# LAB2/4

## Pathconf()

* Sukurkite programą **loginas\_pathconf.c** kuri *pathconf()* funkcijos pagalba sužinotų OS parametrus:
  + Maksimalų failo vardo ilgį ( \_PC\_NAME\_MAX ): ?
  + Maksimalų kelio ilgį ( \_PC\_PATH\_MAX ): ?

|  |
| --- |
| /\* Mildaras Karvelis milkar3 \*/  /\* Failas: milkar3\_sablonas.c \*/  #include <stdio.h>  #include <unistd.h>  int main() {  printf( "(C) 2023 Mildaras Karvelis, %s\n", \_\_FILE\_\_ );  long name\_max = pathconf("/", \_PC\_NAME\_MAX);  long path\_max = pathconf("/", \_PC\_PATH\_MAX);  if (name\_max != -1)  printf("Maksimalus failo vardo ilgis: %ld\n", name\_max);  else  printf("Nepavyko gauti maksimalaus failo vardo ilgio informacijos.\n");  if (path\_max != -1)  printf("Maksimalus kelio ilgis: %ld\n", path\_max);  else  printf("Nepavyko gauti maksimalaus kelio ilgio informacijos.\n");  return 0;  } |

## Getcwd();

* Sukurkite C programą **loginas\_getcwd02.c** (loginas pakeiskite į savo loginą), kuri:
  1. gautų ir atspausdintų einamo katalogo vardą su getcwd() (ir kviečiant getcwd() naudotų pathconf(".",\_PC\_PATH\_MAX) grąžinamą reikšmę)
  2. atidarytų einamą katalogą su open(), įsimintų ir atspausdintų jo deskriptorių
  3. nueitų į /tmp katalogą su chdir()
  4. patikrintų su getcwd() ir atspausdintų koks dabar yra einamasis katalogas
  5. grįžtų į 2-ame žingsnyje atidarytą katalogą su fchdir() (ir patikrintų/parodytų, kad tikrai grįžo)

|  |
| --- |
| /\* Mildaras Karvelis milkar3 \*/  /\* Failas: milkar3\_sablonas.c \*/  #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <fcntl.h>  #include <limits.h>  int main() {  printf( "(C) 2023 Mildaras Karvelis, %s\n", \_\_FILE\_\_ );  char current\_dir[PATH\_MAX];  long path\_max = pathconf(".", \_PC\_PATH\_MAX);  // Gauname einamojo katalogo vardą  if (getcwd(current\_dir, path\_max) != NULL)  printf("Einamasis katalogas: %s\n", current\_dir);  else {  perror("Klaida gaudant einamojo katalogo vardą");  return EXIT\_FAILURE;  }  // Atidarome einamąjį katalogą  int dir\_fd = open(current\_dir, O\_RDONLY);  if (dir\_fd == -1) {  perror("Klaida atidarant einamąjį katalogą");  return EXIT\_FAILURE;  }  // Spausdiname einamojo katalogo deskriptorių  printf("Einamojo katalogo deskriptorius: %d\n", dir\_fd);  // Pereiname į /tmp katalogą  if (chdir("/tmp") == -1) {  perror("Klaida keičiant katalogą į /tmp");  return EXIT\_FAILURE;  }  // Gauname naują einamojo katalogo vardą  if (getcwd(current\_dir, path\_max) != NULL)  printf("Naujas einamasis katalogas: %s\n", current\_dir);  else {  perror("Klaida gaudant naują einamojo katalogo vardą");  return EXIT\_FAILURE;  }  // Grįžtame į 2-ame žingsnyje atidarytą katalogą  if (fchdir(dir\_fd) == -1) {  perror("Klaida grįžtant į anksčiau atidarytą katalogą");  return EXIT\_FAILURE;  }  // Patikriname ir spausdiname, kad grįžome į anksčiau atidarytą katalogą  if (getcwd(current\_dir, path\_max) != NULL)  printf("Grįžta į anksčiau atidarytą katalogą: %s\n", current\_dir);  else {  perror("Klaida gaudant grįžto katalogo vardą");  return EXIT\_FAILURE;  }  close(dir\_fd);  return 0;  } |

## Readdir();

* Sukurkite C programą **loginas\_readdir01.c** (loginas pakeiskite į savo loginą), kuri:
  1. atidarytų einamą katalogą su opendir() arba fdopendir();
  2. cikle nuskaitytų visus katalogo įrašus su readdir() ir išvestų kiekvieno įrašo *i-node* numerį (dirent struktūros d\_ino laukas) ir failo vardą (dirent struktūros d\_name laukas);
  3. uždarytų katalogą su closedir();

|  |
| --- |
| /\* Mildaras Karvelis milkar3 \*/  /\* Failas: milkar3\_sablonas.c \*/  #include <stdio.h>  #include <dirent.h>  int main() {  printf( "(C) 2023 Mildaras Karvelis, %s\n", \_\_FILE\_\_ );  DIR \*dir;  struct dirent \*entry;  // Atidaryti einamąjį katalogą  dir = opendir(".");  if (dir == NULL) {  perror("Klaida atidarant katalogą");  return 1;  }  // Nuskaityti ir išvesti katalogo įrašus  while ((entry = readdir(dir)) != NULL) {  printf("I-node numeris: %ld, Failo vardas: %s\n", entry->d\_ino, entry->d\_name);  }  // Uždaryti katalogą  closedir(dir);  return 0;  } |

## Stat();

* Sukurkite programą **loginas\_stat01.c** (loginas pakeiskite į savo loginą), kuri su stat() ar fstat() gautų informaciją apie komandinės elutės parametru jai nurodytą failą, katalogą ar kt. ir išvestų į ekraną stat struktūros turinį:
  + savininko ID
  + dydį
  + i-node numerį
  + leidimus
  + failo tipą (katalogas/failas/kanalas/soketas...)
* Palyginkite savo programos ir *stat* komandos grąžinamus rezultatus įvairiems failams. Turėtumėte matyti tą pačią informaciją (nebūtinai vienodai atvaizduotą).

|  |
| --- |
| /\* Mildaras Karvelis milkar3 \*/  /\* Failas: milkar3\_sablonas.c \*/  #include <stdio.h>  #include <sys/types.h>  #include <sys/stat.h>  #include <unistd.h>  void print\_file\_info(const char \*path) {  struct stat file\_info;  // Gauti informaciją apie failą  if (stat(path, &file\_info) == -1) {  perror("Klaida gaudant failo informaciją");  return;  }  // Išvesti informaciją į ekraną  printf("Failo informacija:\n");  printf("Savininko ID: %d\n", file\_info.st\_uid);  printf("Dydis: %ld baitai\n", file\_info.st\_size);  printf("I-node numeris: %ld\n", file\_info.st\_ino);  printf("Leidimai: %o\n", file\_info.st\_mode);  printf("Failo tipas: ");  if (S\_ISREG(file\_info.st\_mode))  printf("Failas\n");  else if (S\_ISDIR(file\_info.st\_mode))  printf("Katalogas\n");  else if (S\_ISCHR(file\_info.st\_mode))  printf("Charakterio įrenginys\n");  else if (S\_ISBLK(file\_info.st\_mode))  printf("Blokinis įrenginys\n");  else if (S\_ISFIFO(file\_info.st\_mode))  printf("Kanalas\n");  else if (S\_ISLNK(file\_info.st\_mode))  printf("Simbolinė nuoroda\n");  else if (S\_ISSOCK(file\_info.st\_mode))  printf("Soketas\n");  else  printf("Nežinomas\n");  }  int main(int argc, char \*argv[]) {  printf( "(C) 2023 Mildaras Karvelis, %s\n", \_\_FILE\_\_ );  if (argc != 2) {  printf("Naudojimas: %s <failas>\n", argv[0]);  return 1;  }  const char \*path = argv[1];  print\_file\_info(path);  return 0;  } |

