****

INFORMATION TECHNOLOGY AND MEDIA VIETNAM – KOREA UNIVERSITY

**FACULTY COMPUTER SCIENCE**

**SYSTEM PROGRAMMING REPORT**

**Implementing and Demo for FIFO in IPC**

**Member of group 2: Tran Quoc Bao ID: 22IT021**

**Ngo Van Bao ID:**

**Ho The Bao ID:**

**Lecturer: TS. Hoang Huu Duc**

***Da Nang, date 12 month 10 year 2024***

COMMENTS OF LECTURER

**Lecture’s signature**

**ACKNOWLEDGMENTS**

To complete this project, our group has received a great deal of support and assistance from our advisor – Dr. Hoang Huu Duc.

With heartfelt and sincere feelings, we would like to express our gratitude to her and to all our friends who helped us and collaborated in researching throughout the process of carrying out this project. Due to time constraints and our limited experience, this report inevitably has shortcomings. We kindly hope to receive guidance and feedback from the instructors so that we can further improve our experience, serving us better in future projects.

Our group sincerely thanks you!

**TABLE OF CONTENTS**

[COMMENTS OF LECTURER i](#_Toc181236078)

[INTRODUCTION 1](#_Toc181236079)

[CHAPTER 1: OVERVIEW OF THE TOPIC 2](#_Toc181236080)

[1.1 The purpose of the topic 2](#_Toc181236081)

[1.2 Tools and operating systems used for the topic 2](#_Toc181236082)

[1.2.1 Information about oracle vm virtualbox 2](#_Toc181236083)

[1.2.2 Operating systems Ubuntu/Linux 5](#_Toc181236084)

[1.2.2.1 Introduction to Ubuntu 5](#_Toc181236085)

[1.2.2.2 The different between Ubuntu and Linux 5](#_Toc181236086)

[1.2.2.3 Reasons why ubuntu is so popular for both computer and private servers 6](#_Toc181236087)

[1.2.2.4 A brief history lesson 10](#_Toc181236088)

[1.2.2.5 Opening a terminal 12](#_Toc181236089)

[1.2.2.6 Conclusion 16](#_Toc181236090)

[CHAPTER 2: THEORETICAL BASIS FOR “FIFO IN IPC” 16](#_Toc181236091)

[2.1 16](#_Toc181236092)

INTRODUCTION

In the context of the ever-evolving landscape of information technology, inter-process communication (IPC) plays an increasingly crucial role. IPC enables independent processes to exchange data and synchronize operations, facilitating the creation of complex and efficient applications. To ensure that data exchange occurs smoothly and efficiently, selecting the appropriate data flow management mechanism is essential.

FIFO (First In, First Out) is one of the most common data flow management mechanisms in IPC. Adhering to the principle of "first in, first out," FIFO ensures that requests are processed in the order they are received, preventing bottlenecks and data loss. However, implementing FIFO effectively in real-world IPC systems is not a simple task, especially when faced with demands for high performance, reliability, and security.

# CHAPTER 1: OVERVIEW OF THE TOPIC

## The purpose of the topic

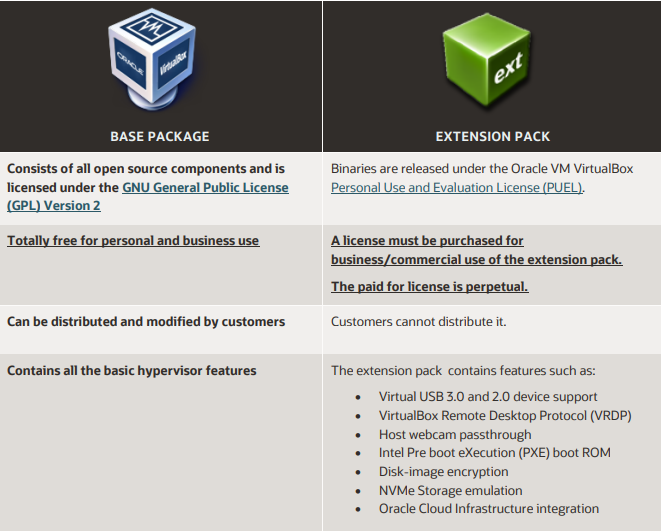
FIFO (First In First Out) is a fundamental mechanism for inter-process communication in Unix-like systems. Understanding FIFOs will help you:

