GHULAM ISHAQ KHAN INSTITUTE OF ENGINEERING SCIENCES AND TECHNOLOGY • PAKISTAN •

رَبِّ زِدْنِی عِلْمًا

CS-311
OPERATING SYSTEM
Directory Synchronizer Report

Team Members	Reg Numbers
Aoun Abdullah	2023381
Ahmed Abdullah	2023299
Muhammad Huzaifa	2023442

Directory Synchronizer: A Kernel-Level File Synchronization System

Project Overview

The Directory Synchronizer is a comprehensive file synchronization utility developed in three progressive phases, culminating in a real-time monitoring system with custom kernel-level system call integration. The project demonstrates the evolution from basic file copying mechanisms to an advanced synchronization tool that leverages Linux kernel features for efficient file metadata comparison and real-time change detection using inotify.

The primary objective of this project is to create a robust directory synchronization tool that maintains an exact replica of a source directory in a destination directory, handling file creation, modification, deletion, and directory structure changes automatically and efficiently.

Phase 1: Basic Directory Synchronization

Objectives

Phase 1 focused on implementing fundamental directory synchronization functionality using standard POSIX system calls without any custom kernel modifications.

Implementation Details

The initial implementation provides basic synchronization capabilities with the following features:

Core Functionality:

- File copying from source to destination directory
- Directory structure validation and creation
- Size-based file comparison for synchronization decisions
- Flat directory handling (non-recursive)

System Calls Used:

- open() - Opening source and destination files
- read() - Reading data from source files
- write() - Writing data to destination files
- close() - Closing file descriptors
- opendir() and readdir() - Directory traversal

- stat() - Retrieving file metadata
- mkdir() - Creating destination directories

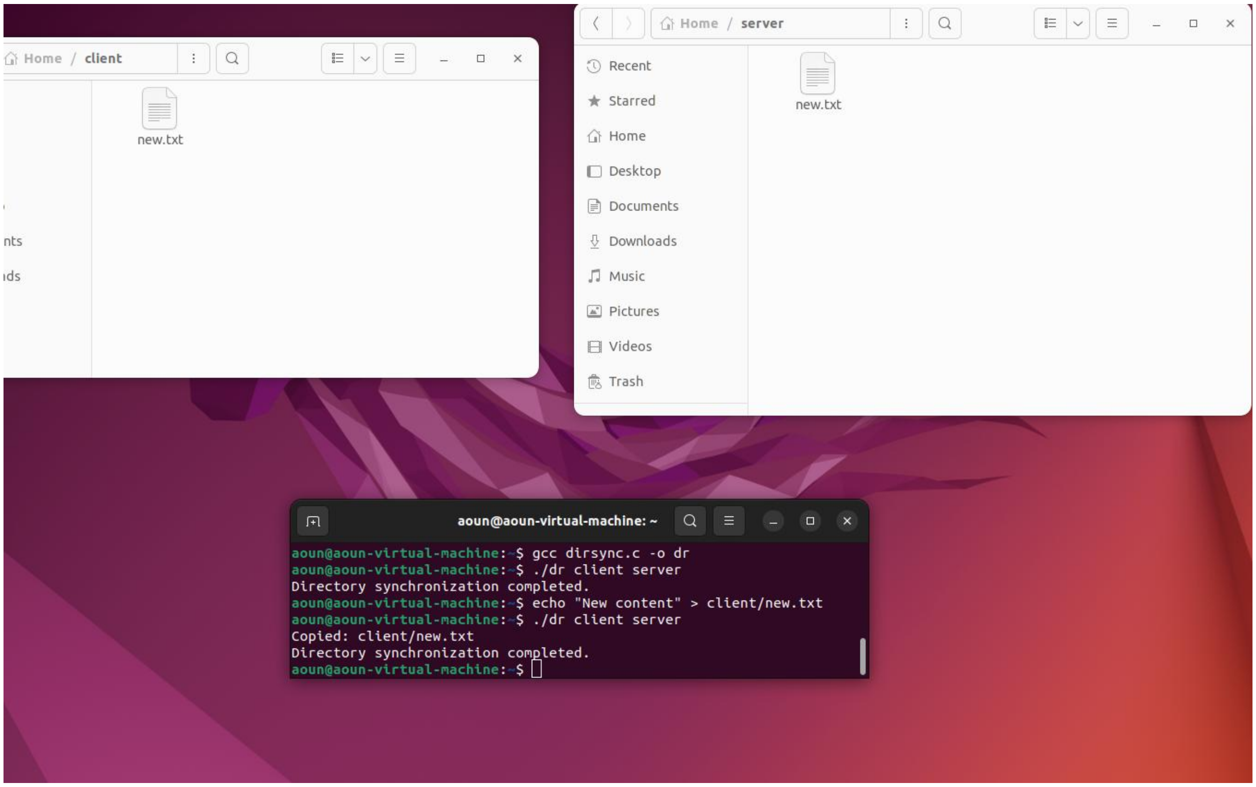
Key Functions:

1. **copy_file()**: Implements file copying using buffered I/O with a 4KB buffer for efficient data transfer.
2. **sync_directories()**: Main synchronization logic that:
    - Opens and reads the source directory
    - Creates destination directory if missing
    - Iterates through source files
    - Compares file sizes between source and destination
    - Copies files that don't exist or have different sizes
    - Skips subdirectories (Phase 1 limitation)

Limitations:

- No recursive directory support
- Simple size-based comparison (ignores modification time)
- No deletion of extra files in destination
- Manual execution required for each sync operation

Output Example



```
aoun@aoun-virtual-machine:~$ gcc dirsync.c -o dr
aoun@aoun-virtual-machine:~$ ./dr client server
Directory synchronization completed.
aoun@aoun-virtual-machine:~$ echo "New content" > client/new.txt
aoun@aoun-virtual-machine:~$ ./dr client server
Copied: client/new.txt
Directory synchronization completed.
aoun@aoun-virtual-machine:~$ 
```

Kernel System Call Implementation

Before proceeding to Phase 2, a custom kernel system call was implemented to enable efficient file metadata retrieval at the kernel level.