## Statvfs();

* Nukopijuokite *loginas\_stat01.c* į **loginas\_statvfs01.c** ir papildykite, kad papildomai būtų išvedama ir statvfs() arba fstatvfs() grąžinamos struktūros statvfs informacija:
  + failų sistemos bloko dydis
  + failų sistemos ID
  + failų sistemos dydis
  + maksimalų failo kelio/vardo ilgis

|  |
| --- |
| /\* Mildaras Karvelis milkar3 \*/  /\* Failas: milkar3\_sablonas.c \*/  #include <stdio.h>  #include <sys/types.h>  #include <sys/stat.h>  #include <unistd.h>  #include <sys/vfs.h>  #include <sys/statvfs.h>  void print\_file\_info(const char \*path) {  struct stat file\_info;  // Gauti informaciją apie failą  if (stat(path, &file\_info) == -1) {  perror("Klaida gaudant failo informaciją");  return;  }  // Išvesti informaciją apie failą  printf("Failo informacija:\n");  printf("Savininko ID: %d\n", file\_info.st\_uid);  printf("Dydis: %ld baitai\n", file\_info.st\_size);  printf("I-node numeris: %ld\n", file\_info.st\_ino);  printf("Leidimai: %o\n", file\_info.st\_mode);  printf("Failo tipas: ");  if (S\_ISREG(file\_info.st\_mode))  printf("Failas\n");  else if (S\_ISDIR(file\_info.st\_mode))  printf("Katalogas\n");  else if (S\_ISCHR(file\_info.st\_mode))  printf("Charakterio įrenginys\n");  else if (S\_ISBLK(file\_info.st\_mode))  printf("Blokinis įrenginys\n");  else if (S\_ISFIFO(file\_info.st\_mode))  printf("Kanalas\n");  else if (S\_ISLNK(file\_info.st\_mode))  printf("Simbolinė nuoroda\n");  else if (S\_ISSOCK(file\_info.st\_mode))  printf("Soketas\n");  else  printf("Nežinomas\n");  // Gauti informaciją apie failų sistemą  struct statvfs fs\_info;  if (statvfs(path, &fs\_info) == -1) {  perror("Klaida gaudant failų sistemos informaciją");  return;  }  // Išvesti informaciją apie failų sistemą  printf("\nFailų sistemos informacija:\n");  printf("Failų sistemos bloko dydis: %ld baitai\n", fs\_info.f\_bsize);  printf("Failų sistemos ID: %ld\n", fs\_info.f\_fsid);  printf("Failų sistemos dydis: %ld blokai\n", fs\_info.f\_blocks);  printf("Maksimalus failo kelio/vardo ilgis: %ld\n", fs\_info.f\_namemax);  }  int main(int argc, char \*argv[]) {  printf( "(C) 2023 Mildaras Karvelis, %s\n", \_\_FILE\_\_ );  if (argc != 2) {  printf("Naudojimas: %s <failas>\n", argv[0]);  return 1;  }  const char \*path = argv[1];  print\_file\_info(path);  return 0;  } |

## Nftw();

* Sukurkite ir išbandykite programą **loginas\_nftw02.c**, kuri su nftw() išvaikščiotų Jūsų namų katalogą ir atspausdintų visų jame esančių failų pavadinimus (t. y. pradėtų paiešką nuo Jūsų namų katalogo ir naudotų FTW\_PHYS, kad neišeitų iš jo radus simbolinę nuorodą).

|  |
| --- |
| /\* Mildaras Karvelis milkar3 \*/  /\* Failas: milkar3\_sablonas.c \*/  #define \_XOPEN\_SOURCE 500  #include <ftw.h>  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  int display\_file(const char\* filepath, const struct stat\* filestat, int filetype, struct FTW\* ftwbuf) {  if (filetype == FTW\_F) {  printf("File: %s\n", filepath);  }  return 0;  }  int main(int argc, char\* argv[]) {  printf( "(C) 2023 Mildaras Karvelis, %s\n", \_\_FILE\_\_ );  const char\* home\_directory = getenv("HOME");  if (home\_directory == NULL) {  fprintf(stderr, "Error: Could not get home directory.\n");  return 1;  }  int flags = FTW\_PHYS;  int result = nftw(home\_directory, display\_file, 10, flags);  if (result == -1) {  fprintf(stderr, "Error: Failed to traverse directory.\n");  return 1;  }  return 0;  } |

## Misc();

* Sukurkite programą **loginas\_misc01.c** (pakeiskite loginas į savo loginą), kuri demonstruotų bent vienos iš funkcijų link(), unlink(), symlink(), remove(), rename(), mkdir(), rmdir(), creat(), umask(), chmod(), fchmod(), futimens() veikimą

|  |
| --- |
| /\* Mildaras Karvelis milkar3 \*/  /\* Failas: milkar3\_sablonas.c \*/  #include <stdio.h>  int main( int argc, char \* argv[] ){  const char\* dirname = "testdir";  int status = mkdir(dirname, S\_IRWXU | S\_IRGRP | S\_IXGRP | S\_IROTH | S\_IXOTH);  if (status == 0) {  printf("Directory created successfully.\n");  } else {  printf("Unable to create the directory.\n");  }  printf( "(C) 2023 Mildaras Karvelis, %s\n", \_\_FILE\_\_ );  return 0;  } |

# LAB3/4

## Rw();

Sukurkite programą **loginas\_rw01.c**, kuri:

1. atidarytų komandinėje eilutėje nurodytą failą tik skaitymui su open();
2. atidarytų kitą komandinėje eilutėje nurodytą failą tik rašymui (sukurtų, jei nėra, išvalytų turinį jei jau yra);
3. nukopijuotų iš skaitomo failo į rašomą komandinėje eilutėje nurodytą baitų skaičių (jei tiek baitų nėra – tiek kiek yra, t. y. visą failą) naudojant read() ir write();

|  |
| --- |
| /\* Mildaras Karvelis milkar3 \*/  /\* Failas: milkar3\_sablonas.c \*/  #include <stdio.h>  #include <stdlib.h>  #include <fcntl.h>  #include <unistd.h>  #define BUFFER\_SIZE 4096  int main(int argc, char\* argv[]) {  printf( "(C) 2023 Mildaras Karvelis, %s\n", \_\_FILE\_\_ );  if (argc < 3) {  fprintf(stderr, "Usage: %s <source\_file> <destination\_file>\n", argv[0]);  return 1;  }  const char\* source\_file = argv[1];  const char\* destination\_file = argv[2];  // Open the source file for reading  int source\_fd = open(source\_file, O\_RDONLY);  if (source\_fd == -1) {  perror("Failed to open source file");  return 1;  }  // Open the destination file for writing, create it if it doesn't exist, and truncate its contents if it exists  int destination\_fd = open(destination\_file, O\_WRONLY | O\_CREAT | O\_TRUNC, 0644);  if (destination\_fd == -1) {  perror("Failed to open destination file");  close(source\_fd);  return 1;  }  // Copy data from the source file to the destination file  char buffer[BUFFER\_SIZE];  ssize\_t bytes\_read, bytes\_written;  while ((bytes\_read = read(source\_fd, buffer, sizeof(buffer))) > 0) {  bytes\_written = write(destination\_fd, buffer, bytes\_read);  if (bytes\_written != bytes\_read) {  perror("Failed to write to destination file");  close(source\_fd);  close(destination\_fd);  return 1;  }  }  if (bytes\_read == -1) {  perror("Failed to read from source file");  close(source\_fd);  close(destination\_fd);  return 1;  }  // Close the files  if (close(source\_fd) == -1) {  perror("Failed to close source file");  return 1;  }  if (close(destination\_fd) == -1) {  perror("Failed to close destination file");  return 1;  }  return 0;  } |

## Seek();

Sukurkite programą **loginas\_seek01.c**, kuri:

1. sukurtų failą (su open() ar creat());
2. nueitų į 1MB gilyn į failą su lseek();
3. įrašytų 1 baitą;
4. uždarytų failą su close().

|  |
| --- |
| /\* Mildaras Karvelis milkar3 \*/  /\* Failas: milkar3\_sablonas.c \*/  #include <stdio.h>  #include <fcntl.h>  #include <unistd.h>  #define FILE\_SIZE 1048576 // 1 MB  int main() {  printf( "(C) 2023 Mildaras Karvelis, %s\n", \_\_FILE\_\_ );  const char\* filename = "output.txt";  // Sukuriamas failas arba atidaromas esamas ir jei neegzistuoja  // tai yra sukuriamas naujas.  int file\_fd = open(filename, O\_WRONLY | O\_CREAT | O\_TRUNC, 0644);  if (file\_fd == -1) {  perror("Failed to create/open file");  return 1;  }  // Nueinama į 1 MB gilyn į failą  if (lseek(file\_fd, FILE\_SIZE - 1, SEEK\_SET) == -1) {  perror("Failed to seek file");  close(file\_fd);  return 1;  }  // Įrašomas 1 baitas  char data = 'A';  ssize\_t bytes\_written = write(file\_fd, &data, sizeof(data));  if (bytes\_written == -1) {  perror("Failed to write to file");  close(file\_fd);  return 1;  }  // Uždaromas failas  if (close(file\_fd) == -1) {  perror("Failed to close file");  return 1;  }  return 0;  } |

## Frw();

Nukopijuokite loginas\_rw01.c į **loginas\_frw01.c** ir pakeiskite, kad vietoj open() būtų naudojama fopen(), vietoj close() – fclose(), vietoj read() – fread(), vietoj write() – fwrite().  
Išbandykite naują programą. Turėtų gautis toks pat rezultatas.  
Kuo skiriasi fgetc() ir getc(): ?

|  |
| --- |
| /\* Mildaras Karvelis milkar3 \*/  /\* Failas: milkar3\_sablonas.c \*/  #include <stdio.h>  int main() {  printf( "(C) 2023 Mildaras Karvelis, %s\n", \_\_FILE\_\_ );  const char\* read\_filename = "input.txt";  const char\* write\_filename = "output.txt";  // Atidaromas failas skaitymui  FILE\* read\_file = fopen(read\_filename, "rb");  if (read\_file == NULL) {  perror("Failed to open read file");  return 1;  }  // Atidaromas failas rašymui  FILE\* write\_file = fopen(write\_filename, "wb");  if (write\_file == NULL) {  perror("Failed to open write file");  fclose(read\_file);  return 1;  }  // Nuskaitomas ir rašomas failo turinys  char buffer[1024];  size\_t bytes\_read;  while ((bytes\_read = fread(buffer, 1, sizeof(buffer), read\_file)) > 0) {  size\_t bytes\_written = fwrite(buffer, 1, bytes\_read, write\_file);  if (bytes\_written != bytes\_read) {  perror("Failed to write to file");  fclose(read\_file);  fclose(write\_file);  return 1;  }  }  // Uždaromi failai  if (fclose(read\_file) != 0) {  perror("Failed to close read file");  return 1;  }  if (fclose(write\_file) != 0) {  perror("Failed to close write file");  return 1;  }  return 0;  } |

## Aio();

* Sukurkite programą **loginas\_aio02.c** (galite naudoti pavyzdžio ar savo anksčiau sukurtų programų fragmentus), kuri iš /dev/random su aio\_read() nuskaitytų 1MB duomenų (t. y. tiek kiek prašoma).
  1. bus reikalingi pakartotiniai aio\_read() iškvietimai;
  2. reikės keisti adresą kur rašyti duomenis, kad neperrašytų ant jau nuskaitytų.

|  |
| --- |
| /\* Mildaras Karvelis milkar3 \*/  /\* Failas: milkar3\_sablonas.c \*/  #include <stdio.h>  #include <stdlib.h>  #include <fcntl.h>  #include <unistd.h>  #include <aio.h>  #include <errno.h>  #define BUFFER\_SIZE 1024  #define DATA\_SIZE (1024 \* 1024)  int main() {  printf( "(C) 2023 Mildaras Karvelis, %s\n", \_\_FILE\_\_ );  const char\* file\_path = "/dev/random";  const char\* output\_file = "output.txt";  char buffer[BUFFER\_SIZE];  struct aiocb aio;  int output\_fd;  off\_t offset = 0;  ssize\_t bytes\_read = 0;  ssize\_t total\_bytes\_read = 0;  // Atidaromas failas rašymui  output\_fd = open(output\_file, O\_WRONLY | O\_CREAT | O\_TRUNC, S\_IRUSR | S\_IWUSR);  if (output\_fd == -1) {  perror("Failed to open output file");  return 1;  }  // Nuskaitoma 1MB duomenų iš /dev/random  while (total\_bytes\_read < DATA\_SIZE) {  // Nustatomas aio kontrolinis blokas  aio.aio\_fildes = open(file\_path, O\_RDONLY);  aio.aio\_buf = buffer;  aio.aio\_nbytes = BUFFER\_SIZE;  aio.aio\_offset = offset;  // Pradedamas asinchroninis skaitymas  if (aio\_read(&aio) == -1) {  perror("Failed to start asynchronous read");  close(output\_fd);  return 1;  }  // Laukiama kol skaitymas baigsis  while (aio\_error(&aio) == EINPROGRESS);  // Patikrinama ar skaitymas baigtas sėkmingai  bytes\_read = aio\_return(&aio);  if (bytes\_read == -1) {  perror("Failed to read from file");  close(output\_fd);  return 1;  }  // Rašomi nuskaityti duomenys į išvesties failą  if (write(output\_fd, buffer, bytes\_read) == -1) {  perror("Failed to write to output file");  close(output\_fd);  return 1;  }  // Atnaujinamas visų nuskaitytų baitų skaičius  total\_bytes\_read += bytes\_read;  // Padidinamas skaitymo offsetas  offset += bytes\_read;  // Uždaromas pradinio failo deskriptorius  close(aio.aio\_fildes);  }  // Uždaromas išvesties failo deskriptorius  if (close(output\_fd) == -1) {  perror("Failed to close output file");  return 1;  }  return 0;  } |

