Programming Assignment: Fast File Duplication

Operating Systems@NYCU

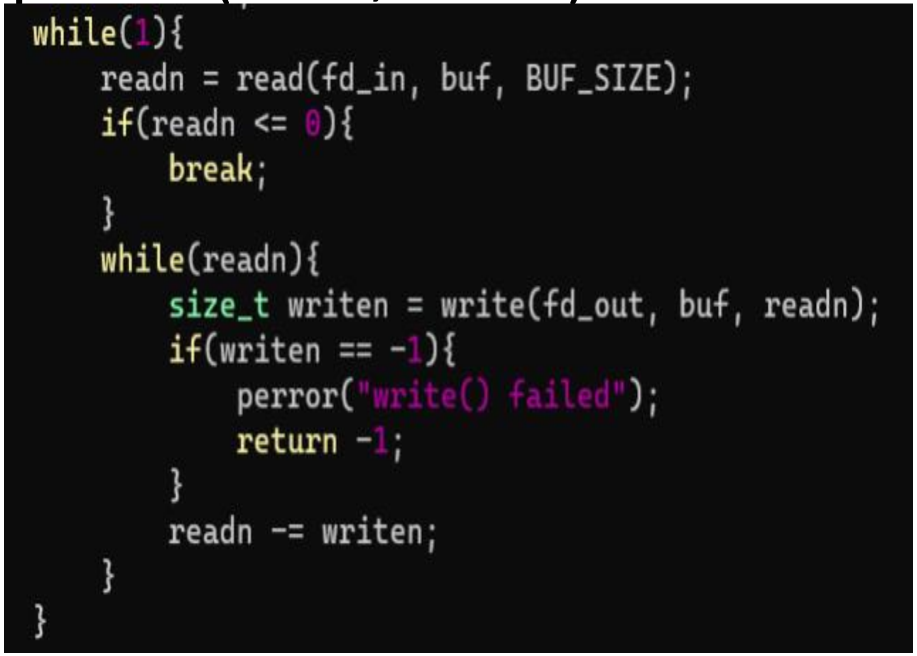
Prof. Li-Pin Chang

Objectives

- Understand the overheads involved in file operations
- Exploit kernel primitives to avoid (or to reduce) these overheads

The Baseline Method

- open→ (read, write)*N



```
while(1){
    readn = read(fd_in, buf, BUF_SIZE);
    if(readn <= 0){
        break;
    }
    while(readn){
        size_t writen = write(fd_out, buf, readn);
        if(writen == -1){
            perror("write() failed");
            return -1;
        }
        readn -= writen;
    }
}
```

- Can you do better (faster) than this?

Overheads of File Rding(Wrting)

- Switching from the user mode to the kernel mode
- Copying data from the storage to kernel pages
- Copying data from the kernel pages to user buffer
- Switching from the kernel mode to the user mode

- In this assignment, we duplicate a huge file (1GB) on *tmpfs*, a RAM-based file system, to focus on the OS-level overhead rather than the storage latency

Test Procedure

- Pre-conditioning
  - mount tmpfs: `mount -t tmpfs -o size=3G tmpfs <mountpoint>`
  - clear swap: `swapoff -a && swapon -a`
  - drop cache: `echo 3 > /proc/sys/vm/drop_caches`
- Run your program
- Cleaning-up
  - diff: “diff source destination”

- Report the time spent on the 2nd step
  - The diff result must report no difference
  - **No need to do fsync() on the destination file

Reference time

- TA's time to duplicate a 1GB file on tmpfs

Method	Baseline	Method 1	Method 2	Method 3	Method 4
Time (s)	0.755	0.350	0.622	0.287	0.255
Time	100%	46%	82%	38%	34%

* The time may vary depending on your hardware configuration.

- TA's test platform
  - Ubuntu 20.04.3 LTS
  - Intel Core i5-10400 with 32GB ram

Requirement

- Implement your own method of file duplication
- Your method must be *faster* than Baseline
- Do not use link() to “duplicate” the file
- Turn in a 2-page report to E3 (pdf)
  - Compare the performance of yours vs. baseline
  - Describing all the implementation details
  - Explain why your method improves the performance
- Turn in your program to E3 (.zip, .c, or .cxx)
  - TA will compile your program and verify the performance of your method

Template of Your 2-Page Report

1. Section 1: Analysis of Baseline
    - This section describes the performance bottleneck of the baseline method
2. Section 2: Proposed method
    - This section describes your proposed method
3. Section 3: Performance evaluation
    - Present the measured performance results of Baseline and your method
    - Discuss why and how your method will perform better than. Baseline
4. References

How to Get a High Mark

- Grading Policy
  - 50% quality of your report
  - 50% time efficiency of your method

- (Program) Duplicate the file correctly and considerably improve upon Baseline
- (Report) Provide insightful discussion, not just show the raw numbers

Source Code of Baseline

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>

#define BUF_SIZE 1024

int main(int argc, char *argv[]){
    if(argc != 3){
        printf("Usage: %s <source> <destination>\n", argv[0]);
        return -1;
    }

    int fd_in = open(argv[1], O_RDONLY);
    if(fd_in == -1){
        perror("readwrite: open");
        return -1;
    }

    int fd_out = creat(argv[2], 0644);
    if(fd_out == -1){
        perror("readwrite: creat");
        return -1;
    }

    char buf[BUF_SIZE];
    size_t readn;
    while(1){
        readn = read(fd_in, buf, BUF_SIZE);
        if(readn <= 0){
            break;
        }
        while(readn){
            size_t writen = write(fd_out, buf, readn);
            if(writen == -1){
                perror("readwrite: write");
                return -1;
            }
            readn -= writen;
        }
    }

    if(readn == -1){
        perror("readwrite: read");
        return -1;
    }

    close(fd_in);
    close(fd_out);
    return 0;
}
```

Deadline

23:55, June 17, 2022

Turn-in all your materials to E3