| LCOV - code coverage report |
| --- |
|  |
| | Current view: | [top level](http://docs.google.com/index.html) - [Common/src](http://docs.google.com/index.html) - RetryFunc.cpp (source / [functions](http://docs.google.com/RetryFunc.cpp.func.html)) |  |  | Hit | Total | Coverage | | --- | --- | --- | --- | --- | --- | --- | | Test: | EBM9K Converter Coverage Report |  | Lines: | 44 | 183 | 24.0 % | | Date: | 2012-03-23 |  | Functions: | 7 | 11 | 63.6 % | | Legend: | Lines: hit not hit | Branches: + taken - not taken # not executed |  | Branches: | 6 | 92 | 6.5 % | |  | |
|  |

|  |
| --- |
| Branch data Line data Source code  1 : : #include <cstdlib>  2 : : #include <sys/ipc.h>  3 : : #include <sys/shm.h>   4 : : #include "RetryFunc.h"  5 : : #include "ErrorHandler.h"  6 : : #include "RetryAgent.h"  7 : : #include "RetryManager.h"  8 : : #include "print.h"  9 : : namespace nft {  10 : :   11 : : FILE \*  12 : 2126 : nftFopenWithRetry(const char \*fileName, const char \* mode) throw(DPL::Error \*) {  13 : : //e9k\_debug("nftFopenWithRetry\n");  14 : 2126 : int iErrNo;  15 : 2126 : RetryAgent retryAgent = RetryManager::instance()->getAgent();  16 : 2126 : do {  17 : : // FILE \*p = fopen( fileName, mode );  18 : : #ifdef TEST\_RETRY\_10  19 : : FILE \*p = NULL;  20 : : static int a = 1;  21 : : if(a >= 5 )   22 : : {  23 : : p = fopen( fileName, mode );  24 : : }  25 : : else  26 : : {  27 : : e9k\_debug("nftFopenWithRetry EMFILE\n");  28 : : }  29 : : #elif TEST\_RETRY\_11  30 : : FILE \*p = NULL;  31 : : static int a = 1;  32 : : if(a >= 5 )   33 : : {  34 : : p = fopen( fileName, mode );  35 : : }  36 : : else  37 : : {  38 : : e9k\_debug("nftFopenWithRetry ENFILE\n");  39 : : }  40 : : #elif TEST\_RETRY\_12  41 : : FILE \*p = NULL;  42 : : static int a = 1;  43 : : if(a >= 5 )   44 : : {  45 : : p = fopen( fileName, mode );  46 : : }  47 : : else  48 : : {  49 : : e9k\_debug("nftFopenWithRetry ENOMEM\n");  50 : : }  51 : : #elif TEST\_RETRY\_13  52 : : FILE \*p = NULL;  53 : : static int a = 1;  54 : : if(a >= 5 )   55 : : {  56 : : p = fopen( fileName, mode );  57 : : }  58 : : else  59 : : {  60 : : e9k\_debug("nftFopenWithRetry ENSPC\n");  61 : : }  62 : : #elif TEST\_RETRY\_43  63 : : FILE \*p = NULL;  64 : : static int a = 1;  65 : : if(a >= 8 )  66 : : {  67 : : p = fopen( fileName, mode );  68 : : }  69 : : else  70 : : {  71 : : e9k\_debug("nftFopenWithRetry TEST\_RETRY\_43\n");  72 : : }  73 : : #else  74 : 2126 : FILE \*p = fopen( fileName, mode );  75 : : #ifdef TEST\_RETRY\_9  76 : : e9k\_debug("TEST\_RETRY\_9 nftFopenWithRetry\n");  77 : : #endif  78 : : #endif  79 : : //#ifdef ERROR\_SEED //<<Test\_Id:nftFopenWithRetry\_iErrNo\_Set\_To\_Null>>  80 : : // p = NULL;  81 : : //#endif  82 [ - + ]: 2126 : if(NULL != p) {  83 : 2126 : return p;  84 : : } else {  85 : 0 : iErrNo = errno; //System error no  86 : : #ifdef TEST\_RETRY\_10  87 : : if( a < 5 ) {  88 : : iErrNo = EMFILE;  89 : : a++;  90 : : }  91 : : #elif TEST\_RETRY\_11  92 : : if( a < 5 ) {  93 : : iErrNo = ENFILE;  94 : : a++;  95 : : }  96 : : #elif TEST\_RETRY\_12  97 : : if( a < 5 ) {  98 : : iErrNo = ENOMEM;  99 : : a++;  100 : : }  101 : : #elif TEST\_RETRY\_13  102 : : if( a < 5 ) {  103 : : iErrNo = ENOSPC;  104 : : a++;  105 : : }  106 : : #elif TEST\_RETRY\_43  107 : : if( a < 8 ) {  108 : : iErrNo = ENOSPC;  109 : : a++;  110 : : }  111 : : #endif  112 : : //#ifdef ERROR\_SEED //<<Test\_Id:nftFopenWithRetry\_iErrNo\_Set\_To\_Null>>  113 : : // iErrNo = NULL;  114 : : //#endif  115 [ # # ][ # # ]: 0 : if ( iErrNo != EEXIST && iErrNo != EMFILE  [ # # ]  116 : : && iErrNo != ENFILE && iErrNo != ENOMEM  117 : : && iErrNo != ENOSPC) {  118 : : //unrecoverable error  119 : 0 : char systemErrorMsg[BUFSIZ];  120 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  121 : 0 : pErr->errorAdd(NULL, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  122 : : ErrorHandler::eErrMemoryAllocation,  123 : : pErr->getErrorMsg(  124 : : ErrorHandler::eErrMemoryAllocation,  125 : : strerror\_r( iErrNo, systemErrorMsg, BUFSIZ )));  126 : 0 : return NULL;  127 : : }  128 : : }  129 [ # # ]: 0 : } while (eAgain == retryAgent.waitForSpecifiedTime());  130 : 0 : char systemErrorMsg[BUFSIZ];  131 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  132 : 0 : pErr->errorAdd(NULL, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  133 : : ErrorHandler::eErrMemoryAllocation,  134 : : pErr->getErrorMsg(  135 : : ErrorHandler::eErrMemoryAllocation,  136 : : strerror\_r( iErrNo, systemErrorMsg, BUFSIZ )));  137 : :   138 [ # # ]: 0 : return NULL;  139 : : }  140 : :   141 : : int  142 : 2096 : nftFcloseWithRetry(FILE \*stream) throw(DPL::Error \*) {  143 : 2096 : int iErrNo;  144 : 2096 : RetryAgent retryAgent = RetryManager::instance()->getAgent();  145 : 2096 : do {  146 : : // int p = fclose( stream );  147 : : #ifdef TEST\_RETRY\_15  148 : : int p;  149 : : static int a = 1;  150 : : if(a < 4 ) {  151 : : p = -1;  152 : : e9k\_debug("nftFcloseWithRetry EINTR\n");  153 : : } else {  154 : : p = fclose( stream );  155 : : }  156 : : #elif TEST\_RETRY\_16  157 : : int p;  158 : : static int a = 1;  159 : : if(a < 4 ) {  160 : : p = -1;  161 : : e9k\_debug("nftFcloseWithRetry EIO\n");  162 : : } else {  163 : : p = fclose( stream );  164 : : }  165 : : #else  166 : 2096 : int p = fclose( stream );  167 : : #ifdef TEST\_RETRY\_14  168 : : e9k\_debug("TEST\_RETRY\_14 nftFcloseWithRetry\n");  169 : : #endif  170 : : #endif  171 : : //#ifdef ERROR\_SEED //<<Test\_Id:nftFcloseWithRetry\_iErrNo\_Set\_To\_Null>>  172 : : // p = 1;  173 : : //#endif   174 [ - + ]: 2096 : if(0 == p) {  175 : 2096 : return p;  176 : : } else {  177 : 0 : iErrNo = errno; //System error no  178 : :   179 : : #ifdef TEST\_RETRY\_15  180 : : if( a < 4 ) {  181 : : iErrNo = EINTR;  182 : : a++;  183 : : }  184 : : #elif TEST\_RETRY\_16  185 : : if( a < 4 ) {  186 : : iErrNo = EIO;  187 : : a++;  188 : : }  189 : : #endif  190 : : //#ifdef ERROR\_SEED //<<Test\_Id:nftFcloseWithRetry\_iErrNo\_Set\_To\_Null>>  191 : : // iErrNo = NULL;  192 : : //#endif  193 [ # # ]: 0 : if ( iErrNo != EINTR && iErrNo != EIO ) {  194 : : //unrecoverable error  195 : 0 : char systemErrorMsg[BUFSIZ];  196 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  197 : 0 : pErr->errorAdd(NULL, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  198 : : ErrorHandler::eErrMemoryAllocation,  199 : : pErr->getErrorMsg(  200 : : ErrorHandler::eErrMemoryAllocation,  201 : : strerror\_r( iErrNo, systemErrorMsg, BUFSIZ )));  202 : 0 : return EOF;  203 : : }  204 : : }  205 [ # # ]: 0 : } while (eAgain == retryAgent.waitForSpecifiedTime());  206 : 0 : char systemErrorMsg[BUFSIZ];  207 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  208 : 0 : pErr->errorAdd(NULL, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  209 : : ErrorHandler::eErrMemoryAllocation,  210 : : pErr->getErrorMsg(  211 : : ErrorHandler::eErrMemoryAllocation,  212 : : strerror\_r( iErrNo, systemErrorMsg, BUFSIZ )));  213 : :   214 [ # # ]: 0 : return EOF;  215 : :   216 : : }  217 : :   218 : : int  219 : 12559 : nftOpenWithRetry(const char \*pathname, int flags) throw(DPL::Error \*) {  220 : 12559 : int iErrNo;  221 : 12559 : RetryAgent retryAgent = RetryManager::instance()->getAgent();  222 : 12559 : do {  223 : : //int p = open( pathname, flags );  224 : : #ifdef TEST\_RETRY\_2  225 : : int p = -1;  226 : : static int a = 1;  227 : : if(a >= 5 )   228 : : {  229 : : p = open( pathname, flags );  230 : : }  231 : : else  232 : : {  233 : : e9k\_debug("nftOpenWithRetry EMFILE\n");  234 : : }  235 : : #elif TEST\_RETRY\_3  236 : : int p = -1;  237 : : static int a = 1;  238 : : if(a >= 5 )  239 : : {  240 : : p = open( pathname, flags );  241 : : }  242 : : else  243 : : {  244 : : e9k\_debug("nftOpenWithRetry ENFILE\n");  245 : : }  246 : :   247 : : #elif TEST\_RETRY\_4  248 : : int p = -1;  249 : : static int a = 1;  250 : : if(a >= 5 )  251 : : {  252 : : p = open( pathname, flags );  253 : : }  254 : : else  255 : : {  256 : : e9k\_debug("nftOpenWithRetry ENOMEM\n");  257 : : }  258 : : #elif TEST\_RETRY\_5  259 : : int p = -1;  260 : : static int a = 1;  261 : : if(a >= 5 )  262 : : {  263 : : p = open( pathname, flags );  264 : : }  265 : : else  266 : : {  267 : : e9k\_debug("nftOpenWithRetry ENOSPC\n");  268 : : }  269 : : /\*#elif ERROR\_SEED //<<Test\_Id:nftOpenWithRetry\_iErrNo\_Set\_To\_ENOSPC>>  270 : : int p = -1;  271 : : static int a = 1;  272 : : if(a >= 5 )  273 : : {  274 : : p = open( pathname, flags );  275 : : }  276 : : else  277 : : {  278 : : e9k\_debug("nftOpenWithRetry ENOSPC\n");  279 : : }\*/  280 : : #else  281 : 12559 : int p = open( pathname, flags );  282 : : #ifdef TEST\_RETRY\_1  283 : : e9k\_debug("TEST\_RETRY\_1 nftOpenWithRetry\n");  284 : : #endif  285 : : #endif  286 : : //#ifdef ERROR\_SEED //<<Test\_Id:nftOpenWithRetry\_iErrNo\_Set\_To\_Null>>  287 : : // p = -1;  288 : : //#endif  289 [ - + ]: 12559 : if(-1 != p) {  290 : 12559 : return p;  291 : : } else {  292 : 0 : iErrNo = errno; //System error no  293 : : #ifdef TEST\_RETRY\_2  294 : : if( a < 5 ) {  295 : : iErrNo = EMFILE;  296 : : a++;  297 : : }  298 : : #elif TEST\_RETRY\_3  299 : : if( a < 5 ) {  300 : : iErrNo = ENFILE;  301 : : a++;  302 : : }  303 : : #elif TEST\_RETRY\_4  304 : : if( a < 5 ) {  305 : : iErrNo = ENOMEM;  306 : : a++;  307 : : }  308 : : #elif TEST\_RETRY\_5  309 : : if( a < 5 ) {  310 : : iErrNo = ENOSPC;  311 : : a++;  312 : : }  313 : : /\*#elif ERROR\_SEED //<<Test\_Id:nftOpenWithRetry\_iErrNo\_Set\_To\_ENOSPC>>  314 : : if( a < 5 ) {  315 : : iErrNo = ENOSPC;  316 : : a++;  317 : : }\*/  318 : : #endif  319 : : //#ifdef ERROR\_SEED //<<Test\_Id:nftOpenWithRetry\_iErrNo\_Set\_To\_Null>>  320 : : // iErrNo = NULL;  321 : : //#endif  322 [ # # ][ # # ]: 0 : if ( iErrNo != EEXIST && iErrNo != EMFILE  [ # # ]  323 : : && iErrNo != ENFILE && iErrNo != ENOMEM  324 : : && iErrNo != ENOSPC) {  325 : : //unrecoverable error  326 : 0 : char systemErrorMsg[BUFSIZ];  327 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  328 : 0 : pErr->errorAdd(NULL, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  329 : : ErrorHandler::eErrMemoryAllocation,  330 : : pErr->getErrorMsg(  331 : : ErrorHandler::eErrMemoryAllocation,  332 : : strerror\_r( iErrNo, systemErrorMsg, BUFSIZ )));  333 : 0 : return -1;  334 : : }  335 : : }  336 [ # # ]: 0 : } while (eAgain == retryAgent.waitForSpecifiedTime());  337 : 0 : char systemErrorMsg[BUFSIZ];  338 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  339 : 0 : pErr->errorAdd(NULL, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  340 : : ErrorHandler::eErrMemoryAllocation,  341 : : pErr->getErrorMsg(  342 : : ErrorHandler::eErrMemoryAllocation,  343 : : strerror\_r( iErrNo, systemErrorMsg, BUFSIZ )));  344 : :   345 [ # # ]: 0 : return -1;  346 : :   347 : : }  348 : :   349 : : int  350 : 0 : nftOpenWithRetry(const char \*pathname, int flags, mode\_t mode) throw(DPL::Error \*) {  351 : 0 : int iErrNo;  352 : 