Processing Large Files in C

Mike Peters, 10-03-2008

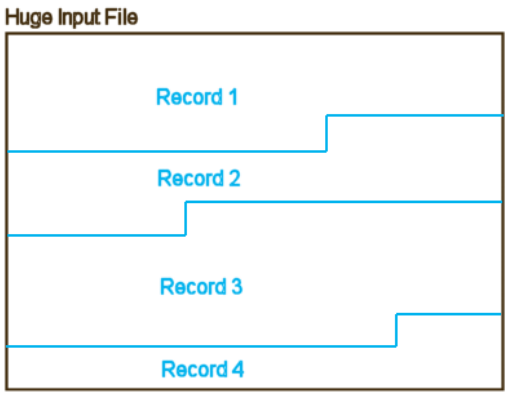
Imagine this scenario -

You are faced with the task of writing a parser that will be capable of handling files of **unlimited size**. Every record in the file has a varying length. You cannot read the entire file into memory and you cannot use fgets() or fgetcsv() type functions to read records one at a time.

As part of this post I will present a simple, yet highly effective, algorithm that will enable you to process files of unlimited size at ease. We use this extensively whenever Importing or Parsing files.

The key principle is using a single fixed-size read-buffer along with a left-over-buffer.

Before we jump into the code, let's review the data layout of a typical huge input file:



Notice how every record has a different size in the huge input file.

The most efficient method to handle such a file is as follows:

1. Read as many bytes as our read-buffer can store from huge input file

2. Pass read-buffer to our ProcessData() function. The ProcessData function will scan through the passed read-buffer, search for end-of-record separator (end of line, semicolon or whatever end-of-record separator you use) and process the records.

3. Upon completion, ProcessData() will have a left-over buffer it cannot process. A beginning of a new record with no end-of-record separator (see diagram below). ProcessData() returns that left-over buffer to the calling function

4. Copy left-over-buffer to the beginning of our read-buffer and read size_of_buffer - size_of_leftover new bytes from the file

5. Goto 2

The read-buffer should be big enough to hold a record of the maximum allowed record-size.

For example, let's assume the read-buffer can hold all of records 1 and 2. Our algorithm will iterate twice as follows:

Step 1:

Read as much as possible into read-buffer and pass to ProcessData function:

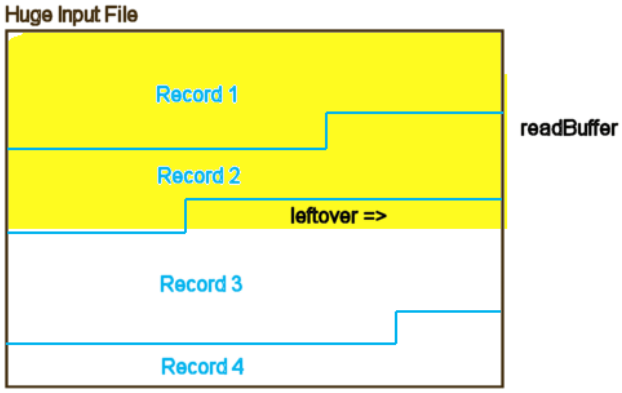
Recent Posts

Monitoring services with xinetd
Optimizing NGINX and PHP-fpm for high traf
How to install htop
How to delete files when argument list too l
6 Months with GlusterFS: a Distributed File S
How to: Install PHP w/ FPM + Memcached +
NTP for Accurate Global Time Synchronizatic
Cassandra for PHP Sessions
redis: a persistent key-value store
How to hide .php extension in your urls with

