CSE 344 – System Programming

Homework #5

Due Date: 3 June 2023, Saturday, 23.59

This is a simplified version of the homework. This is not the original homework file. Please refer to the original homework, in any misunderstandings.

No responsibility is taken in what this PDF says.

Here's a simplified breakdown of what you need to do:

1. Understand the problem:
    - You're asked to create a multithreaded file copying utility in C that performs the copying tasks in parallel.

2. Producer Thread:
    - Initialize the producer thread which takes two directory pathnames as parameters.
    - Traverse the first directory (source directory) and for each file, perform the following:
        - Open the file for reading.
        - Create or truncate a file of the same name in the second directory (destination directory) for writing.
        - Handle any errors that occur while opening files, close the files and send an error message to standard output.
    - Create a buffer to pass the open file descriptors and file names.
        - Manage this buffer properly, handling whether it's empty or full and controlling access to it.
    - Set a 'done' flag when the buffer has been filled with all file names from the source directory and then terminate the producer thread.

3. Consumer Threads:
    - Each consumer thread should:
        - Read an item from the buffer.
        - Copy the content from the source file to the destination file.
        - Close the files.
        - Write a message to standard output with the file name and the completion status of the operation.
    - These threads should be aware of the critical section where both the producer and consumer threads are writing to standard output, and this section should be protected.
    - Each thread should terminate if the 'done' flag has been set by the producer and there are no more items in the buffer.

4. Main Program:
    - Accept four command-line arguments: buffer size, number of consumer threads, source directory, and destination directory.
    - Start the producer and consumer threads.
    - Wait for the threads to complete using pthread_join.
    - Measure and display the total time it takes to copy the files.
    - Ensure that both regular files and FIFOs are copied and do this recursively for subdirectories.
    - Keep statistics about the number and types of files copied and the total number of bytes copied.

5. Additional Handling and Reporting:
    - Experiment with different buffer sizes and numbers of consumer threads and write a report on your findings.
    - Handle scenarios where the number of open file descriptors exceeds the per-process limit, and check if this produces an error.
    - Check your program for memory leaks.
    - Handle signals properly.
    - Based on your observations, comment on the combinations of buffer size/number of consumer threads that produce the best results.

In the next page, a more detailed version of the project is being told.

Here's a more detailed breakdown of the project needs and requirements:

1. Directory Copying Utility:
   - Implement a directory copying utility called "pCp" using the C language.
   - The utility should allow the user to copy files and sub-directories from a source directory to a destination directory.

2. Thread-Based Parallel Processing:
   - Create a new thread for each file and sub-directory to perform the copying task in parallel.
   - Each thread will handle the copying of a single file or sub-directory.

3. Worker Thread Pool:
   - Implement a worker thread pool (fixed number) to regulate the number of active threads at any time.
   - The number of threads in the pool should be configurable based on a command-line argument.
   - The worker threads will handle the copying workload.

4. Producer-Consumer Synchronization:
   - Use a producer-consumer synchronization mechanism to coordinate the work between the main thread (producer) and the worker threads (consumers).
   - The producer will produce items (file names) and put them in a buffer, while the consumer threads will consume these items and perform the copying operation.

5. Producer Function:
   - Create a producer thread function that takes an array of at least two directory paths as a parameter.
   - Iterate over the files in the source directory and its sub-directories recursively.
   - For each file, open it for reading and create a corresponding file in the destination directory for writing.
   - If a file with the same name already exists in the destination directory, open and truncate it.
   - If there is an error in opening either file, close both files and output an informative message.
   - Pass the open file descriptors and file names into the buffer for the consumer threads to process.
   - Properly manage the buffer to handle its empty or full state, allowing graceful termination of threads.

6. Consumer Function:
    - Create multiple consumer threads that will consume items (file names) from the buffer.
    - Each consumer thread will copy the file from the source file descriptor to the destination file descriptor.
    - After copying, close the files and write a message to the standard output indicating the file name and the completion status.
    - Implement proper synchronization mechanisms to protect the critical section of writing to the standard output.

7. Main Program:
    - Accept command-line arguments for buffer size, number of consumer threads, source directory, and destination directory.
    - Create the necessary threads (one producer thread and multiple consumer threads) based on the provided arguments.
    - Use pthread_join to wait for all threads to complete before the program exits.
    - Measure the time taken for the copying process using gettimeofday or a similar time tracking mechanism.
    - Display the total time taken to copy all files in the directory.
    - Ensure that your utility can copy both regular files and FIFOs (named pipes).
    - Implement recursive copying to handle sub-directories, ensuring that all files and sub-directories are copied.
    - Keep track of statistics such as the number and types of files copied, and the total number of bytes copied.
    - Experiment with different buffer sizes and numbers of consumer threads to find the optimal performance for copying large directories.
    - Write a report on your results, including observations and comments on the combinations of buffer size and the number of consumer threads that produce the best performance.
    - Test the program for handling errors when exceeding the per-process limit on the number of open file descriptors.
    - Handle signals properly to allow for graceful termination or interruption of the program.
    - Conduct memory leak checks to ensure your program is free from memory leaks.

Bir sonraki sayfada aynı açıklama Türkçe olarak yapılmaktadır.

