hàm thông thường, việc chuyển quyền điều khiển tới một trình xử lý ngắt hoặc ngoại lệ sử dụng ngăn xếp để lưu trữ thông tin cần thiết để trở về mã bị ngắt.

Như được thấy trong hình dưới đây, một ngắt đẩy (push) thanh ghi EFLAGS trước khi lưu trữ địa chỉ của lệnh bị ngắt. Một số loại ngoại lệ cũng khiến cho một mã lỗi (error code) được đẩy (push) vào ngăn xếp để hỗ trợ gỡ lỗi ngoại lệ.



Hình 4.7 Stack of interrupt handler

### Xử lí yêu cầu ngắt

Sau khi một yêu cầu ngắt được tạo ra, bộ xử lý (CPU) thực hiện một chuỗi sự kiện cuối cùng là chạy trình xử lý ngắt của hạt nhân (kernel interrupt handler):

* CPU kiểm tra mức đặc quyền hiện tại.
* Nếu cần thay đổi mức đặc quyền:
* Thay đổi ngăn xếp (stack) bằng ngăn xếp liên kết với mức đặc quyền mới.
* Lưu thông tin ngăn xếp cũ trên ngăn xếp mới.
* Lưu trữ EFLAGS, CS, EIP trên ngăn xếp.
* Lưu trữ mã lỗi trên ngăn xếp trong trường hợp có sự cố (abort).
* Thực thi trình xử lý ngắt của hạt nhân.

## Xử lí ngắt trong Linux

Trong Linux, việc xử lý ngắt được thực hiện qua ba giai đoạn: giai đoạn quan trọng (critical), giai đoạn tức thì (immediate) và giai đoạn trì hoãn (deferred).

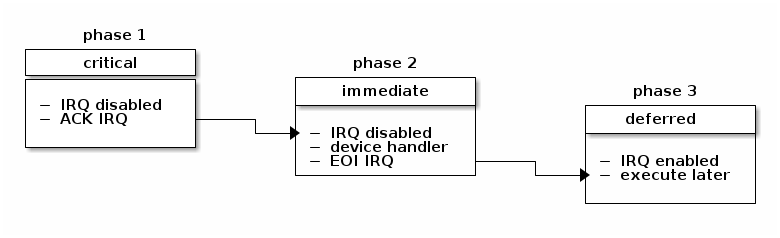
Trong giai đoạn đầu tiên, hạt nhân sẽ chạy trình xử lý ngắt chung (generic interrupt handler) để xác định số ngắt, trình xử lý ngắt cho ngắt cụ thể này và bộ điều khiển ngắt (interrupt controller). Tại thời điểm này, các hoạt động quan trọng về thời gian cũng sẽ được thực hiện (ví dụ: xác nhận ngắt ở mức bộ điều khiển ngắt). Các ngắt cục bộ trên bộ xử lý sẽ bị vô hiệu hóa trong suốt giai đoạn này và tiếp tục bị vô hiệu hóa trong giai đoạn tiếp theo.

Trong giai đoạn thứ hai, tất cả các trình xử lý của trình điều khiển thiết bị liên quan đến ngắt này sẽ được thực thi. Khi kết thúc giai đoạn này, phương thức "end of interrupt" của bộ điều khiển ngắt được gọi để cho phép bộ điều khiển ngắt khởi lại ngắt này. Các ngắt cục bộ trên bộ xử lý được kích hoạt tại thời điểm này.

***Lưu ý:***

Có thể xảy ra tình huống một ngắt liên kết với nhiều thiết bị và trong trường hợp này, ngắt được coi là được chia sẻ (shared). Thông thường, khi sử dụng ngắt chia sẻ, trách nhiệm xác định xem ngắt có đích đến là thiết bị của nó hay không thuộc trình điều khiển thiết bị.

Cuối cùng, trong giai đoạn cuối cùng của việc xử lý ngắt, các hoạt động có thể trì hoãn trong ngữ cảnh ngắt sẽ được thực hiện. Đôi khi chúng còn được gọi là "nửa dưới" của ngắt (nửa trên là phần xử lý ngắt chạy với ngắt bị vô hiệu hóa). Tại thời điểm này, các ngắt được kích hoạt trên bộ xử lý cục bộ.



