

Dokumentaatio: Käyttöjärjestelmät ja systeemiohjelmointi

Koodit on saatavissa:

<https://github.com/Alexi-Karhu/KasyHT1>

<https://github.com/Alexi-Karhu/KasyHT2>

<https://github.com/Alexi-Karhu/KasyHT4>

1)

Ensimmäisen harjoitustehtävän oli tarkoitus tutustuttaa c-ohjelmoinnin pariin. Se toimi lämmittelynä tuleville tehtäville. Ohjelmassa oli tarkoitus luoda reverse-niminen, joka kääntää tekstin väärin päin. Ohjelma ottaa sisäänsä kolme komentoa:

```
prompt> ./reverse  
prompt> ./reverse input.txt  
prompt> ./reverse input.txt output.txt
```

Ohjelma kääntyy seuraavalla käskyllä:

```
aleksi@ubuntu:~/Desktop/Käyttöjärjestelmät ja systeemiohjelmointi/T1 gcc reverse.c -o  
reverse -Wall -Werror
```

Lähdin ratkaisemaan ongelmaa ensin tutustumalla netistä löytyvään tietoon samanlaisista ratkaisuksista. Päädyin käyttämään readline-komentoa sillä sen kanssa muistinkäsittely on helpompaa kuin esimerkiksi fgetsin kanssa. Siihen löytyi myös hyvä dokumentaatio ja siinä hyödynnettiin stdin jolloin varmasti täytän tehtävän vaatimukset. Netistä löytyi myös eräs esimerkkiratkaisu, kuinka listan saa luettua väärin päin. Sain tuosta inspiraatiota listan käsittelyyn. Loput tulivatkin omasta osaamisesta.

Ensimmäinen komento antaa käyttäjän kirjoittaa komentoriville haluamansa määrän tekstiä, joka lopussa käännetään väärin päin jolloin viimeiseksi kirjoitetut lauseet tulostetaan käyttäjälle ensimmäisenä. Aluksi määritellään muuttujat ja tulostetaan käyttäjälle ohjeita. Tämän jälkeen avataan while-loop, jossa tarkkaillaan lopetuskomentoa ja kopioidaan uudet rivit listan alkioiksi. Kun käyttäjä lopettaa kirjoittamisen avataan for-loop, joka suoritetaan i-1 kertaa, aina i:tä pienentäen. Jokaisella iteraatiolla tulostetaan uusi rivi listasta. Lopuksi vapautetaan muisti ja suljetaan ohjelma.

```

//Funktio joka kysyy käyttäjältä syötettä ja tulostaa sen käänteisenä komentoriville
void reverseInTerminal() {
    int i, count = 0;
    char *input = NULL;
    size_t size;
    char list[BUFFER][BUFFER];

    //Määritetään !end lopetuskomennoksi
    fprintf(stdout, "Type something to be reversed. End the typing by typing !end.\n\n");

    //Käytetään getlinea käyttäjä syöttämän rivin hakemiseksi
    while(getline(&input, &size, stdin) != -1) {
        if(strcmp(input, "!end\n") == 0) {
            break;
        }

        //Kopioidaan rivi listaan ja kasvatetaan laskuria
        strcpy(list[count], input);
        count++;
    }

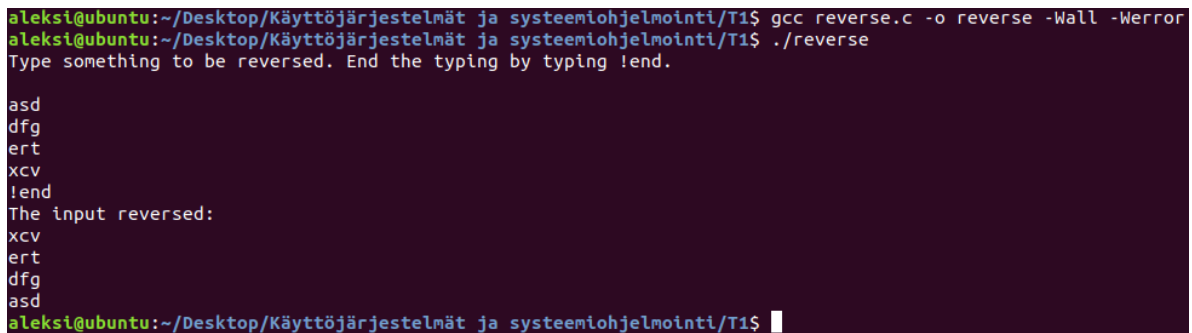
    fprintf(stdout, "The input reversed:\n");

    //Käytään lista n-1 kertaa läpi ja pienennetään laskuria
    for(i = count-1; i >=0; i--){

        //Tulostetaan listan alkiot
        fprintf(stdout, "%s", list[i]);
    }

    //Muistin vapautus ja ohjelman lopetus
    free(input);
    exit(1);
}

