CS 180A Assignment 1a (Logger) - Ryman Barnett

Generated by Doxygen 1.8.17

1 File Index

1.1 File List

Here is a list of all documented files with brief descriptions:

logger.c
Source file for logger functions to open and write to a file using buffered output that will dump to file when the buffer is full **1**

logger.h
Header file for logger functions **6**

2 File Documentation

2.1 logger.c File Reference

Source file for logger functions to open and write to a file using buffered output that will dump to file when the buffer is full.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <unistd.h>
#include "logger.h"
```

Functions

- LOG_RESULT log_open (const char ∗filename, unsigned int buffersize)

  opens a file, and allocated a buffer for writting to it
- LOG_RESULT log_flush (void)

  write contents of the buffer to the disk and then reset the buffer
- LOG_RESULT log_write (const unsigned char ∗text)

  write bytes from a given string into the buffer and flush it to disk when buffer is full
- LOG_RESULT log_close (void)

  closes a file, deallocates all buffers , and resets counters
- void log_dump (void)

  dumps contents of the buffer to the command line

Variables

- static unsigned char ∗ LOG_BUFFER

  The memory buffer.
- static unsigned int LOG_BUFSIZE

  The size of the buffer.
- static unsigned int LOG_BYTECOUNT

  How many bytes in the buffer?
- static char ∗ LOG_FILENAME

  Name of the file to write the log to.
- static int LOG_FH = -1

  The file handle of the open file.

2.1.1 Detailed Description

Source file for logger functions to open and write to a file using buffered output that will dump to file when the buffer is full.

Author

Ryman Barnett

email: ryman.b@digipen.edu

DigiPen login: ryman.b

Course: CS180

Section: A

Assignment #1a

Date

2022-09-13

2.1.2 Function Documentation

2.1.2.1 log_close()

```
LOG_RESULT log_close (
        void  )
```

closes a file, deallocates all buffers , and resets counters

Returns

LOG_WRITEFAIL if buffer could'nt be written to file, and LOG_SUCCESS otherwise

Definition at line 167 of file logger.c.

```
{
  // flush remaining buffer into file
  LOG_RESULT writeRes =  log_flush();

  close(LOG_FH);       // close file
  free(LOG_BUFFER);    // free buffer
  free(LOG_FILENAME); // free filename

  LOG_BUFFER = NULL;   // reset dangling pointer
  LOG_FH = -1;         // reset dangling pointer
  LOG_FILENAME = NULL;// reset dangling pointer
  LOG_BUFSIZE = 0;     // reset size counter
  LOG_BYTECOUNT = 0;   // reset bytes read

  return writeRes; // result of write
}
```

References LOG_BUFFER, LOG_BUFSIZE, LOG_BYTECOUNT, LOG_FH, LOG_FILENAME, and log_flush().

Referenced by log_flush(), and log_write().

2.1.2.2 log_flush() `LOG_RESULT log_flush ( void )`

write contents of the buffer to the disk and then reset the buffer

Returns

LOG_WRITEFAIL if buffer cant be written to disk, and LOG_SUCCESS otherwise

Definition at line 86 of file logger.c.

```
{
  // try to write buffer to file
  if (LOG_BYTECOUNT == 0 || write(LOG_FH, LOG_BUFFER, LOG_BYTECOUNT))
  {
    // sync file
    if (fsync(LOG_FH) != -1)
    {
      LOG_BYTECOUNT = 0;  // reset buffer counter

      return LOG_SUCCESS; // success
    }
  }

  log_close(); // close for error

  return LOG_WRITEFAIL; // failed to write or could not sync file
}
```

References LOG_BUFFER, LOG_BYTECOUNT, log_close(), LOG_FH, LOG_SUCCESS, and LOG_WRITEFAIL.

Referenced by log_close(), and log_write().

2.1.2.3 log_open() `LOG_RESULT log_open ( const char * filename, unsigned int buffersize )`

opens a file, and allocated a buffer for writting to it

Parameters

filename	const char∗ to the name of the file to open
buffersize	unsigned int to size of buffer to allocate

Returns

LOG_NOMEM if buffer cant be allocated, LOG_OPENFAIL if file not opened, and LOG_SUCCESS otherwise

Definition at line 44 of file logger.c.

