Report: Directory Copying Utility "MWCp"

EMRE YAVUZ 200104004003 HW4 CSE 344

Objective

The goal is to develop a directory copying utility called "MWCp" that copies files and subdirectories in parallel using a worker-manager approach. The program synchronizes thread activities using POSIX and standard C libraries. The program is tested with various buffer sizes and worker numbers.

General Structure

1. **Initialization:**
    - Accept command-line arguments: buffer size, number of workers, source directory, and destination directory.
    - Initialize global variables and buffer.
    - Set up signal handlers for graceful termination.
    - Measure start time for performance evaluation.
2. **Manager Thread:**
    - Traverse the source directory and for each file or directory:
        - If a directory, create the corresponding directory in the destination.
        - If a file, open the source and destination files and pass their descriptors to the buffer.
    - Signal completion when traversal is done.
3. **Worker Threads:**
    - Continuously read file descriptors from the buffer.
    - Copy the contents from the source file to the destination file.
    - Update statistics (number of files copied, total bytes copied).
    - Terminate when signaled.
4. **Buffer Management:**
    - Implement a circular buffer to hold file descriptors.
    - Use mutexes and condition variables to synchronize access to the buffer.
    - Ensure that the buffer is neither overfilled nor underfilled.

5. **Signal Handling:**
    - Handle **SIGINT** (Ctrl+C) and **SIGTSTP** (Ctrl+Z) to terminate the program gracefully.
    - Broadcast condition variables to wake up waiting threads.
6. **Performance Measurement:**
    - Measure end time and calculate the elapsed time.
    - Print statistics including the number of files and directories copied, total bytes copied, and total time taken.

Pseudocode :

```
// Initialization
parse_command_line_arguments()
initialize_global_variables()
set_signal_handlers()
start_timer()

// Create and start manager thread
create_thread(manager_thread, args)

// Create and start worker threads
for each worker in num_workers:
    create_thread(worker_thread, NULL)

// Wait for threads to finish
join_thread(manager_thread)
for each worker in num_workers:
    join_thread(worker_thread)

// Clean up resources
destroy_mutexes_and_conditions()
destroy_buffer()
stop_timer()
print_statistics()

// Manager thread function
manager_thread(args):
    src_dir, dest_dir = args
    traverse_directory(src_dir, dest_dir)
    lock(buffer_mutex)
    set done to 1
    broadcast_conditions()
    unlock(buffer_mutex)
    return
```

```
// Copy file function
copy_file(src_fd, dest_fd):
    total_bytes = 0
    while bytes_read from src_fd:
        write bytes_read to dest_fd
        if write fails:
            log_error
            break
        total_bytes += bytes_written
    return total_bytes

// Signal handler function
handle_signal(sig):
    set done to 1
    broadcast_conditions()
    return

// Print statistics function
print_statistics():
    print("-------- STATISTICS --------")
    print("Consumers: ", num_workers)
    print("Buffer Size: ", buffer_size)
    print("Number of Regular Files: ", files_copied)
    print("Number of Directories: ", dirs_created)
    print("TOTAL BYTES COPIED: ", total_bytes_copied)
    print("TOTAL TIME: ", elapsed_time)
    return
```

```
// Traverse directory function
traverse_directory(src_dir, dest_dir):
    for each entry in src_dir:
        if entry is directory:
            create_corresponding_directory_in_dest()
            traverse_directory(src_subdir, dest_subdir)
        else if entry is file:
            open_src_and_dest_files()
            file_data = create_file_data(src_fd, dest_fd, src_path, dest_path)
            lock(buffer_mutex)
            while buffer_is_full and not done:
                wait(buffer_not_full, buffer_mutex)
            if done:
                unlock(buffer_mutex)
                close_files(src_fd, dest_fd)
                return
            buffer_write(file_data)
            signal(buffer_not_empty)
            unlock(buffer_mutex)
    return

// Worker thread function
worker_thread(args):
    while true:
        lock(buffer_mutex)
        while buffer_is_empty and not done:
            wait(buffer_not_empty, buffer_mutex)
        if done and buffer_is_empty:
            unlock(buffer_mutex)
            break
        file_data = buffer_read()
        signal(buffer_not_full)
        unlock(buffer_mutex)
        bytes_copied = copy_file(file_data.src_fd, file_data.dest_fd)
        lock(stats_mutex)
        update_statistics(bytes_copied, file_data.src_name, file_data.dest_name)
        unlock(stats_mutex)
        close_files(file_data.src_fd, file_data.dest_fd)
    return
```

Testing Scenarios

The program is tested with the following scenarios:

1. **Test1:**

valgrind ./MWCp 10 10 ../testdir/src/libvterm ../tocopy

- Memory leak checking, buffer size = 10, number of workers = 10

```
emre@emre-VirtualBox:~/Desktop/hw4/hw4test/put_your_codes_here$ valgrind ./MWCp 10 10 ..
/testdir/src/libvterm ../tocopy
==4762== Memcheck, a memory error detector
==4762== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==4762== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==4762== Command: ./MWCp 10 10 ../testdir/src/libvterm ../tocopy
==4762==




---------------STATISTICS--------------------
Consumers: 10 - Buffer Size: 10
Number of Regular Files: 194
Number of FIFO Files: 0
Number of Directories: 7
TOTAL BYTES COPIED: 25009680
TOTAL TIME: 00:03.196 (min:sec.mili)
==4762==
==4762== HEAP SUMMARY:
==4762==     in use at exit: 0 bytes in 0 blocks
==4762==   total heap usage: 21 allocs, 21 frees, 348,544 bytes allocated
==4762==
==4762== All heap blocks were freed -- no leaks are possible
==4762==
==4762== For lists of detected and suppressed errors, rerun with: -s
==4762== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
emre@emre-VirtualBox:~/Desktop/hw4/hw4test/put_your_codes_here$
emre@emre-VirtualBox:~/Desktop/hw4/hw4test/put_your_codes_here$
emre@emre-VirtualBox:~/Desktop/hw4/hw4test/put_your_codes_here$
emre@emre-VirtualBox:~/Desktop/hw4/hw4test/put_your_codes_here$ ./MWCp 10 10 ../testdir/
src/libvterm ../tocopy

---------------STATISTICS--------------------
Consumers: 10 - Buffer Size: 10
Number of Regular Files: 194
Number of FIFO Files: 0
Number of Directories: 7
TOTAL BYTES COPIED: 25009680
TOTAL TIME: 00:02.-809 (min:sec.mili)
```

2. **Test2:**

./MWCp 10 4 ../testdir/src/libvterm/src ../toCopy

- Buffer size = 10, number of workers = 4

```
emre@emre-VirtualBox:~/Desktop/hw4/hw4test/put_your_codes_here$ valgrind ./MWCp 10 4 ../
testdir/src/libvterm/src ../toCopy
==4816== Memcheck, a memory error detector
==4816== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==4816== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==4816== Command: ./MWCp 10 4 ../testdir/src/libvterm/src ../toCopy
==4816==



--------------STATISTICS-------------------
Consumers: 4 - Buffer Size: 10
Number of Regular Files: 140
Number of FIFO Files: 0
Number of Directories: 2
TOTAL BYTES COPIED: 24873082
TOTAL TIME: 00:03.-34 (min:sec.mili)
==4816==
==4816== HEAP SUMMARY:
==4816==     in use at exit: 0 bytes in 0 blocks
==4816==   total heap usage: 10 allocs, 10 frees, 182,832 bytes allocated
==4816==
==4816== All heap blocks were freed -- no leaks are possible
==4816==
==4816== For lists of detected and suppressed errors, rerun with: -s
==4816== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
emre@emre-VirtualBox:~/Desktop/hw4/hw4test/put_your_codes_here$
emre@emre-VirtualBox:~/Desktop/hw4/hw4test/put_your_codes_here$
emre@emre-VirtualBox:~/Desktop/hw4/hw4test/put_your_codes_here$ ./MWCp 10 4 ../testdir/s
rc/libvterm/src ../toCopy

--------------STATISTICS-------------------
Consumers: 4 - Buffer Size: 10
Number of Regular Files: 140
Number of FIFO Files: 0
Number of Directories: 2
TOTAL BYTES COPIED: 24873082
TOTAL TIME: 00:00.498 (min:sec.mili)
```

3. **Test3:**

./MWCp 10 10 ../testdir ../toCopy

- Buffer size = 10, number of workers = 10

```
emre@emre-VirtualBox:~/Desktop/hw4/hw4test/put_your_codes_here$ valgrind ./MWCp 10 10 ..
/testdir ../toCopy
==4874== Memcheck, a memory error detector
==4874== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==4874== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==4874== Command: ./MWCp 10 10 ../testdir ../toCopy
==4874==

---------------STATISTICS--------------------
Consumers: 10 - Buffer Size: 10
Number of Regular Files: 3116
Number of FIFO Files: 0
Number of Directories: 151
TOTAL BYTES COPIED: 73520554
TOTAL TIME: 00:15.-528 (min:sec.mili)
==4874==
==4874== HEAP SUMMARY:
==4874==     in use at exit: 0 bytes in 0 blocks
==4874==   total heap usage: 165 allocs, 165 frees, 5,074,048 bytes allocated
==4874==
==4874== All heap blocks were freed -- no leaks are possible
==4874==
==4874== For lists of detected and suppressed errors, rerun with: -s
==4874== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
emre@emre-VirtualBox:~/Desktop/hw4/hw4test/put_your_codes_here$ ./MWCp 10 10 ../testdir
../toCopy

---------------STATISTICS--------------------
Consumers: 10 - Buffer Size: 10
Number of Regular Files: 3116
Number of FIFO Files: 0
Number of Directories: 151
TOTAL BYTES COPIED: 73520554
TOTAL TIME: 00:07.-06 (min:sec.mili)
```

Observations

- The program handles large files and directories efficiently with multiple workers.
- Proper synchronization ensures no race conditions or deadlocks.
- Signal handling allows for graceful termination and resource cleanup.
- Performance metrics provide insights into the execution time and efficiency of the program.

Conclusion

The "MWCp" utility successfully implements a parallel directory copying mechanism using a worker-manager approach. It efficiently handles synchronization, signal handling, and resource management, making it a robust solution for copying large directories.