Subject: Performance comparison of mmap+MemStream to FileIn
Posted by jjacksonRIAB on Wed, 14 Feb 2024 07:22:41 GMT
View Forum Message <> Reply to Message

```
#include <Core/Core.h>
#include <Functions4U/Functions4U.h>
#include <sys/mman.h>

using namespace Upp;

class MemoryMapStream : public MemStream {
public:
   void Load(String filename) {
      file.Open(filename);
      map = (char*)mmap(0, file.GetSize(), PROT_READ, MAP_SHARED, file.GetHandle(), 0);
      Create(map, file.GetSize());
   }

   MemoryMapStream(String filename) {
      Load(filename);
   }

   ~MemoryMapStream() {
      if(map) {
         munmap(map, file.GetSize());
      }
   }

private:
   FileIn    file;
   char*     map {};
};

CONSOLE_APP_MAIN {
   Vector<int64> positions;

   // 1 gigabyte test csv file
   const String fname = "test.csv";

   {
      FileIn f(fname);

      while(!f.IsEof()) {
         GetCsvLine(f, ',', CHARSET_DEFAULT);
         positions.Add() = f.GetPos();
      }
```

```
    positions.Drop();
    Shuffle(positions);
}

Vector<String> lines;

{
    MemoryMapStream mms(fname);

    TimeStop tm;
    RTIMESTOP("MemoryMapStream serial performance");

    while(!mms.IsEof()) {
        RTIMING("GetCsvLine MemorymapStream");
        lines = GetCsvLine(mms, ',', CHARSET_DEFAULT);
    }

    RDUMP(tm.Elapsed());
}

{
    FileIn f(fname);

    TimeStop tm;
    RTIMESTOP("FileIn serial performance");

    while(!f.IsEof()) {
        RTIMING("GetCsvLine FileIn");
        lines = GetCsvLine(f, ',', CHARSET_DEFAULT);
    }

    RDUMP(tm.Elapsed());
}

{
    MemoryMapStream mms(fname);

    TimeStop tm;
    RTIMESTOP("MemoryMapStream random performance");

    for(int i = 0; i < positions.GetCount(); i++) {
        RTIMING("GetCsvLine MemoryMapStream (random seek)");

        mms.Seek(positions[i]);
        lines = GetCsvLine(mms, ',', CHARSET_DEFAULT);
    }

    RDUMP(tm.Elapsed());
```

```
    }

    {
        FileIn f(fname);

        TimeStop tm;
        RTIMESTOP("FileIn random performance");

        for(int i = 0; i < positions.GetCount(); i++) {
   RTIMING("GetCsvLine FileIn (random seek)");

            f.Seek(positions[i]);
            lines = GetCsvLine(f, ',', CHARSET_DEFAULT);
        }

        RDUMP(tm.Elapsed());
    }
}
```

```
tm.Elapsed() = 6755842
MemoryMapStream serial performance 6.755847 s
tm.Elapsed() = 7023237
FileIn serial performance 7.023258 s
tm.Elapsed() = 7768212
MemoryMapStream random performance 7.768234 s
tm.Elapsed() = 16070698
FileIn random performance 16.070718 s
TIMING GetCsvLine FileIn (random seek): 15.70 s  -  2.35 us (15.83 s  / 6674559 ), min:  0.00 ns, max:  1.00 ms, nesting: 0 - 6674559
TIMING GetCsvLine MemoryMapStream (random seek):  7.43 s  -  1.11 us ( 7.55 s  / 6674559 ), min:  0.00 ns, max:  1.00 ms, nesting: 0 - 6674559
TIMING GetCsvLine FileIn:  6.69 s  -  1.00 us ( 6.81 s  / 6674560 ), min:  0.00 ns, max:  1.00 ms, nesting: 0 - 6674560
TIMING GetCsvLine MemorymapStream:  6.39 s  - 956.63 ns ( 6.51 s  / 6674560 ), min:  0.00 ns, max:  1.00 ms, nesting: 0 - 6674560
```

For sequential reads the performance is roughly the same, but for random seeks, skips, etc the mmapped stream performs significantly better. This test might even understate how much better it performs. Recently I found myself parsing MP4 containers and for a 1.9 gigabyte video the serialization time for the container was around 0.5 seconds. MP4 contains a lot of tags called atoms that are interspersed with binary throughout the video, leading to a lot of skips to get to the next tag. I was able to improve that by increasing FileIn's buffer size to around 0.33 seconds.

Wanting to see if I could do even better, I mmapped the whole video and I got the read time down to 0.03 seconds, a significant improvement. Perhaps the greatest improvement in that case was a reduction in stalled cycles. It went from around 7000% to 188.87%.