System Call Design

System Call Number: 451
Function Signature: sys_dirsync_stat(const char __user *path, struct dirsync_info __user *info)

Data Structure:

```
struct dirsync_info {
  int exists;
  long size;
  long mtime;
};
```

This structure encapsulates three critical pieces of file metadata:

- **exists**: Boolean flag indicating file existence

- **size**: File size in bytes
- **mtime**: Last modification timestamp (seconds since epoch)

Implementation Steps

1. Kernel Source Preparation

```
cd /usr/src
sudo wget https://cdn.kernel.org/pub/linux/kernel/v6.x/linux-6.1.tar.xz
sudo tar -xvf linux-6.1.tar.xz
cd linux-6.1
```

2. System Call Registration

Modified arch/x86/entry/syscalls/syscall_64.tbl:

```
451    common   dirsync_stat    sys_dirsync_stat
```

3. System Call Prototype Declaration

Added to include/linux/syscalls.h:

```
asmlinkage long sys_dirsync_stat(const char __user *path,
                         struct dirsync_info __user *info);
```

4. System Call Implementation

Created kernel/dirsync_stat.c with the following logic:

- Uses vfs_stat() to retrieve kernel-level file statistics
- Handles non-existent files gracefully by setting exists flag to 0
- Safely copies data to user space using copy_to_user()
- Returns appropriate error codes

5. Build System Integration

Modified kernel/Makefile:

```
obj-y += dirsync_stat.o
```

6. Kernel Compilation and Installation

```
cp /boot/config-$(uname -r) .config
yes "" | make oldconfig
sudo make -j$(nproc)
sudo make modules_install
sudo make install
sudo update-grub
sudo reboot
```

Advantages of Custom System Call

The custom system call provides several benefits over traditional user-space stat operations:

- **Single kernel transition**: Reduces context switching overhead
- **Atomic operation**: Ensures consistent metadata retrieval
- **Existence checking**: Built-in handling for non-existent files
- **Performance**: Eliminates multiple syscall overhead for metadata comparison

---

Phase 2: Recursive Synchronization with Custom System Call

Objectives

Phase 2 enhanced the synchronization system with recursive directory support, bidirectional synchronization (including deletions), and integration of the custom kernel system call for efficient file comparison.

Implementation Details

Major Enhancements:

1. **Recursive Directory Traversal**: Full support for nested directory structures with automatic subdirectory synchronization.
2. **Custom System Call Integration**: Replaced standard stat() calls with syscall(__NR_dirsync_stat, ...) for file comparison.
3. **Bidirectional Synchronization**: Added deletion detection and removal of files that exist in destination but not in source.
4. **Enhanced File Comparison**: Uses both size and modification time for accurate change detection.

Key Functions:

1. **files_are_different()**:
    - Uses custom syscall for both source and destination
    - Compares size and modification time
    - Returns 1 if files differ or don't exist

2. **sync_directory()**: Enhanced recursive version with two passes:
    - **First Pass**: Copy new/modified files and sync subdirectories
    - **Second Pass**: Delete files from destination that no longer exist in source

Syscall Usage:

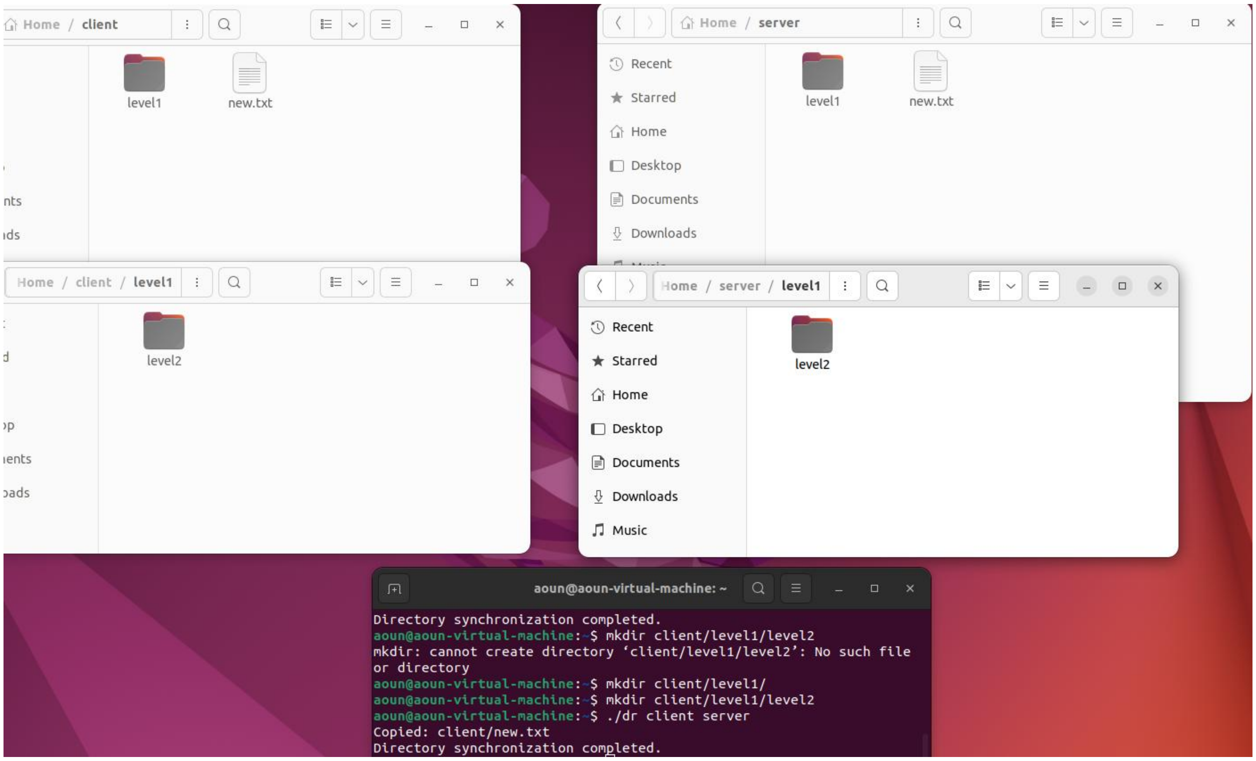
```
#define __NR_dirsync_stat 451

if (syscall(__NR_dirsync_stat, src, &src_info) < 0)
  return 1;
```

Synchronization Logic Flow

1. Open source directory
2. Create destination directory if needed
3. For each entry in source:
    - If directory: recursively sync
    - If file: compare using syscall and copy if different
4. Open destination directory
5. For each entry in destination:
    - If not in source: delete (file or directory)

Output Example



```
Directory synchronization completed.
aoun@aoun-virtual-machine:~$ mkdir client/level1/level2
mkdir: cannot create directory ‘client/level1/level2’: No such file
or directory
aoun@aoun-virtual-machine:~$ mkdir client/level1/
aoun@aoun-virtual-machine:~$ mkdir client/level1/level2
aoun@aoun-virtual-machine:~$ ./dr client server
Copied: client/new.txt
Directory synchronization completed.
```

Phase 3: Real-Time Monitoring with Inotify

Objectives

The final phase implements real-time directory monitoring using Linux inotify API, transforming the synchronizer from a one-time execution tool into a continuous background daemon.

