CSE 344 System Programming HW5 REPORT

Student Name: AYKUT SERT

Student ID: 200104004104

Makefile commands:

make compile => creates MWCp.out

make clean => removes out file

1-Barrier Declaration

pthread_barrier_t worker_barrier => This global variable is used to define the barrier which will synchronize the threads.

2. Barrier Initialization

pthread_barrier_init (&worker_barrier, NULL, num_workers + 1) => The barrier is initialized to synchronize num_workers worker threads plus one manager thread.

3. Barrier Wait in Manager Function

pthread_barrier_wait(&worker_barrier) => The manager thread waits at the barrier until all worker threads have completed their tasks.

4. Barrier Wait in Worker Function

pthread_barrier_wait(&worker_barrier) => Each worker thread waits at the barrier, ensuring they all synchronize before finishing their execution.

5. Barrier Destruction

pthread_barrier_destroy(&worker_barrier) => The barrier is destroyed to clean up resources.

Impact of the Changes

- **Synchronization:** Ensures all threads (both worker and manager) complete their tasks before the program proceeds to cleanup and prints the statistics.
- **Stability:** Improves the program's stability by preventing premature termination of any thread, ensuring all file operations are completed.
- **Coordination:** Enhances coordination among threads, making the program more reliable and efficient.

These changes make the program more robust and help to avoid issues related to race conditions and incomplete operations, providing a clear and coordinated end to the multithreaded file copying process.

Places where Barrier is added:

```c
// Buffer structure
typedef struct {
    FileInformation *data;
    int size;
    int count;
    int head;
    int tail;
} Buffer;

Buffer buffer;
pthread_mutex_t buffer_mutex;
pthread_cond_t buffer_not_full;
pthread_cond_t buffer_not_empty;
pthread_barrier_t worker_barrier; // Barrier declaration
int done = 0;
int num_files_copied = 0;
int num_directories = 0;
int num_fifo_files = 0;
long long total_bytes_copied = 0;
struct timeval start_time, end_time;
int num_workers;

pthread_t *worker_threads;
```

```c
int main(int argc, char *argv[]) {
    if (argc != 5) {
        print_usage_and_exit(); // Print usage and exit if argumen
    }

    int buffer_size = atoi(argv[1]); // Convert buffer size to int
    num_workers = atoi(argv[2]); // Convert number of workers to i
    char *src_dir = argv[3]; // Source directory
    char *dst_dir = argv[4]; // Destination directory

    // Initialize buffer, mutex, condition variables, and barrier
    initialize_buffer(buffer_size);
    pthread_mutex_init(&buffer_mutex, NULL);
    pthread_cond_init(&buffer_not_full, NULL);
    pthread_cond_init(&buffer_not_empty, NULL);
    pthread_barrier_init(&worker_barrier, NULL, num_workers + 1);
```

A barrier variable has been created globally. It was initiated in Main. Manager worker threads will wait. Each worker thread waits at the barrier, ensuring they all synchronize before finishing their execution.

```c
void *manager_function(void *arg) {
    char **src_dst_dirs = (char **)arg;
    char *src_dir = src_dst_dirs[0];
    char *dst_dir = src_dst_dirs[1];

    traverse_directory(src_dir, dst_dir);

    pthread_mutex_lock(&buffer_mutex);
    done = 1;
    pthread_cond_broadcast(&buffer_not_empty);
    pthread_mutex_unlock(&buffer_mutex);

    pthread_barrier_wait(&worker_barrier); // Wa

    return NULL;
}
```

```c
void *worker_function(void *arg) {
    while (1) {
        pthread_mutex_lock(&buffer_mutex);
        while (buffer_is_empty() && !done) {
            pthread_cond_wait(&buffer_not_empty, &buffer_mutex);
        }

        if (buffer_is_empty() && done) {
            pthread_mutex_unlock(&buffer_mutex);
            break;
        }

        FileInformation fd_pair = buffer_get();
        pthread_cond_signal(&buffer_not_full);
        pthread_mutex_unlock(&buffer_mutex);

        copy_file_contents(fd_pair.src_fd, fd_pair.dst_fd);
        close(fd_pair.src_fd);
        close(fd_pair.dst_fd);

        // Update counters with mutex lock
        pthread_mutex_lock(&buffer_mutex);
        num_files_copied++;
        pthread_mutex_unlock(&buffer_mutex);
    }

    pthread_barrier_wait(&worker_barrier); // Wait at the barrier

    return NULL;
}
```