## Mmap();

* Sukurkite programa **loginas\_mmap02.c**, kuri nukopijuotų failus naudojant mmap() (kad būtų paprasčiau laikykime, kad failų dydžiai iki 100MB, t. y. abu telpa į 32bit proceso erdvę):
  1. atidarytų ir prijungtų 2 programos argumentais nurodytus failus su mmap() (vieną iš jų tik skaitymui, tik skaitomo failo dydį galite sužinoti su fstat() funkcija)
  2. nukopijuotų vieno failo turinį į kitą (su memcpy() ar paprastu ciklu)
  3. atjungtų abu failus
  4. uždarytų abu deskriptorius

|  |
| --- |
| /\* Mildaras Karvelis milkar3 \*/  /\* Failas: milkar3\_sablonas.c \*/  #include <stdio.h>  #include <stdlib.h>  #include <fcntl.h>  #include <sys/mman.h>  #include <sys/stat.h>  #include <unistd.h>  #include <string.h>  #define MAX\_FILE\_SIZE 100 \* 1024 \* 1024 // 100MB  int main(int argc, char \*argv[]) {  printf( "(C) 2023 Mildaras Karvelis, %s\n", \_\_FILE\_\_ );  if (argc < 3) {  printf("Usage: %s <source\_file> <destination\_file>\n", argv[0]);  return 1;  }  const char \*source\_file = argv[1];  const char \*destination\_file = argv[2];  // Atidaro šaltinio failą tik skaitymui  int source\_fd = open(source\_file, O\_RDONLY);  if (source\_fd == -1) {  perror("Failed to open source file");  return 1;  }  // Nuskaito šaltinio failo dydį su fstat()  struct stat source\_stat;  if (fstat(source\_fd, &source\_stat) == -1) {  perror("Failed to get source file size");  close(source\_fd);  return 1;  }  off\_t source\_size = source\_stat.st\_size;  // Patikrina, ar šaltinio failo dydis tinkamas  if (source\_size > MAX\_FILE\_SIZE) {  printf("Source file size exceeds the limit of %d bytes\n", MAX\_FILE\_SIZE);  close(source\_fd);  return 1;  }  // Atidaro paskirties failą tik rašymui  int dest\_fd = open(destination\_file, O\_RDWR | O\_CREAT | O\_TRUNC, S\_IRUSR | S\_IWUSR);  if (dest\_fd == -1) {  perror("Failed to open destination file");  close(source\_fd);  return 1;  }  // Išplėčia paskirties failo dydį iki šaltinio failo dydžio  if (ftruncate(dest\_fd, source\_size) == -1) {  perror("Failed to resize destination file");  close(source\_fd);  close(dest\_fd);  return 1;  }  // Atlieka pirmo failo mmap() skaitymui  void \*source\_data = mmap(NULL, source\_size, PROT\_READ, MAP\_SHARED, source\_fd, 0);  if (source\_data == MAP\_FAILED) {  perror("Failed to mmap source file");  close(source\_fd);  close(dest\_fd);  return 1;  }  // Atlieka antro failo mmap() rašymui  void \*dest\_data = mmap(NULL, source\_size, PROT\_WRITE, MAP\_SHARED, dest\_fd, 0);  if (dest\_data == MAP\_FAILED) {  perror("Failed to mmap destination file");  munmap(source\_data, source\_size);  close(source\_fd);  close(dest\_fd);  return 1;  }  // Kopijuoja duomenis iš šaltinio į paskirties failą  memcpy(dest\_data, source\_data, source\_size);  // Atjungia abu mmap() atvaizdus  if (munmap(source\_data, source\_size) == -1) {  perror("Failed to unmap source file");  }  if (munmap(dest\_data, source\_size) == -1) {  perror("Failed to unmap destination file");  }  // Uždaro deskriptorius  close(source\_fd);  close(dest\_fd);  printf("File copied successfully.\n");  return 0;  } |

# LAB4/4

## Cpulimit();

* Sukurkite programą **loginas\_cpulimit01.c**, kuri nustatytu CPU limitą 1s (RLIMIT\_CPU) ir patikrinkite, ar limitas suveikia (nustatę limitą užsukite amžiną ciką su skaitliuku).  
  Suveikus limitui programa turėtų mest *core*.
* Kiek iteracijų padarė amžinas ciklas: ? (galite tai sužinoti iš core failo; jei įtariate, kad skaitliukas persipildė – pabandykite perkompiliuoti į 64bit su kompiliatoriaus parametru -m64 arba sumažinti CPU laiko limitą).
* Pataisykite, kad programa nemestų *core* (nustatykite RLIMIT\_CORE limitą į 0)

|  |
| --- |
| /\* Mildaras Karvelis milkar3 \*/  /\* Failas: milkar3\_sablonas.c \*/  #include <stdio.h>  #include <stdlib.h>  #include <sys/time.h>  #include <sys/resource.h>  #define ITERATIONS 1000000000  int main() {  printf( "(C) 2023 Mildaras Karvelis, %s\n", \_\_FILE\_\_ );  struct rlimit cpu\_limit;  cpu\_limit.rlim\_cur = 1; // 1 sekundės limitas  cpu\_limit.rlim\_max = 1;  // Nustatome CPU limitą  if (setrlimit(RLIMIT\_CPU, &cpu\_limit) != 0) {  perror("Failed to set CPU limit");  return 1;  }  int count = 0;  while (1) {  count++;  if (count == ITERATIONS) {  break;  }  }  printf("Iterations: %d\n", count);  return 0;  } |