Implementation Details

Major Features:

1. **Inotify Integration**: Monitors source directory for real-time file system events.
2. **Watch Descriptor Management**: Custom mapping system to track watched directories and their paths.
3. **Event-Driven Synchronization**: Responds immediately to file system changes rather than periodic scanning.
4. **Automatic Directory Watching**: Dynamically adds watches to newly created subdirectories.

Monitored Events:

- IN_CREATE - New file/directory creation
- IN_MODIFY - File content modification
- IN_DELETE - File/directory deletion
- IN_MOVED_FROM - File moved out of watched directory
- IN_MOVED_TO - File moved into watched directory
- IN_DELETE_SELF - Watched directory itself deleted

Watch Descriptor Mapping System

To efficiently track which directory triggered an event, a custom mapping system was implemented:

```
typedef struct {
  int wd;
  char path[PATH_MAX];
} wd_map_t;

wd_map_t wd_map[MAX_WATCHES];
```

Key Functions:

1. **add_wd_to_map()**: Registers new watch descriptors with their corresponding paths.
2. **get_path_from_wd()**: Retrieves directory path from watch descriptor during event processing.
3. **remove_wd_from_map()**: Cleans up mapping when directories are deleted.

Recursive Watch Setup

The add_watch_recursive() function ensures comprehensive monitoring:

- Adds inotify watch to specified directory
- Records watch descriptor and path in mapping
- Recursively traverses subdirectories
- Adds watches to all subdirectories

Event Processing Loop

The main event loop operates as follows:

1. **Initial Full Sync**: Performs complete synchronization before monitoring begins.
2. **Inotify Initialization**: Creates inotify instance and sets up recursive watches.

3. **Continuous Monitoring**:
    - Reads inotify events from kernel buffer
    - Looks up parent directory path using watch descriptor
    - Reconstructs full source and destination paths
    - Handles events based on type:
        - Creation/Modification: Calls handle_file() to sync
        - Deletion: Removes corresponding destination file
        - New directories: Adds recursive watches
4. **Path Reconstruction Logic**:

```
// Get parent directory from watch descriptor
const char *parent_path = get_path_from_wd(event->wd);

// Build full source path
snprintf(src_path, sizeof(src_path), "%s/%s", parent_path, event->name);

// Calculate relative path and build destination path
const char *relative_path = src_path + strlen(argv[1]);
snprintf(dst_path, sizeof(dst_path), "%s/%s", argv[2], relative_path);
```

Enhanced File Handling

The handle_file() function provides intelligent file operation handling:

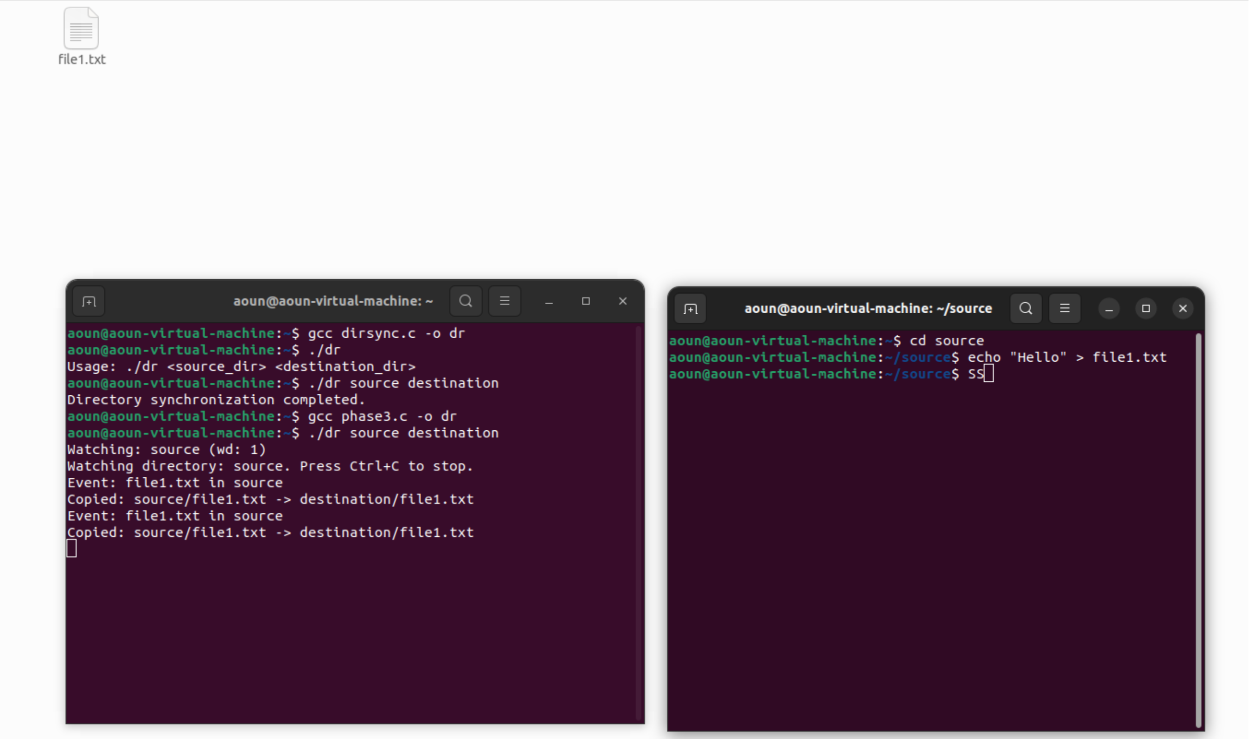
- Detects if source file/directory still exists
- Creates destination directories as needed
- Copies modified files using custom syscall comparison
- Deletes destination items when source is removed
- Handles both files and directories appropriately

Continuous Operation

Phase 3 transforms the synchronizer into a daemon-like process:

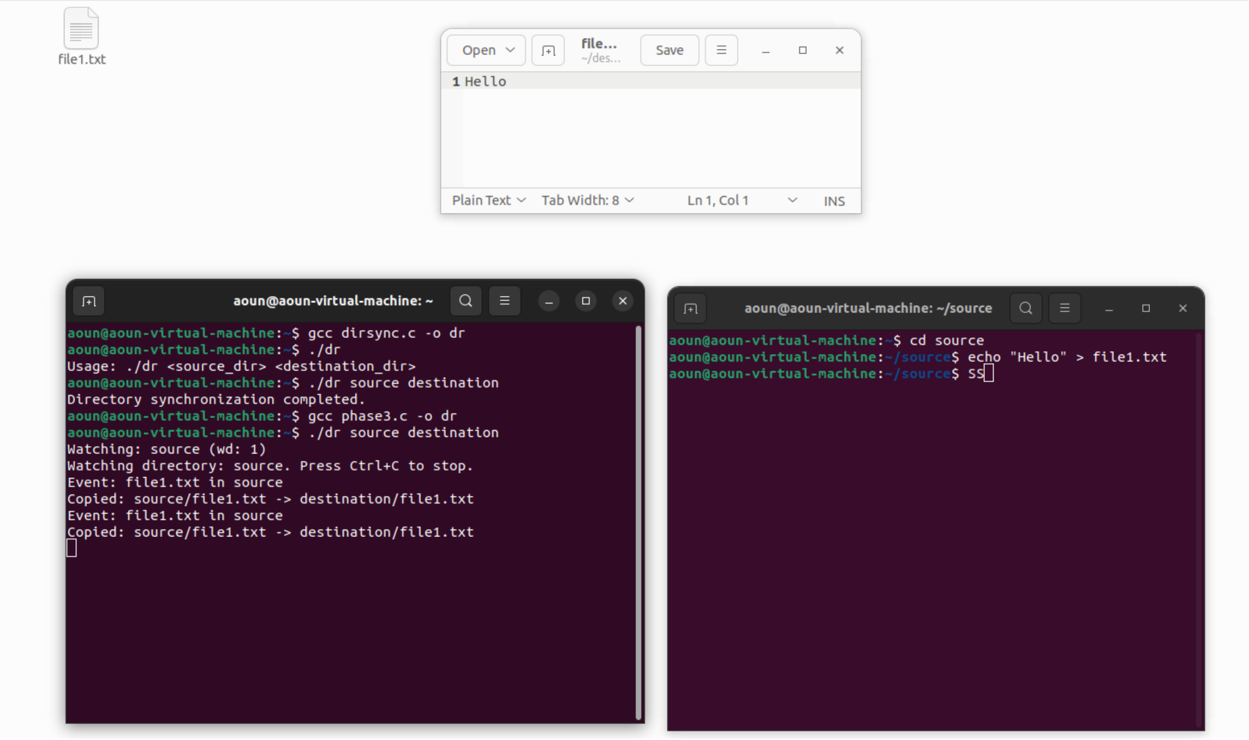
- Runs continuously until manually terminated (Ctrl+C)
- Responds to changes in real-time (typically milliseconds)
- Minimal CPU usage when idle (blocking read on inotify fd)
- Handles multiple simultaneous changes efficiently

Output Examples


file1.txt
aoun@aoun-virtual-machine: ~
aoun@aoun-virtual-machine: ~/source

```
aoun@aoun-virtual-machine:~$ gcc dirsync.c -o dr
aoun@aoun-virtual-machine:~$ ./dr
Usage: ./dr <source_dir> <destination_dir>
aoun@aoun-virtual-machine:~$ ./dr source destination
Directory synchronization completed.
aoun@aoun-virtual-machine:~$ gcc phase3.c -o dr
aoun@aoun-virtual-machine:~$ ./dr source destination
Watching: source (wd: 1)
Watching directory: source. Press Ctrl+C to stop.
Event: file1.txt in source
Copied: source/file1.txt -> destination/file1.txt
Event: file1.txt in source
Copied: source/file1.txt -> destination/file1.txt
```

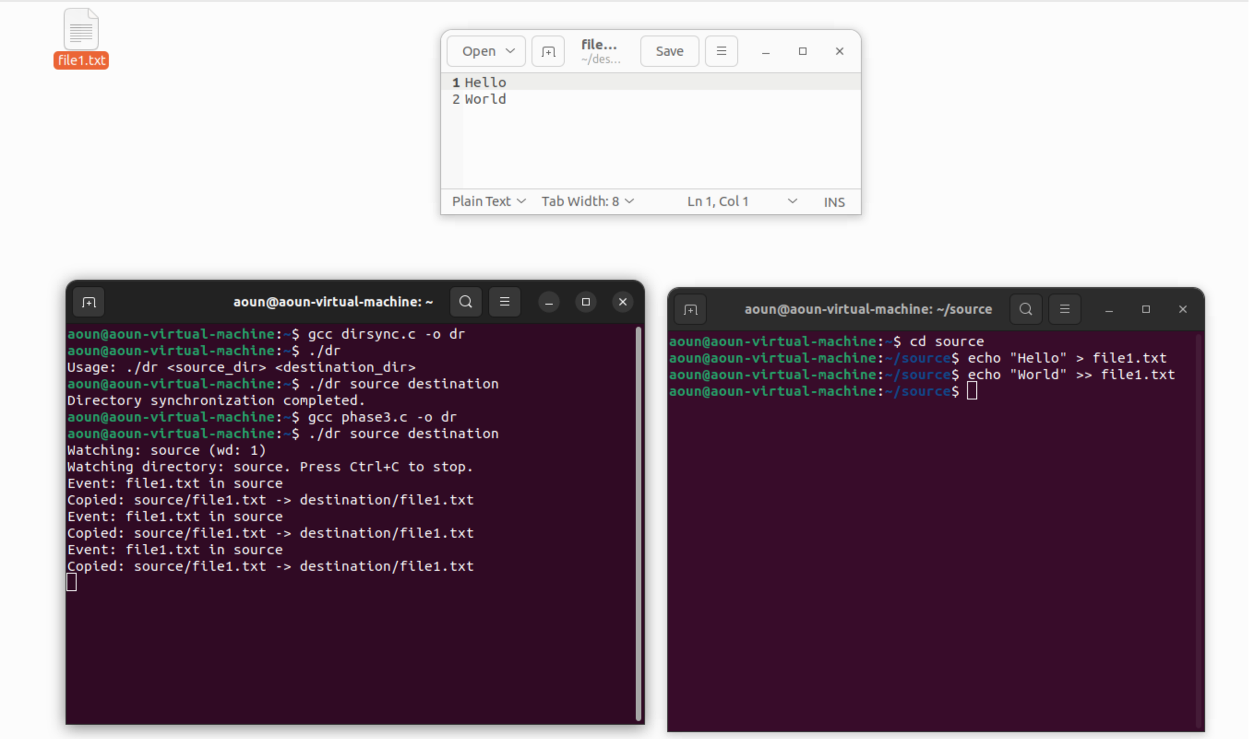
```
aoun@aoun-virtual-machine:~$ cd source
aoun@aoun-virtual-machine:~/source$ echo "Hello" > file1.txt
aoun@aoun-virtual-machine:~/source$ SS
```



```
1 Hello
```

```
aoun@aoun-virtual-machine:~$ gcc dirsync.c -o dr
aoun@aoun-virtual-machine:~$ ./dr
Usage: ./dr <source_dir> <destination_dir>
aoun@aoun-virtual-machine:~$ ./dr source destination
Directory synchronization completed.
aoun@aoun-virtual-machine:~$ gcc phase3.c -o dr
aoun@aoun-virtual-machine:~$ ./dr source destination
Watching: source (wd: 1)
Watching directory: source. Press Ctrl+C to stop.
Event: file1.txt in source
Copied: source/file1.txt -> destination/file1.txt
Event: file1.txt in source
Copied: source/file1.txt -> destination/file1.txt
```

