GEBZE TECHNICAL UNIVERSITY
COMPUTER ENGINEERING FACULTY

CSE 344 SYSTEM PROGRAMMING

HOMEWORK 5 REPORT

ENES AYSU
1901042671

Program Description

This program facilitates multi-threaded file copying and directory management using POSIX threads (pthread) in C. It employs a task buffer to queue file copy tasks, with worker threads responsible for copying files concurrently. The program supports copying directories recursively while maintaining statistics on the copied files, directories, and total bytes transferred.

Key Components:

Task Buffer: A thread-safe buffer manages file copy tasks, allowing multiple worker threads to process tasks concurrently.

Worker Threads: Concurrent threads perform file copying operations by reading from the task buffer and copying files from source to destination.

Manager Thread: Coordinates directory copy operations, initiates file copy tasks, and manages cleanup operations upon completion.

Synchronization: Mutexes and condition variables ensure thread safety, while a barrier synchronizes worker threads at specific points to coordinate task processing.

Signal Handling: The program handles SIGINT signals for graceful termination, performing cleanup actions before exiting.

The program's design enables efficient parallel file copying and directory management, making it suitable for scenarios requiring concurrent file operations and resource-intensive tasks.

Function and Processes

cleanup():

This function is responsible for cleaning up dynamically allocated memory and destroying synchronization primitives.

- It frees the memory allocated for workers and buffer.tasks.
- It destroys the mutexes (buffer.mutex and stats_mutex), condition variables (buffer.cond_full and buffer.cond_empty), and the barrier (barrier).

sigint_handler(int sig):

This function is a signal handler for the SIGINT signal (interrupt signal, typically generated by pressing Ctrl+C).

- It prints a message indicating that a SIGINT signal has been received and performs cleanup actions using the cleanup() function before exiting the program.

***handle_error(const char msg):**

This function is a generic error handler that prints an error message corresponding to the given error message string and exits the program with EXIT_FAILURE.

***worker_thread_func(void arg):**

This is the main function executed by worker threads.

- It enters a loop where it locks the buffer mutex, waits for tasks in the buffer (buffer.tasks), processes the tasks, and updates statistics.
- After processing all tasks, it waits at the barrier (pthread_barrier_wait(&barrier);) before exiting the thread.

****add_task(const char src_path, const char dest_path):**

This function adds a new task (file copy operation) to the buffer.

- It locks the buffer mutex, checks if the buffer is full, adds the task to the buffer, updates the buffer's count, signals the condition variable for empty buffer (buffer.cond_empty), and unlocks the mutex.

***delete_directory_contents(const char path):**

This function recursively deletes the contents of a directory.

- It opens the directory, reads its entries, and for each entry, it recursively deletes subdirectories/files using delete_directory_contents() or removes the file/directory using unlink() or rmdir().

****copy_directory(const char src_dir, const char dest_dir):**

This function recursively copies the contents of a directory to another directory.

- It opens the source directory, reads its entries, and for each entry, it creates a corresponding entry in the destination directory using mkdir() (for directories) or adds a copy task to the buffer using add_task() (for regular files).

***manager_thread_func(void arg):**

This function is executed by the manager thread.

- It receives source and destination directory paths as arguments, deletes the contents of the destination directory using delete_directory_contents(), and then initiates the directory copy process using copy_directory().
- After completing the copy operation, it signals that the copy operation is done (buffer.done = 1) and broadcasts the condition variable for an empty buffer (buffer.cond_empty).

***usage(const char prog_name):**

This function prints a usage message indicating how to run the program and exits with EXIT_FAILURE.

***main(int argc, char argv[]):**

This is the main entry point of the program.

- It parses command-line arguments for buffer size, number of workers, source directory, and destination directory.
- It initializes synchronization primitives (mutexes, condition variables, and barrier), allocates memory for workers and buffer.tasks, sets up a SIGINT signal handler, and creates threads for the manager and worker functions.
- After the manager thread completes, it joins all threads, calculates, and prints statistics, and cleans up resources before exiting the program.

