Gebze Tecnhical University

CSE344 System Programming

-

Homework #4 Report

Abdulsamet Aslan

200104004098

**Problem overview**

The problem addressed in this project is the efficient copying of files and sub-directories within a large directory tree using a utility called "pCp." Existing utilities may struggle to handle the resource requirements when working with such large directory trees. To overcome this limitation, the implementation of pCp aims to achieve parallel processing by creating a new thread for each file and sub-directory copy operation.

**Problem Solving Approach**

*Declaration of the Buffer*

I used a queue implementation to create a buffer that stores the file description and filenames.

*Creating Threads*

After taking commands from user, a producer thread and consumer threads which are taken from command line are created. While producer thread takes directory names as parameter, consumer threads doesn’t take any parameter.

*Mutexes and Condition Variables*

I created two different mutexes. First one is created to establish the write-read synchronizations. Second one is used to protect the critical section which the producer and the multiple consumers are writing to the standard output.

*Producer Thread*

Firstly, It checks whether destination file exists. If it exists, creates a directory that shares same name with source directory. Then producer thread calls a recursive function to copy subdirectories. It finds the files and directories in the subdirectory. If there is a directory, it calls itself recursively. Otherwise, it opens the source file to read and destination file to write. Finally, it locks the required mutex and push the file descriptors and filename to queue. Then it unlocks the mutex and signal to ‘full’ condition variable to inform consumer threads.

*Consumer Threads*

Consumer Threads locks the mutex and pop an element from the buffer, signals to ‘empty’ condition variable to inform producer thread and unlocks the mutex. After that It reads from source file descriptor and writes to destination file descriptor as chunks. If a signal occurs when this process is executing, the consumer threads finish their work with the last chunk at that moment and breaks the loop to terminate.

*Handle signals*

*Terminating Threads*

End of the main loop, program waits for the thread to terminate using *phtread\_join.* Furthermore, it destroys all threads and condition variables.

Statistics