0 : RetryAgent retryAgent = RetryManager::instance()->getAgent();  353 : 0 : do {  354 : : // int p = open( pathname, flags, mode );  355 : : #ifdef TEST\_RETRY\_2  356 : : int p = -1;  357 : : static int a = 1;  358 : : if(a >= 5 )  359 : : {  360 : : p = open( pathname, flags, mode );  361 : : }  362 : : else  363 : : {  364 : : e9k\_debug("nftOpenWithRetry EMFILE\n");  365 : : }  366 : : #elif TEST\_RETRY\_3  367 : : int p = -1;  368 : : static int a = 1;  369 : : if(a >= 5 )  370 : : {  371 : : p = open( pathname, flags, mode );  372 : : }  373 : : else  374 : : {  375 : : e9k\_debug("nftOpenWithRetry ENFILE\n");  376 : : }  377 : :   378 : : #elif TEST\_RETRY\_4  379 : : int p = -1;  380 : : static int a = 1;  381 : : if(a >= 5 )  382 : : {  383 : : p = open( pathname, flags, mode );  384 : : }  385 : : else  386 : : {  387 : : e9k\_debug("nftOpenWithRetry ENOMEM\n");  388 : : }  389 : : #elif TEST\_RETRY\_5  390 : : int p = -1;  391 : : static int a = 1;  392 : : if(a >= 5 )  393 : : {  394 : : p = open( pathname, flags, mode );  395 : : }  396 : : else  397 : : {  398 : : e9k\_debug("nftOpenWithRetry ENOSPC\n");  399 : : }  400 : : #else  401 : 0 : int p = open( pathname, flags, mode );  402 : : #ifdef TEST\_RETRY\_1  403 : : e9k\_debug("TEST\_RETRY\_1 nftOpenWithRetry\n");  404 : : #endif  405 : :   406 : : #endif  407 [ # # ]: 0 : if(-1 != p) {  408 : 0 : return p;  409 : : } else {  410 : 0 : iErrNo = errno; //System error no  411 : : #ifdef TEST\_RETRY\_2  412 : : if( a < 5 ) {  413 : : iErrNo = EMFILE;  414 : : a++;  415 : : }  416 : : #elif TEST\_RETRY\_3  417 : : if( a < 5 ) {  418 : : iErrNo = ENFILE;  419 : : a++;  420 : : }  421 : : #elif TEST\_RETRY\_4  422 : : if( a < 5 ) {  423 : : iErrNo = ENOMEM;  424 : : a++;  425 : : }  426 : : #elif TEST\_RETRY\_5  427 : : if( a < 5 ) {  428 : : iErrNo = ENOSPC;  429 : : a++;  430 : : }  431 : : #endif  432 [ # # ][ # # ]: 0 : if ( iErrNo != EEXIST && iErrNo != EMFILE  [ # # ]  433 : : && iErrNo != ENFILE && iErrNo != ENOMEM  434 : : && iErrNo != ENOSPC) {  435 : : //unrecoverable error  436 : 0 : char systemErrorMsg[BUFSIZ];  437 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  438 : 0 : pErr->errorAdd(NULL, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  439 : : ErrorHandler::eErrMemoryAllocation,  440 : : pErr->getErrorMsg(  441 : : ErrorHandler::eErrMemoryAllocation,  442 : : strerror\_r( iErrNo, systemErrorMsg, BUFSIZ )));  443 : 0 : return -1;  444 : : }  445 : : }  446 [ # # ]: 0 : } while (eAgain == retryAgent.waitForSpecifiedTime());  447 : 0 : char systemErrorMsg[BUFSIZ];  448 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  449 : 0 : pErr->errorAdd(NULL, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  450 : : ErrorHandler::eErrMemoryAllocation,  451 : : pErr->getErrorMsg(  452 : : ErrorHandler::eErrMemoryAllocation,  453 : : strerror\_r( iErrNo, systemErrorMsg, BUFSIZ )));  454 : :   455 [ # # ]: 0 : return -1;  456 : :   457 : : }  458 : :   459 : : int  460 : 12559 : nftCloseWithRetry(int fd)throw(DPL::Error \*) {  461 : 12559 : int iErrNo;  462 : 12559 : RetryAgent retryAgent = RetryManager::instance()->getAgent();  463 : 12559 : do {  464 : : //int p = close( fd );  465 : : #ifdef TEST\_RETRY\_7  466 : : int p;  467 : : static int a = 1;  468 : : if(a < 4 ) {  469 : : p = -1;  470 : : e9k\_debug("nftCloseWithRetry EINTR\n");  471 : : } else {  472 : : p = close( fd );  473 : : }  474 : : #elif TEST\_RETRY\_8  475 : : int p;  476 : : static int a = 1;  477 : : if(a < 4 ) {  478 : : p = -1;  479 : : e9k\_debug("nftCloseWithRetry EIO\n");  480 : : } else {  481 : : p = close( fd );  482 : : }  483 : : /\*#elif ERROR\_SEED //<<Test\_Id:nftCloseWithRetry\_iErrNo\_Set\_To\_EIO>>  484 : : int p;  485 : : static int a = 1;  486 : : if(a < 4 ) {  487 : : p = -1;  488 : : e9k\_debug("nftCloseWithRetry EIO\n");  489 : : }\*/  490 : : #else  491 : 12559 : int p = close( fd );  492 : : #ifdef TEST\_RETRY\_6  493 : : e9k\_debug("TEST\_RETRY\_6 nftCloseWithRetry\n");  494 : : #endif  495 : : #endif  496 : : //#ifdef ERROR\_SEED //<<Test\_Id:nftCloseWithRetry\_iErrNo\_Set\_To\_Null>>  497 : : // p = 1;  498 : : //#endif  499 [ - + ]: 12559 : if(0 == p) {  500 : 12559 : return p;  501 : : } else {  502 : 0 : iErrNo = errno; //System error no  503 : : #ifdef TEST\_RETRY\_7  504 : : if( a < 4 ) {  505 : : iErrNo = EINTR;  506 : : a++;  507 : : }  508 : : #elif TEST\_RETRY\_8  509 : : if( a < 4 ) {  510 : : iErrNo = EIO;  511 : : a++;  512 : : }  513 : : /\*#elif ERROR\_SEED //<<Test\_Id:nftCloseWithRetry\_iErrNo\_Set\_To\_EIO>>  514 : : if( a < 4 ) {  515 : : iErrNo = EIO;  516 : : a++;  517 : : }\*/  518 : : #endif  519 : : //#ifdef ERROR\_SEED //<<Test\_Id:nftCloseWithRetry\_iErrNo\_Set\_To\_Null>>  520 : : // iErrNo = NULL;  521 : : //#endif  522 [ # # ]: 0 : if ( iErrNo != EINTR && iErrNo != EIO ) {  523 : : //unrecoverable error  524 : 0 : char systemErrorMsg[BUFSIZ];  525 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  526 : 0 : pErr->errorAdd(NULL, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  527 : : ErrorHandler::eErrMemoryAllocation,  528 : : pErr->getErrorMsg(  529 : : ErrorHandler::eErrMemoryAllocation,  530 : : strerror\_r( iErrNo, systemErrorMsg, BUFSIZ )));  531 : 0 : return -1;  532 : : }  533 : : }  534 [ # # ]: 0 : } while (eAgain == retryAgent.waitForSpecifiedTime());  535 : 0 : char systemErrorMsg[BUFSIZ];  536 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  537 : 0 : pErr->errorAdd(NULL, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  538 : : ErrorHandler::eErrMemoryAllocation,  539 : : pErr->getErrorMsg(  540 : : ErrorHandler::eErrMemoryAllocation,  541 : : strerror\_r( iErrNo, systemErrorMsg, BUFSIZ )));  542 : :   543 [ # # ]: 0 : return -1;  544 : : }  545 : :   546 : :   547 : : void \*  548 : 12559 : nftMmapWithRetry(void \*start, size\_t length, int prot, int flags, int fd, off\_t offset) throw(DPL::Error \*) {  549 : 12559 : int iErrNo;  550 : 12559 : RetryAgent retryAgent = RetryManager::instance()->getAgent();  551 : 12559 : do {  552 : : //void \*p = mmap( start, length, prot, flags, fd, offset );  553 : : #ifdef TEST\_RETRY\_22  554 : : void \*p = MAP\_FAILED;  555 : : static int a = 1;  556 : : if(a >= 5 )  557 : : {  558 : : p = mmap( start, length, prot, flags, fd, offset );  559 : : }  560 : : else  561 : : {  562 : : e9k\_debug("nftMmapWithRetry EAGAIN\n");  563 : : }  564 : : #elif TEST\_RETRY\_23  565 : : void \*p = MAP\_FAILED;  566 : : static int a = 1;  567 : : if(a >= 5 )  568 : : {  569 : : p = mmap( start, length, prot, flags, fd, offset );  570 : : }  571 : : else  572 : : {  573 : : e9k\_debug("nftMmapWithRetry ENOMEM\n");  574 : : }  575 : : #elif TEST\_RETRY\_44  576 : : void \*p = MAP\_FAILED;  577 : : static int a = 1;  578 : : if(a >= 8 )  579 : : {  580 : : p = mmap( start, length, prot, flags, fd, offset );  581 : : }  582 : : else  583 : : {  584 : : e9k\_debug("nftMmapWithRetry TEST\_RETRY\_44\n");  585 : : }  586 : :   587 : : #else  588 : 12559 : void \*p = mmap( start, length, prot, flags, fd, offset );  589 : : #ifdef TEST\_RETRY\_21  590 : : e9k\_debug("TEST\_RETRY\_21 nftMmapWithRetry\n");  591 : : #endif  592 : : #endif  593 : : //#ifdef ERROR\_SEED //<<Test\_Id:nftMmapWithRetry\_p\_Set\_To\_MAP\_FAILED>>  594 : : // p = MAP\_FAILED;  595 : : //#endif  596 [ - + ]: 