Project Report-Xv6 & Unix utilities

University of Science - VNUHCM

- **Date**: 15/02/2025
- **Subject**: *Operating System*
- **Class**: *23CLC03*
- **Environment**: *Ubuntu-24.04*
- **Intructors:**
  - *Phan Quốc Kỳ*
  - *Lê Hà Minh*
  - *Lê Giang Thanh*

I. Group Information

No	Student's ID	Student's Full Name	Contribution
1	23127004	Lê Nhật Khôi	`33%` : *pingpong*, *primes*
2	23127113	Nguyễn Trần Phú Quý	`33%` : *xargs*, *writeups*
3	23127165	Nguyễn Hải Đăng	`33%` : *pingpong*, *find*

II. Submission's Details

Here are all of the assignments stated in the lab.

1. pingpong

The program creates two processes that exchange a byte over pipes.

Key system calls:

- `pipe()` : Creates pipes for bidirectional communication
- `fork()` : Creates a child process
- `read()` and `write()` : Exchange data through pipes

- `getpid()`: Gets process IDs for output

Testing

When run in the xv6 shell, the program produces the expected output:

```
$ pingpong
4: received ping
3: received pong
$
```

2. primes

We used **Eratosthenes's Sieve** to implement this assignment.

> *Eratosthenes's Sieve* is an ancient algorithm for finding prime numbers up to any given limit.
>
> It works by iteratively marking the multiples of each prime starting from 2, ensuring that only prime numbers remain unmarked.

> **The algorithm of *Eratosthenes's Sieve***

The algorithm for finding all the prime numbers to a given integer `n` is based on the following steps

1. **Step 1:** Create a list of consecutive integers from 2 to `n` and initially let `p = 2` which is the smallest prime numbers
2. **Step 2:** Remove the multiples of `p` (except `p`) such as: `2*p`, `3*p`,...
3. **Step 3:** Find the smallest number in the list greater than `p` and set this new number as `p`
4. **Step 4:** Looping conditions:
    - If there are not any numbers larger than `p` in the list, the process stops. All of the remaining numbers in the lists are prime numbers.
    - Else, repeat from **Step 2**.

> **Using processes and pipes to implement *Eratosthenes's Sieve***

To implement *Eratosthenes's Sieve*, each processes is responsible for filtering a prime number. Along with processes are pipes which are responsible for passing numbers between processes. This algorithm is simulated through the following examples:

```
Input:  2  3  4  5  6  7  8  9  10  11

Process 1 (2):
  - Print 2
  - Send → 3  5  7  9  11
  - Drop: 4  6  8  10

Process 2 (3):
  - Print 3
  - Send → 5  7  11
  - Drop: 9

Process 3 (5):
  - Print 5
  - Send → 7  11

Process 4 (7):
  - Print 7
  - Send → 11

Process 5 (11):
  - Print 11
```

3. find

On **Linux**, the basic functionality of `find` could be replicate by this command:

```
ls -R | grep "filename.ext"
```

So initially, we try combining the `ls` and `grep` commands to make `find`, but this approach failed because:

- The xv6 `ls` is missing the `-R` option and so it doesn't recursively traverse directories
- The shell's piping `|` mechanism is limited for this specific task
- This approach wouldn't handle the proper display of full pathnames

Instead, we developed a dedicated solution that correctly implements the required functionality.

Key Components

- **Directory Traversal**: Adapted code from `user/ls.c` to read directory entries
- **Recursive Search**: Implemented recursion to explore subdirectories
- **Path Handling**: Properly constructed file paths and extracted filenames
- Skipped `.` and `..` entries to avoid infinite recursion

Testing

When tested with the required directory structure, the program produces the expected output:

```
$ find . b
./b
./a/b
./a/aa/b
```

4. xargs

The program reads input lines from `stdin` and executes a specified command with each line as an additional argument.

The implementation meets the basic requirements but could be enhanced with more robust error handling and input processing.

Key Implementation Details

- Used basic system calls: `fork()`, `exec()`, `wait()`, `read()`
- Processes input one character at a time until newline
- Handles command execution by creating child processes
- Passes command-line arguments along with input line to executed command

Limitations

- Does not support the `-n` optimization found in *UNIX* xargs
- Maximum line length fixed at *100* characters
- No support for multiple arguments per input line
- Input buffer could overflow with very long lines

Testing

Successfully tested with basic use cases:

- `echo hello too | xargs echo bye`
- `find . b | xargs grep hello`

IV. References

- *Lab Answers*
- *GeeksforGeeks - Linux Commands*
- *File Descriptors and Open Files*