CSE344-SYSTEMS PROGRAMMING

HW5 REPORT

ADEM MURAT ÖZDEMİR-200104004110

Compile the Code: “make”

Run the program : “./MWCp <buffer_size> <num_workers> <src_directory> <dest_directory>”

For Test1 : “valgrind ./MWCp 10 10 ../testdir/src/libvterm ../tocopy”

For Test2 : “./MWCp 10 4 ../testdir/src/libvterm/src ../toCopy”

For Test3 : “./MWCp 10 100 ../testdir ../toCopy”

Clean the .o files and MWCp program : “make clean”

IMPORTANT : For HW4 and HW5, I used condition variable and barrier structures in both assignments. That's why I uploaded the same source codes and report to both assignments.

Problem Overview and How to Solve

In this assignment, we are tasked with developing an enhanced version of the directory copying utility "MWCp". The objective is to implement a worker-manager model to synchronize thread activities using both condition variables and barriers. This project focuses on efficient file management and transfer using multithreading in C, adhering to POSIX and standard C libraries.

The core functionality of the program involves copying files and directories from a source directory to a destination directory. The program is designed to handle different types of files, including regular files, FIFO files, and directories. It uses a buffer to manage tasks, where each task represents a file or directory to be copied. The buffer ensures thread-safe access and synchronization among multiple threads.

Key components of the solution include:

- **Buffer Management:** A circular buffer is implemented to store file tasks. The buffer uses mutex locks and condition variables to handle synchronization between the producer (manager thread) and consumers (worker threads). The buffer supports operations to add and remove tasks, as well as track statistics such as total bytes copied and the number of different file types processed.
- **Manager Thread:** The manager thread is responsible for scanning the source directory and adding tasks to the buffer. It recursively traverses directories, adding file tasks to the buffer for worker threads to process.
- **Worker Threads:** Multiple worker threads are spawned to process tasks from the buffer. Each worker thread reads from the buffer, copies files, and updates the buffer with the

progress. The worker threads wait for tasks to become available if the buffer is empty and signal the manager thread when the buffer has space available.

- **Synchronization Mechanisms:** Condition variables are used to signal the state of the buffer (empty or full), ensuring efficient synchronization without busy waiting. Barriers are utilized to ensure that all worker threads wait at a certain point before proceeding, which helps in managing phases of processing.
- **Signal Handling:** Proper signal handling is implemented to gracefully terminate the program upon receiving a SIGINT signal (Ctrl+C). This ensures that all resources are released, and the program exits cleanly.

The program also includes robust error handling, memory management, and performance measurement to provide detailed statistics about the file copying process. The enhancements made in this version aim to improve efficiency, reliability, and maintainability of the file copying utility, leveraging multithreading and synchronization techniques.

Pseudocode :

MAIN FUNCTION:

1. Setup signal handling for SIGINT (Ctrl+C)

2. Parse command-line arguments to get buffer size, number of workers, source directory, and destination directory

3. Create destination directory if it does not exist

4. Initialize the buffer with the specified size

5. Initialize the barrier for synchronization

6. Record the start time

7. Create the manager thread with the source and destination directories and the buffer

8. Create worker threads with the buffer and the barrier

9. Wait for the manager thread to complete

10. Signal the worker threads to finish by setting the buffer done flag

11. Wait at the barrier for all threads to finish

12. Record the end time

13. Calculate and print statistics (total bytes copied, number of files and directories, total time)

14. Clean up and exit

MANAGER THREAD FUNCTION:

1. Traverse the source directory
2. For each entry in the directory:
   a. If the entry is a directory:
      i. Create the corresponding directory in the destination
      ii. Recursively process the directory
   b. If the entry is a regular file or FIFO:
      i. Create a file task
      ii. Add the file task to the buffer
      iii. Update the buffer statistics
3. Set the buffer done flag to indicate that all tasks have been added
4. Exit the thread

WORKER THREAD FUNCTION:

1. Loop until there are no more tasks to process or a stop signal is received:
   a. Remove a task from the buffer
   b. If no task is available and the buffer is done, exit the loop
   c. Open the source file for reading
   d. Open the destination file for writing
   e. Copy data from the source file to the destination file
   f. Update the buffer statistics with the number of bytes copied
   g. Close the source and destination files
2. Wait at the barrier for all threads to finish
3. Exit the thread

BUFFER FUNCTIONS:

- Initialize the buffer and allocate memory for tasks
- Destroy the buffer and release allocated resources
- Add a task to the buffer (wait if the buffer is full)
- Remove a task from the buffer (wait if the buffer is empty and not done)
- Set the buffer done flag
- Add to the total bytes copied

- Retrieve the total bytes copied

- Increment and retrieve the number of regular files

- Increment and retrieve the number of FIFO files

- Increment and retrieve the number of directories

Some Explanations

In this program, condition variables and barriers are used to ensure proper synchronization and coordination among threads. Condition variables prevent busy waiting by allowing threads to wait for specific conditions to be met. The producer thread (manager) waits when the buffer is full (buffer->not_full), and the consumer threads (workers) wait when the buffer is empty (buffer->not_empty). This efficient waiting mechanism ensures that CPU resources are not wasted.