* Build Efficient Multi-Process Applications:
* **Data Exchange:** FIFOs provide a simple, efficient channel for processes to exchange data, enabling the creation of complex applications.
* **Workflow Organization:** FIFOs help organize tasks in a specific order, ensuring that tasks are processed sequentially.
* **Performance Improvement:** By using FIFOs, you can optimize system resource utilization and enhance application execution speed.
* Deepen Your Understanding of Operating Systems:
* **Inter-Process Communication:** FIFOs are a core mechanism for process interaction. Studying FIFOs will help you grasp how processes communicate with each other.
* **Resource Management:** FIFOs play a vital role in managing system resources like CPU and memory.
* Solve Real-world Problems:
* **Pipeline Systems:** FIFOs are the foundation for creating pipeline systems where data is processed sequentially through multiple stages.
* **Messaging Systems:** FIFOs can be used to build simple messaging systems where processes can send and receive messages.
* **Process Synchronization:** FIFOs can be used to synchronize processes, ensuring that tasks are executed in the correct order.
* Expand Your Systems Programming Knowledge:
* **System Calls:** Working with FIFOs introduces you to system calls like mkfifo, open, read, and write, enhancing your understanding of system interactions.
* **Resource Management:** You'll learn how to efficiently manage system resources, preventing issues like deadlocks and race conditions.

## Tools and operating systems used for the topic

### Information about oracle vm virtualbox

**Introduction:** Oracle VM VirtualBox is cross-platform virtualization software. It allows users to extend their existing computer to run multiple operating systems including Microsoft Windows, Mac OS X, Linux, and Oracle Solaris, at the same time. Designed for IT professionals and developers, Oracle VM VirtualBox is ideal for testing, developing, demonstrating, and deploying solutions across multiple platforms from one machine. The following table summarizes each of the components:



Oracle VM VirtualBox has been designed to take advantage of the innovations introduced in the x86 modern hardware platform, and it is lightweight and easy to install and use. Yet, under the simple exterior lies an extremely fast and powerful virtualization engine. With a well-earned reputation for speed and agility, Oracle VM VirtualBox contains innovative features to deliver tangible benefits: excellent performance; a powerful virtualization system; and a wide range of supported guest operating systems.

Oracle VM VirtualBox is a bridge to open source and cloud development. The latest release allows users to create and deploy virtual machines nearly everywhere, upload to the cloud, download from the cloud, and review and make changes offline.

With thousands of downloads each day, Oracle VM VirtualBox is the world’s most popular free and open source, cross-platform virtualization software, based on vibrant community participation combined with world-class development and support supplied by Oracle.

A box with a logo

Description automatically generatedOracle VM VirtualBox simplifies cloud deployment by allowing developers to create multiplatform environments and to develop applications for container and virtualization technologies within Oracle VM VirtualBox on a single machine. Operating system and application updates can be done within Oracle VM VirtualBox virtual machines (VMs), and VMs can subsequently be deployed to server virtualization environments such as Oracle Linux KVM or Oracle Private Cloud Appliance.

Oracle VM VirtualBox Enterprise is an ideal choice for a next-generation development solution. The latest release introduces paravirtualization support for Linux and Windows virtual machines and support for xHCI/USB 3.0 devices and new platforms, and it provides enhanced CPU capabilities and support for bidirectional drag and drop between a host and its guest virtual machines. It also introduces disk-image encryption and many other enhancements.

Oracle VM VirtualBox Enterprise provides world-class support for both the base package and the extension pack and licenses for commercial use of the extension pack.

**Oracle VM VirtualBox Enterprise use cases:**

Development platform for the cloud. Software developers rely on Oracle VM VirtualBox Enterprise for the development and debugging of their applications in multiple operating systems and environments on one device. Developers can clone an environment on their personal desktop/laptop without impact to production services.

A computer screen with cloud computing and cloud computing

Description automatically generated

### Operating systems Ubuntu/Linux

#### 1.2.2.1 Introduction to Ubuntu

Ubuntu is a popular free and open-source Linux-based operating system you can use on a computer or virtual private server. Ubuntu was introduced in 2004 by a British company Canonical. It was based on Debian – a popular distro back then – which was difficult to install. As a result, Ubuntu was proposed as a more user-friendly alternative. Ubuntu has multiple editions, including core, server, and desktop, that allow it to run across different types of machines. It can be used on personal computers, servers, supercomputers, in cloud computing, and more.

#### 1.2.2.2 The different between Ubuntu and Linux

Linux is a family of operating systems based on the Linux kernel – the core of an operating system. It enables the communication between hardware and software components.

