|  |
| --- |
| Trường Đại học Khoa học tự nhiên – Khoa Công nghệ thông tin. |
| Đồ án số 01 |
| HỆ ĐIỀU HÀNH – Operating System |

|  |
| --- |
| TP. Hồ Chí Minh.  Tháng 10, 2024. |

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**KHOA CÔNG NGHỆ THÔNG TIN**

**📖👨‍💻📖**

**A blue and white logo

Description automatically generated**

**ĐỒ ÁN THỰC HÀNH SỐ 01**

**Bộ môn:** Hệ điều hành.

**Tên đề tài:**

*“XV6 and Unix Utilities”.*

**Tên nhóm:** 3T.

**Thành viên:**

1. Trần Đức Trí – 22120387.
2. Vũ Hoàng Nhật Trường – 22120398.
3. Nguyễn Anh Tường – 22120412.

# **Thông tin chung:**

1. **Bộ môn:** Hệ điều hành.
2. **Giảng viên lý thuyết:** Thầy Trần Trung Dũng.
3. **Giảng viên thực hành:** Thầy Nguyễn Thanh Quân.
4. **Mã lớp:** 22\_4.
5. **Tên nhóm:** 3T.
6. **Danh sách thành viên:**
   1. Nguyễn Đình Trí – 22120384.
   2. Vũ Hoàng Nhật Trường – 22120398.
   3. Nguyễn Anh Tường – 22120412.

MỤC LỤC

[**ĐỒ ÁN THỰC HÀNH SỐ 01** 2](#_Toc179484821)

[**Thông tin chung:** 3](#_Toc179484822)

[**Phần giới thiệu:** 5](#_Toc179484823)

[**Giới thiệu về các thành phần được sử dụng.** 6](#_Toc179484824)

[**Yêu cầu 1:** 7](#_Toc179484825)

[**I.** **Thành viên thực hiện:** 8](#_Toc179484826)

[**II.** **Mô tả thực hiện:** 8](#_Toc179484827)

[**III.** **Kết quả chương trình:** 10](#_Toc179484828)

[**Yêu cầu 2:** 11](#_Toc179484829)

[**I.** **Thành viên thực hiện:** 12](#_Toc179484830)

[**II.** **Mô tả thực hiện:** 12](#_Toc179484831)

[**III.** **Một số vấn đề đáng chú ý để hoàn thành yêu cầu 2 ( đã được giải quyết ):** 17](#_Toc179484832)

[**IV. Kết quả chương trình:** 19](#_Toc179484833)

[**Yêu cầu 3:** 20](#_Toc179484834)

[**Yêu cầu 4:** 22](#_Toc179484835)

# **Phần giới thiệu:**

*GIỚI THIỆU CHUNG*

## **Giới thiệu về các thành phần được sử dụng.**

1. **Môi trường lập trình:**

* Linux: Máy ảo ( Ubuntu 24.04.1 LTS ).
* WSL: Ubuntu - Phiên bản 24.04.5 LTS.
* Qemu: QEMU emulator version 7.2.0.

1. **Công cụ lập trình:**

* Visual Studio Code.
* Riscv64-linux-gnu-gcc ( Debian 10.3.0-8) 10.3.0.
* Riscv64-unknown-elf-gcc ( GCC ) 10.1.0.
* Riscv64-unknown-linux-gnu-gcc ( GCC ) 10.1.0.

1. **Phiên bản XV6:**

* XV6 - UNIX 6th Edition – 2024.
  + 1st Edition ( June 14, 2000 ).
  + ISBN: 1-57398-013-7
  + **Link repo:** [Link here.](git://g.csail.mit.edu/xv6-labs-2024)