Barriers are used to synchronize all threads at a specific point, ensuring they all reach a particular stage before proceeding. In this program, after completing their tasks, the manager and worker threads wait at a barrier (pthread_barrier_wait). This ensures that all threads finish their work before the program continues, enabling coordinated execution and proper synchronization.

In this program, the O_TRUNC flag is used in the open system call to ensure that the destination file is truncated to zero length if it already exists. This is crucial because it guarantees that any existing data in the file is erased before writing new data. By truncating the file, we prevent leftover data from previous contents, ensuring a clean and accurate copy of the source file. This maintains the integrity and consistency of the file copy operation, making sure the destination file matches the source file exactly.

In this program, worker threads terminate under two specific conditions:

1. **Buffer Exhaustion and Done Signal**: Worker threads continuously attempt to remove tasks from the buffer for processing. They use a condition variable (not_empty) to wait if the buffer is empty. When the buffer becomes non-empty (i.e., when the manager thread adds tasks to it), the worker threads are signaled and proceed to remove tasks. The manager thread, after finishing its task of adding all the directory contents into the buffer, sets the done flag to indicate that no more tasks will be added to the buffer (buffer_set_done). This flag, combined with an empty buffer, signals the worker threads to stop waiting for new tasks. Specifically, when a worker thread finds the buffer empty and the done flag set, it exits the task removal loop and proceeds to terminate.
2. **SIGINT (Ctrl+C) Signal Handling**: The program is equipped with signal handling for the SIGINT signal, typically generated by pressing Ctrl+C. When the SIGINT signal is received, the signal handler sets a global stop flag to indicate that the program should terminate. Worker threads, while waiting to remove tasks from the buffer or while processing tasks, periodically check the stop flag. If the stop flag is set, worker threads break out of their processing loop and proceed to terminate. This ensures a graceful shutdown of the program in response to user interruption, ensuring that all threads can exit cleanly and resources can be released properly.

In summary, worker threads terminate when they detect that the buffer is empty and no more tasks will be added (indicated by the done flag), or when a SIGINT signal is received and processed, setting the stop flag to initiate a graceful shutdown.

Test Results

Test 1,2,3

```
ademmurat@ademmurat-GL553VD:~/Desktop/system programming2024/hw5/yeni2/hw4-hw5_test/put_your_codes_here$ valgrind ./MWCp 10 10 ../testdi
r/src/libvterm ../tocopy
==5535== Memcheck, a memory error detector
==5535== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==5535== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==5535== Command: ./MWCp 10 10 ../testdir/src/libvterm ../tocopy
==5535==

--------------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: 00:01.-420 (min:sec.milli) - Elapsed: 0.580 seconds
==5535==
==5535== HEAP SUMMARY:
==5535==     in use at exit: 0 bytes in 0 blocks
==5535==   total heap usage: 28 allocs, 28 frees, 352,566 bytes allocated
==5535==
==5535== All heap blocks were freed -- no leaks are possible
==5535==
==5535== For lists of detected and suppressed errors, rerun with: -s
==5535== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
ademmurat@ademmurat-GL553VD:~/Desktop/system programming2024/hw5/yeni2/hw4-hw5_test/put_your_codes_here$ ./MWCp 10 4 ../testdir/src/libv
term/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: 00:00.022 (min:sec.milli) - Elapsed: 0.022 seconds
ademmurat@ademmurat-GL553VD:~/Desktop/system programming2024/hw5/yeni2/hw4-hw5_test/put_your_codes_here$ ./MWCp 10 100 ../testdir ../toC
opy

--------------STATISTICS-------------------
Consumers: 100 - Buffer Size: 10
Number of Regular Files: 3116
Number of FIFO Files: 0
Number of Directories: 151
TOTAL BYTES COPIED: 73520554
TOTAL TIME: 00:00.091 (min:sec.milli) - Elapsed: 0.091 seconds
ademmurat@ademmurat-GL553VD:~/Desktop/system programming2024/hw5/yeni2/hw4-hw5_test/put_your_codes_here$
```

If there is no folder with the given name in the path given for the destination, a folder with the given name is created.

If source path is not valid , output will be like this :

```
ademmurat@ademmurat-GL553VD:~/Desktop/system programming2024/hw5/yeni2/hw4-hw5_test/put_your_codes_here$ ./MWCp 10 100 ../randomdir ../t
oCopy
opendir: No such file or directory

--------------STATISTICS-------------------
Consumers: 100 - Buffer Size: 10
Number of Regular Files: 0
Number of FIFO Files: 0
Number of Directories: 0
TOTAL BYTES COPIED: 0
TOTAL TIME: 00:00.007 (min:sec.milli) - Elapsed: 0.008 seconds
```

If we use Ctrl+C while the program is running, the currently running worker thread will be waited for its work to finish and the program will be terminated. The number of bytes copied will be suppressed until Ctrl+C is performed.