General Structure and Functioning

Our program uses multithreading to copy files from the specified source directory to the target directory. The program creates a certain number of worker threads and coordinates file copy operations using a buffer. The program also terminates safely by capturing the Ctrl+C signal during the copying process and notifies the user of the intermediate results. Additionally, barriers are used to synchronize threads, ensuring that all worker threads and the manager thread complete their tasks before proceeding to the cleanup and statistics phase.

Pseudocode

Main Function

1 -> Check the arguments and make sure they are valid.

2 -> Get buffer size and number of worker threads.

3 -> Initialize buffer and synchronization structures.

4 -> Initialize the barrier for synchronizing threads.

5 -> Set the signal handler to capture the Ctrl+C signal.

6 -> Save start time.

7 -> Create the thread that will start the manager.

8 -> Create worker threads.

9 -> Wait for the manager to complete.

10 -> Wait for worker threads to complete.

11 -> Save end time.

12 -> Perform cleanups and print statistics.

Manager Function

1 -> Get source and target directories.

2 -> Scan the source directory and update the buffer for each file.

3 -> Set the done flag and wake up worker threads.

4 -> Wait at the barrier for all worker threads to complete.

Worker Function

While:

1 -> Check if the buffer is empty.

2 -> Exit if the buffer is empty and the done flag is set.

3 -> Get file information from buffer.

4 -> Copy the files.

5 -> Update the number of files and bytes copied.

6 -> Wait at the barrier for all threads to synchronize before finishing.

Buffer Functions

initialize_buffer(size): Start buffer.

buffer_is_empty(): Check if the buffer is empty.

buffer_is_full(): Check if the buffer is full.

buffer_add(src_fd, dst_fd, src_name, dst_name): Add file information to buffer.

buffer_get(): Get file information from buffer.

Directory Traversal Function

1 -> Open the source directory.

2 -> Create the target directory.

Scan directory contents:

1 -> If it's a subdirectory, process it recursively.

2 -> If it is a file, add the file information to the buffer.

3 -> If FIFO file, update counters.

Copy Function

1 -> Copy data from source file to target file using buffer.

2 -> Update the number of bytes copied.

Signal Handler

When Ctrl+C signal is received:

1 -> Set the done flag.

2 -> Wake up all threads.

3 -> Wait for worker threads to complete.

4 -> Perform cleanups and print statistics.

5 -> End the program.

Error Handling

handle_error(message): Print the error message and terminate the program.

Some Test Outputs:

Test1: valgrind ./MWCp 10 10 ../testdir/src/libvterm ../tocopy, #memory leak checking, buffer size=10, number of workers=10, sourceFile, destinationFile

```
==13101== Memcheck, a memory error detector
==13101== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==13101== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==13101== Command: ./MWCp 10 10 ../testdir/src/libvterm ../tocopy/
==13101==
---------------STATISTICS--------------------
Consumers: 10 - Buffer Size: 10
Number of Regular Files: 194
Number of Directories: 7
Number of FIFO Files: 0
TOTAL BYTES COPIED: 25009680
TOTAL TIME: 0.606 seconds
==13101==
==13101== HEAP SUMMARY:
==13101==     in use at exit: 0 bytes in 0 blocks
==13101==   total heap usage: 22 allocs, 22 frees, 348,624 bytes allocated
==13101==
==13101== All heap blocks were freed -- no leaks are possible
==13101==
==13101== For lists of detected and suppressed errors, rerun with: -s
==13101== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Test2: ./MWCp 10 4 ../testdir/src/libvterm/src ../toCopy #buffer size=10, number of workers=4, sourceFile, destinationFile

Test3: ./MWCp 10 10 ../testdir ../toCopy #buffer size=10, number of workers=10, sourceFile, destinationFile