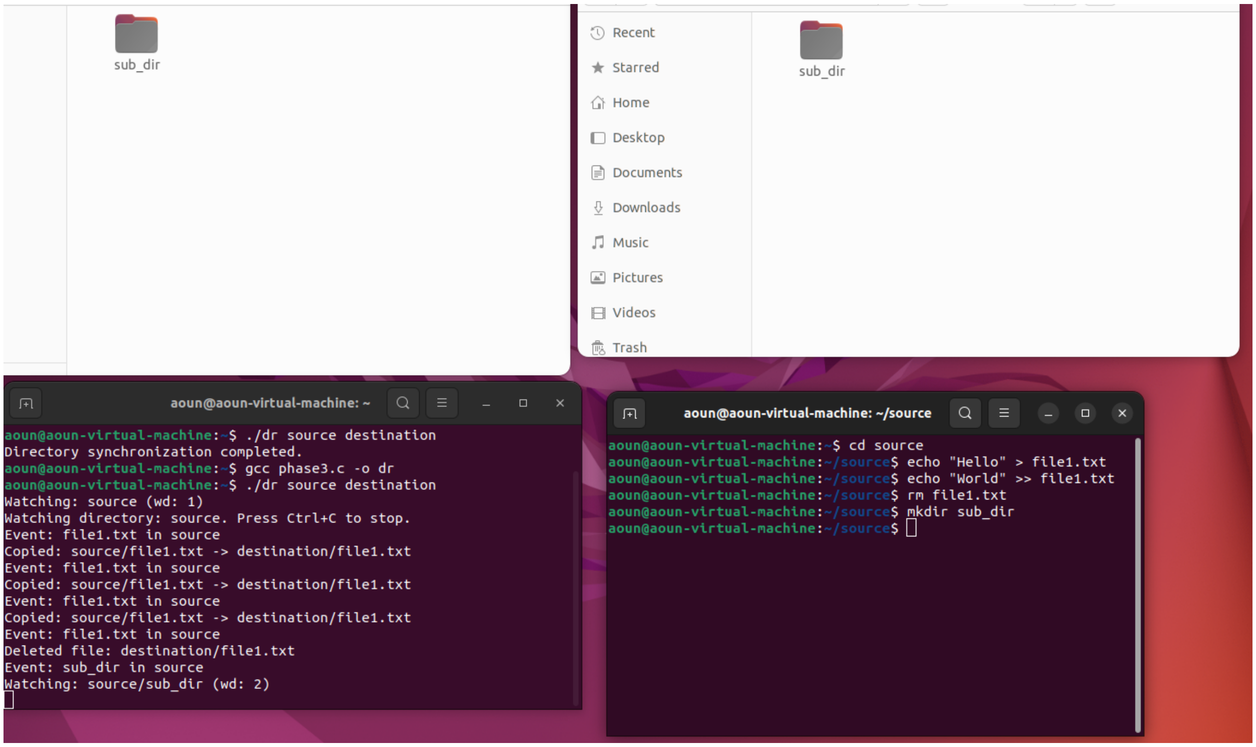
```
aoun@aoun-virtual-machine:~$ cd source
aoun@aoun-virtual-machine:~/source$ echo "Hello" > file1.txt
aoun@aoun-virtual-machine:~/source$ SS
```



```
Hello
World
```

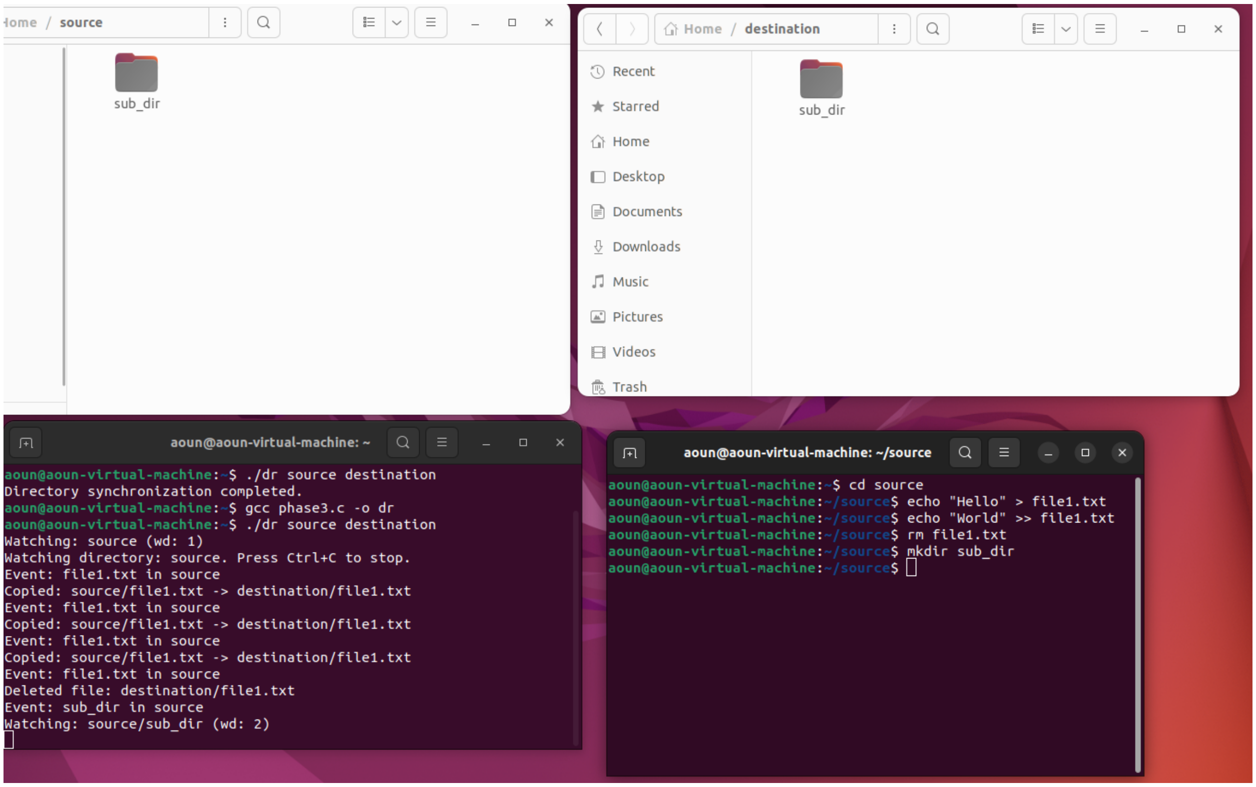
```
aoun@aoun-virtual-machine:~$ gcc dirsync.c -o dr
aoun@aoun-virtual-machine:~$ ./dr
Usage: ./dr <source_dir> <destination_dir>
aoun@aoun-virtual-machine:~$ ./dr source destination
Directory synchronization completed.
aoun@aoun-virtual-machine:~$ gcc phase3.c -o dr
aoun@aoun-virtual-machine:~$ ./dr source destination
Watching: source (wd: 1)
Watching directory: source. Press Ctrl+C to stop.
Event: file1.txt in source
Copied: source/file1.txt -> destination/file1.txt
Event: file1.txt in source
Copied: source/file1.txt -> destination/file1.txt
Event: file1.txt in source
Copied: source/file1.txt -> destination/file1.txt
```

```
aoun@aoun-virtual-machine:~$ cd source
aoun@aoun-virtual-machine:~/source$ echo "Hello" > file1.txt
aoun@aoun-virtual-machine:~/source$ echo "World" >> file1.txt
aoun@aoun-virtual-machine:~/source$
```