```
{
  int len = (strlen(filename) + 1);          // length of filename
  LOG_BUFSIZE = buffersize;                  // store buffersize
  LOG_BYTECOUNT = 0;                         // Initial bytes read are 0
  LOG_FILENAME = (char *)malloc(len);        // allocate space for filename
  LOG_BUFFER = (char *)malloc(LOG_BUFSIZE); // allocate buffer memory

  // make sure both mallocs were successfull
  if (LOG_BUFFER == NULL || LOG_FILENAME == NULL)
  {
    free(LOG_BUFFER);   // free buffer if it was allocated
    free(LOG_FILENAME); // free filename if it was allocated

    return LOG_NOMEM; // memory not allocated
  }

  // copy file name
  strcpy(LOG_FILENAME, filename);

  // open file
  LOG_FH = open(LOG_FILENAME, O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);

  // validate opened file
  if (LOG_FH == -1)
  {
    free(LOG_FILENAME); // free filename if it was allocated
    free(LOG_BUFFER);   // free buffer if it was allocated

    return LOG_OPENFAIL; // file not opened
  }

  return LOG_SUCCESS; // no errors file was opened
}
```

References LOG_BUFFER, LOG_BUFSIZE, LOG_BYTECOUNT, LOG_FH, LOG_FILENAME, LOG_NOMEM, LOG↩ _OPENFAIL, and LOG_SUCCESS.

2.1.2.4 log_write() `LOG_RESULT log_write ( const unsigned char * text )`

write bytes from a given string into the buffer and flush it to disk when buffer is full

Parameters

text	const unsigned char∗ to text to write to file

Returns

LOG_WRITEFAIL if buffer could'nt be written to file, and LOG_SUCCESS otherwise

Definition at line 117 of file logger.c.

```
{
  long unsigned i = 0; // loop counter

  // for each letter in text
  while (i < strlen(text))
  {
    // check if it should flush
    if (LOG_BYTECOUNT < LOG_BUFSIZE)
    {
      // if not write the text char into buffer
      LOG_BUFFER[LOG_BYTECOUNT] = text[i];
      ++LOG_BYTECOUNT; // another byte read
      ++i; // move text buffer counter
    }
    else // if it should flush
    {
      // flush buffer into file
      if (log_flush() == LOG_WRITEFAIL)
      {
        log_close();

        return LOG_WRITEFAIL; // failed to write
      }
    }
  }

  // check if final buffer is full
  if (LOG_BYTECOUNT >= LOG_BUFSIZE)
  {
    // flush buffer into file
    if (log_flush() == LOG_WRITEFAIL)
    {
      log_close();

      return LOG_WRITEFAIL; // failed to write
    }
  }

  return LOG_SUCCESS; // succeded
}
```

References LOG_BUFFER, LOG_BUFSIZE, LOG_BYTECOUNT, log_close(), log_flush(), LOG_SUCCESS, and LO↩ G_WRITEFAIL.

2.2 logger.h File Reference

Header file for logger functions.

Typedefs

- typedef enum LOG_RESULT LOG_RESULT

  < enum for return results of log functions

Enumerations

- enum LOG_RESULT { LOG_SUCCESS, LOG_NOMEM, LOG_OPENFAIL, LOG_WRITEFAIL }

  < enum for return results of log functions

Functions

- LOG_RESULT log_open (const char ∗filename, unsigned int buffersize)

  opens a file, and allocated a buffer for writting to it
- LOG_RESULT log_write (const unsigned char ∗text)

  write bytes from a given string into the buffer and flush it to disk when buffer is full
- LOG_RESULT log_flush (void)

  write contents of the buffer to the disk and then reset the buffer
- LOG_RESULT log_close (void)

  closes a file, deallocates all buffers , and resets counters
- void log_dump (void)

  dumps contents of the buffer to the command line

2.2.1 Detailed Description

Header file for logger functions.

Author

Ryman Barnett

email: ryman.b@digipen.edu

DigiPen login: ryman.b

Course: CS180

Section: A

Assignment #1a

Date

2022-09-13

2.2.2 Enumeration Type Documentation

2.2.2.1 LOG_RESULT enum LOG_RESULT

< enum for return results of log functions

Enumerator

LOG_SUCCESS	No errors.
LOG_NOMEM	Couldn't allocate buffer.
LOG_OPENFAIL	File couldn't be opened.
LOG_WRITEFAIL	Disk write failed.

Definition at line 18 of file logger.h.

```
{
  LOG_SUCCESS,    //!< No errors
  LOG_NOMEM,      //!< Couldn't allocate buffer
  LOG_OPENFAIL,   //!< File couldn't be opened
  LOG_WRITEFAIL,  //!< Disk write failed
}LOG_RESULT;
```

2.2.3 Function Documentation

2.2.3.1 log_close() `LOG_RESULT log_close ( void )`

closes a file, deallocates all buffers , and resets counters

Returns

LOG_WRITEFAIL if buffer could'nt be written to file, and LOG_SUCCESS otherwise

Definition at line 167 of file logger.c.