Condition Variables and Barrier Implementation

Condition Variables:

- Used for the producer-consumer problem to manage access to the shared task buffer.
- Prevents race conditions by ensuring that producers wait if the buffer is full and consumers wait if the buffer is empty.
- Synchronizes access using “pthread_cond_wait” and “pthread_cond_signal”.

Barriers:

- Used to synchronize the completion of all worker threads.
- Ensures that all threads reach a specific point (completion of their work) before any of them proceeds.
- Achieves this using “pthread_barrier_wait”.

By using these mechanisms, the code ensures that tasks are safely added to and removed from the buffer without conflicts and that all worker threads finish their tasks before the main thread calculates and displays statistics.

Execution Steps of the Program

Run the Program:

- Execute the compiled program with the required command-line arguments.

Required command-line arguments:

- <buffer_size>: Size of the task buffer (integer).
- <num_workers>: Number of worker threads (integer).
- <src_dir>: Source directory path (string).
- <dest_dir>: Destination directory path (string).

Execution Flow:

- The program initializes synchronization primitives (mutexes, condition variables, and barrier), allocates memory for tasks and worker threads, and sets up a signal handler for SIGINT.
- It creates a manager thread to coordinate directory copy operations and worker threads to perform file copy tasks.
- The manager thread recursively copies the contents of the source directory to the destination directory while managing concurrent file copy tasks using the task buffer.
- Worker threads process tasks from the task buffer, copying files from the source to destination, and updating statistics.
- Upon completion, the manager thread signals the end of copy operations, and all threads join back to the main thread for cleanup and statistics reporting.

Signal Handling:

- Press Ctrl+C (SIGINT) to send an interrupt signal to the program.
- The program's signal handler (sigint_handler) catches the SIGINT signal, performs cleanup actions, and exits gracefully.

View Statistics:

- After successful execution, the program prints statistics including the number of regular files copied, FIFO files encountered, directories copied, total bytes transferred, and the total execution time.

Cleanup:

- The program automatically performs cleanup actions during exit, freeing allocated memory and destroying synchronization primitives.

Tests

Test 1

For Compilation

- $ make
- $ valgrind ./1901042671_main 10 10 ../testdir/src/libvterm ../toCopy

Result

```
---------------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: 0.896817 seconds
==31898==
==31898== HEAP SUMMARY:
==31898==     in use at exit: 0 bytes in 0 blocks
==31898==   total heap usage: 174 allocs, 174 frees, 5,336,576 bytes allocated
==31898==
==31898== All heap blocks were freed -- no leaks are possible
==31898==
==31898== For lists of detected and suppressed errors, rerun with: -s
==31898== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Test 2

For Compilation

- $ make
- $ ./1901042671_main 10 4 ../testdir/src/libvterm/src ../toCopy

Result

```
---------------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: 0.173866 seconds
```

Test 3

For Compilation

- $ make
- $ ./1901042671_main 10 10 ../testdir ../toCopy

Result

```
---------------STATISTICS--------------------
Consumers: 10 - Buffer Size: 10
Number of Regular Files: 3117
Number of FIFO Files: 0
Number of Directories: 151
TOTAL BYTES COPIED: 73526702
TOTAL TIME: 2.604543 seconds
```

Summary

The provided C program facilitates concurrent file copying and directory management through POSIX threads, offering a robust framework for efficient parallel processing. It employs a task buffer to queue file copy tasks, managed by worker threads that copy files from source to destination while maintaining statistics. A manager thread coordinates directory copy operations and cleanup tasks, ensuring orderly execution and synchronization between threads using mutexes, condition variables, and a barrier for synchronization. Signal handling for SIGINT signals enables graceful termination, and the program delivers scalability and performance benefits for resource-intensive file operations in a concurrent environment.