Contents

[Getcwd 1](#_Toc40431434)

[Nftw 2](#_Toc40431435)

[openClose 3](#_Toc40431436)

[Pathconf 5](#_Toc40431437)

[Readdir 6](#_Toc40431438)

[Rename 7](#_Toc40431439)

[Stat 8](#_Toc40431440)

[Statcfs 9](#_Toc40431441)

[Statvfs 10](#_Toc40431442)

[Aio 11](#_Toc40431443)

[Frw 14](#_Toc40431444)

[Mmap 16](#_Toc40431445)

[Rw 20](#_Toc40431446)

[Lseek 21](#_Toc40431447)

# /var/lab/files/submit kd3.c

# Getcwd

/\* Arnas Švenčionis IFF-8/11 arnsve2 \*/

/\* Failas: arnsve2\_getcwd01.c \*/

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

int as\_test\_getcwd();

int as\_test\_open();

int as\_test\_chdir();

int as\_test\_fchdir(int dskr);

int as\_test\_fchdir(int dskr){

fchdir(dskr);

as\_test\_getcwd();

return 1;

}

int as\_test\_chdir(){

chdir("/tmp");

as\_test\_getcwd();

return 1;

}

int as\_test\_open(){

int dskr;

dskr = open( ".", O\_RDONLY );

printf( "Deskriptorius: %d\n", dskr );

return dskr;

}

int as\_test\_getcwd(){

char \*cwd;

cwd = getcwd(NULL, pathconf( ".", \_PC\_PATH\_MAX) );

puts( cwd );

free( cwd );

return 1;

}

int main( int argc, char \* argv[] ){

printf( "(C) 2020 Arnas Švenčionis, %s\n", \_\_FILE\_\_ );

as\_test\_getcwd(argv[1]);

int dskr;

dskr = as\_test\_open();

as\_test\_chdir();

as\_test\_fchdir(dskr);

return 0;

}

# Nftw

#include <sys/stat.h>

#include <stdio.h>

#include <stdlib.h>

#include <sys/statvfs.h>

#include <ftw.h>

int as\_test\_nftw(const char \*fpath, const struct stat \*sb,

int tflag, struct FTW \*ftwbuf);

int as\_test\_display\_info(const char \*fpath, const struct stat \*sb,

int tflag, struct FTW \*ftwbuf){

printf("%s\n", fpath);

return 0;

}

int main( int argc, char \*argv[] ){

printf( "(C) 2020 Arnas Švenčionis, %s\n", \_\_FILE\_\_ );

nftw("/home/arnsve2", as\_test\_display\_info, 20, FTW\_PHYS);

return 0;

}

# openClose

/\* Kęstutis Paulikas KTK kespaul \*/

/\* Failas: kespaul\_open01.c \*/

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

int kp\_test\_open(const char \*name);

int kp\_test\_close(int fd);

int kp\_test\_open(const char \*name){

int dskr;

dskr = open( name, O\_RDONLY );

if( dskr == -1 ){

perror( name );

exit(1);

}

printf( "dskr = %d\n", dskr );

return dskr;

}

int kp\_test\_close(int fd){

int rv;

rv = close( fd );

if( rv != 0 ) perror ( "close() failed" );

else puts( "closed" );

return rv;

}

int main( int argc, char \*argv[] ){

printf( "(C) 2020 Arnas Švenčionis, %s\n", \_\_FILE\_\_ );

int d;

if( argc != 2 ){

printf( "Naudojimas:\n %s failas\_ar\_katalogas\n", argv[0] );

exit( 255 );

}

d = kp\_test\_open( argv[1] );

kp\_test\_close( d );

kp\_test\_close( d ); /\* turi mesti close klaida \*/

return 0;

}

# Pathconf

/\* Arnas Švenčionis arnsve2 \*/

/\* Failas: loginas\_sablonas.c \*/

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

int arnsve2\_pathconf(const char \*name);

int arnsve2\_pathconf(const char \*name){

long namemax;

namemax = pathconf(name, \_PC\_NAME\_MAX);

printf( "NAME\_MAX = %ld\n", namemax );

long pathmax;

pathmax = pathconf(name, \_PC\_PATH\_MAX);

printf( "Path max = %ld\n", pathmax );

return 0;

}

int main( int argc, char \* argv[] ){

if(argc != 2)

{

printf( "Naudojimas:\n %s failas\_ar\_katalogas\n", argv[0] );

exit(255);

}

arnsve2\_pathconf(argv[1]);

return 0;