```
{
  // flush remaining buffer into file
  LOG_RESULT writeRes =  log_flush();

  close(LOG_FH);       // close file
  free(LOG_BUFFER);    // free buffer
  free(LOG_FILENAME);  // free filename

  LOG_BUFFER = NULL;   // reset dangling pointer
  LOG_FH = -1;         // reset dangling pointer
  LOG_FILENAME = NULL;// reset dangling pointer
  LOG_BUFSIZE = 0;     // reset size counter
  LOG_BYTECOUNT = 0;   // reset bytes read

  return writeRes; // result of write
}
```

References LOG_BUFFER, LOG_BUFSIZE, LOG_BYTECOUNT, LOG_FH, LOG_FILENAME, and log_flush().

Referenced by log_flush(), and log_write().

2.2.3.2 log_flush() `LOG_RESULT log_flush ( void )`

write contents of the buffer to the disk and then reset the buffer

Returns

LOG_WRITEFAIL if buffer cant be written to disk, and LOG_SUCCESS otherwise

Definition at line 86 of file logger.c.

```
{
  // try to write buffer to file
  if (LOG_BYTECOUNT == 0 || write(LOG_FH, LOG_BUFFER, LOG_BYTECOUNT))
  {
    // sync file
    if (fsync(LOG_FH) != -1)
    {
      LOG_BYTECOUNT = 0;  // reset buffer counter

      return LOG_SUCCESS; // success
    }
  }

  log_close(); // close for error

  return LOG_WRITEFAIL; // failed to write or could not sync file
}
```

References LOG_BUFFER, LOG_BYTECOUNT, log_close(), LOG_FH, LOG_SUCCESS, and LOG_WRITEFAIL.

Referenced by log_close(), and log_write().

2.2.3.3 log_open()

```
LOG_RESULT log_open (
        const char * filename,
        unsigned int buffersize )
```

opens a file, and allocated a buffer for writting to it

Parameters

filename	const char∗ to the name of the file to open
buffersize	unsigned int to size of buffer to allocate

Returns

LOG_NOMEM if buffer cant be allocated, LOG_OPENFAIL if file not opened, and LOG_SUCCESS otherwise

Definition at line 44 of file logger.c.

```
{
  int len = (strlen(filename) + 1);          // length of filename
  LOG_BUFSIZE = buffersize;                  // store buffersize
  LOG_BYTECOUNT = 0;                         // Initial bytes read are 0
  LOG_FILENAME = (char *)malloc(len);        // allocate space for filename
  LOG_BUFFER = (char *)malloc(LOG_BUFSIZE); // allocate buffer memory

  // make sure both mallocs were successfull
  if (LOG_BUFFER == NULL || LOG_FILENAME == NULL)
  {
    free(LOG_BUFFER);   // free buffer if it was allocated
    free(LOG_FILENAME); // free filename if it was allocated

    return LOG_NOMEM; // memory not allocated
  }

  // copy file name
  strcpy(LOG_FILENAME, filename);

  // open file
  LOG_FH = open(LOG_FILENAME, O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);

  // validate opened file
  if (LOG_FH == -1)
  {
    free(LOG_FILENAME); // free filename if it was allocated
    free(LOG_BUFFER);   // free buffer if it was allocated

    return LOG_OPENFAIL; // file not opened
  }

  return LOG_SUCCESS; // no errors file was opened
}
```

References LOG_BUFFER, LOG_BUFSIZE, LOG_BYTECOUNT, LOG_FH, LOG_FILENAME, LOG_NOMEM, LOG↩
_OPENFAIL, and LOG_SUCCESS.

2.2.3.4 log_write()

```
LOG_RESULT log_write (
        const unsigned char * text )
```

write bytes from a given string into the buffer and flush it to disk when buffer is full

Parameters

text	const unsigned char∗ to text to write to file

Returns

LOG_WRITEFAIL if buffer could'nt be written to file, and LOG_SUCCESS otherwise

Definition at line 117 of file logger.c.

```
{
  long unsigned i = 0; // loop counter

  // for each letter in text
  while (i < strlen(text))
  {
    // check if it should flush
    if (LOG_BYTECOUNT < LOG_BUFSIZE)
    {
      // if not write the text char into buffer
      LOG_BUFFER[LOG_BYTECOUNT] = text[i];
      ++LOG_BYTECOUNT; // another byte read
      ++i; // move text buffer counter
    }
    else // if it should flush
    {
      // flush buffer into file
      if (log_flush() == LOG_WRITEFAIL)
      {
        log_close();

        return LOG_WRITEFAIL; // failed to write
      }
    }
  }

  // check if final buffer is full
  if (LOG_BYTECOUNT >= LOG_BUFSIZE)
  {
    // flush buffer into file
    if (log_flush() == LOG_WRITEFAIL)
    {
      log_close();

      return LOG_WRITEFAIL; // failed to write
    }
  }

  return LOG_SUCCESS; // succeded
}
```

References LOG_BUFFER, LOG_BUFSIZE, LOG_BYTECOUNT, log_close(), log_flush(), LOG_SUCCESS, and LO↩ G_WRITEFAIL.

Index