| LCOV - code coverage report |
| --- |
|  |
| | Current view: | [top level](http://docs.google.com/index.html) - [Common/src](http://docs.google.com/index.html) - Memory.cpp (source / [functions](http://docs.google.com/Memory.cpp.func.html)) |  |  | Hit | Total | Coverage | | --- | --- | --- | --- | --- | --- | --- | | Test: | EBM9K Converter Coverage Report |  | Lines: | 70 | 124 | 56.5 % | | Date: | 2012-03-23 |  | Functions: | 13 | 13 | 100.0 % | | Legend: | Lines: hit not hit | Branches: + taken - not taken # not executed |  | Branches: | 21 | 90 | 23.3 % | |  | |
|  |

|  |
| --- |
| Branch data Line data Source code  1 : : #include <cstdlib>  2 : : #include <iostream>  3 : : #include <list>  4 : : #include <boost/thread/mutex.hpp>  5 : :   6 : : #include "ErrorHandler.h"  7 : : #include "Memory.h"  8 : : #include "RetryAgent.h"  9 : : #include "RetryManager.h"  10 : : #include "print.h"  11 : : #include "Logger.h"  12 : :   13 : : namespace nft {  14 : :   15 : : namespace \_impl {  16 : : class ChunkAllocator {  17 : : typedef char \* Chunk;  18 : : typedef std::list<Chunk> ChunkList;  19 : : public:  20 : : static ChunkAllocator \*  21 : 293367 : instance() {  22 [ - + ][ # # ]: 195558 : static ChunkAllocator obj;  [ - + ][ # # ]  [ - + ][ # # ]  [ # # ][ # # ]  [ - + ][ # # ]  [ + + ][ + - ]  23 : : return &obj;  24 : : }  25 : :   26 : : void \*  27 : : allocate( size\_t inSize )  28 : 293427 : {  29 : 97809 : boost::mutex::scoped\_lock l(mMutex);  30 : 97809 : Chunk pChunk = mCurrentChunk;  31 [ # # ][ - + ]: 97809 : if( mCurrentOffset + inSize > mChunkSize )  [ + + ]  32 : : {  33 [ # # ][ # # ]: 1110 : mChunkSize = ( inSize > mChunkSize ) ? inSize : mChunkSize;   [ - + ]  34 : 1110 : Chunk newChunk = (Chunk)malloc( mChunkSize );  35 : 1110 : mChunks.push\_back( newChunk );  36 : 1110 : mCurrentChunk = newChunk;  37 : 1110 : mCurrentOffset = 0;  38 : 1110 : pChunk = newChunk;  39 : : }  40 : 97809 : char \*pMem = pChunk + mCurrentOffset;  41 : 97809 : mCurrentOffset += inSize;  42 : : return pMem;  43 : : }  44 : :   45 : : void  46 : : free( void \*p )  47 : 97749 : {  48 [ # # ][ # # ]: 85759 : }  [ # # ][ # # ]  [ # # ]  49 : :   50 : : ChunkAllocator()  51 : 1110 : :mChunkSize( 1073741824 ), //100MB  52 : : mChunks(),  53 : : mCurrentChunk(),  54 : 1110 : mCurrentOffset( mChunkSize ) {  55 : : }  56 : :   57 : 1110 : ~ChunkAllocator() {  58 : 4440 : for( ChunkList::iterator it = mChunks.begin(); it != mChunks.end();  59 : : ++it )  60 : : {  61 : 1110 : ::free(\*it);  62 : : }  63 : 1110 : mChunks.clear();  64 : 1110 : mCurrentChunk = 0;  65 : : }  66 : :   67 : : private:  68 : : size\_t mChunkSize;  69 : :   70 : : ChunkList mChunks;  71 : : Chunk mCurrentChunk;  72 : : size\_t mCurrentOffset;  73 : : boost::mutex mMutex;  74 : : };  75 : :   76 : : } //namespace \_impl  77 : :   78 : :   79 : : void \*  80 : 46645 : nftMalloc( size\_t inSize ) throw(DPL::Error \*) {  81 : 46645 : int iErrNo;  82 : 46645 : RetryAgent retryAgent = RetryManager::instance()->getAgent();  83 : 46645 : do {  84 : : #ifdef TEST\_RETRY\_20  85 : : void \*p = NULL;  86 : : static int a = 1;  87 : : if( a >= 5 )   88 : : {  89 : : p = malloc( inSize );  90 : : }  91 : : else  92 : : {  93 : : e9k\_debug("nftMallocWithRetry ENOMEM\n");  94 : : }   95 : : #else  96 : : void \*p = \_impl::ChunkAllocator::instance()  97 : 46645 : ->allocate( inSize );  98 : : #ifdef TEST\_RETRY\_19  99 : : e9k\_debug("TEST\_RETRY\_19 nftMallocWithRetry\n");  100 : : #endif   101 : : #endif  102 : : //#ifdef ERROR\_SEED //<<Test\_Id:Memory\_Malloc\_iErrno\_Set\_to\_Null>>  103 : : // p = NULL;  104 : : //#endif  105 [ - + ]: 46645 : if( NULL != p) {  106 : 46645 : return p;  107 : : } else {  108 : 0 : iErrNo = errno; //System error no  109 : : #ifdef TEST\_RETRY\_20  110 : : if( a < 5 ) {  111 : : iErrNo = ENOMEM;  112 : : a++;  113 : : }  114 : : #endif  115 : : //#ifdef ERROR\_SEED //<<Test\_Id:Memory\_Malloc\_iErrno\_Set\_to\_Null>>  116 : : // iErrNo=NULL;  117 : : //#endif  118 [ # # ]: 0 : if ( iErrNo != ENOMEM && iErrNo != EAGAIN) {  119 : : //unrecoverable error  120 : 0 : char systemErrorMsg[BUFSIZ];  121 : 0 : Logger::instance()->log(LogMessageTable::eMEMORY\_ALLOC);  122 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  123 : 0 : pErr->errorAdd(NULL, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  124 : : ErrorHandler::eErrMemoryAllocation,  125 : : pErr->getErrorMsg(  126 : : ErrorHandler::eErrMemoryAllocation,  127 : : strerror\_r( iErrNo, systemErrorMsg, BUFSIZ )));  128 : 0 : return NULL;  129 : : }  130 : : }  131 [ # # ]: 0 : } while (eAgain == retryAgent.waitForSpecifiedTime());  132 : 0 : char systemErrorMsg[BUFSIZ];  133 : 0 : Logger::instance()->log(LogMessageTable::eMEMORY\_ALLOC);  134 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  135 : 0 : pErr->errorAdd(NULL, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  136 : : ErrorHandler::eErrMemoryAllocation,  137 : : pErr->getErrorMsg(  138 : : ErrorHandler::eErrMemoryAllocation,  139 : : strerror\_r( iErrNo, systemErrorMsg, BUFSIZ )));  140 : :   141 [ # # ]: 0 : return NULL;  142 : : }  143 : :   144 : : void \*  145 : 51164 : nftMallocNoThrow( size\_t inSize ) {  146 : 51164 : e9k\_debug("nftMallocNoThrow\n");  147 : 51164 : RetryAgent retryAgent = RetryManager::instance()->getAgent();  148 : 51164 : do {  149 : 51164 : void \*p = \_impl::ChunkAllocator::instance()->allocate( inSize );  150 [ - + ]: 51164 : if(NULL != p) {  151 : 51164 : return p;  152 : : } else {  153 [ # # ]: 0 : if ( errno != ENOMEM && errno != EAGAIN) {  154 : : //unrecoverable error  155 : 0 : return NULL;  156 : : }  157 : : }  158 [ # # ]: 0 : } while (eAgain == retryAgent.waitForSpecifiedTime());  159 : :   160 : 0 : return NULL;  161 : : }  162 : :   163 : : void \*  164 : 0 : nftCalloc( size\_t inNumElements, size\_t inSize ) throw(DPL::Error \*) {  165 : 0 : int iErrNo;  166 : 0 : RetryAgent retryAgent = RetryManager::instance()->getAgent();  167 : 0 : do {  168 : 0 : void \*p = \_impl::ChunkAllocator::instance()->allocate( inSize );  169 [ # # ]: 0 : if(NULL != p) {  170 : 0 : return p;  171 : : } else {  172 : 0 : iErrNo = errno; //System error no  173 [ # # ]: 0 : if ( errno != ENOMEM && errno != EAGAIN) {  174 : : //unrecoverable error  175 : 0 : char systemErrorMsg[BUFSIZ];  176 : 0 : Logger::instance()->log(LogMessageTable::eMEMORY\_ALLOC);  177 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  178 : 0 : pErr->errorAdd(NULL, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  179 : : ErrorHandler::eErrMemoryAllocation,  180 : : pErr->getErrorMsg(  181 : : ErrorHandler::eErrMemoryAllocation,  182 : : strerror\_r( iErrNo, systemErrorMsg, BUFSIZ )));  183 : 0 : return NULL;  184 : : }  185 : : }  186 [ # # ]: 0 : } while (eAgain == retryAgent.waitForSpecifiedTime());  187 : 0 : char systemErrorMsg[BUFSIZ];  188 : 0 : Logger::instance()->log(LogMessageTable::eMEMORY\_ALLOC);  189 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  190 : 0 : pErr->errorAdd(NULL, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  191 : : ErrorHandler::eErrMemoryAllocation,  192 : : pErr->getErrorMsg(  193 : : ErrorHandler::eErrMemoryAllocation,  194 : : strerror\_r( iErrNo, systemErrorMsg, BUFSIZ )));  195 : :   196 [ # # ]: 0 : return NULL;  197 : : }  198 : :   199 : : void \*  200 : 5940 : nftRealloc( void \*inOldPtr, size\_t inSize ) throw(DPL::Error \*) {  201 : 5940 : int iErrNo;  202 : 5940 : RetryAgent retryAgent = RetryManager::instance()->getAgent();  203 : 5940 : do {  204 : 5940 : void \*p = realloc( inOldPtr, inSize );  205 : : //#ifdef ERROR\_SEED //<<Test\_Id:Memory\_Realloc\_errno\_Set\_to\_Null>>  206 : : // p=NULL;  