12559 : if(MAP\_FAILED != p) {  597 : 12559 : return p;  598 : : } else {  599 : 0 : iErrNo = errno; //System error no  600 : : #ifdef TEST\_RETRY\_22  601 : : if( a < 5 ) {  602 : : iErrNo = EAGAIN;  603 : : a++;  604 : : }  605 : : #elif TEST\_RETRY\_23  606 : : if( a < 5 ) {  607 : : iErrNo = ENOMEM;  608 : : a++;  609 : : }  610 : : #elif TEST\_RETRY\_44  611 : : if( a < 8 ) {  612 : : iErrNo = ENOMEM;  613 : : a++;  614 : : }  615 : : #endif  616 : : //#ifdef ERROR\_SEED //<<Test\_Id:nftMmapWithRetry\_p\_Set\_To\_MAP\_FAILED>>  617 : : // iErrNo = NULL;  618 : : //#endif  619 [ # # ]: 0 : if ( iErrNo != EAGAIN && iErrNo != ENOMEM ) {  620 : : //unrecoverable error  621 : 0 : char systemErrorMsg[BUFSIZ];  622 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  623 : 0 : pErr->errorAdd(NULL, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  624 : : ErrorHandler::eErrMemoryAllocation,  625 : : pErr->getErrorMsg(  626 : : ErrorHandler::eErrMemoryAllocation,  627 : : strerror\_r( iErrNo, systemErrorMsg, BUFSIZ )));  628 : 0 : return MAP\_FAILED;  629 : : }  630 : : }  631 [ # # ]: 0 : } while (eAgain == retryAgent.waitForSpecifiedTime());  632 : 0 : char systemErrorMsg[BUFSIZ];  633 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  634 : 0 : pErr->errorAdd(NULL, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  635 : : ErrorHandler::eErrMemoryAllocation,  636 : : pErr->getErrorMsg(  637 : : ErrorHandler::eErrMemoryAllocation,  638 : : strerror\_r( iErrNo, systemErrorMsg, BUFSIZ )));  639 : :   640 [ # # ]: 0 : return MAP\_FAILED;  641 : : }  642 : :   643 : : int  644 : 12559 : nftMunmapWithRetry(void \*start, size\_t length) throw(DPL::Error \*) {  645 : 12559 : int iErrNo;  646 : 12559 : RetryAgent retryAgent = RetryManager::instance()->getAgent();  647 : 12559 : do {  648 : 12559 : int p = munmap( start, length );  649 : : //#ifdef ERROR\_SEED //<<Test\_Id:n/ftMunmapWithRetry\_errno\_Set\_To\_EAGAIN>>  650 : : // p = -1;  651 : : // errno = EAGAIN;  652 : : //#endif  653 : : //#ifdef ERROR\_SEED //<<Test\_Id:nftMunmapWithRetry\_errno\_Set\_To\_Null>>  654 : : // p = -1;  655 : : // errno = NULL;  656 : : //#endif  657 [ - + ]: 12559 : if(-1 != p) {  658 : 12559 : return p;  659 : : } else {  660 : 0 : iErrNo = errno; //System error no  661 [ # # ]: 0 : if ( errno != EAGAIN && errno != ENOMEM ) {  662 : : //unrecoverable error  663 : 0 : char systemErrorMsg[BUFSIZ];  664 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  665 : 0 : pErr->errorAdd(NULL, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  666 : : ErrorHandler::eErrMemoryAllocation,  667 : : pErr->getErrorMsg(  668 : : ErrorHandler::eErrMemoryAllocation,  669 : : strerror\_r( iErrNo, systemErrorMsg, BUFSIZ )));  670 : 0 : return -1;  671 : : }  672 : : }  673 [ # # ]: 0 : } while (eAgain == retryAgent.waitForSpecifiedTime());  674 : 0 : char systemErrorMsg[BUFSIZ];  675 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  676 : 0 : pErr->errorAdd(NULL, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  677 : : ErrorHandler::eErrMemoryAllocation,  678 : : pErr->getErrorMsg(  679 : : ErrorHandler::eErrMemoryAllocation,  680 : : strerror\_r( iErrNo, systemErrorMsg, BUFSIZ )));  681 : :   682 [ # # ]: 0 : return -1;  683 : :   684 : : }  685 : :   686 : : int  687 : 0 : nftFtruncateWithRetry(int fd, off\_t length) throw(DPL::Error \*) {  688 : 0 : e9k\_debug("nftFtruncateWithRetry called\n");  689 : 0 : int iErrNo;  690 : 0 : RetryAgent retryAgent = RetryManager::instance()->getAgent();  691 : 0 : do {  692 : : // int p = ftruncate( fd, length );  693 : : #ifdef TEST\_RETRY\_26  694 : : int p = -1;  695 : : static int a = 1;  