# **Yêu cầu 1:**

***A blue line drawing of a server

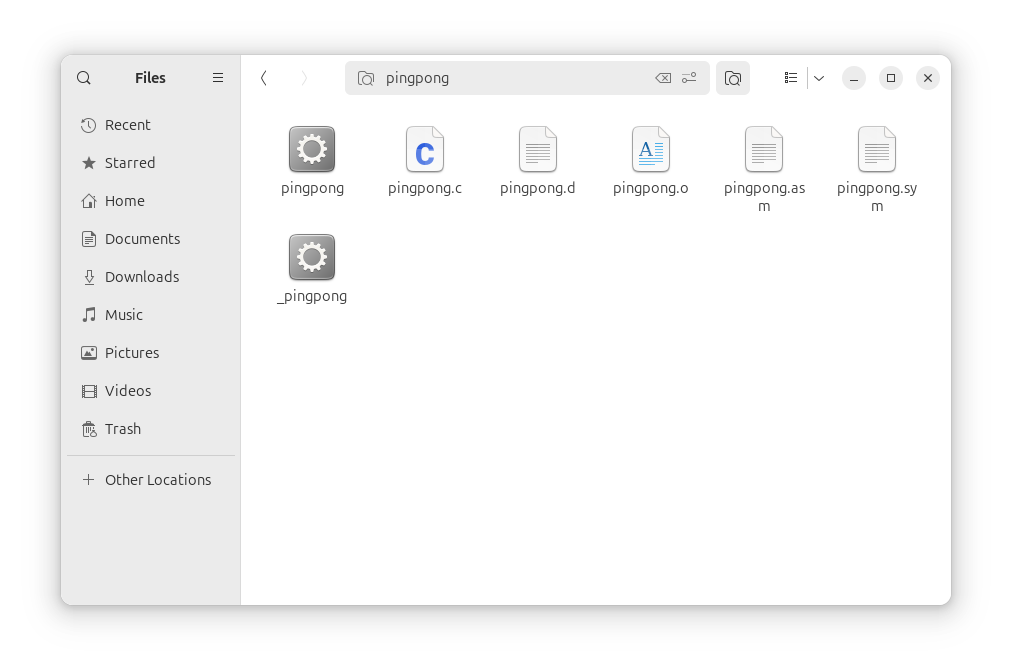
Description automatically generated****PINGPONG*

1. **Thành viên thực hiện:**

* Nguyễn Anh Tường – 22120412.

1. **Mô tả thực hiện:**

**Bước 1:** Tạo file *pingpong.c* trong thư mục **user** của XV6.



**Bước 2:** Vào môi trường VS code để lập trình cho file pingpong.c trên.

**Bước 3:** Sử dụng các thư viện được cung cấp bởi hệ điều hành XV6.

#include "kernel/types.h"

#include "kernel/stat.h"

#include "user/user.h"

**Bước 4:** Tạo tiến trình con bằng hàm fork(). Tạo 2 pipe bằng hàm pipe(<pipe>).

int proc = fork();

...

int check\_pipe1 = pipe(pipe\_one); // Create a parent-to-child pipe.

int check\_pipe2 = pipe(pipe\_two); // Create a child-to-parent pipe.

**Bước 5:** Kiểm tra việc tạo pipe và fork có thành công hay không?

if (check\_pipe1 == -1 || check\_pipe2 == -1) {

    printf("Cannot create pipe! - Error: pipe() function failed!");

    return 0;

}

if (proc < 0) {

    printf("Cannot create a child process! - Error: fork() function failed!");

    return 0;

}

**Bước 6:** Kiểm tra **pid,** nếu bằng 0 tức là ta đang ở tiến trình con.

* Vì tiến trình con đọc trước nên ta đóng đầu ghi của pipe\_one.
* Đọc kí tự được gởi từ tiến trình cha.
* Sau khi nhận được ta in ra màn hình lệnh ping.
* Đóng đầu đọc của pipe\_one ngay sau đó.
* Ta đóng đầu đọc của pipe\_two để tiến hành ghi kí tự gởi đến cha.
* Sau khi ghi xong thì ta đóng đầu ghi của pipe\_two lại.

else if (proc == 0) { // Child process.

        char mess;

        close(pipe\_one[1]); // Close the write end of parent-to-child pipe.

        read(pipe\_one[0], &mess, 1); // Read from parent-to-child pipe.

        printf("%d: received ping\n", getpid());

        close(pipe\_one[0]); // Close the read end of parent-to-child pipe.

        close(pipe\_two[0]); // Close the read end of child-to-parent pipe.

        write(pipe\_two[1], &mess, 1); // Write to child-to-parent pipe.

        close(pipe\_two[1]); // Close the write end of child-to-parent pipe.