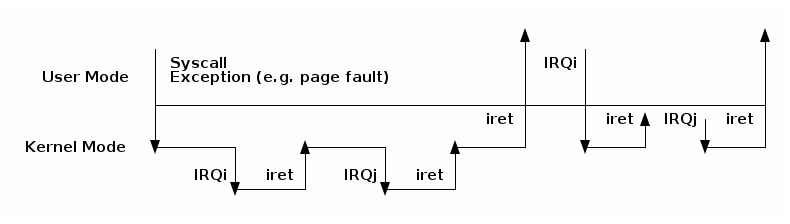
Hình 4.8 Interrupt handling in Linux

### Nested interrupts and exceptions

Trước đây, Linux hỗ trợ ngắt lồng nhau nhưng điều này đã được loại bỏ trong một thời gian để tránh các giải pháp ngày càng phức tạp cho các vấn đề tràn ngăn xếp - chỉ cho phép một cấp lồng nhau, cho phép nhiều cấp lồng nhau đến một độ sâu ngăn xếp hạt nhân nhất định, v.v.

Tuy nhiên, vẫn có thể có sự lồng nhau giữa các ngoại lệ và ngắt, nhưng các quy tắc khá hạn chế:

* Một ngoại lệ (ví dụ: lỗi trang, gọi hệ thống) không thể gián đoạn một ngắt; nếu điều này xảy ra, nó được coi là một lỗi.
* Một ngắt có thể gián đoạn một ngoại lệ.
* Một ngắt không thể gián đoạn một ngắt khác (trước đây có thể thực hiện điều này).



Hình 4.9 Nested interrupts and exceptions

### Ngữ cảnh ngắt

Trong quá trình xử lý một ngắt (từ thời điểm CPU nhảy vào trình xử lý ngắt cho đến khi trình xử lý ngắt trả về - ví dụ: IRET được thực thi), ta nói rằng mã chạy trong "ngữ cảnh ngắt" (interrupt context).

Mã chạy trong ngữ cảnh ngắt có các đặc điểm sau đây:

* Nó chạy như một kết quả của một IRQ (không phải là một ngoại lệ).
* Không có ngữ cảnh quy trình (process context) rõ ràng được liên kết.
* Không được phép kích hoạt một chuyển đổi ngữ cảnh (context switch) (không sleep, schedule hoặc truy cập bộ nhớ người dùng).

### Các hoạt động có thể trì hoãn

Các hoạt động có thể trì hoãn được sử dụng để chạy các hàm gọi lại (callback functions) vào một thời điểm sau. Nếu các hoạt động có thể trì hoãn được lên lịch từ một trình xử lý ngắt, hàm gọi lại liên quan sẽ chạy sau khi trình xử lý ngắt đã hoàn thành.

Có hai danh mục lớn của các hoạt động có thể trì hoãn: các hoạt động chạy trong ngữ cảnh ngắt và các hoạt động chạy trong ngữ cảnh quy trình.

Mục đích của các hoạt động có thể trì hoãn trong ngữ cảnh ngắt là tránh làm quá nhiều công việc trong hàm xử lý ngắt. Chạy quá lâu với các ngắt bị vô hiệu hóa có thể có tác động không mong muốn như gia tăng độ trễ hoặc hiệu suất hệ thống kém do bỏ lỡ các ngắt khác (ví dụ: gói mạng bị mất vì CPU không kịp thời phản hồi để giải hàng đợi gói từ giao diện mạng và bộ đệm của thẻ mạng đầy).

Các hoạt động có thể trì hoãn có API để: khởi tạo một phiên bản, kích hoạt hoặc lên lịch cho hoạt động và vô hiệu hóa và kích hoạt/làm cho việc thực thi của hàm gọi lại. Thao tác cuối cùng được sử dụng cho mục đích đồng bộ hóa giữa hàm gọi lại và các ngữ cảnh khác.