207 : : // errno=NULL;  208 : : //#endif   209 [ - + ]: 5940 : if(NULL != p) {  210 : 5940 : return p;  211 : : } else {  212 : 0 : iErrNo = errno; //System error no  213 [ # # ]: 0 : if ( errno != ENOMEM && errno != EAGAIN) {  214 : : //unrecoverable error  215 : 0 : char systemErrorMsg[BUFSIZ];  216 : 0 : Logger::instance()->log(LogMessageTable::eMEMORY\_ALLOC);  217 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  218 : 0 : pErr->errorAdd(NULL, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  219 : : ErrorHandler::eErrMemoryAllocation,  220 : : pErr->getErrorMsg(  221 : : ErrorHandler::eErrMemoryAllocation,  222 : : strerror\_r( iErrNo, systemErrorMsg, BUFSIZ )));  223 : 0 : return NULL;  224 : : }  225 : : }  226 [ # # ]: 0 : } while (eAgain == retryAgent.waitForSpecifiedTime());  227 : 0 : char systemErrorMsg[BUFSIZ];  228 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  229 : 0 : Logger::instance()->log(LogMessageTable::eMEMORY\_ALLOC);  230 : 0 : pErr->errorAdd(NULL, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  231 : : ErrorHandler::eErrMemoryAllocation,  232 : : pErr->getErrorMsg(  233 : : ErrorHandler::eErrMemoryAllocation,  234 : : strerror\_r( iErrNo, systemErrorMsg, BUFSIZ )));  235 : :   236 [ # # ]: 0 : return NULL;  237 : : }  238 : :   239 : : void  240 : 127236 : nftFree( void \*inPtr, bool inIsFromSystem ) throw(DPL::Error \*) {  241 [ + + ][ + + ]: 127236 : if( !inPtr )  [ + - ]  242 : 84665 : {  243 : 41465 : return;  244 : : }  245 [ + + ]: 41465 : if( inIsFromSystem )  246 : : {  247 : 28381 : free( inPtr );  248 : 28381 : return;  249 : : }  250 : 97749 : \_impl::ChunkAllocator::instance()->free( inPtr );  251 [ # # ]: 41477 : return;  252 : : }  253 : :   254 : : //new delete overload  255 : : void\*   256 : 33561 : NftNewDeleteBase::operator new(size\_t numberOfBytes) throw(DPL::Error \*) {  257 : : //e9k\_debug("overloaded operator new \n");  258 : 33561 : void \*p = NULL;  259 : 33561 : try  260 : : {  261 : 33561 : p = nftMalloc(numberOfBytes);  262 : : }  263 [ # # ]: 0 : catch( DPL::Error \*pChild ) {  264 : :   265 : 0 : Logger::instance()->log(LogMessageTable::eMEMORY\_ALLOC);  266 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  267 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  268 : : ErrorHandler::eErrFunction,  269 : : pErr->getErrorMsg(  270 : : ErrorHandler::eErrFunction,  271 : : "RetryFunc::nftMalloc\_with\_retry()"));  272 : :   273 : : }  274 : 33561 : return p;  275 : : }  276 : :   277 : : void\*   278 : 2190 : NftNewDeleteBase::operator new[](size\_t numberOfBytes) throw(DPL::Error \*) {  279 : : //e9k\_debug("overloaded operator new[] \n");  280 [ # # ]: 2190 : return operator new(numberOfBytes);  281 : : }  282 : :   283 : : void\*  284 : 24042 : NftNewDeleteBase::operator new(size\_t numberOfBytes, const std::nothrow\_t& t) {  285 : : //e9k\_debug("overloaded operator new(nothrow) \n");  286 : 51164 : void \*p = nftMallocNoThrow(numberOfBytes);  287 : : return p;  288 : : }  289 : :   290 : : void\*  291 : 24042 : NftNewDeleteBase::operator new[](size\_t numberOfBytes, const std::nothrow\_t& t) {  292 : : //e9k\_debug("overloaded operator new(nothrow)[] \n");  293 : 24042 : return operator new(numberOfBytes, t);  294 : : }  295 : :   296 : : void   297 : 85759 : NftNewDeleteBase::operator delete(void\* inPtr) throw(DPL::Error \*) {  298 : : //e9k\_debug("overloaded operator delete \n");  299 : 85759 : nftFree(inPtr);  300 : : }  301 : :   302 : : void   303 : 26230 : NftNewDeleteBase::operator delete[](void\* inPtr) throw(DPL::Error \*) {  304 : : //e9k\_debug("overloaded operator delete[] \n");  305 : 26230 : operator delete(inPtr);  306 : : }  307 : :   308 : :   309 : 1215 : } //namespace nft  310 : 2430 : |

|  |
| --- |
| Generated by: [LCOV version 1.9](http://ltp.sourceforge.net/coverage/lcov.php) |