}

**Bước 6.2:** Kiểm tra nếu pid là một con số bất kì ( > 0 ) thì ta đang ở tiến trình cha.

* Ta tiến hành đóng đầu đọc của pipe\_one.
* Tiến hành viết kí tự vào đầu ghi của pipe\_one để gởi đến tiến trình con.
* Đóng đầu ghi của pipe\_one sau khi ghi xong.
* Đóng đầu ghi của pipe\_two.
* Mở đầu đọc của pipe\_two để tiến hành nhận thông tin từ tiến trình con.
* Sau khi nhận được thì ta in ra màn hình lệnh “pong”.
* Đóng đầu đọc của pipe\_two.
* Chờ cho tiến trình con kết thúc để kết thúc chương trình.

    else { // Parent process

        char mess = 'T'; // Because all of the member's name starts by 'T' letter

        close(pipe\_one[0]); // Close the read end of parent-to-child pipe.

        write(pipe\_one[1], &mess, 1); // Write to parent-to-child pipe.

        close(pipe\_one[1]); // Close the write end of parent-to-child pipe.

        close(pipe\_two[1]); // Close the write end of child-to-parent pipe.

        read(pipe\_two[0], &mess, 1); // Read from child-to-parent pipe.

        printf("%d: received pong\n", getpid());

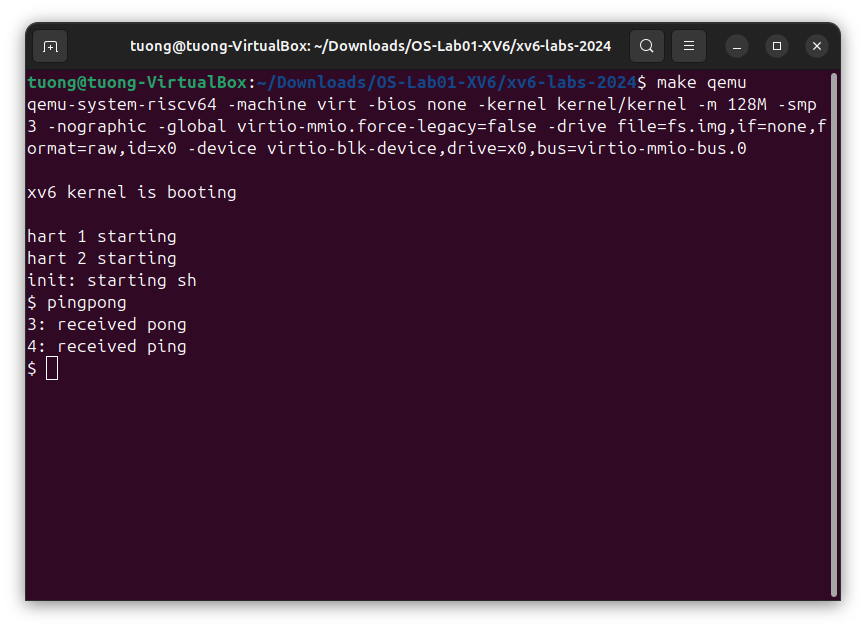
        close(pipe\_two[0]); // Close the read end of child-to-parent pipe.

        wait(0); // Wait for child process to finish.