Linux is based on Unix and built around the Linux kernel. It was released in 1991 and is available for web servers, gaming consoles, embedded systems, desktops, and personal computers. It comes in many different versions called distributions.

Ubuntu is a Linux distro based on Debian. It is suitable for cloud computing, servers, desktops, and internet of things (IoT) devices. The main difference between Linux and Ubuntu is that the former is an operating system family based on Unix, while Ubuntu is a Linux distribution.

#### 1.2.2.3 Reasons why ubuntu is so popular for both computer and private servers

* **User-Friendliness:** Ubuntu uses Linux desktop environments for its interface. Since Ubuntu 17.10, GNOME is the default one. GNOME doesn’t clutter the screen with descriptions, instead using icons to facilitate navigation.

By default, GNOME features the Activities panel on the left taskbar.

**A screenshot of a computer

Description automatically generated**

Controls are situated on the top-right corner of the screen.

A screenshot of a computer

Description automatically generated

A full applications overview can be viewed by clicking the grid button on the bottom-left corner of the screen.

A screenshot of a computer

Description automatically generated

System navigation is made easy because all configuration and application elements are accessible from the main screen.

* **Strong Security:** Ubuntu is open-source, undergoing constant checks and reviews by its community members. As a result, any security vulnerabilities can be identified and eliminated quickly. Generally, Linux distributions have fewer security flaws compared to other operating systems.

What’s more, Ubuntu employs AppArmor, a kernel enhancement that restricts how programs behave and limits their resources. It works when you have profiles inserted into the kernel.

These consist of text files containing access rules for each application. AppArmor can mitigate the extent of security breaches since programs don’t have unlimited permissions.

In addition, there are a number of security practices Ubuntu supports, such as automatically installing security updates, using sudo instead of root [Linux user](https://www.hostinger.com/tutorials/how-to-see-system-users-in-ubuntu-linux-vps/), implementing complex passwords, [setting up a VPN server](https://www.hostinger.com/tutorials/how-to-set-up-a-linux-vpn-server-with-openvpn/), configuring [firewall using ufw](https://www.hostinger.com/tutorials/how-to-configure-firewall-on-ubuntu-using-ufw/), and enabling [iptables](https://www.hostinger.com/tutorials/iptables-tutorial).

* **More Software Options:** Most of the popular macOS and Windows applications such as Slack, Spotify, and Firefox are also available for Linux users and can be installed via the Ubuntu Software Center. Even if you can’t find the application you want, chances are there is a quality alternative available. For example, Libre Office works just as well as Microsoft Office.

A screenshot of a computer

Description automatically generated

Another option you can use aside from the Ubuntu Software Center is Snapcraft. It is an application created by Canonical that contains open-source and proprietary [software packages](https://www.hostinger.com/tutorials/how-to-list-installed-packages-on-ubuntu/) available for Linux-based operating systems. One major advantage of Snapcraft is that it uses the snapd daemon that automatically checks and updates applications.

* **Enhanced Privacy:** Just like any other operating system, Ubuntu has its data privacy policy.

There are four fundamental principles that Ubuntu follows in terms of personal information processing:

* Ubuntu doesn’t ask for personal data unless it truly needs such information for legal purposes.
* Ubuntu doesn’t share its users’ personal information with anyone except to provide its customers with products and services, comply with the law, and protect its rights.
* Ubuntu doesn’t store personal information except if it is required for the operation of services, to provide products, comply with the law, or protect its rights.

Ubuntu also collects some hardware information as well as location and usage data. However, you can always stop it from doing so. For example, location services can be disabled via the Privacy settings, as shown below.

A screenshot of a computer

Description automatically generated

* **Lightweight Performance:** Ubuntu is not resource-intensive – it operates smoothly on low-end devices. The default interface can run on less than 1 GB of RAM. What’s more, a lot of Ubuntu desktop environments are even more lightweight. For example, Lubuntu can run on systems with as little as 512 MB of RAM.

In comparison, both Windows and macOS require considerably more resources – both macOS Big Sur and Windows 11 need a minimum of 4 GB of RAM to run. This is because these operating systems have resource-heavy user interfaces (UIs) with advanced features incorporated in them.

* **Free of Charge:** Ubuntu is a free open-source operating system that you can download from its official website. You can also modify its source code as you see fit – as of now, there are numerous projects based on Ubuntu.

