CSE 344
SYSTEM PROGRAMMING

HW5
REPORT

OGUZ MUTLU
1801042624

1. PROBLEM DEFINITION

The task is to develop a directory copying utility called "pCp" that efficiently copies files and sub-directories from a source directory to a destination directory. The utility should utilize parallel processing by creating multiple worker threads to perform the copying task concurrently. This approach is necessary to handle large directory trees without exhausting system resources. Main program should take the buffer size, number of consumers and the source and destination directories as command-line arguments.

1.Copy files and subdirectories
2.Thread Pool Creation
3.Syncronization and buffering
4.Efficiency
These are the main objextives of our system.

2. PROBLEM SOLUTION APPROACH

For this assignment, it is primarily aimed to use the producer consumer structure for more than one thread. In order to serve this purpose, a single producer has been created using more than one consumer thread.

First of all, I defined a producer thread, the function of this thread is to copy the created buffer to the folder or file to be copied to the destination without content information, and to keep the file descriptors that are opened after the copying process is in the struct buffer. Mutex and conditional variables are used for necessary synchronization.

Producer

```
for(;;)
   lock(m)
   while(count == N)
      cwait(empty,m)
   produce_item()
   count++
   broadcast(full)
   unlock(m)
```

Consumer

```
   for(;;)
      lock(m)
      while(count == 0)
         cwait(full, m)
      consume_item
      count--
      broadcast(empty)
      unlock(m)
```

If there are other folders or files in the folder (if there is a folder hierarchy), the code works recursively. First of all, if the case that it is a file /folder is found, the same function with the given path is called again and the folder specified as target is created. It is taken into account that they are created in the same hierarchy.

```c
void directorySearch(const char *source, char *destination)
{
    DIR *dir = opendir(source);

    if (dir)
    {
        char path[512] = "\0", *endptr = path;
        struct dirent *entry;
        strcpy(path, source);
        strcat(path, "/");

        endptr += strlen(source);
        int source_len = strlen(path);

        while ((entry = readdir(dir)) != NULL)
        {
            if (strcmp(entry->d_name, ".") == 0 || strcmp(entry->d_name, "..") ==
            {
                continue;
            }
            else
            {
                char new_temp[512] = "\0";
                strncpy(new_temp, path, source_len);
                fflush(stdout);
                memset(path + source_len, 0, sizeof(path) - source_len);
                strcpy(path, new_temp);
                strncpy(new_temp, path, source_len);
                struct stat info;
                strcat(endptr, entry->d_name);
                if (!stat(path, &info))
                {
                    if (S_ISDIR(info.st_mode))
                    {
                        char new_target[512] = "\0";
                        strcat(new_target, destination);
                        strcat(new_target, "/");
                        strcat(new_target, entry->d_name);

                        if (mkdir(new_target, info.st_mode) == -1 && errno != EEXIST)
                        {
                            perror("mkdir");
                            return;
                        }
                        num_dirs_copied++;
                        directorySearch(path, new_target);
                        strcpy(path, new_temp);
                        strcat(path, "/");
                    }
                    else if (S_ISREG(info.st_mode))
                    {
                        copyFile(path, destination);
                        num_files_copied++;
                        num_bytes_copied += info.st_size;
                    }else if (S_ISFIFO(info.st_mode)) {
                        char new_target[512] = "\0";
                        strcat(new_target, destination);
```

If the file, not the folder, is detected, this time only a file will be created in the same hierarchy, and the content will not be copied in the producer. This created file is opened first and then the source file is opened. The descriptors and names of the opened files are written to the buffer, mutex is used at the beginning of this writing process, and unlocked when finished.

The working logic of mutex and conditional varibale is given below

1.The thread locks the mutex in preparation for checking the state of the shared variable.
2.The state of the shared variable is checked.
3.If the shared variable is not in the desired state, then the thread must unlock the mutex (so that other threads can access the shared variable) before it goes to sleep on the condition variable.
4.When the thread is reawakened because the condition variable has been signaled, the mutex must once more be locked, since, typically, the thread then immediately accesses the shared variable.

```c
void *producer(void *arg)
{
    char filenames[2][512] = {{"\0"}};
    struct filenames_buffer *tempStruct = (struct filenames_buffer *)arg;
    strcpy(filenames[0], tempStruct->filenames[0]);
    strcpy(filenames[1], tempStruct->filenames[1]);

    directorySearch(filenames[0], filenames[1]);
    done = 1;
    pthread_exit(NULL);
}
```

When consumer threads are created, the necessary syscall errors in the files are checked first. Then, again using the necessary thread synchronization, read and write are performed in a single thread. If there is more than one thread, the other thread values will continue to process until the buffer is full or the done flag works.

```c
void *consumer(void *arg)
{
    while(1)
    {
        int s = pthread_mutex_lock(&mtx);
        if (s != 0)
        {
            perror("pthread_mutex_lock");
            return (void*)1;
        }

        while (bufferCounter == 0 && !done)
        {
            pthread_cond_wait(&cond_consume, &mtx);
        }

        if((bufferCounter == 0 || done) && signalint != 1){
            pthread_mutex_unlock(&mtx);
            break;
        }

        bufferCounter--;
        consume();
        sleep(1);

        buffer_index = (buffer_index + 1)%buffer_size;
        // printf("file %s, copied\n", buffer[bufferCounter+1].source_file

        s = pthread_cond_broadcast(&cond_produce);
        if (s != 0)
        {
```

When the main program starts, threads are created after taking the necessary arguments, the created buffer is created as dynamic.