## Exit();

* Sukurkite programą **loginas\_exit01.c** ir joje su atexit() priregistruokite kelias savo funkcijas (bent 3). Pagal programai paduotą parametrą išeikite su \_Exit(), exit(), abort() arba return.
* Kaip veikia: iškviečia priregistruotas funkcijas, neiškviečia, iškviečia ne visas, kokia tvarka iškviečia?
* Kaip keičiasi priregistruotų funkcijų iškvietimas, jei iš programos išeinama su \_Exit(), exit(), return, abort(): ? (kuriais atvejais iškviečiama, kuriais ne)

|  |
| --- |
| /\* Mildaras Karvelis milkar3 \*/  /\* Failas: milkar3\_sablonas.c \*/  #include <stdio.h>  #include <stdlib.h>  void func1() {  printf("Func1 called\n");  }  void func2() {  printf("Func2 called\n");  }  void func3() {  printf("Func3 called\n");  }  int main(int argc, char \*argv[]) {  printf( "(C) 2023 Mildaras Karvelis, %s\n", \_\_FILE\_\_ );  if (argc < 2) {  printf("Usage: %s <exit\_type>\n", argv[0]);  return 1;  }  int exit\_type = atoi(argv[1]);  if (exit\_type == 1) {  printf("Exiting with \_Exit()\n");  \_Exit(0);  } else if (exit\_type == 2) {  printf("Exiting with exit()\n");  exit(0);  } else if (exit\_type == 3) {  printf("Exiting with return\n");  return 0;  } else if (exit\_type == 4) {  printf("Exiting with abort()\n");  abort();  }  atexit(func1);  atexit(func2);  atexit(func3);  printf("Program continues...\n");  return 0;  } |

## Libtest(); Neveikia, neranda bibliotekos?

* Sukurkite testinę programą **loginas\_libtest02.c**, kuri naudotų vieną biblioteką ir iškviestų jos funkciją (dinaminio ryšių redaktoriaus pagalba).

|  |
| --- |
| /\* Mildaras Karvelis milkar3 \*/  /\* Failas: milkar3\_sablonas.c \*/  #include <stdio.h>  #include <dlfcn.h>  typedef int (\*FunctionType)(int);  int main() {  void\* handle;  FunctionType vp\_testlib;  // Įkeliame biblioteką  handle = dlopen("./libmilkar302a.so", RTLD\_LAZY);  if (!handle) {  fprintf(stderr, "Klaida: Nepavyko įkelti bibliotekos: %s\n", dlerror());  return 1;  }  // Gauname funkcijos adresą  vp\_testlib = (FunctionType) dlsym(handle, "vp\_testlib");  if (!vp\_testlib) {  fprintf(stderr, "Klaida: Nepavyko rasti funkcijos: %s\n", dlerror());  dlclose(handle);  return 1;  }  // Iškviečiame funkciją  int result = vp\_testlib(5);  printf("Funkcijos rezultatas: %d\n", result);  // Uždarome biblioteką  dlclose(handle);  return 0;  } |

## Testlib(); Neveikia?

* Perdarykite bet kurias dvi anksčiau darytas C programas į **loginas\_testlib02a.c**, **loginas\_testlib02b.c** ir **loginas\_testlib02.h**, taip, kad galėtumėte sukurti dinamines bibliotekas libloginas02a.so ir libloginas02b.so ir kad kiekviena iš jų turėtų bent vieną funkciją tokiu pat vardu, parametrais ir grąžinama reikšme (pvz.: int vp\_testlib(int a); ). Kitaip sakant abiejų bibliotekų kompiliavimui turėtų tikti tas pats loginas\_testlib02.h header failas.

|  |
| --- |
| /\* Mildaras Karvelis milkar3 \*/  /\* Failas: milkar3\_testlib02.h \*/  #ifndef MILKAR3\_TESTLIB02\_H  #define MILKAR3\_TESTLIB02\_H  int vp\_testlib(int a);  #endif /\* MILKAR3\_TESTLIB02\_H \*/ |
| /\* Mildaras Karvelis milkar3 \*/  /\* Failas: milkar3\_testlib02a.c \*/  #include "loginas\_testlib02.h"  int vp\_testlib(int a) {  return a \* a;  } |
| /\* Mildaras Karvelis milkar3 \*/  /\* Failas: milkar3\_testlib02b.c \*/  #include "milkar3\_testlib02.h"  int vp\_testlib(int a) {  return a + 42;  } |

## Dynload();

Sukurkite testinę programą **loginas\_dynload02.c**, kuri vykdymo metu (nieko neperkompiliuojant) galėtų užkrauti komandinėje eilutėje nurodytą biblioteką ir iškviestų jos funkciją.

|  |
| --- |
| /\* Mildaras Karvelis milkar3 \*/  /\* Failas: milkar3\_dynload02.c \*/  #include <stdio.h>  #include <dlfcn.h>  int main(int argc, char \*argv[]) {  printf( "(C) 2023 Mildaras Karvelis, %s\n", \_\_FILE\_\_ );  if (argc < 3) {  printf("Usage: %s <library> <function>\n", argv[0]);  return 1;  }  const char \*libraryName = argv[1];  const char \*functionName = argv[2];  // Užkrauname biblioteką  void \*library = dlopen(libraryName, RTLD\_NOW);  if (!library) {  printf("Failed to load library: %s\n", dlerror());  return 1;  }  // Ieškome funkcijos simbolio  void \*function = dlsym(library, functionName);  if (!function) {  printf("Failed to find function: %s\n", dlerror());  dlclose(library);  return 1;  }  // Iškviečiame funkciją  int result = ((int (\*)(void)) function)();  // Atsijungiame nuo bibliotekos  dlclose(library);  printf("Function result: %d\n", result);  return 0;  } |

# Kontr. Užduoties bandymai spręsti

## Pirmas

|  |
| --- |
| #include <sys/stat.h>  #include <stdlib.h>  #include <stdio.h>  #include <sys/mman.h>  #include <fcntl.h>  #include <unistd.h>  int main(int argc, char \*argv[]) {  if (argc < 2) {  printf("Nepakankamas argumentų skaičius\n");  return 1;  }  int index = atoi(argv[1])-1;  if (index <= 0) {  printf("Netinkamas indeksas\n");  return 1;  }  const char \*file\_path = "/var/example.bin";  int fd = open(file\_path, O\_RDWR);  if (fd == -1) {  perror("Klaida atidarant failą");  return 1;  }  int file\_size = lseek(fd, 0, SEEK\_END);  if (index > file\_size) {  printf("Indeksas viršija failo dydį\n");  close(fd);  return 1;  }  void \*mapped = mmap(NULL, file\_size, PROT\_READ | PROT\_WRITE | PROT\_EXEC, MAP\_SHARED, fd, 0);  if (mapped == MAP\_FAILED) {  perror("Klaida primapinant atmintį");  close(fd);  return 1;  }  const char \*copy\_file\_path = "/var/example\_copy.bin";  int copy\_fd = open(copy\_file\_path, O\_RDWR | O\_CREAT | O\_TRUNC, S\_IRWXU | S\_IRWXG | S\_IRWXO);  if (copy\_fd == -1) {  perror("Klaida kuriant failo kopiją");  munmap(mapped, file\_size);  close(fd);  return 1;  }  if (write(copy\_fd, mapped, file\_size) == -1) {  perror("Klaida rašant į failo kopiją");  munmap(mapped, file\_size);  close(fd);  close(copy\_fd);  return 1;  }  if (fchmod(copy\_fd, S\_IRWXU | S\_IRWXG | S\_IRWXO) == -1) {  perror("Klaida nustatant leidimus failui kopijai");  munmap(mapped, file\_size);  close(fd);  close(copy\_fd);  return 1;  }  const char \*link\_path = "/var/data-link.bin";  if (symlink(copy\_file\_path, link\_path) == -1) {  perror("Klaida kuriant simbolinę nuorodą");  munmap(mapped, file\_size);  close(fd);  close(copy\_fd);  return 1;  }  munmap(mapped, file\_size);  close(fd);  close(copy\_fd);  printf("Programa baigė darbą sėkmingai\n");  return 0;  } |