In comparison, macOS and Windows are closed-source operating systems. To use Windows, you need to buy a computer that comes with it or purchase a license which starts at $139/license. Meanwhile, macOS is not available for purchase – it comes pre-installed with Mac devices.

#### 1.2.2.4 A brief history lesson

During the formative years of the computer industry, one of the early operating systems was called Unix. It was designed to run as a multi-user system on mainframe computers, with users connecting to it remotely via individual ***terminals***. These terminals were pretty basic by modern standards: just a keyboard and screen, with no power to run programs locally. Instead they would just send keystrokes to the server and display any data they received on the screen. There was no mouse, no fancy graphics, not even any choice of colour. Everything was sent as text, and received as text. Obviously, therefore, any programs that ran on the mainframe had to produce text as an output and accept text as an input.

Compared with graphics, text is very light on resources. Even on machines from the 1970s, running hundreds of terminals across glacially slow network connections (by today’s standards), users were still able to interact with programs quickly and efficiently. The commands were also kept very terse to reduce the number of keystrokes needed, speeding up people’s use of the terminal even more. This speed and efficiency is one reason why this text interface is still widely used today.

When logged into a Unix mainframe via a terminal users still had to manage the sort of file management tasks that you might now perform with a mouse and a couple of windows. Whether creating files, renaming them, putting them into subdirectories or moving them around on disk, users in the 70s could do everything entirely with a textual interface.

Each of these tasks required its own program or command: one to change directories (cd), another to list their contents (ls), a third to rename or move files (mv), and so on. In order to coordinate the execution of each of these programs, the user would connect to one single master program that could then be used to launch any of the others. By wrapping the user’s commands this “shell” program, as it was known, could provide common capabilities to any of them, such as the ability to pass data from one command straight into another, or to use special wildcard characters to work with lots of similarly named files at once. Users could even write simple code (called “shell scripts”) which could be used to automate long series of shell commands in order to make complex tasks easier. The original Unix shell program was just called sh, but it has been extended and superceded over the years, so on a modern Linux system you’re most likely to be using a shell called bash. Don’t worry too much about which shell you have, all the content in this tutorial will work on just about all of them.

Linux is a sort-of-descendent of Unix. The core part of Linux is designed to behave similarly to a Unix system, such that most of the old shells and other text-based programs run on it quite happily. In theory you could even hook up one of those old 1970s terminals to a modern Linux box, and access the shell through that. But these days it’s far more common to use a software terminal: that same old Unix-style text interface, but running in a window alongside your graphical programs.

#### 1.2.2.5 Opening a terminal

On a Ubuntu 18.04 system you can find a launcher for the terminal by clicking on the Activities item at the top left of the screen, then typing the first few letters of “terminal”, “command”, “prompt” or “shell”. Yes, the developers have set up the launcher with all the most common synonyms, so you should have no problems finding it.

A screen shot of a computer

Description automatically generated

Other versions of Linux, or other flavours of Ubuntu, will usually have a terminal launcher located in the same place as your other application launchers. It might be hidden away in a submenu or you might have to search for it from within your launcher, but it’s likely to be there somewhere.

If you can’t find a launcher, or if you just want a faster way to bring up the terminal, most Linux systems use the same default keyboard shortcut to start it: **Ctrl-Alt-T**.

However you launch your terminal, you should end up with a rather dull looking window with an odd bit of text at the top, much like the image below. Depending on your Linux system the colours may not be the same, and the text will likely say something different, but the general layout of a window with a large (mostly empty) text area should be similar.

A screenshot of a computer

Description automatically generated

Let’s run our first command. Click the mouse into the window to make sure that’s where your keystrokes will go, then type the following command, ***all in lower case***, before pressing the **Enter** or **Return** key to run it.

****

You should see a directory path printed out (probably something like /home/YOUR\_USERNAME), then another copy of that odd bit of text.

A screenshot of a computer

Description automatically generated

There are a couple of basics to understand here, before we get into the detail of what the command actually did. First is that when you type a command it appears on the same line as the odd text. That text is there to tell you the computer is ready to accept a command, it’s the computer’s way of prompting you. In fact it’s usually referred to as the *prompt*, and you might sometimes see instructions that say “bring up a prompt”, “open a command prompt”, “at the bash prompt” or similar. They’re all just different ways of asking you to open a terminal to get to a shell.