Aşağıda, proje ihtiyaçlarının ve gereksinimlerinin daha ayrıntılı bir dökümü verilmiştir:

1. Dizin Kopyalama Aracı:
   - C dilini kullanarak "pCp" adında bir dizin kopyalama aracı uygulayın.
   - Bu araç, kullanıcının kaynak dizinden hedef dizine dosyaları ve alt dizinleri kopyalamasına izin vermeli.

2. İş Parçacığı (Thread) Tabanlı Paralel İşleme:
   - Kopyalama görevini paralel olarak gerçekleştirmek için her dosya ve alt dizin için yeni bir iş parçacığı oluşturun.
   - Her iş parçacığı, bir dosya veya alt dizinin kopyalanmasını yönetecektir.

3. İş Parçacığı Havuzu (Thread pool):
   - Belirli bir sayıda iş parçacığından oluşan bir iş parçacığı havuzu uygulayarak etkin iş parçacığı sayısını düzenleyin.
   - Havuzdaki iş parçacığı sayısı, komut satırı argümanına dayalı olarak yapılandırılabilir olmalıdır.
   - İş parçacıkları, kopyalama iş yükünü yönetecektir.

4. Üretici-Tüketici Senkronizasyonu:
   - Ana iş parçacığı (üretici) ile iş parçacığı havuzundaki iş parçacıkları (tüketiciler) arasındaki çalışmayı koordine etmek için üretici-tüketici senkronizasyon mekanizmasını kullanın.
   - Üretici, öğeleri (dosya adları) üretecek ve bir tampona koyacak, tüketici iş parçacıkları ise bu öğeleri tüketip kopyalama işlemini gerçekleştirecektir.

5. Üretici Fonksiyonu:
   - En az iki dizin yolunu parametre olarak alan bir üretici iş parçacığı fonksiyonu oluşturun.
   - Kaynak dizinde ve alt dizinlerinde özyinelemeli olarak dosyalara dolaşın.
   - Her dosya için, okuma için açın ve yazma için hedef dizinde aynı ada sahip bir dosya oluşturun.
   - Hedef dizinde aynı ada sahip bir dosya zaten varsa, onu açıp içeriğini silin.
   - Herhangi bir dosyayı açarken bir hata oluşursa, her iki dosyayı da kapatın ve bilgilendirici bir mesaj çıktısı verin.
   - Açık dosya tanımlayıcılarını ve dosya adlarını tüketici iş parçacıklarının işleyeceği tampona geçirin.
   - Tamponu (buffer) doğru bir şekilde yönetin, doluluk durumunu ve boşalma durumunu kontrol ederek iş parçacıklarının düzgün biçimde sonlandırılmasını sağlayın.

6. Tüketici Fonksiyonu:
    - Tampondan (buffer) öğeleri tüketen birden fazla tüketici iş parçacığı oluşturun.
    - Her tüketici iş parçacığı, kaynak dosya tanımlayıcısından hedef dosya tanımlayıcısına dosyayı kopyalayacaktır.
    - Kopyalama işleminden sonra dosyaları kapatın ve dosya adını ve tamamlanma durumunu belirten bir mesajı standart çıktıya yazın.
    - Standart çıktıya yazma bölümünü korumak için senkronizasyon mekanizmalarını uygulayın.

7. Ana Program:
    - Tampon boyutu, tüketici iş parçacığı sayısı, kaynak dizini ve hedef dizini için komut satırı argümanlarını kabul edin.
    - Verilen argümanlara göre gerekli iş parçacıklarını (bir üretici iş parçacığı ve birden fazla tüketici iş parçacığı) oluşturun.
    - Programın sonlandığında tüm iş parçacıklarının görevlerini tamamlamasını sağlamak için pthread_join kullanın.
    - Kopyalama işlemi için geçen süreyi gettimeofday veya benzer bir zaman takip mekanizmasıyla ölçün.
    - Dizindeki tüm dosyaların kopyalanması için geçen toplam süreyi görüntüleyin.
    - Araç, normal dosyaları ve FIFO'ları (named pipe) kopyalayabilecek şekilde tasarlanmalıdır.
    - Alt dizinleri kopyalamak için özyinelemeli kopyalama gerçekleştirin ve tüm dosya ve alt dizinlerin kopyalandığından emin olun.
    - Kopyalanan dosya sayısı, dosya türleri ve toplam kopyalanan bayt sayısı gibi istatistikleri takip edin.
    - Büyük dizinleri kopyalamak için farklı tampon boyutları ve tüketici iş parçacığı sayılarıyla deneyler yaparak en iyi performansı bulun.
    - En iyi performansı sağlayan tampon boyutu ve tüketici iş parçacığı sayısı kombinasyonlarıyla ilgili gözlemler ve yorumlar içeren bir rapor yazın.
    - Açık dosya tanımlayıcılarının işlem sınırını aşması durumunda ortaya çıkan hataları işlemek için programı test edin.
    - Programın zarif bir şekilde sonlandırılması veya kesintiye uğraması için sinyalleri uygun şekilde işleyin.
    - Programınızın bellek sızıntılarından arındığından emin olmak için bellek sızıntısı kontrolleri yapın.