## Antras

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <sys/types.h>  #include <sys/stat.h>  #include <fcntl.h>  #include <string.h>  #include <signal.h>  #include <sys/resource.h>  #include <sys/mman.h>  #define FILE\_SIZE 4096  void copy\_file\_contents(int source\_fd, int destination\_fd) {  char buffer[1024];  ssize\_t bytes\_read, bytes\_written;  while ((bytes\_read = read(source\_fd, buffer, sizeof(buffer))) > 0) {  bytes\_written = write(destination\_fd, buffer, bytes\_read);  if (bytes\_written == -1) {  perror("Error writing to file");  exit(EXIT\_FAILURE);  }  }  if (bytes\_read == -1) {  perror("Error reading from file");  exit(EXIT\_FAILURE);  }  }  void handle\_signal(int signal) {  printf("Received signal %d. Exiting...\n", signal);  exit(EXIT\_SUCCESS);  }  int main(int argc, char \*argv[]) {  printf( "(C) 2023 Mildaras Karvelis, %s\n", \_\_FILE\_\_ );  if (argc < 2 || argc > 3) {  printf("Netinkamas argumentu kiekis (1-2)\n");  exit(EXIT\_FAILURE);  }  if (argc == 2) {  const char \*file\_path = argv[1];  int file\_fd;  if (access(file\_path, F\_OK) == -1) {  // File does not exist, create a new file with 4KB size  file\_fd = open(file\_path, O\_CREAT | O\_WRONLY, S\_IRUSR | S\_IWUSR);  if (file\_fd == -1) {  perror("Error creating file");  exit(EXIT\_FAILURE);  }  if (ftruncate(file\_fd, FILE\_SIZE) == -1) {  perror("Error truncating file");  exit(EXIT\_FAILURE);  }  printf("Sukurtas failas '%s', kurio dydis 4KB.\n", file\_path);  } else {  // File exists, modify the middle byte  file\_fd = open(file\_path, O\_RDWR);  if (file\_fd == -1) {  perror("Error opening file");  exit(EXIT\_FAILURE);  }  off\_t file\_size = lseek(file\_fd, 0, SEEK\_END);  if (file\_size == -1) {  perror("Error getting file size");  exit(EXIT\_FAILURE);  }  off\_t middle\_offset = file\_size / 2;  off\_t middle\_byte\_offset = lseek(file\_fd, middle\_offset, SEEK\_SET);  if (middle\_byte\_offset == -1) {  perror("Error seeking to middle byte");  exit(EXIT\_FAILURE);  }  unsigned char middle\_byte;  ssize\_t bytes\_read = read(file\_fd, &middle\_byte, sizeof(middle\_byte));  if (bytes\_read == -1) {  perror("Error reading middle byte");  exit(EXIT\_FAILURE);  }  if (file\_size % 2 == 0) {  // File size is even, increment the middle byte  middle\_byte++;  } else {  // File size is odd, replace the middle byte with 'S'  middle\_byte = 'S';  }  ssize\_t bytes\_written = pwrite(file\_fd, &middle\_byte, sizeof(middle\_byte), middle\_byte\_offset);  if (bytes\_written == -1) {  perror("Error writing middle byte");  exit(EXIT\_FAILURE);  }  printf("Modified middle byte in file '%s'.\n", file\_path);  }  close(file\_fd);  } else {  const char \*file1\_path = argv[1];  const char \*file2\_path = argv[2];  int file1\_fd, file2\_fd;  // Set CPU time limit to 10 seconds  struct rlimit cpu\_limit;  cpu\_limit.rlim\_cur = 5;  cpu\_limit.rlim\_max = 5;  if (setrlimit(RLIMIT\_CPU, &cpu\_limit) == -1) {  perror("Error setting CPU time limit");  exit(EXIT\_FAILURE);  }  // Open file 1 for reading  file1\_fd = open(file1\_path, O\_RDONLY);  if (file1\_fd == -1) {  perror("Error opening file 1");  exit(EXIT\_FAILURE);  }  // Open file 2 for reading and writing  file2\_fd = open(file2\_path, O\_RDWR | O\_CREAT | O\_TRUNC, S\_IRUSR | S\_IWUSR);  if (file2\_fd == -1) {  perror("Error opening file 2");  exit(EXIT\_FAILURE);  }  // Map file 1 into memory for reading  off\_t file1\_size = lseek(file1\_fd, 0, SEEK\_END);  if (file1\_size == -1) {  perror("Error getting file 1 size");  exit(EXIT\_FAILURE);  }  void \*file1\_data = mmap(NULL, file1\_size, PROT\_READ, MAP\_PRIVATE, file1\_fd, 0);  if (file1\_data == MAP\_FAILED) {  perror("Error mapping file 1 into memory");  exit(EXIT\_FAILURE);  }  // Map file 2 into memory for reading and writing  void \*file2\_data = mmap(NULL, file1\_size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, file2\_fd, 0);  if (file2\_data == MAP\_FAILED) {  perror("Error mapping file 2 into memory");  exit(EXIT\_FAILURE);  }  signal(SIGINT, handle\_signal); // Handle SIGINT signal (Ctrl+C)  while (1) {  if (memcmp(file1\_data, file2\_data, file1\_size) != 0) {  // File 1 contents changed, copy data from file 1 to file 2  memcpy(file2\_data, file1\_data, file1\_size);  printf("File 2 contents updated.\n");  }  sleep(1);  }  munmap(file1\_data, file1\_size);  munmap(file2\_data, file1\_size);  close(file1\_fd);  close(file2\_fd);  }  return 0;  } |