        }

**Bước 7:** Hoàn thành chương trình.

1. **Kết quả chương trình:**

****

# **Yêu cầu 2:**

***A blue line drawing of a server

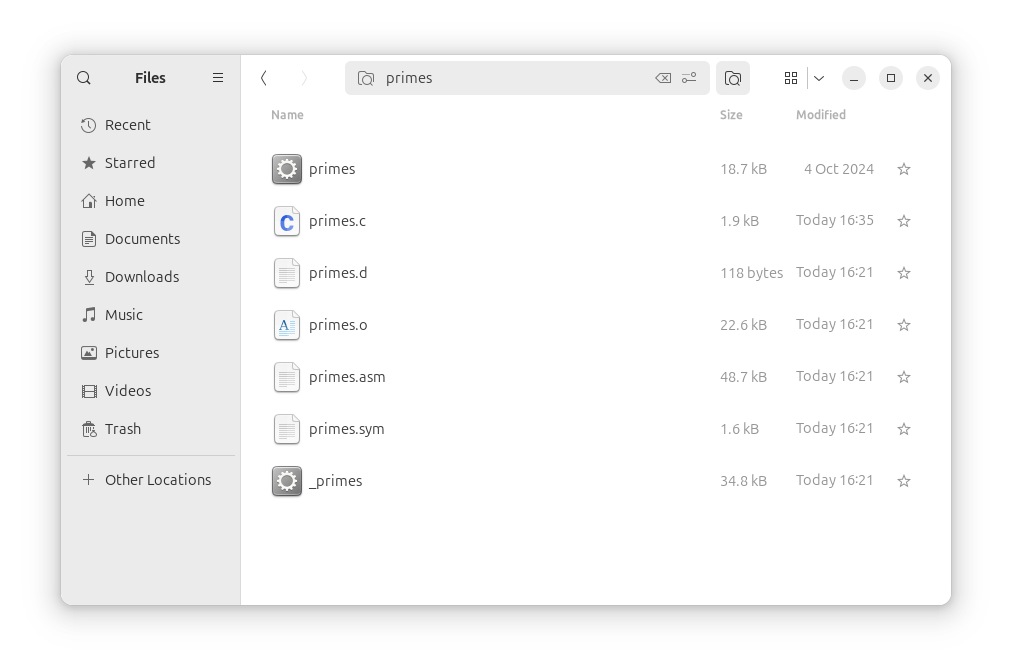
Description automatically generated****PRIMES*

1. **Thành viên thực hiện:**

Nguyễn Anh Tường – 22120412.

1. **Mô tả thực hiện:**

**Bước 1:** Tạo file *primes.c* trong thư mục **user** của XV6.



**Bước 2:** Vào môi trường VS code để lập trình cho file pingpong.c trên.

**Bước 3:** Sử dụng các thư viện được cung cấp bởi hệ điều hành XV6.

#include "kernel/types.h"

#include "kernel/stat.h"

#include "user/user.h"

**Bước 4:** Tạo tiến trình con và pipe đầu tiên lần lượt bằng hàm fork() và pipe(). Đồng thời kiểm tra xem có tại được không?

int fd[2];

    int c\_pipe = pipe(fd); //fd[0]: read | fd[1]: write

    if(c\_pipe == -1){

        printf("Cannot create pipe! - Error: pipe() function failed!");

        exit(0);

    }

    int pid = fork();

    if(pid < 0){

        printf("RaiseError: Cannot create process!");

        exit(0);

    }

**Bước 5:** Kiểm tra tiến trình bằng PID, nếu là tiến trình con thì bằng 0, lớn hơn không là tiến trình cha.

**Nếu là tiến trình con.**

* Tiến hành đóng đầu ghi của pipe lại.
* Ta chạy hàm primes – là một hàm đệ quy – để thực hiện tìm kiếm các số nguyên tố.

**Nếu là tiến trình cha.**

* Tiến hành đóng đầu đọc của pipe lại.
* Chạy một vòng lặp FOR từ 2 đến 280, bước nhảy là 1.
  + Với mỗi vòng lặp ta tiến hành ghi số nguyên ghi số nguyên hiện tại vào trong pipe.
* Sau khi hết vòng lặp, ta tiến hành đóng pipe lại.
* Đợi tiến trình con kết thúc.

else if (pid == 0){

        //This is the child process

        close(fd[1]);

        primes(fd[0]);

    }

 else{

        //This is the parent process

        close(fd[0]);

        for( int i = 2; i <= 280; i++){

            write(fd[1], &i, sizeof(i));

        }

        close(fd[1]);

        wait(0);

  }

**Bước 6:** Thực thi hàm Primes. Có tham số là một pipe và không có giá trị trả về.

* **Lưu ý:** Ta thêm dòng code sau để tránh bị lỗi vòng lặp vô hạn trong Qemu.

void primes(int cur\_pipe)\_\_attribute\_\_((noreturn));

void primes(int cur\_pipe){

...

