GTU Department of Computer Engineering

CSE 344 – Spring 2024

Homework 5 Report

Barış Batuhan Bolat

210104004029

Note

- I used mutexes and condition variables in homework 4. Now I added barriers to make homework 5.

Functions

1. Main

- Parses command-line arguments into **buffer_size**, **num_workers**, **src_dir** and **dest_dir**.
- Allocates memory for the worker threads using malloc.
- Initializes the barrier with **pthread_barrier_init**, where **num_workers + 1** includes the manager thread and all worker threads.
- Sets up a signal handler for **SIGINT** (**Ctrl+C**) using **sigaction**.
- Starts the manager thread using **pthread_create** to initiate the copy process.
- Creates the specified number of worker threads using **pthread_create**. These threads will wait for files to be added to the shared buffer and copy them.
- Waits for the manager thread to finish using pthread_join.
- Waits for all worker threads to finish using pthread_join (ensures all files are copied before exiting).
- Calculates the elapsed time using the **clock_gettime** function before and after the copy process.
- Prints various statistics to standard output:
    - Elapsed time for the copy operation.
    - Number of regular files copied.
    - Number of directories copied.
    - Number of FIFO files copied.
    - Total number of files processed.
    - Total number of bytes copied.
- Frees the memory allocated for the worker threads using **free**.
- Destroys the mutex, condition variables and barrier used for thread synchronization using **pthread_mutex_destroy**, **pthread_cond_destroy and pthread_barrier_destroy**.

2. signal_handler

- Prints a message to the standard output indicating that **SIGINT** was received and the program is cleaning up.
- Sets the **done** flag to **1**, which signals to other threads that the program is exiting.
- Broadcasts a signal to both the **buffer_cond_full** and **buffer_cond_empty** condition variables, which can unblock any threads waiting on them.
- Frees the memory allocated for the worker threads using free.
- Exits the program with an exit code of 1.

3. manager

- Checks if the destination directory exists and has the correct permissions using **stat**. If not it creates.
- Manager thread waits with **pthread_barrier_wait** until all worker threads are ready before starting the directory copy process.

- Calls **copy_directory** to start the recursive copy process from the source directory to the destination directory.
- Sets the done flag to 1 to signal to other threads that the copying process is complete.
- Broadcasts a signal to the **buffer_cond_full** condition variable to unblock any worker threads waiting for new items in the buffer.

4. worker

- Continuously loops to process files:
- Each worker thread calls **pthread_barrier_wait** to wait until all threads have reached this point, ensuring synchronization before they start processing files from the buffer.

    - Calls **remove_buffer** to retrieve a **FileInfo** structure containing source and destination file paths.
    - If the retrieved **FileInfo** structure has empty source and destination paths (indicated by '\0'), it breaks out of the loop, signifying there are no more files to copy and the program is exiting.
    - Calls **copy_file** to copy the file from the source path to the destination path.
    - Prints a message to standard error indicating which file was copied.

5. copy_directory

- Opens the source directory using **opendir**.
- Iterates through the directory entries using **readdir**.
- Skips entries for "." and ".." directories.
- Retrieves file status information using **lstat**.
- If the file is a directory:
    - Creates the corresponding directory in the destination path using **mkdir**. It sets the directory permissions to match the source directory's permissions.
    - Calls **copy_directory** recursively to process the contents of the subdirectory.
- If the file is a regular file:
    - Creates a **FileInfo** structure containing the source and destination file paths.
    - Calls **insert_buffer** to add the **FileInfo** structure to the shared buffer, allowing worker threads to copy the file later.
    - Increments the **num_regular_files** counter to track the number of **regular files** copied.
- If the file is a FIFO file (based on the S_ISFIFO macro):
    - Increments the **num_fifo_files** counter to track the number of **FIFO** files encountered.