}

# Readdir

/\* Arnas Švenčionis arnsve2 \*/

/\* Failas: loginas\_sablonas.c \*/

#include <dirent.h>

#include <stdio.h>

#include<stdlib.h>

#include <unistd.h>

#include <fcntl.h>

DIR \*as\_test\_opendir();

DIR \*as\_test\_opendir(){

DIR \*dir;

struct dirent \*dp;

if((dir = opendir(".")) == NULL){

printf("Katalogo negalima atidaryti");

exit(1);

}

while((dp = readdir(dir)) != NULL){

printf("i-node: %ld, vardas: %s \n",dp->d\_ino, dp->d\_name);

}

closedir(dir);

return 0;

}

int main( int argc, char \* argv[] ){

printf( "(C) 2020 Arnas Švenčionis, %s\n", \_\_FILE\_\_ );

//readdir(opendir("."));

//DIR dir;

//dir = as\_test\_opendir();

as\_test\_opendir();

return 0;

}

# Rename

#include <sys/stat.h>

#include <stdio.h>

#include <stdlib.h>

#include <sys/statvfs.h>

#include <ftw.h>

int as\_test\_rename(const char \*name);

int as\_test\_rename(const char \*name){

rename(name, "renamed");

return 0;

}

int main( int argc, char \*argv[] ){

printf( "(C) 2020 Arnas Švenčionis, %s\n", \_\_FILE\_\_ );

if( argc != 2 ){

printf( "Naudojimas:\n %s failas\_ar\_katalogas\n", argv[0] );

exit( 255 );

}

as\_test\_rename(argv[1]);

return 0;

}

# Stat

#include <sys/stat.h>

#include <stdio.h>

#include <stdlib.h>

int as\_test\_stat(const char \*name);

int as\_test\_stat(const char \*name){

struct stat statbuf;

stat( name, &statbuf );

printf("Savininko id:%d\n", statbuf.st\_uid);

printf("Dydis: %ld\n", statbuf.st\_size);

printf("I-node numeris: %ld\n", statbuf.st\_ino);

printf("Leidimai: %d\n", statbuf.st\_mode);

//printf("Failo tipas: %d\n", S\_IFMT(statbuf.st\_mode)

return 0;

}

int main( int argc, char \*argv[] ){

printf( "(C) 2020 Arnas Švenčionis, %s\n", \_\_FILE\_\_ );

if( argc != 2 ){

printf( "Naudojimas:\n %s failas\_ar\_katalogas\n", argv[0] );

exit( 255 );

}

as\_test\_stat(argv[1]);

return 0;

}

# Statcfs

#include <sys/stat.h>

#include <stdio.h>

#include <stdlib.h>

int as\_test\_stat(const char \*name);

int as\_test\_stat(const char \*name){

struct stat statbuf;

stat( name, &statbuf );

printf("Savininko id:%d\n", statbuf.st\_uid);

printf("Dydis: %ld\n", statbuf.st\_size);

printf("I-node numeris: %ld\n", statbuf.st\_ino);

printf("Leidimai: %d\n", statbuf.st\_mode);

//printf("Failo tipas: %d\n", S\_IFMT(statbuf.st\_mode)

return 0;

}

int main( int argc, char \*argv[] ){

printf( "(C) 2020 Arnas Švenčionis, %s\n", \_\_FILE\_\_ );

if( argc != 2 ){

printf( "Naudojimas:\n %s failas\_ar\_katalogas\n", argv[0] );

exit( 255 );

}

as\_test\_stat(argv[1]);

return 0;

}

# Statvfs

#include <sys/stat.h>

#include <stdio.h>

#include <stdlib.h>

#include <sys/statvfs.h>

int as\_test\_stat(const char \*name);

int as\_test\_statvfs(const char \*name);

int as\_test\_statvfs(const char \*name){

struct statvfs buffer;

statvfs( name, &buffer );

printf("statvfs:\n");

printf("Failų sistemos bloko dydis: %ld\n", buffer.f\_bsize);

printf("Failų sistemos ID: %ld\n", buffer.f\_fsid);

printf("Failų sistemos dydis: %ld\n", buffer.f\_frsize);

printf("Maksimals failo kelio/vardo ilgis: %ld\n", buffer.f\_namemax);

return 0;

}

int as\_test\_stat(const char \*name){

struct stat statbuf;

stat( name, &statbuf );

printf("stat:\n");

printf("Savininko id:%d\n", statbuf.st\_uid);

printf("Dydis: %ld\n", statbuf.st\_size);

printf("I-node numeris: %ld\n", statbuf.st\_ino);

printf("Leidimai: %d\n", statbuf.st\_mode);

//printf("Failo tipas: %d\n", S\_IFMT(statbuf.st\_mode)

return 0;