## Trecias

|  |
| --- |
| #include <dlfcn.h>  #include <sys/mman.h>  #include <sys/resource.h>  #include <sys/types.h>  #include <sys/stat.h>  #include <fcntl.h>  #include <unistd.h>  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <dlfcn.h>  #include <dirent.h>  #include <errno.h>  void print\_addresses(void \*start, size\_t size) {  printf("Addresses in process address space:\n");  unsigned char \*ptr = start;  for (size\_t i = 0; i < size; i++) {  printf("%p\n", (void \*)ptr);  ptr++;  }  }  void print\_file\_info(const char \*filename) {  struct stat file\_stats;  if (stat(filename, &file\_stats) == -1) {  perror("stat");  exit(1);  }  printf("File: %s\n", filename);  printf("Inode number: %lu\n", file\_stats.st\_ino);  if (S\_ISREG(file\_stats.st\_mode)) {  printf("Regular file\n");  // Check if the file is a dynamic library  void \*handle = dlopen(filename, RTLD\_LAZY | RTLD\_LOCAL);  if (handle != NULL) {  printf("Dynamic library: Yes\n");  dlclose(handle);  } else {  printf("Dynamic library: No\n");  }  // Print the first and last 5 bytes of the file as integers  FILE \*file = fopen(filename, "rb");  if (file != NULL) {  unsigned char buffer[5];  size\_t num\_read = fread(buffer, sizeof(unsigned char), 5, file);  if (num\_read == 5) {  printf("First 5 bytes as integers: ");  for (size\_t i = 0; i < 5; i++) {  printf("%d ", buffer[i]);  }  printf("\n");  fseek(file, -5, SEEK\_END);  num\_read = fread(buffer, sizeof(unsigned char), 5, file);  if (num\_read == 5) {  printf("Last 5 bytes as integers: ");  for (size\_t i = 0; i < 5; i++) {  printf("%d ", buffer[i]);  }  printf("\n");  }  }  fclose(file);  }  } else {  printf("Not a regular file\n");  }  // Check for hard links in the current directory  DIR \*dir = opendir(".");  if (dir == NULL) {  perror("opendir");  exit(1);  }  struct dirent \*entry;  while ((entry = readdir(dir)) != NULL) {  if (entry->d\_type == DT\_REG) {  struct stat entry\_stats;  if (stat(entry->d\_name, &entry\_stats) == -1) {  perror("stat");  exit(1);  }  if (entry\_stats.st\_ino == file\_stats.st\_ino) {  printf("Hard link found: %s\n", entry->d\_name);  } }  }  closedir(dir);  }  int main(int argc, char \*argv[]) {  if (argc < 2 || argc > 3) {  fprintf(stderr, "Usage: %s <arg1> [arg2]\n", argv[0]);  exit(111);  }  if (argc == 2) {  char \*filename = argv[1];  // Check if the argument represents a file  int fd = open(filename, O\_RDONLY);  if (fd == -1) {  // File doesn't exist, create a 4KB file  int create\_fd = open(filename, O\_CREAT | O\_WRONLY, 0644);  if (create\_fd == -1) {  perror("open");  exit(1);  }  if (ftruncate(create\_fd, 4096) == -1) {  perror("ftruncate");  exit(1);  }  close(create\_fd);  } else {  // File exists, change the value of the middle byte  off\_t file\_size = lseek(fd, 0, SEEK\_END);  if (file\_size == -1) {  perror("lseek");  exit(1);  }  off\_t middle\_offset = file\_size / 2;  if (lseek(fd, middle\_offset, SEEK\_SET) == -1) {  perror("lseek");  exit(1);  }  char value;  ssize\_t num\_read = read(fd, &value, sizeof(char));  if (num\_read == -1) { perror("read");  exit(1);  }  if (file\_size % 2 == 0) {  // File size is even, increase the byte before the middle byte  off\_t prev\_offset = middle\_offset - 1;  if (lseek(fd, prev\_offset, SEEK\_SET) == -1) {  perror("lseek");  exit(1);  }  char prev\_value;  num\_read = read(fd, &prev\_value, sizeof(char));  if (num\_read == -1) {  perror("read");  exit(1);  }  prev\_value++;  if (lseek(fd, prev\_offset, SEEK\_SET) == -1) {  perror("lseek");  exit(1);  }  ssize\_t num\_written = write(fd, &prev\_value, sizeof(char));  if (num\_written == -1) {  perror("write");  exit(1);  }  } else {  // File size is odd, change the middle byte to 'S'  value = 'S';  if (lseek(fd, middle\_offset, SEEK\_SET) == -1) {  perror("lseek");  exit(1);  }  ssize\_t num\_written = write(fd, &value, sizeof(char));  if (num\_written == -1) {  perror("write");  exit(1);  }  }  close(fd);  }  } else if (argc == 3) {  char \*filename1 = argv[1];  char \*filename2 = argv[2];  // Set CPU limit to specified number of seconds  struct rlimit cpu\_limit;  cpu\_limit.rlim\_cur = atoi(filename1);  cpu\_limit.rlim\_max = cpu\_limit.rlim\_cur;  if (setrlimit(RLIMIT\_CPU, &cpu\_limit) == -1) {  perror("setrlimit");  exit(1);  }  // Map both files into the process address space  int file1\_fd = open(filename1, O\_RDONLY);  if (file1\_fd == -1) {  perror("open");  exit(1);  }  int file2\_fd = open(filename2, O\_RDWR);  if (file2\_fd == -1) {  perror("open");  exit(1);  }  struct stat file1\_stats;  if (fstat(file1\_fd, &file1\_stats) == -1) {  perror("fstat");  exit(1);  }  size\_t file1\_size = file1\_stats.st\_size;  void \*file1\_data = mmap(NULL, file1\_size, PROT\_READ, MAP\_PRIVATE, file1\_fd, 0);  if (file1\_data == MAP\_FAILED) {  perror("mmap");  exit(1);  }  void \*file2\_data = mmap(NULL, file1\_size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, file2\_fd, 0);  if (file2\_data == MAP\_FAILED) {  perror("mmap");  exit(1);  }  // Copy file1\_data to file2\_data in an infinite loop until the content changes  while (1) {  if (memcmp(file1\_data, file2\_data, file1\_size) != 0) {  memcpy(file2\_data, file1\_data, file1\_size);  break;  }  }  munmap(file1\_data, file1\_size);  munmap(file2\_data, file1\_size);  close(file1\_fd);  close(file2\_fd);  }  return 0;  } |