The program is terminated by printing the counter values kept at the end of the program.

Test Cases

2 size of buffer

```
oguz@oguz-ubuntu:~/Desktop/System Programming/System-Programming-/hw5$ ./cp 2 6
source target
Copied: source/a.txt, total bytes copies : 19 bytes)
Copied: source/subfolder3/subsubfolder/subfile.txt, total bytes copies : 26 byte
s)
Copied: source/subfolder3/xx.txt, total bytes copies : 44 bytes)
Copied: source/subfolder/b.txt, total bytes copies : 28 bytes)
Total time : 3.001630
Total files copied: 4
Total directories copied: 4
Total fifo copied: 0
Total bytes copied: 117
Total time taken: 3.001630 seconds
oguz@oguz-ubuntu:~/Desktop/System Programming/System-Programming-/hw5$
```

1 size of buffer

```
oguz@oguz-ubuntu:~/Desktop/System Programming/System-Programming-/hw5$ ./cp 1 6
source target
Copied: source/a.txt, total bytes copies : 19 bytes)
Copied: source/subfolder3/subsubfolder/subfile.txt, total bytes copies : 26 byte
s)
Copied: source/subfolder3/xx.txt, total bytes copies : 44 bytes)
Copied: source/subfolder/b.txt, total bytes copies : 28 bytes)
Total time : 3.002740
Total files copied: 4
Total directories copied: 4
Total fifo copied: 0
Total bytes copied: 117
Total time taken: 3.002740 seconds
oguz@oguz-ubuntu:~/Desktop/System Programming/System-Programming-/hw5$
```

6size of buffering

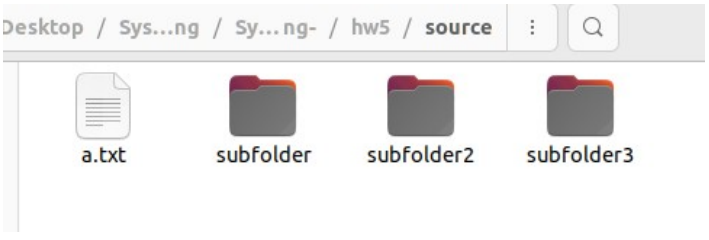
```
oguz@oguz-ubuntu:~/Desktop/System Programming/System-Programming-/hw5$ ./cp 6 6
source target
Copied: source/a.txt, total bytes copies : 19 bytes)
Copied: source/subfolder3/subsubfolder/subfile.txt, total bytes copies : 26 byte
s)
Copied: source/subfolder3/xx.txt, total bytes copies : 44 bytes)
Copied: source/subfolder/b.txt, total bytes copies : 28 bytes)
Total time : 3.003004
Total files copied: 4
Total directories copied: 4
Total fifo copied: 0
Total bytes copied: 117
Total time taken: 3.003004 seconds
oguz@oguz-ubuntu:~/Desktop/System Programming/System-Programming-/hw5$
```

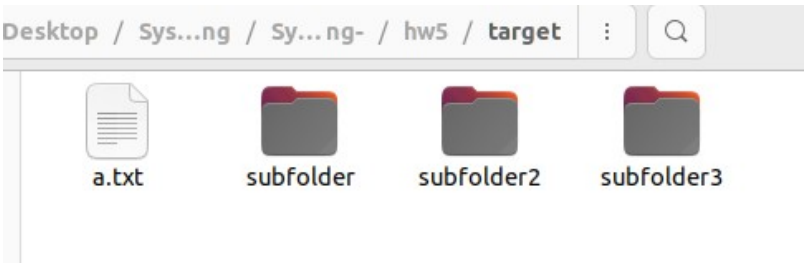
2 Thread

```
oguz@oguz-ubuntu:~/Desktop/System Programming/System-Programming-/hw5$ ./cp 6 2
source target
Copied: source/a.txt, total bytes copies : 19 bytes)
Copied: source/subfolder3/subsubfolder/subfile.txt, total bytes copies : 26 byte
s)
Copied: source/subfolder3/xx.txt, total bytes copies : 44 bytes)
Copied: source/subfolder/b.txt, total bytes copies : 28 bytes)
Total time : 3.001809
Total files copied: 4
Total directories copied: 4
Total fifo copied: 0
Total bytes copied: 117
Total time taken: 3.001809 seconds
oguz@oguz-ubuntu:~/Desktop/System Programming/System-Programming-/hw5$
```

6 Thread

```
oguz@oguz-ubuntu:~/Desktop/System Programming/System-Programming-/hw5$ ./cp 6 6
source target
Copied: source/a.txt, total bytes copies : 19 bytes)
Copied: source/subfolder3/subsubfolder/subfile.txt, total bytes copies : 26 byte
s)
Copied: source/subfolder3/xx.txt, total bytes copies : 44 bytes)
Copied: source/subfolder/b.txt, total bytes copies : 28 bytes)
Total time : 3.003004
Total files copied: 4
Total directories copied: 4
Total fifo copied: 0
Total bytes copied: 117
Total time taken: 3.003004 seconds
oguz@oguz-ubuntu:~/Desktop/System Programming/System-Programming-/hw5$
```





The program can copy .pdf, .pptx, .png files.

3. NOTES

While reading the files, I used dynamic memory allocation explicitly in only one place, then valgrind results in a possible leak of 1632 bytes, even though I used free. I could not solve the problem in any way, 1632 memory leak can be counted as missing.

Best regards