6. copy_file

- Opens the source file using open with read-only permissions.
- Creates the destination file using open with write-only, create, and truncate permissions. It also sets the file permissions to allow read access for the owner, group, and others.
- Reads data from the source using read.
- Writes the read data to the destination file in chunks using write.
- Checks for errors during read and write operations. If an error occurs, it prints an error message to the standard error output and closes both files before returning.
- Closes both the source and destination files using close.

7. insert_buffer

- Acquires the lock on the **buffer_mutex** mutex to ensure thread safety when accessing the shared buffer.
- It checks if the buffer is full.
- If the buffer is full, the function waits on the **buffer_cond_empty** condition variable, releasing the lock on the mutex in the meantime. This allows other threads to acquire the lock and potentially add space to the buffer.
- Once the buffer has space, the function reacquires the lock on the mutex and copies the **FileInfo** structure into the next available slot in the buffer.
- It increments the **buffer_count** variable to reflect the addition of a new element.
- Signals the buffer_cond_full condition variable to unblock any worker threads waiting for new items in the buffer.
- Finally, releases the lock on the **buffer_mutex**.

8. remove_buffer

- Acquires the lock on the **buffer_mutex** mutex to ensure thread safety when accessing the shared buffer.
- It checks if the buffer is empty and the **done** flag is not set.
- If the buffer is empty and the program is not exiting, the function waits on the **buffer_cond_full** condition variable, releasing the lock on the mutex in the meantime. This allows other threads to acquire the lock and potentially add new items to the buffer.
- If the buffer is empty and the program is exiting, the function reacquires the lock on the mutex and sets both source and destination file paths in the **FileInfo** structure to empty strings. This signals to worker threads that there are no more files to copy.
- Otherwise, the function retrieves the **FileInfo** structure from the head of the buffer, increments the **buffer_head** index to point to the next element, and decrements the **buffer_count** variable to reflect the removal of an element.
- Signals the **buffer_cond_empty** condition variable to unblock any manager threads that might be waiting for space to add new items to the buffer.
- Finally, releases the lock on the **buffer_mutex** and returns the retrieved FileInfo structure.

General Structure

```
main:

        parse command-line arguments

        set up signal handler for SIGINT

        create manager thread

        create worker threads

        wait for manager and worker threads to finish

        print statistics

        clean up resources

        exit

manager thread:
```

```
	check if destination directory exists, create if necessary
	call copy_directory(source_dir, dest_dir)
	set done flag
	signal worker threads
copy_directory(src_dir, dest_dir):
	for each entry in src_dir:
		if entry is a regular file:
			increment num_regular_files
			add FileInfo to shared buffer
		else if entry is a directory:
			increment num_directories
			create corresponding directory in dest_dir
			call copy_directory(entry_path, dest_dir/entry_name)
		else if entry is a FIFO file:
			increment num_fifo_files
worker thread:
	while not done or buffer not empty:
		remove FileInfo from shared buffer
		call copy_file(src_file, dest_file)
		print copy message

copy_file(src_file, dest_file):
	open source file
	create destination file
	while data available in source file:
		read data from source file
		write data to destination file
	close source and destination files
```

Tests

1. Test 1

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

```
Elapsed time: 2.186 seconds
Number of regular files copied: 194
Number of directories copied: 7
Number of FIFO files copied: 0
Number of total files copied: 201
Total bytes copied: 25009680
==39707==
==39707== HEAP SUMMARY:
==39707==     in use at exit: 0 bytes in 0 blocks
==39707==   total heap usage: 20 allocs, 20 frees, 265,600 bytes allocated
==39707==
==39707== All heap blocks were freed -- no leaks are possible
==39707==
==39707== For lists of detected and suppressed errors, rerun with: -s
==39707== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

2. Test 2

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

```
Elapsed time: 1.214 seconds
Number of regular files copied: 140
Number of directories copied: 2
Number of FIFO files copied: 0
Number of total files copied: 142
Total bytes copied: 24873082
```

3. Test 3

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

```
Elapsed time: 9.939 seconds
Number of regular files copied: 3116
Number of directories copied: 151
Number of FIFO files copied: 0
Number of total files copied: 3267
Total bytes copied: 73513699
```