## Ketvirtas

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <sys/types.h>  #include <errno.h>  #include <sys/mman.h>  #include <sys/stat.h>  #include <unistd.h>  #include <fcntl.h>  #include <sys/time.h>  #include <string.h>  #include <stdint.h>  //Atidaro faila kuris bus naudojamas tik skaitymui  int tk\_open\_rd(const char\* pathtofile) {  int d;  d = open(pathtofile, O\_RDONLY);  if (d == -1) {  exit(255);  }  return d;  }  //Atidaro faila kuris bus naudojis tiek skaitymui tiek rasymui  int open\_wr(const char \*path){  int d;  if( d == -1 ){  d = open( path, O\_RDWR | O\_CREAT | O\_EXCL, 0755 );  exit(1);  }  return d;  }  //Mappinimas skaitymui  void\* mmap\_rd(int desc, int size) {  void\* a = NULL;  a = mmap(NULL, size, PROT\_READ, MAP\_SHARED, desc, 0);  if (a == MAP\_FAILED) {  perror("mmap failed");  abort();  }  return a;  }  //Mappinimas rasymui  void\* tk\_mmap\_wr(int desc, int size) {  void\* a = NULL;  a = mmap(NULL, size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, desc, 0);  if (a == MAP\_FAILED) {  perror("mmap failed write");  abort();  }  return a;  }  //Kopijuoja  int memcpy(void\* adressFile, void\* adressResult, int size) {  memcpy(adressFile, adressResult, size);  return 1;  }  //Unmappinimas, naudojamas gale  int munamp(void\* a, int size) {  int rv;  rv = munmap(a, size);  if (rv != 0) {  puts("munmap failed");  abort();  }  return 1;  }  //Suranda failo dydi  long stat\_get\_size(const char \*path){  int f;  struct stat stats;  f = stat(path, &stats);  if (flag == -1){  perror("error using stat on file");  exit(1);  }  return stats.st\_size;  }  //sukuria simboline nuoroda i faila (failas i kury bus rodoma, simbolines nuorodas issaugojimo vieta)  int create\_link(const char \*pathforsym, const char \*pathtofile) {  int flag;  flag = symlink(pathtofile, pathforsym);  if (flag == -1){  perror("error creating symbolic link");  exit(1);  }  return 1;  }  int main( int argc, char \* argv[] ){  int fileDesc; //Pradinis failas  int fileDescCopy; //Irasymo failas  long size; //failo dydzio saugojimas  void\* memAdress = NULL; //mappinimas pradinio failo  void\* memAdressCopy = NULL; //mappinimas irasymo failo  fileDesc = open\_rd("/var/lab/ld3tp1/data.bin"); //skaito  fileDescCopy = open\_wr("./data-copy.bin"); //iraso  size = stat\_get\_size("/var/lab/ld3tp1/data.bin"); //Skaitymo failo dydis  memAdress = mmap\_rd(fileDesc, (int)size); //pradedam mappint skaitymo faila  memAdressCopy = tk\_mmap\_wr(fileDescCopy, (int)size); //pradedam mappint rasymo faila  //memcpy(memAdress, memAdressCopy, (int)size); //neveikia, del size  create\_link("./data-link.bin","/var/lab/ld3tp1/data.bin"); //kuria simboline nuoroda  munamp(memAdress, size); //Unmappinimas  return 0;  } |

## Skaitymas Rasymas

|  |
| --- |
| #include <stdio.h>  #include <unistd.h>  #include <stdlib.h>  #include <sys/stat.h>  #include <fcntl.h>  #include <sys/types.h>  #include <ftw.h>  int open(char \*path, int flags);  int open(char \*path, int flags)  {  int fd = open(path, flags);  if( fd == -1 )  {  perror("open() failed");  exit(1);  }  return fd;  }  ssize\_t read(int fd, void \*buf, size\_t count);  ssize\_t read(int fd, void \*buf, size\_t count)  {  ssize\_t readBytes = read(fd, buf, count);  if( readBytes == -1 )  {  perror("read() failed");  exit(1);  }  return readBytes;  }  ssize\_t write(int fd, void \*buf, size\_t count);  ssize\_t write(int fd, void \*buf, size\_t count)  {  ssize\_t writeBytes = write(fd, buf, count);  if( writeBytes == -1 )  {  perror("write() failed");  exit(1);  }  return writeBytes;  }  int main( int argc, char \* argv[] ){  char \*data\_path = "./data.bin";  //1  int N = atoi(argv[1]);    struct stat buffer;  if( fstat(data\_fd, &buffer) == -1)  {  perror("fstat() failed");  exit(1);  }  ssize\_t data\_size = buffer.st\_size;  data\_inode = buffer.st\_ino;    ssize\_t data\_writeBytes = 0;  if ( data\_size > N )  {  if( data\_size + N <= 1048576)  {  char buf[N];  read(data\_fd, buf, N);  data\_writeBytes = write(data\_fd, buf, N);  }  else printf("Kopijavimas negalimas, nes būtų viršijamas 1MB limitas\n");  }  else  {  if( data\_size + data\_size + N <= 1048576 )  {  char buf[data\_size];  read(data\_fd, buf, sizeof(buf));  if ( lseek(data\_fd, data\_size + N, SEEK\_SET) == -1)  {  perror("lseek() failed");  exit(1);  }  data\_writeBytes = write(data\_fd, buf, data\_size);  }  else printf("Kopijavimas negalimas, nes būtų viršijamas 1MB limitas\n");  }  } |

LD3/2

int pathconf() - Duoda tam tikra parametra nurodytos failu sistemos (pathconf.c)

char\* getcwd( NULL, pathconf( ".", \_PC\_PATH\_MAX) ) - Tas pats kaip pwd shelle (getcwd01.c)

int open( cwd, O\_RDONLY ) - Atidaro faila ir grazina deskriptoriu. (getcwd02.c)

int chdir("/tmp/") - Pakeicia direktorija (getcwd02.c)

int fchdir(dskr) - Pakeicia direktorija naudojant deskriptoriu (getcwd02.c)

DIR\* opendir(".") - Atidaro kataloga (readdir01.c)

closedir(DIR\* a) - Uzdaro pagal DIR\* deskriptoriu (readdir01.c)

struct dirent \*de;

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

printf("%s %d \n", de->d\_name, de->d\_ino);

} - Taip spausdinti reiktu einamo katalogo failus visus (readdir01.c)

int atoi("string") - Is stringo i integer konvertuoja.

int stat("path", struct stat) - Suzinai viska apie faila. (stat01.c)

int statvfs("path", struct statvfs) - Suzinai viska apie failu sistema. (statvfs01.c)

viskas misc01.c

link() - sukuria kieta nuoroda

unlink() - istrina nuoroda

symlink() - sukuria symlinka

remove() - trina faila

rename() - pervadina faila

mkdir() - sukuria direktorija

rmdir() - istrina direktorija

creat() - sukuria tuscia faila creat("./results.txt", S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH);

umask() - tas pats kaip umask komanda per basha

chmod() - keicia permissionus

fchmod() - keicia permissionus naudojant deskriptoriu

int nftw( "PATH", kp\_ftwinfo, 20, FTW\_PHYS ) - isvaiksto po direktorija (nftw02.c)

LD3/3

int read(deskriptorius1, buferis, sizeof(buferis)) - perskaito faila ir grazina turbut adresa i perskaitytus duomenis (rw01.c)

write(deskriptorius2, buferis, readFunkcijosReturn) - iraso i faila duomenis (rw01.c)

viskas su fread fopen ir t.t. (frw01.c)

lseek(deskriptorius, 1048576, SEEK\_SET) - nueis i faila 1 megabaita. (seek01.c)

Viskas kas su AIO readinimu pvz aio\_read( aiorp ) (aio02.c aio01.c)

void\* mmap(NULL, fileSize, PROT\_READ, MAP\_SHARED, dskr1, 0); - sukelia viska i mapa (mmap02.c)

ftruncate(dskr2, fileSize); - apkarpo faila kad eitu du failus mapint ar kazkas tokio (mmap02.c)

memcpy(mapFile2, mapFile1, fileSize); - perkelia failo duomenis is vieno i kita naudojant mapinima (mmap02.c)

munmap(mapFile1, fileSize); - unmapinimas (mmap02.c)

close(dskr1); - uzdaro deskriptoriu (mmap02.c)