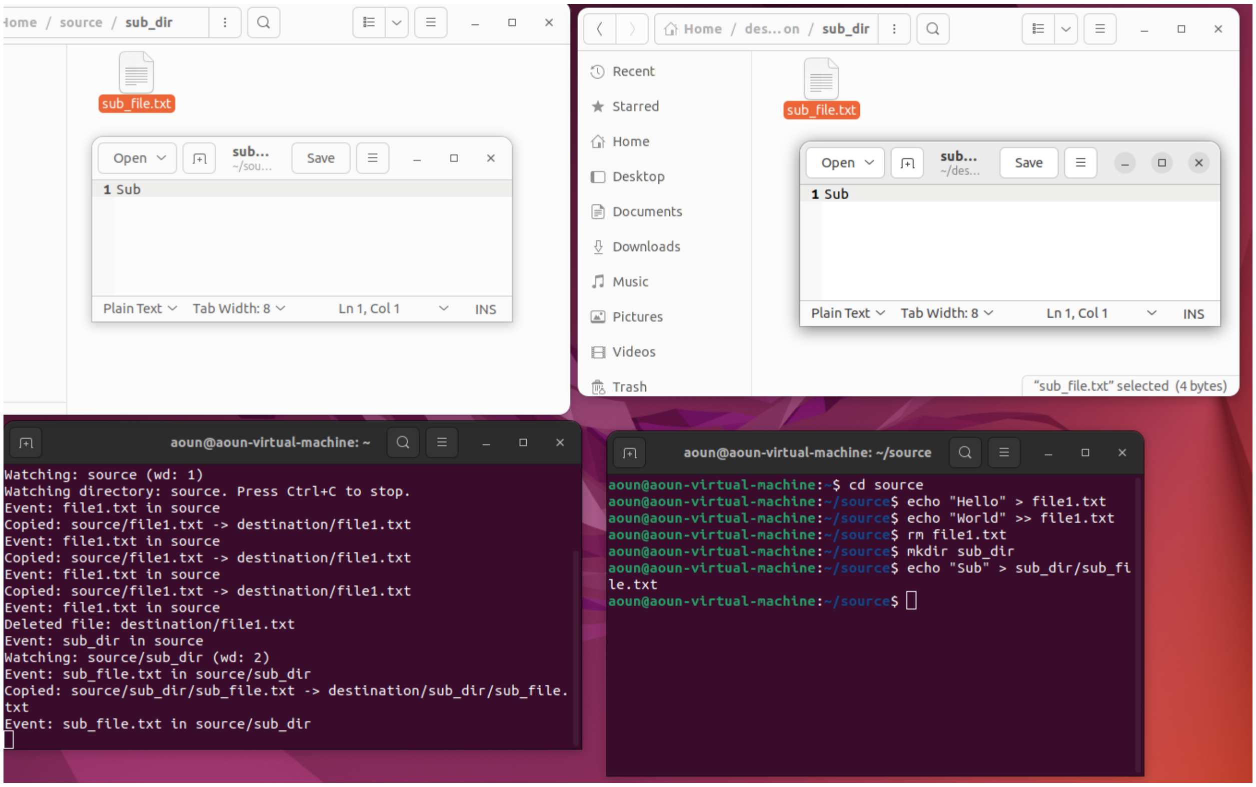
```
aoun@aoun-virtual-machine:~$ ./dr source destination
Directory synchronization completed.
aoun@aoun-virtual-machine:~$ gcc phase3.c -o dr
aoun@aoun-virtual-machine:~$ ./dr source destination
Watching: source (wd: 1)
Watching directory: source. Press Ctrl+C to stop.
Event: file1.txt in source
Copied: source/file1.txt -> destination/file1.txt
Event: file1.txt in source
Copied: source/file1.txt -> destination/file1.txt
Event: file1.txt in source
Copied: source/file1.txt -> destination/file1.txt
Event: file1.txt in source
Deleted file: destination/file1.txt
Event: sub_dir in source
Watching: source/sub_dir (wd: 2)
```

```
aoun@aoun-virtual-machine:~$ cd source
aoun@aoun-virtual-machine:~/source$ echo "Hello" > file1.txt
aoun@aoun-virtual-machine:~/source$ echo "World" >> file1.txt
aoun@aoun-virtual-machine:~/source$ rm file1.txt
aoun@aoun-virtual-machine:~/source$ mkdir sub_dir
aoun@aoun-virtual-machine:~/source$
```