696 : : if(a >= 5 )  697 : : {  698 : : p = ftruncate( fd, length );  699 : : }  700 : : else  701 : : {  702 : : e9k\_debug("nftFtruncateWithRetry EIO\n");  703 : : }  704 : : #elif TEST\_RETRY\_27  705 : : int p = -1;  706 : : static int a = 1;  707 : : if(a >= 5 )  708 : : {  709 : : p = ftruncate( fd, length );  710 : : }  711 : : else  712 : : {  713 : : e9k\_debug("nftFtruncateWithRetry ENOMEM\n");  714 : : }  715 : : #else  716 : 0 : int p = ftruncate( fd, length );  717 : : #ifdef TEST\_RETRY\_25  718 : : e9k\_debug("TEST\_RETRY\_25 nftFtruncateWithRetry\n");  719 : : #endif  720 : : #endif  721 : :   722 [ # # ]: 0 : if(-1 != p) {  723 : 0 : return p;  724 : : } else {  725 : 0 : iErrNo = errno; //System error no  726 : : #ifdef TEST\_RETRY\_26  727 : : if( a < 5 ) {  728 : : iErrNo = EIO;  729 : : a++;  730 : : }  731 : : #elif TEST\_RETRY\_27  732 : : if( a < 5 ) {  733 : : iErrNo = ENOMEM;  734 : : a++;  735 : : }  736 : : #endif  737 : :   738 [ # # ]: 0 : if ( iErrNo != EIO && iErrNo != ENOMEM ) {  739 : : //unrecoverable error  740 : 0 : char systemErrorMsg[BUFSIZ];  741 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  742 : 0 : pErr->errorAdd(NULL, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  743 : : ErrorHandler::eErrMemoryAllocation,  744 : : pErr->getErrorMsg(  745 : : ErrorHandler::eErrMemoryAllocation,  746 : : strerror\_r( iErrNo, systemErrorMsg, BUFSIZ )));  747 : 0 : return -1;  748 : : }  749 : : }  750 [ # # ]: 0 : } while (eAgain == retryAgent.waitForSpecifiedTime());  751 : 0 : char systemErrorMsg[BUFSIZ];  752 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  753 : 0 : pErr->errorAdd(NULL, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  754 : : ErrorHandler::eErrMemoryAllocation,  755 : : pErr->getErrorMsg(  756 : : ErrorHandler::eErrMemoryAllocation,  757 : : strerror\_r( iErrNo, systemErrorMsg, BUFSIZ )));  758 : :   759 [ # # ]: 0 : return -1;  760 : : }  761 : :   762 : : int  763 : 0 : nftShmgetWithRetry(key\_t key, size\_t size, int shmflg) throw(DPL::Error \*) {  764 : 0 : int iErrNo;  765 : 0 : RetryAgent retryAgent = RetryManager::instance()->getAgent();  766 : 0 : do {  767 : : //int p = shmget( key, size, shmflg );  768 : : #ifdef TEST\_RETRY\_29  769 : : int p = -1;  770 : : static int a = 1;  771 : : if(a >= 5 )  772 : : {  773 : : p = shmget( key, size, shmflg );  774 : : }  775 : : else  776 : : {  777 : : e9k\_debug("nftShmgetWithRetry ENOSPC\n");  778 : : }  779 : : #elif TEST\_RETRY\_30  780 : : int p = -1;  781 : : static int a = 1;  782 : : if(a >= 5 )  783 : : {  784 : : p = shmget( key, size, shmflg );  785 : : }  786 : : else  787 : : {  788 : : e9k\_debug("nftShmgetWithRetry ENOMEM\n");  789 : : }  790 : : /\*#elif ERROR\_SEED //<<Test\_Id:nftShmgetWithRetry\_iErrNo\_Set\_To\_ENOMEM>>  791 : : int p = -1;  792 : : static int a = 1;  793 : : if(a >= 5 )  794 : : {  795 : : p = shmget( key, size, shmflg );  796 : : }  797 : : else  798 : : {  799 : : e9k\_debug("nftShmgetWithRetry ENOMEM\n");  800 : : }\*/  801 : : #else  802 : 0 : int p = shmget( key, size, shmflg );  803 : : #ifdef SHOT\_READ\_SMW  804 : : p = -1;  805 : : #endif   806 : :   807 : : #ifdef TEST\_RETRY\_28  808 : : e9k\_debug("TEST\_RETRY\_28 nftShmgetWithRetry\n");  809 : : #endif  810 : : #endif  811 [ # # ]: 0 : if(-1 != p) {  812 : 0 : return p;  813 : : } else {  814 : 0 : iErrNo = errno; //System error no  815 : : #ifdef TEST\_RETRY\_29  816 : : if( a < 5 ) {  817 : : iErrNo = ENOSPC;  818 : : a++;  819 : : }  820 : : #elif TEST\_RETRY\_30  821 : : if( a < 5 ) {  822 : : iErrNo = ENOMEM;  823 : : a++;  824 : : }  825 : : /\*#elif ERROR\_SEED //<<Test\_Id:nftShmgetWithRetry\_iErrNo\_Set\_To\_ENOMEM>>  826 : : if( a < 5 ) {  827 : : iErrNo = ENOMEM;  828 : : a++;  829 : : }\*/  830 : : #endif  831 [ # # ]: 0 : if ( iErrNo != ENOSPC && iErrNo != ENOMEM ) {  832 : : //unrecoverable error  833 : 0 : char systemErrorMsg[BUFSIZ];  834 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  835 : 0 : pErr->errorAdd(NULL, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  836 : : ErrorHandler::eErrMemoryAllocation,  837 : : pErr->getErrorMsg(  838 : : ErrorHandler::eErrMemoryAllocation,  839 : : strerror\_r( iErrNo, systemErrorMsg, BUFSIZ )));  840 : 0 : return -1;  841 : : }  842 : : }  843 [ # # ]: 0 : } while (eAgain == retryAgent.waitForSpecifiedTime());  844 : 0 : char systemErrorMsg[BUFSIZ];  845 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  846 : 0 : pErr->errorAdd(NULL, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  847 : : ErrorHandler::eErrMemoryAllocation,  848 : : pErr->getErrorMsg(  849 : : ErrorHandler::eErrMemoryAllocation,  850 : : strerror\_r( iErrNo, systemErrorMsg, BUFSIZ )));  851 : :   852 [ # # ]: 0 : return -1;  853 : : }  854 : :   855 : : void \*  856 : 0 : nftShmatWithRetry(int shmid, const void \*shmaddr, int shmflg) throw(DPL::Error \*) {  857 : 0 : int iErrNo;  858 : 0 : RetryAgent retryAgent = RetryManager::instance()->getAgent();  859 : 0 : do {  860 : : //void \*p = shmat( shmid, shmaddr, shmflg );  861 : : #ifdef TEST\_RETRY\_32  862 : : void \*p = (void\*)-1;  863 : : static int a = 1;  864 : : if(a >= 5 )  865 : : {  866 : : p = shmat( shmid, shmaddr, shmflg );  867 : : }  868 : : else  869 : : {  870 : : e9k\_debug("nftShmatWithRetry ENOMEM\n");  871 : : }  872 : : /\*#elif ERROR\_SEED //<<Test\_Id:nftShmatWithRetry\_iErrNo\_Set\_To\_ENOMEM>>  873 : : void \*p = (void\*)-1;  874 : : static int a = 1;  875 : : if(a >= 5 )  876 : : {  877 : : p = shmat( shmid, shmaddr, shmflg );  878 : : }  879 : : else  880 : : {  881 : : e9k\_debug("nftShmatWithRetry ENOMEM\n");  882 : : }\*/  883 : : #else  884 : 0 : void \*p = shmat( shmid, shmaddr, shmflg );  885 : : #ifdef TEST\_RETRY\_31  886 : : e9k\_debug("TEST\_RETRY\_31 nftShmatWithRetry\n");  887 : : #endif  888 : : #endif  889 [ # # ]: 0 : if((void \*) -1 != p) {  890 : 0 : return p;  891 : : } else {  892 : 0 : iErrNo = errno; //System error no  893 : : #ifdef TEST\_RETRY\_32  894 : : if( a < 5 ) {  895 : : iErrNo = ENOMEM;  896 : : a++;  897 : : }  898 : : /\*#elif ERROR\_SEED //<<Test\_Id:nftShmatWithRetry\_iErrNo\_Set\_To\_ENOMEM>>  899 : : if( a < 5 ) {  900 : : iErrNo = ENOMEM;  901 : : a++;  902 : : }\*/  903 : : #endif  904 [ # # ]: 0 : if ( iErrNo != ENOMEM ) {  905 : : //unrecoverable error  906 : 0 : char systemErrorMsg[BUFSIZ];  907 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  908 : 0 : pErr->errorAdd(NULL, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  909 : : ErrorHandler::eErrMemoryAllocation,  910 : : pErr->getErrorMsg(  911 : : ErrorHandler::eErrMemoryAllocation,  912 : : strerror\_r( iErrNo, systemErrorMsg, BUFSIZ )));  913 : 0 : return (void \*) -1;  914 : : }  915 : : }  916 [ # # ]: 0 : } while (eAgain == retryAgent.waitForSpecifiedTime());  917 : 0 : char systemErrorMsg[BUFSIZ];  918 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  919 : 0 : pErr->errorAdd(NULL, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  920 : : ErrorHandler::eErrMemoryAllocation,  921 : : pErr->getErrorMsg(  922 : : ErrorHandler::eErrMemoryAllocation,  923 : : strerror\_r( iErrNo, systemErrorMsg, BUFSIZ )));  924 : :   925 [ # # ]: 0 : return (void \*) -1;  926 : :   927 : : }  928 : :   929 : 1215 : } //nft namespace |

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