Protokoll von 5. Praktikum

Betriebsystem - Prof. Dr. Ronald Moore

Praktikant/-in: Huong Ly Nguyen MatrikelNr: 772829
Praktikant/-in: Saad Bourbouh MatrikelNr:1115826

1. Unfaire Lösung:

```
…
#include <semaphore>

// The worker thread
std::counting_semaphore db{1};
std::counting_semaphore mutex{1};
int rc = 0;

void readerwriter( int workerID, double percentReader, int numSeconds ) {
   …
      if ( reader ) {
         mutex.acquire();
         rc++;
         if (rc == 1)
            db.acquire();
         mutex.release();
         result = theDatabase.read( workerID );
         mutex.acquire();
         rc--;
         if(rc == 0)
            db.release();
         mutex.release();
         ++reads;
      } else // if writer
      {
         db.acquire();
         result = theDatabase.write( workerID );
         db.release();
         ++writes;
      }; // end if writer
      ++tests;

      // NON-CRITICAL AREAD
      // Sleep a while...
      …

   } // repeat until time used is up
```

```
…
} // end worker function

// inspiriert von
// https://en.wikipedia.org/wiki/Readers%E2%80%93writers_problem
```

Wir haben mit unterschiedlichen Werten für den Parameter „percentage Reader“ getestet (alle mit einer Zeit von 5 Minuten und mit 100 Threads).

Percentage Reader	Leseoperationen	Schreiboperationen
10	72	732
50	656	680
90	5079	563

Unter welchen Bedingungen wird z.B. die Benachteiligung der Schreiber deutlich?

Anhand der oben genannten Statistiken lässt sich die Benachteiligung der Schreiber am deutlichsten beim starken Leserpräsenz erkennen. Wenn es viele Leser gibt, die häufig auf die gemeinsam genutzten Ressourcen zugreifen möchten, können die Schreiber aufgrund der hohen Nachfrage der Leser Schwierigkeiten haben, Zugriff auf die Ressourcen zu erhalten. Dies kann zu längeren Wartezeiten für die Schreiber führen.

In einer unfair Lösung kann die Implementierung so gestaltet sein, dass den Lesern bevorzugt Zugriff auf die Ressourcen gewährt wird. Infolgedessen können die Schreiber länger warten müssen, bevor sie auf die Ressourcen zugreifen dürfen.

Um dieses Problem zu lösen, wurde eine fairere Lösung vorgeschlagen.

2. Faire Lösung:

```
#include <semaphore>

// The worker thread
std::counting_semaphore db{1};
std::counting_semaphore mutex{1};
std::counting_semaphore queue{1};
int rc = 0;

void readerwriter( int workerID, double percentReader, int numSeconds ) {
    …
      if ( reader ) {
         queue.acquire();
         mutex.acquire();
         rc++;
         if (rc == 1)
            db.acquire();
         queue.release();
```

```
            mutex.release();
            result = theDatabase.read( workerID );
            ++reads;
            mutex.acquire();
            rc--;
            if (rc == 0)
               db.release();
            mutex.release();

         } else // if writer
         {
            queue.acquire();
            db.acquire();
            queue.release();
            result = theDatabase.write( workerID );
            ++writes;
            db.release();

         }; // end if writer
         ++tests;

         // NON-CRITICAL AREAD
         // Sleep a while...
         …

      } // repeat until time used is up
         …

   } // end worker function

    // inspiriert von
    // https://en.wikipedia.org/wiki/Readers%E2%80%93writers_problem
```

Wir haben auch hier mit verschiedenen Mischungen von lesenden und schreibenden Threads experimentiert - genau wie bei erster Lösung. (alle mit einer Zeit von 5 Minuten und mit 100 Threads).

PercentageReader	Leseoperationen	Schreiboperationen
10	71	739
50	721	674
90	5062	561

Es gibt hier vielleicht irgendwo einen Fehler. Normalerweise sollte hier die Anzahl der Lese- und Schreiboperatio ungefähr gleich sein. Dies zeigt, dass die faire Methode versucht, eine gute Balance zwischen Lese- und Schreibvorgängen zu erreichen.