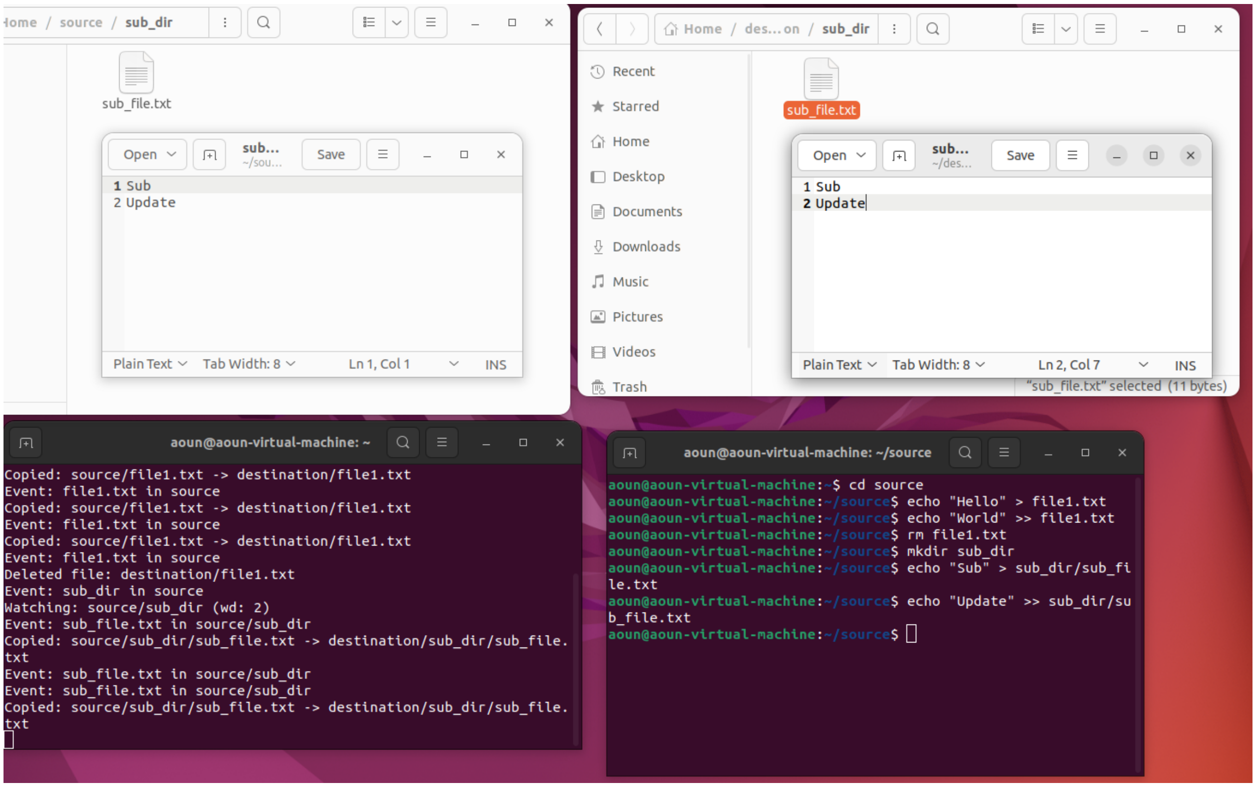
```
aoun@aoun-virtual-machine:~$ ./dr source destination
Directory synchronization completed.
aoun@aoun-virtual-machine:~$ gcc phase3.c -o dr
aoun@aoun-virtual-machine:~$ ./dr source destination
Watching: source (wd: 1)
Watching directory: source. Press Ctrl+C to stop.
Event: file1.txt in source
Copied: source/file1.txt -> destination/file1.txt
Event: file1.txt in source
Copied: source/file1.txt -> destination/file1.txt
Event: file1.txt in source
Copied: source/file1.txt -> destination/file1.txt
Event: file1.txt in source
Deleted file: destination/file1.txt
Event: sub_dir in source
Watching: source/sub_dir (wd: 2)
```

```
aoun@aoun-virtual-machine:~$ cd source
aoun@aoun-virtual-machine:~/source$ echo "Hello" > file1.txt
aoun@aoun-virtual-machine:~/source$ echo "World" >> file1.txt
aoun@aoun-virtual-machine:~/source$ rm file1.txt
aoun@aoun-virtual-machine:~/source$ mkdir sub_dir
aoun@aoun-virtual-machine:~/source$ 
```



```
Watching: source (wd: 1)
Watching directory: source. Press Ctrl+C to stop.
Event: file1.txt in source
Copied: source/file1.txt -> destination/file1.txt
Event: file1.txt in source
Copied: source/file1.txt -> destination/file1.txt
Event: file1.txt in source
Copied: source/file1.txt -> destination/file1.txt
Event: file1.txt in source
Deleted file: destination/file1.txt
Event: sub_dir in source
Watching: source/sub_dir (wd: 2)
Event: sub_file.txt in source/sub_dir
Copied: source/sub_dir/sub_file.txt -> destination/sub_dir/sub_file.
txt
Event: sub_file.txt in source/sub_dir
```