Thông thường, trình điều khiển thiết bị sẽ khởi tạo cấu trúc hoạt động có thể trì hoãn trong quá trình khởi tạo phiên bản thiết bị và sẽ kích hoạt/lên lịch cho hoạt động có thể trì hoãn từ trình xử lý ngắt.

### Soft IRQs (Interrupt Requests mềm)

Soft IRQs là thuật ngữ được sử dụng để chỉ cơ chế cấp thấp thực hiện việc trì hoãn công việc từ các trình xử lý ngắt nhưng vẫn chạy trong ngữ cảnh ngắt.

Soft IRQs có các API sau:

* Khởi tạo: open\_softirq()
* Kích hoạt: raise\_softirq()
* Vô hiệu hóa: local\_bh\_disable(), local\_bh\_enable()

Sau khi được kích hoạt, hàm gọi lại do\_softirq() chạy sau:

* Sau một trình xử lý ngắt hoặc
* Từ tiểu trình kernel ksoftirqd

Do soft IRQs có thể lên lịch cho chính chúng hoặc các ngắt khác có thể xảy ra và lên lịch cho chúng, chúng có thể dẫn đến tình trạng đói tiến trình (process starvation) (tạm thời) nếu không đặt các kiểm tra vào chỗ. Hiện tại, hạt nhân Linux không cho phép chạy soft IRQs quá MAX\_SOFTIRQ\_TIME hoặc lên lịch cho quá MAX\_SOFTIRQ\_RESTAR lần tiếp theo.

Khi đạt đến giới hạn này, một tiểu trình đặc biệt của hạt nhân, ksoftirqd, sẽ được đánh thức và tất cả các soft IRQs đang chờ sẽ được chạy từ ngữ cảnh của tiểu trình kernel này.

Việc sử dụng soft IRQs bị hạn chế, chúng được sử dụng bởi một số hệ thống con có yêu cầu độ trễ thấp và tần suất cao.

### Tasklets (Công việc lệnh)

Tasklets là một loại công việc trì hoãn động (không giới hạn số lượng) chạy trong ngữ cảnh ngắt.

Tasklets API:

* Khởi tạo: tasklet\_init()
* Kích hoạt: tasklet\_schedule()
* Vô hiệu hóa: tasklet\_disable(), tasklet\_enable()

Tasklets được triển khai trên hai soft IRQs cụ thể: TASKLET\_SOFTIRQ và HI\_SOFTIRQ.

Tasklets cũng được tuần tự hóa, tức là cùng một tasklet chỉ có thể thực thi trên một bộ xử lý.

### Workqueues (Hàng đợi công việc)

Workqueues là một loại công việc trì hoãn chạy trong ngữ cảnh quy trình.

Chúng được triển khai trên các tiểu trình kernel.

Workqueues API:

* Khởi tạo: INIT\_WORK
* Kích hoạt: schedule\_work()

### Timers (Bộ định thời)

Bộ hẹn giờ được triển khai trên soft IRQ TIMER\_SOFTIRQ.

API của bộ hẹn giờ:

* Khởi tạo: setup\_timer()
* Kích hoạt: mod\_timer()

**Dưới đây là một tóm tắt về các hoạt động có thể trì hoãn trong Linux:**

* SoftIRQ
* Chạy trong ngữ cảnh ngắt
* Được cấp phát tĩnh
* Cùng một trình xử lý ngắt có thể chạy song song trên nhiều lõi
* Tasklet
* Chạy trong ngữ cảnh ngắt
* Có thể được cấp phát động
* Cùng một trình xử lý ngắt chạy tuần tự
* Workqueues
* Chạy trong ngữ cảnh quy trình

## Thực thi

Trong phần này, để có thể đưa ra một ví dụ cho Interrupt, nhóm em thực hiện một bài tập nhỏ đó là tạo một Driver đơn giản cho bàn phím, sử dụng Linux Kernel:

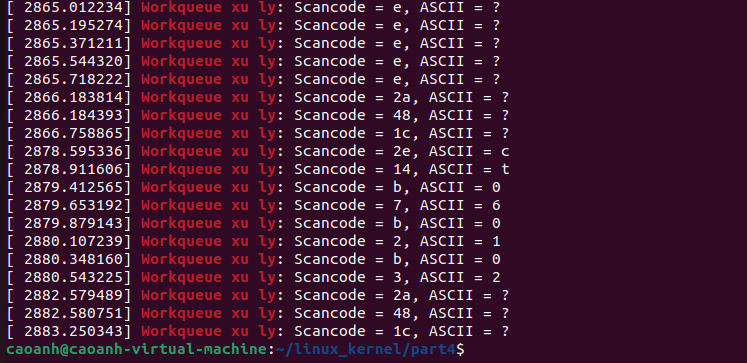
|  |
| --- |
| obj-m += kbd\_driver.o  all:  make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules  clean:  make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean |

Code 4.1 Makefile

|  |
| --- |
| #include <linux/module.h>  #include <linux/init.h>  #include <linux/fs.h>  #include <linux/uaccess.h>  #include <linux/interrupt.h>  #include <linux/cdev.h>  #include <linux/device.h>  #include <linux/spinlock.h>  #include <linux/workqueue.h>  #include <asm/io.h>  MODULE\_LICENSE("GPL");  MODULE\_AUTHOR("Nhom3");  MODULE\_DESCRIPTION("Keyboard IRQ Handler with Workqueue");  #define DEVICE\_NAME "kbd\_log"  #define BUFFER\_SIZE 1024  #define KBD\_IRQ 1  static char buffer[BUFFER\_SIZE];  static int head = 0;  static int tail = 0;  static spinlock\_t buf\_lock;  static int major;  static struct class \*kbd\_class;  static struct cdev kbd\_cdev;  /\* Workqueue \*/  static struct workqueue\_struct \*kbd\_wq;  static DECLARE\_WORK(kbd\_work, NULL);  static unsigned char last\_scancode = 0;  /\* Chuyển scancode sang ASCII đơn giản \*/  static char scancode\_to\_ascii(unsigned char scancode) {  static const char \*keymap = "??1234567890-=??qwertyuiop[]??asdfghjkl;'`??zxcvbnm,./?";  if (scancode < 0x3A)  return keymap[scancode];  return '?';  }  /\* Hàm xử lý workqueue \*/  static void kbd\_work\_handler(struct work\_struct \*work) {  char ascii;  unsigned long flags;  ascii = scancode\_to\_ascii(last\_scancode);  spin\_lock\_irqsave(&buf\_lock, flags);  buffer[head] = ascii;  head = (head + 1) % BUFFER\_SIZE;  if (head == tail)  tail = (tail + 1) % BUFFER\_SIZE;  spin\_unlock\_irqrestore(&buf\_lock, flags);  pr\_info("Workqueue xu ly: Scancode = %x, ASCII = %c\n", last\_scancode, ascii);  }  /\* IRQ handler – chỉ giao việc \*/  static irqreturn\_t irq\_handler(int irq, void \*dev\_id) {  unsigned char scancode = inb(0x60);  if (!(scancode & 0x80)) {  last\_scancode = scancode;  queue\_work(kbd\_wq, &kbd\_work);  }  return IRQ\_HANDLED;  }  static ssize\_t kbd\_read(struct file \*file, char \_\_user \*user\_buf, size\_t count, loff\_t \*ppos) {  unsigned long flags;  char kbuf[BUFFER\_SIZE];  int copied = 0;  spin\_lock\_irqsave(&buf\_lock, flags);  while (head != tail && copied < count) {  kbuf[copied++] = buffer[tail];  tail = (tail + 1) % BUFFER\_SIZE;  }  spin\_unlock\_irqrestore(&buf\_lock, flags);  if (copied == 0)  return 0;  if (copy\_to\_user(user\_buf, kbuf, copied))  return -EFAULT;  return copied;  }  static struct file\_operations fops = {  .owner = THIS\_MODULE,  .read = kbd\_read,  };  static int \_\_init kbd\_init(void) {  dev\_t dev;  int ret;  spin\_lock\_init(&buf\_lock);  ret = alloc\_chrdev\_region(&dev, 0, 1, DEVICE\_NAME);  major = MAJOR(dev);  pr\_info("Allocated major number: %d\n", major);  cdev\_init(&kbd\_cdev, &fops);  cdev\_add(&kbd\_cdev, dev, 1);  kbd\_class = class\_create("kbd\_class");  if (IS\_ERR(kbd\_class)) {  pr\_err("Không thể tạo class\n");  return PTR\_ERR(kbd\_class);  }  device\_create(kbd\_class, NULL, dev, NULL, DEVICE\_NAME);  INIT\_WORK(&kbd\_work, kbd\_work\_handler);  kbd\_wq = create\_singlethread\_workqueue("kbd\_wq");  ret = request\_irq(KBD\_IRQ, irq\_handler, IRQF\_SHARED, "kbd\_driver", (void \*)(irq\_handler));  if (ret) {  pr\_err("Không thể đăng ký IRQ\n");  return ret;  }  pr\_info("kbd\_driver đã được tải.\n");  return 0;  }  static void \_\_exit kbd\_exit(void) {  dev\_t dev = MKDEV(major, 0);  free\_irq(KBD\_IRQ, (void \*)(irq\_handler));  flush\_workqueue(kbd\_wq);  destroy\_workqueue(kbd\_wq);  device\_destroy(kbd\_class, dev);  class\_destroy(kbd\_class);  cdev\_del(&kbd\_cdev);  unregister\_chrdev\_region(dev, 1);  pr\_info("kbd\_driver đã được gỡ bỏ.\n");  }  module\_init(kbd\_init);  module\_exit(kbd\_exit); |