}

**Bước 7:** Ta nhận dữ liệu được gởi từ cha thông qua pipe được truyền vào. Sau đó ta kiểm tra xem có nhận dữ liệu được không, nếu không thì ta đóng pipe và thoát hàm. Nếu dữ liệu nhận được ok thì ta tiến hành in ra màn hình.

int val;

int c\_read = read(cur\_pipe, &val, sizeof(val)); // c\_read: check\_read

if(c\_read == 0){

  close(cur\_pipe);

  exit(0);

}

printf("prime: %d\n",val);

**Bước 8:** Tiến hành tạo pipe, fork tiếp theo và kiếm tra. Nếu tạo không được thì ta in ra lỗi, đóng các pipe hiện có và thoát chương trình.

int fd[2];

if (pipe(fd) == -1) {

   printf("Error: pipe creation failed!\n");

   close(cur\_pipe);

   exit(1);

}

int pid = fork();

if(pid < 0){

   printf("Error: fork failed!\n");

   close(fd[0]);

   close(fd[1]);

   close(cur\_pipe);

   exit(1);

}

**Bước 9:** Kiểm tra xem ta đang ở tiến trình nào?

**Nếu là tiến trình con:** PID = 0.

* Ta đóng pipe ghi của pipe hiện tại.
* Tiến hành gọi đệ quy hàm primes với tham số là pipe đọc của pipe hiện tại

else if(pid == 0){

    close(fd[1]); // Close write pipe in child process.

    primes(fd[0]);

}

**Nếu là tiến trình cha:** PID > 0.

* Tiến hành đọc pipe tham số cho đến khi gặp được một số nguyên tố thì sẽ tiến hành ghi số nguyên tố đó vào pipe tiếp theo và dừng vòng lặp lại.
* Nếu gặp lỗi thì in lỗi ra màn hình, ngược lại chạy tiếp.
* Đóng các pipe hiện tại và đợi cho tiến trình con kết thúc.
* Cuối cùng là thoát hàm.

    else {

        int check\_prime;

        close(fd[0]); // Close read pipe in parent process.

        while((c\_read = read(cur\_pipe, &check\_prime, sizeof(check\_prime))) > 0){

            if(check\_prime % val != 0){

                if (write(fd[1], &check\_prime, sizeof(check\_prime)) == -1) {

                    printf("Error writing to pipe\n");

                    break;

                }

            }

        }

        if(c\_read == -1) {

            printf("Error reading from pipe\n");

        }

        close(fd[1]);

        close(cur\_pipe);

        wait(0);

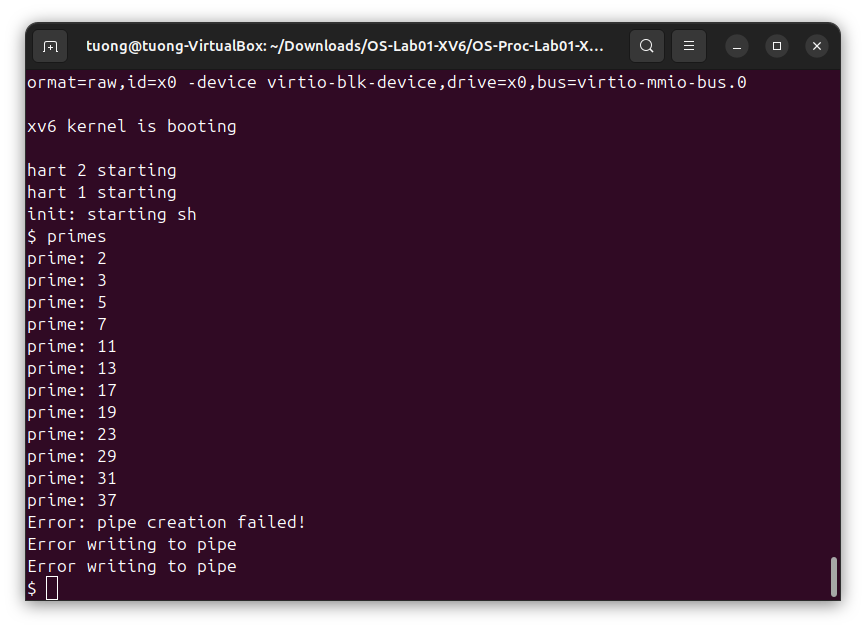
        exit(0);