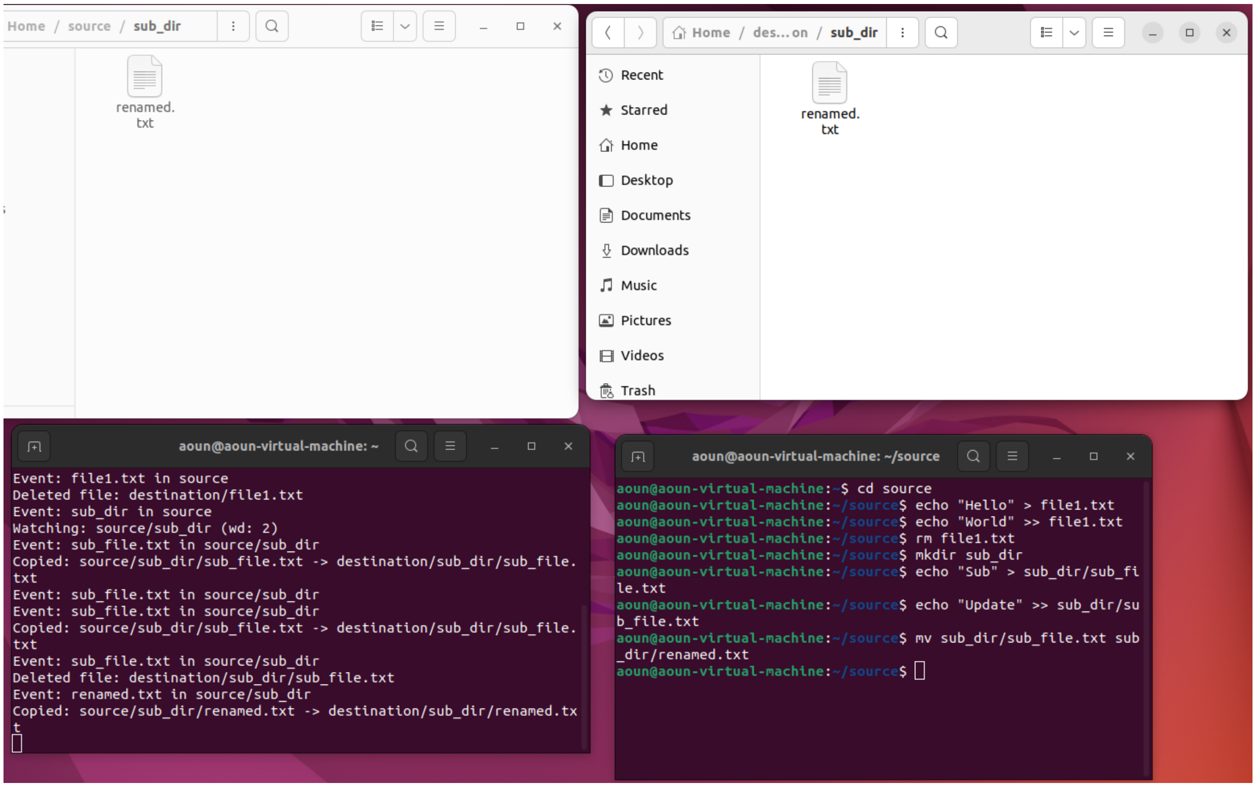
```
aoun@aoun-virtual-machine:~$ cd source
aoun@aoun-virtual-machine:~/source$ echo "Hello" > file1.txt
aoun@aoun-virtual-machine:~/source$ echo "World" >> file1.txt
aoun@aoun-virtual-machine:~/source$ rm file1.txt
aoun@aoun-virtual-machine:~/source$ mkdir sub_dir
aoun@aoun-virtual-machine:~/source$ echo "Sub" > sub_dir/sub_fi
le.txt
aoun@aoun-virtual-machine:~/source$ 
```



```
1 Sub
2 Update
```

```
Copied: source/file1.txt -> destination/file1.txt
Event: file1.txt in source
Copied: source/file1.txt -> destination/file1.txt
Event: file1.txt in source
Copied: source/file1.txt -> destination/file1.txt
Event: file1.txt in source
Deleted file: destination/file1.txt
Event: sub_dir in source
Watching: source/sub_dir (wd: 2)
Event: sub_file.txt in source/sub_dir
Copied: source/sub_dir/sub_file.txt -> destination/sub_dir/sub_file.
txt
Event: sub_file.txt in source/sub_dir
Event: sub_file.txt in source/sub_dir
Copied: source/sub_dir/sub_file.txt -> destination/sub_dir/sub_file.
txt
```

```
aoun@aoun-virtual-machine:~$ cd source
aoun@aoun-virtual-machine:~/source$ echo "Hello" > file1.txt
aoun@aoun-virtual-machine:~/source$ echo "World" >> file1.txt
aoun@aoun-virtual-machine:~/source$ rm file1.txt
aoun@aoun-virtual-machine:~/source$ mkdir sub_dir
aoun@aoun-virtual-machine:~/source$ echo "Sub" > sub_dir/sub_fi
le.txt
aoun@aoun-virtual-machine:~/source$ echo "Update" >> sub_dir/su
b_file.txt
aoun@aoun-virtual-machine:~/source$ 
```



```
Event: file1.txt in source
Deleted file: destination/file1.txt
Event: sub_dir in source
Watching: source/sub_dir (wd: 2)
Event: sub_file.txt in source/sub_dir
Copied: source/sub_dir/sub_file.txt -> destination/sub_dir/sub_file.
txt
Event: sub_file.txt in source/sub_dir
Event: sub_file.txt in source/sub_dir
Copied: source/sub_dir/sub_file.txt -> destination/sub_dir/sub_file.
txt
Event: sub_file.txt in source/sub_dir
Deleted file: destination/sub_dir/sub_file.txt
Event: renamed.txt in source/sub_dir
Copied: source/sub_dir/renamed.txt -> destination/sub_dir/renamed.tx
t
```

```
aoun@aoun-virtual-machine:~$ cd source
aoun@aoun-virtual-machine:~/source$ echo "Hello" > file1.txt
aoun@aoun-virtual-machine:~/source$ echo "World" >> file1.txt
aoun@aoun-virtual-machine:~/source$ rm file1.txt
aoun@aoun-virtual-machine:~/source$ mkdir sub_dir
aoun@aoun-virtual-machine:~/source$ echo "Sub" > sub_dir/sub_fi
le.txt
aoun@aoun-virtual-machine:~/source$ echo "Update" >> sub_dir/su
b_file.txt
aoun@aoun-virtual-machine:~/source$ mv sub_dir/sub_file.txt sub
_dir/renamed.txt
aoun@aoun-virtual-machine:~/source$ 
```

Technical Comparison Across Phases

Feature	Phase 1	Phase 2	Phase 3
Directory Support	Flat only	Recursive	Recursive
File Comparison	Size only	Size + mtime	Size + mtime
System Call	Standard stat()	Custom syscall	Custom syscall
Deletion Handling	No	Yes	Yes
Execution Mode	One-time	One-time	Continuous
Event Monitoring	Manual	Manual	Real-time (inotify)
Watch Management	N/A	N/A	Dynamic mapping
Subdirectory Detection	No	Yes	Automatic

Performance Considerations

Phase 1 & 2

- Performance depends on directory size and file count
- Full directory scan required for each execution
- I/O bound operation limited by disk speed

Phase 3

- Minimal overhead during idle monitoring
- Event-driven approach eliminates unnecessary scans
- Immediate response to file system changes
- Scalable to large directory structures

Compilation and Usage

Compilation

```
gcc -o dirsync dirsync.c
```

Execution

Phase 1 & 2:

```
./dirsync /path/to/source /path/to/destination
```

Phase 3 (Continuous Monitoring):

```
./dirsync /path/to/source /path/to/destination
# Press Ctrl+C to stop monitoring
```

Conclusion

The Directory Synchronizer project successfully demonstrates the progression from basic file operations to advanced kernel-level integration and real-time monitoring. Each phase built upon the previous implementation, addressing limitations and adding sophisticated features:

- **Phase 1** established the foundation with basic synchronization logic
- **Phase 2** introduced recursive support and custom kernel system call integration for efficient metadata comparison
- **Phase 3** achieved real-time monitoring capabilities with inotify, transforming the tool into a production-ready synchronization daemon

The custom kernel system call implementation showcases deep understanding of Linux kernel internals, while the inotify integration demonstrates mastery of advanced system programming concepts. The resulting tool is efficient, robust, and suitable for real-world directory synchronization tasks.

Future Enhancements

Potential improvements for future iterations:

- Configuration file support for multiple sync pairs
- Network synchronization support (remote directories)
- Conflict resolution strategies for bidirectional sync
- Bandwidth throttling and scheduling options
- GUI interface for easier management
- Logging and notification system
- Checksum-based verification for data integrity

References

- Linux Kernel 6.1 Documentation
- The Linux Programming Interface by Michael Kerrisk

- Linux System Call Table - arch/x86/entry/syscalls/
- inotify(7) - Linux manual page
- POSIX.1-2008 Standards Documentation