Code 4.2 Code Kbd\_driver.c

**Kết quả:**Tiến hành gõ trên bàn phím dòng chữ “ct060102”

****

Hình 4.10 Kết quả thực thi

# KẾT LUẬN

Qua quá trình thực hiện bài tập lớn môn Lập trình nhân Linux, nhóm chúng em đã triển khai thành công nhiều chức năng thiết yếu cho việc tương tác với hệ thống Linux ở cấp độ thấp.

Các chức năng đã đạt được gồm:

* Hoàn thiện hệ thống quản lý tệp tin cơ bản trong Shell, bao gồm: tạo, xóa, hiển thị, đổi tên và sao chép tệp tin.
* Xây dựng thành công chức năng auto install và auto uninstall giúp người dùng dễ dàng cài đặt hoặc gỡ bỏ các module shell đã viết.
* Quản lý tiến trình với việc lập lịch, chỉnh sửa, liệt kê và xóa các tác vụ đã được cấu hình bằng crontab.
* Hoàn thành chức năng quản lý thời gian hệ thống, bao gồm: hiển thị thời gian, cài đặt thời gian thủ công và đồng bộ thời gian với máy chủ mạng.
* Viết và nạp module nhân (kernel module) với nhiều chức năng tính toán như: phép cộng ma trận, nhân ma trận, đếm số nguyên tố, tìm phần tử nhỏ nhất/lớn nhất và thực hiện các phép toán đặc thù.

Tuy nhiên, nhóm cũng nhận thấy còn một số điểm chưa thực hiện được hoặc cần hoàn thiện thêm, như:

* Giao diện người dùng chỉ ở mức đơn giản, chưa có hỗ trợ thông báo lỗi chi tiết hay xử lý sai sót đầu vào.
* Các chức năng còn được thực thi tách biệt, chưa có một hệ thống điều khiển tổng thể.
* Việc xử lý tham số trong module nhân còn hạn chế, chưa có xác thực hoặc kiểm tra an toàn bộ nhớ.
* Chưa có báo cáo benchmark đánh giá hiệu suất của từng chức năng trong các tình huống thực tế.

# TÀI LIỆU THAM KHẢO

1. ThS. Phạm Văn Hưởng, slide bài giảng Lập trình nhân Linux, năm 2025
2. Interrupt: <https://linux-kernel-labs.github.io/refs/heads/master/lectures/interrupts.html.>
3. I/O access and Interrupts: <https://linux-kernel-labs.github.io/refs/heads/master/labs/interrupts.html>