    }

1. **Một số vấn đề đáng chú ý để hoàn thành yêu cầu 2 ( đã được giải quyết ):**

Vì XV6 là một hệ điều hành khá đơn giản và nhẹ nhàng, nên việc tạo nhiều pipe và nhiều tiến trình, gọi đệ quy,… đã tốn không ít tài nguyên. Do đó ta phải chỉnh sửa lại một số thông số của hệ điều hành để chương trình có thể chạy một cách yêm xuôi.

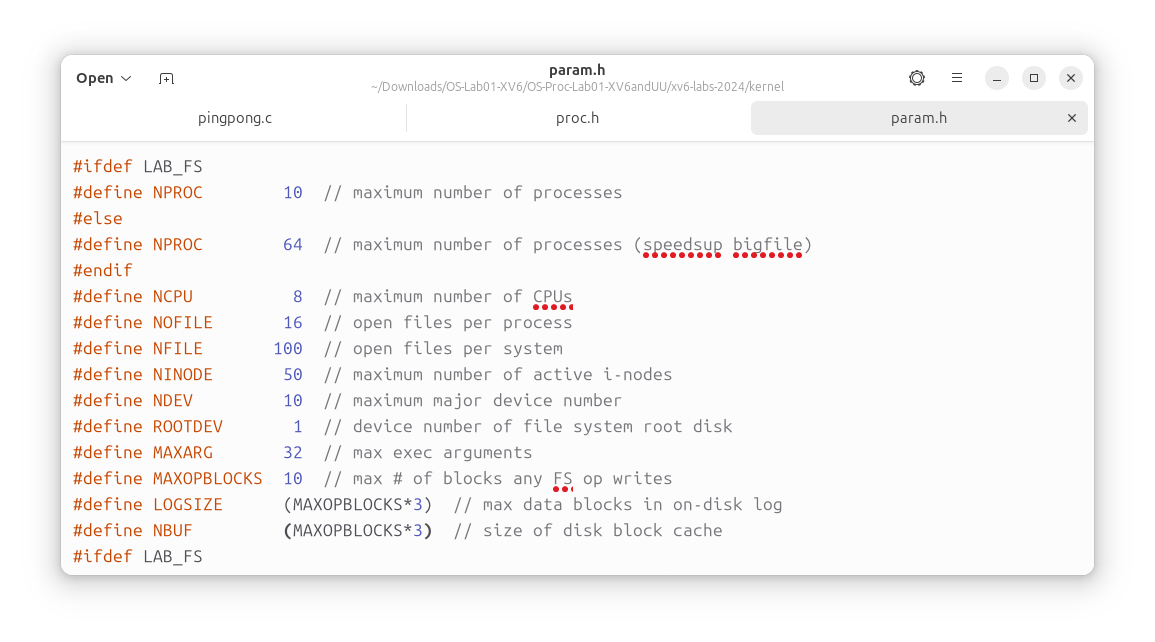
* 1. **Vấn đề ban đầu:**

****

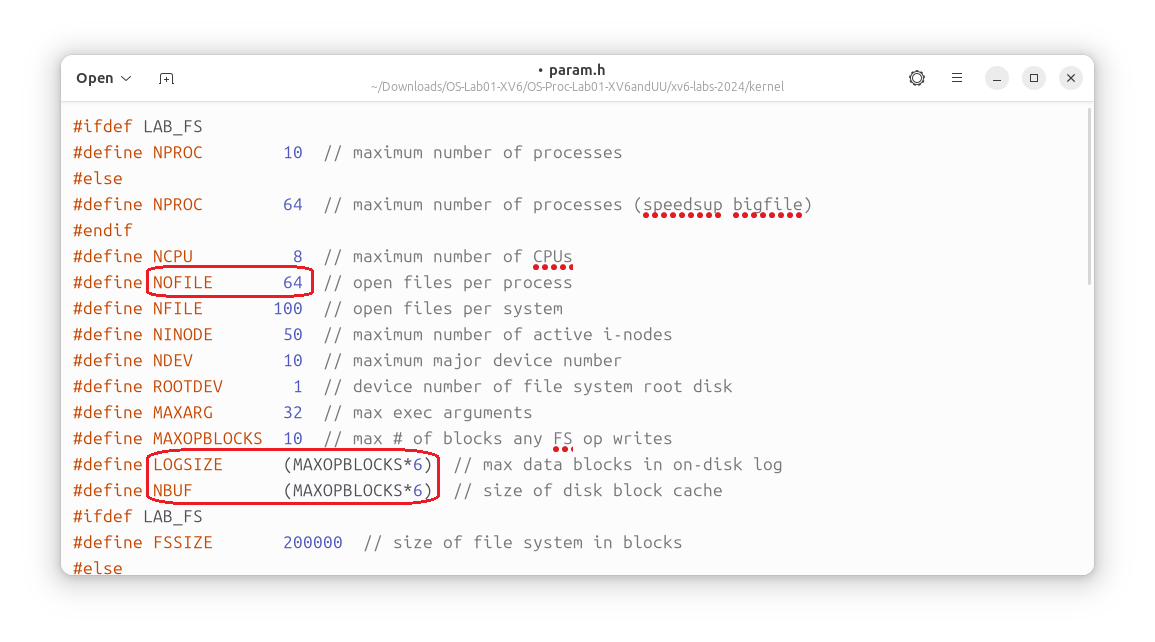