}

int main( int argc, char \*argv[] ){

printf( "(C) 2020 Arnas Švenčionis, %s\n", \_\_FILE\_\_ );

if( argc != 2 ){

printf( "Naudojimas:\n %s failas\_ar\_katalogas\n", argv[0] );

exit( 255 );

}

as\_test\_stat(argv[1]);

as\_test\_statvfs(argv[1]);

return 0;

}

# Aio

/\* Arnas Švenčionis IFF-8/11 arnsve2 \*/

/\* Failas: arnsve2\_aio02.c \*/

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <string.h>

#include <aio.h>

#define BUFFLEN 1048576

int kp\_test\_open(const char \*name);

int kp\_test\_close(int fd);

int kp\_test\_aio\_read\_start( const int d, struct aiocb \*aiorp, void \*buf, const int count );

int kp\_test\_dummy( const void \*data, int n );

int kp\_test\_aio\_read\_waitcomplete( struct aiocb \*aiorp );

int kp\_test\_open(const char \*name){

int dskr;

dskr = open( name, O\_RDONLY );

if( dskr == -1 ){

perror( name );

exit(1);

}

printf( "dskr = %d\n", dskr );

return dskr;

}

int kp\_test\_close(int fd){

int rv;

rv = close( fd );

if( rv != 0 ) perror ( "close() failed" );

else puts( "closed" );

return rv;

}

int kp\_test\_aio\_read\_start( const int d, struct aiocb \*aiorp, void \*buf, const int count ){

int rv = 0;

memset( (void \*)aiorp, 0, sizeof( struct aiocb ) );

aiorp->aio\_fildes = d;

aiorp->aio\_buf = buf;

aiorp->aio\_nbytes = count;

aiorp->aio\_offset = 0;

rv = aio\_read( aiorp );

if( rv != 0 ){

perror( "aio\_read failed" );

abort();

}

return rv;

}

int kp\_test\_dummy( const void \*data, int n ){

int i, cnt = 0;

for( i = 0; i < n; i++ )

if( ((char\*)data)[i] == '\0' ) cnt++;

printf( "Number of '0' in data: %d\n", cnt );

return 1;

}

int kp\_test\_aio\_read\_waitcomplete( struct aiocb \*aiorp ){

const struct aiocb \*aioptr[1];

int rv;

aioptr[0] = aiorp;

rv = aio\_suspend( aioptr, 1, NULL );

if( rv != 0 ){

perror( "aio\_suspend failed" );

abort();

}

rv = aio\_return( aiorp );

printf( "AIO complete, %d bytes read.\n", rv );

return 1;

}

int main( int argc, char \* argv[] ){

int d;

struct aiocb aior;

char buffer[BUFFLEN];

printf( "(C) 2020 Arnas Švenčionis, %s\n", \_\_FILE\_\_ );

//if( argc != 2 ){

// printf( "Naudojimas:\n %s sk\n", argv[0] );

// exit( 255 );

//}

d = kp\_test\_open( "/dev/urandom" );

kp\_test\_aio\_read\_start( d, &aior, buffer, sizeof(buffer) );

kp\_test\_dummy( buffer, sizeof(buffer) );

kp\_test\_aio\_read\_waitcomplete( &aior );

kp\_test\_close( d );

kp\_test\_dummy( buffer, sizeof(buffer) );

return 0;

}

# Frw

/\* Arnas Švenčionis IFF-8/11 arnsve2 \*/

/\* Failas: arnsve2\_frw01.c \*/

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

int kp\_test\_open(const char \*readff,const char \*writef, int baitai);

int kp\_test\_open(const char \*readff,const char \*writef, int baitai){

FILE \*fp;

if ((fp = fopen(readff, "r")) == NULL){

return 1;

}

FILE \*fp2;

if ((fp2 = fopen(writef, "w")) == NULL){

return 1;

}

char buf[baitai];

fread( buf, sizeof(buf), 1, fp );

fwrite( buf, sizeof(buf), 1, fp2 );

fclose(fp);

fclose(fp2);

return 0;

}

int main( int argc, char \*argv[] ){

printf( "(C) 2020 Arnas Švenčionis, %s\n", \_\_FILE\_\_ );

if( argc != 4 ){

printf( "Naudojimas:\n %s failas\_ar\_katalogas failas\_ar\_katalogas2 sk\n", argv[0] );

exit( 255 );

}

kp\_test\_open( argv[1], argv[2], atoi( argv[3] ) );

//kp\_test\_close( d );

//kp\_test\_close( d ); /\* turi mesti close klaida \*/

return 0;