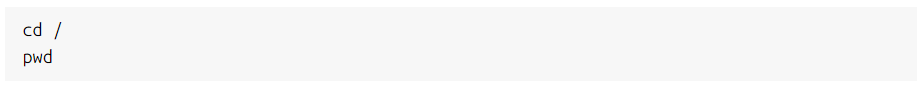
On the subject of synonyms, another way of looking at the prompt is to say that there’s a line in the terminal into which you type commands. A command line, if you will. Again, if you see mention of “command line”, including in the title of this very tutorial, it’s just another way of talking about a shell running in a terminal.

The second thing to understand is that when you run a command any output it produces will usually be printed directly in the terminal, then you’ll be shown another prompt once it’s finished. Some commands can output a lot of text, others will operate silently and won’t output anything at all. Don’t be alarmed if you run a command and another prompt immediately appears, as that usually means the command succeeded. If you think back to the slow network connections of our 1970s terminals, those early programmers decided that if everything went okay they may as well save a few precious bytes of data transfer by not saying anything at all.

**The importance of case:** Be extra careful with case when typing in the command line. Typing PWD instead of pwd will produce an error, but sometimes the wrong case can result in a command appearing to run, but not doing what you expected. We’ll look at case a little more on the next page but, for now, just make sure to type all the following lines in exactly the case that’s shown.

**A sense of location:** One important concept to understand is that the shell has a notion of a default location in which any file operations will take place. This is its working directory. If you try to create new files or directories, view existing files, or even delete them, the shell will assume you’re looking for them in the current working directory unless you take steps to specify otherwise. So it’s quite important to keep an idea of what directory the shell is “in” at any given time, after all, deleting files from the wrong directory could be disastrous. If you’re ever in any doubt, the pwd command will tell you exactly what the current working directory is.

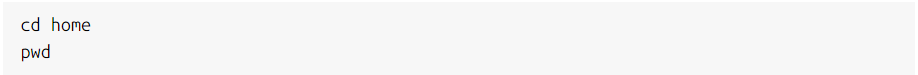
You can change the working directory using the cd command, an abbreviation for ‘change directory’. Try typing the following:



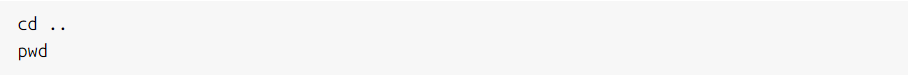
*Note: Note that the directory separator is a forward slash (“/”), not the backslash that you may be used to from Windows or DOS systems*

Now your working directory is “/”. If you’re coming from a Windows background you’re probably used to each drive having its own letter, with your main hard drive typically being “C:”. Unix-like systems don’t split up the drives like that. Instead they have a single unified file system, and individual drives can be attached (“mounted”) to whatever location in the file system makes most sense. The “/” directory, often referred to as the root directory, is the base of that unified file system. From there everything else branches out to form a tree of directories and subdirectories.

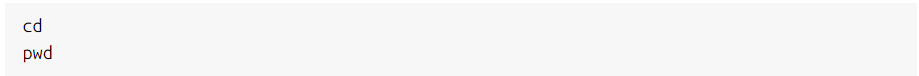
From the root directory, the following command will move you into the “home” directory (which is an immediate subdirectory of “/”):



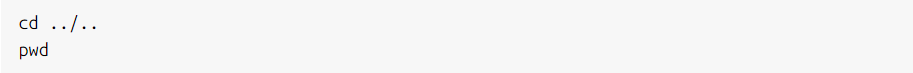
To go up to the parent directory, in this case back to “/”, use the special syntax of two dots (..) when changing directory (note the space between cd and .., unlike in DOS you can’t just type cd.. as one command):



Typing cd on its own is a quick shortcut to get back to your home directory:



You can also use .. more than once if you have to move up through multiple levels of parent directories:



Notice that in the previous example we described a route to take through the directories. The path we used means “starting from the working directory, move to the parent / from that new location move to the parent again”. So if we wanted to go straight from our home directory to the “etc” directory (which is directly inside the root of the file system), we could use this approach:

A white background with a black and white flag

Description automatically generated with medium confidence

#### 1.2.2.6 Conclusion

This section has only been a brief introduction to the Linux command line. We’ve looked at a few common commands for moving around the file system and manipulating files, but no tutorial could hope to provide a comprehensive guide to every available command. What’s more important is that you’ve learnt the key aspects of working with the shell. You’ve been introduced to some widely used terminology (and synonyms) that you might come across online, and have gained an insight into some of the key parts of a typical shell command.

# CHAPTER 2: THEORETICAL BASIS FOR “FIFO IN IPC”

## 2.1