Subscribe to RSS

Sponsors

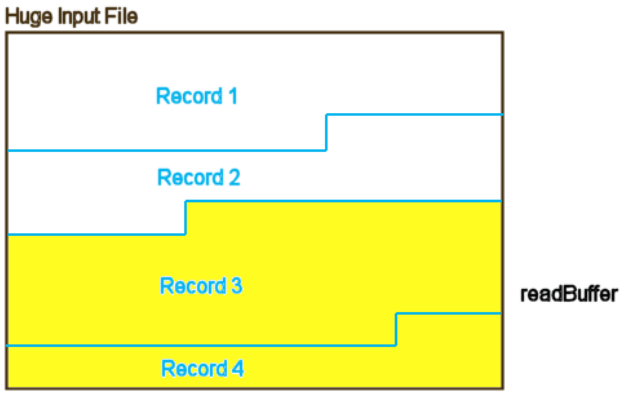
Help



ProcessData will process both Record1 and Record2. It will be unable to process Record3 due to not finding end-of-record separator. As a result ProcessData returns the beginning of Record3 as the left-over buffer.

Step 2:

left-over buffer copied to the beginning of the read-buffer.
Read as much as possible into read-buffer and pass to ProcessData function:



ProcessData can now process Record3 and Record4.

-

The code in C that makes it all happen:

```
 #define       MAXLINELENGTH    1024 // Max record size
#define       BUFSIZE      50000

long          bytesread;
char          buf[BUFSIZE];
int           sizeLeftover=0;
int           bLoopCompleted = 0;
long       pos = 0;

// Open source file
if (!(handle = fopen(Filename,"rb")))
{
// Bail
return 0;
}

do
{

// Read next block from file and save into buf, right after the
// "left over" buffer
   bytesread = fread(buf+sizeLeftover, 1, sizeof(buf)-1-sizeLeftover, handle);
   if (bytesread<1)
   {
      // Turn on 'loop completed' flag so that we know to exit at the bottom
      // Still need to process any block we currently have in the
```

Help

```
        // leftover buffer
        bLoopCompleted = 1;
        bytesread  = 0;
      }

  // Add NULL terminator at the end of our buffer
      buf[bytesread+sizeLeftover] = 0;

  // Process data - Replace with your function
  //
  // Function should return the position in the file or -1 if failed
  //
  // We are also passing bLoopCompleted to let ProcessData know whether this is
  // the last record (in which case - if no end-of-record separator,
  // use eof and process anyway)
      pos = ProcessData(connection, buf, bytesread+sizeLeftover,
      bLoopCompleted);

  // If error occured, bail
      if (pos<1)
      {
        bLoopCompleted = 1;
        pos      = 0;
      }

  // Set Left over buffer size to
  //
  //  * The remaining unprocessed buffer that was not processed
  //  by ProcessData (because it couldn't find end-of-line)
  //
  // For protection if the remaining unprocessed buffer is too big
  // to leave sufficient room for a new line (MAXLINELENGTH), cap it
  // at maximumsize - MAXLINELENGTH
      sizeLeftover = mymin(bytesread+sizeLeftover-pos, sizeof(buf)-MAXLINELENGTH);
  // Extra protection - should never happen but you can never be too safe
      if (sizeLeftover<1) sizeLeftover=0;

  // If we have a leftover unprocessed buffer, move it to the beginning of
  // read buffer so that when reading the next block, it will connect to the
  // current leftover and together complete a full readable line
      if (pos!=0 && sizeLeftover!=0)
      memmove(buf, buf+pos, sizeLeftover);

  } while(!bLoopCompleted);

// Close file
fclose(handle);
```

James, 10-28-2017

Nice work. Unsure what the connection parameter in the ProcessData function is expecting.

Gerard de Jong, 03-20-2018

Look good and I would like to use this algorithm.
But can you give me the code of the following routines:
1. 'ProcessData'
2. 'mymin'
3. 'memmove'

thanks!

Best regards, Gerard

Enjoyed this post? Subscribe to RSS

Subscribe Now to receive new posts via Email as soon as they come out.

COMMENTS

POST YOUR COMMENTS

Name:

Email: (Required - never shown)

URL:

Comments:

Post (please only click once!)

Note: **No link spamming!** If your message contains link/s, it will NOT be published on the site before manually approved by one of our moderators.

Help