* **Chương trình không thể tạo thêm pipe do không đủ tài nguyên.**
  1. **Hướng khắc phục:**

Điều chỉnh thông số cho hệ điều hành để trành tình trạng **Buffer Overflow** này.

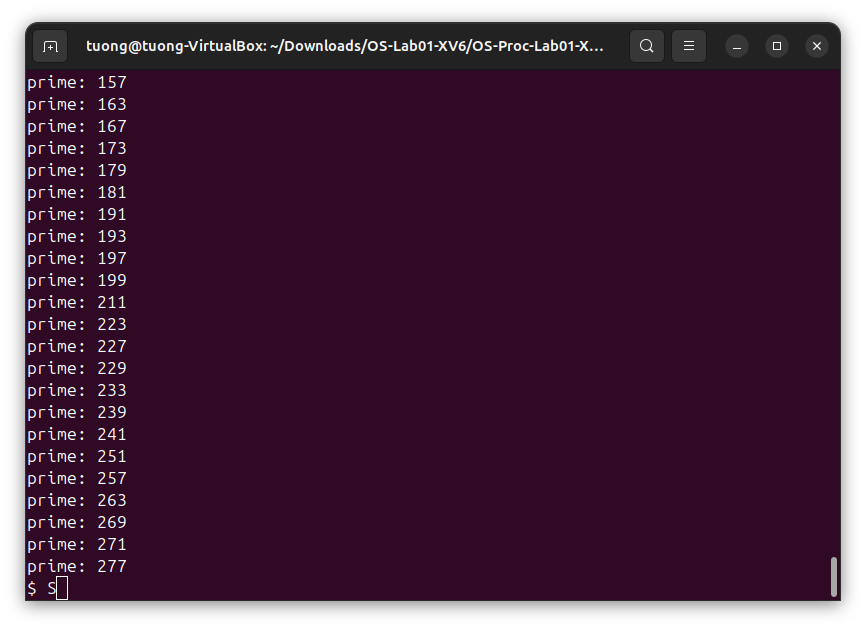
**Thông số ban đầu:**

****

**Thông số sau khi tuỳ chỉnh:**



1. **Kết quả chương trình:**



# **Yêu cầu 3:**

***A blue line drawing of a server

Description automatically generated****FIND*

# **Yêu cầu 4:**

***A blue line drawing of a server

Description automatically generated****XARGS*