}

# Mmap

#include <stdio.h>

#include <stdlib.h>

#include <sys/mman.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/time.h>

#include <string.h>

int md\_open(char \*p);

int md\_openw(const char \*name);

int md\_close(int fd);

void\* md\_mmapr( int d, int size );

void\* md\_mmapw( int d, int size );

int md\_munamp( void \*a, int size );

int md\_usemaped( void \*a, int size );

int md\_open(char \*p)

{

int dskr;

dskr = open(p, O\_RDONLY);

if(dskr == -1)

{

perror(p);

exit(1);

}

printf("Deskriptorius = %d\n", dskr);

return dskr;

}

int md\_openw(const char \*name){

int dskr;

dskr = open( name, O\_RDWR | O\_CREAT, 0640 );

if( dskr == -1 ){

perror( name );

exit( 255 );

}

printf( "dskr = %d\n", dskr );

return dskr;

}

int md\_close(int fd){

int rv;

rv = close( fd );

if( rv != 0 ) perror ( "close() failed" );

else puts( "closed" );

return rv;

}

void\* md\_mmapr( int d, int size ){

void \*a = NULL;

a = mmap( NULL, size, PROT\_READ, MAP\_SHARED, d, 0 );

if( a == MAP\_FAILED ){

perror( "mmap failed" );

abort();

}

return a;

}

void\* md\_mmapw( int d, int size ){

void \*a = NULL;

a = mmap( NULL, size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, d, 0 );

if( a == MAP\_FAILED ){

perror( "mmap failed" );

abort();

}

return a;

}

int md\_munamp( void \*a, int size ){

int rv;

rv = munmap( a, size );

if( rv != 0 ){

puts( "munmap failed" );

abort();

}

return 1;

}

int main( int argc, char \* argv[] ){

int d;

int d2;

void \*a = NULL;

void \*b = NULL;

printf( "(C) 2020 Arnas Švenčionis, %s\n", \_\_FILE\_\_ );

if( argc != 3 ){

printf( "Naudojimas:\n %s failas failas2\n", argv[0] );

exit( 255 );

}

d = md\_open( argv[1] );

d2 = md\_openw(argv[2]);

struct stat buffer;

fstat(d, &buffer);

int size = buffer.st\_size;

a = md\_mmapr( d, size );

b = md\_mmapw(d2,size);

ftruncate(d2, size);

memcpy(b, a, size);

md\_munamp(a,size);

md\_munamp(b,size);

md\_close(d);

md\_close(d2);

return 0;

}

# Rw

/\* Kęstutis Paulikas KTK kespaul \*/

/\* Failas: kespaul\_open01.c \*/

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

int kp\_test\_open(const char \*readff,const char \*writef, int baitai);

int kp\_test\_open(const char \*readff,const char \*writef, int baitai){

int dskr;

dskr = open( readff, O\_RDONLY );

if( dskr == -1 ){

perror( readff );

exit(1);

}

//printf( "dskr = %d\n", dskr );

//return dskr;

int dskr2;

dskr2 = open( writef, O\_WRONLY | O\_CREAT );

if( dskr2 == -1 ){

perror( writef );

exit(1);

}

char buf[baitai];

read(dskr, buf, baitai);

write(dskr2, buf, baitai);

return 0;

}

int main( int argc, char \*argv[] ){

printf( "(C) 2020 Arnas Švenčionis, %s\n", \_\_FILE\_\_ );

if( argc != 4 ){

printf( "Naudojimas:\n %s failas\_ar\_katalogas failas\_ar\_katalogas2 sk\n", argv[0] );

exit( 255 );

}

kp\_test\_open( argv[1], argv[2], atoi( argv[3] ) );

//kp\_test\_close( d );

//kp\_test\_close( d ); /\* turi mesti close klaida \*/

return 0;

}

# Lseek

/\* Arnas Švenčionis IFF-8/11 arnsveč \*/

/\* Failas: arnsve2\_seek01.c \*/

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

int kp\_test\_open();

int kp\_test\_open(){

int dskr;

const char \*readff = "sukurtas";

dskr = open( readff, O\_RDWR | O\_CREAT);

if( dskr == -1 ){

perror( readff );

exit(1);

}

//char\* buf = "123456789012123123123";

lseek( dskr, 1048576, SEEK\_SET );

write( dskr, "1", 1 );

int rv;

rv = close( dskr );

if( rv != 0 ) perror ( "close() failed" );

return rv;

}

int main( int argc, char \*argv[] ){

printf( "(C) 2020 Arnas Švenčionis, %s\n", \_\_FILE\_\_ );

kp\_test\_open();

return 0;

}