```



```

aleksi@ubuntu:~/Desktop/Käyttöjärjestelmät ja systeemiohjelmointi/T1$ gcc reverse.c -o reverse -Wall -Werror
aleksi@ubuntu:~/Desktop/Käyttöjärjestelmät ja systeemiohjelmointi/T1$ ./reverse
Type something to be reversed. End the typing by typing !end.

asd
dfg
ert
xcv
!end
The input reversed:
xcv
ert
dfg
asd
aleksi@ubuntu:~/Desktop/Käyttöjärjestelmät ja systeemiohjelmointi/T1$

```

Toisessa komennossa ohjelmalle annetaan komentoriviparametrina tiedoston nimi, jonka ohjelma lukee rivi kerrallaan ja tallentaa listaan. Lopuksi lista tulostetaan käänteisenä komentoriville. Aluksi määritellään muuttujat ja määritetään lukemisessa käytettävä tiedosto. Tämän jälkeen avataan while-loop, jossa jatketaan kunnes tiedosto loppuu ja kopioidaan uudet rivit listan alkioiksi. Tällä kertaa luetaan tiedostoa eikä stdin:iä. Kun tiedosto on käsitelty avataan for-loop, joka suoritetaan i-1 kertaa, aina i:tä pienentäen. Jokaisella iteraatiolla tulostetaan uusi rivi listasta. Lopuksi vapautetaan muisti ja suljetaan ohjelma sekä tiedosto.

```

//Funktio joka lukee tiedostosta ja kirjoittaa käänteisenä komentoriville
void reverseFromFile(char *filename) {
    char list[BUFFER][BUFFER];
    char *input = NULL;
    int i, count = 0;
    size_t size;
    FILE *file = NULL;

    //Määritetään tiedosto ja tehdään virheen käsittely
    file = fopen(filename, "r");
    if(file == NULL) {
        fprintf(stderr, "Error in opening file");
        exit(1);
    }

    //Luetaan tällä kertaa tiedostosta ja kopioidaan listaan
    while(getline(&input, &size, file) != -1) {
        strcpy(list[count], input);
        count++;
    }

    //Hyödynnetään reverseInTerminal-funktion kirjoitusta
    fprintf(stdout, "The file reversed:");
    for(i = count-1; i >= 0; i--){
aleksi@ubuntu:~/Desktop/Käyttöjärjestelmät ja systeemiohjelmointi/T1$ ./reverse input.txt
The file reversed:
words
of
order
correct
the
is
This
    }

```

Kolmannessa komennossa ohjelmalle annetaan komentoriviparametrina kirjoitettavan ja luettavan tiedoston nimi. Ensimmäinen tiedosto luetaan rivi kerrallaan läpi ja tallentaa listaan. Lopuksi lista kirjoitetaan käänteisenä toiseen tiedostoon. Aluksi määritellään muuttujat ja määritetään lukemisessa sekä kirjoittamisessa käytettävä tiedosto. Tämän jälkeen avataan while-loop, jossa jatketaan kunnes tiedosto loppuu ja kopioidaan uudet rivit listan alkioiksi. Kun tiedosto on käsitelty avataan for-loop, joka suoritetaan i-1 kertaa, aina i:tä pienentäen. Jokaisella iteraatiolla kirjoitetaan uusi rivi listasta tiedostoon. Lopuksi vapautetaan muisti ja suljetaan ohjelma sekä tiedostot.

```

//Funktio joka lukee tiedostosta ja kirjottaa toiseen tiedostoon
void reverseToFile(char *readfile, char *writefile) {
    char list[BUFFER][BUFFER];
    char *input = NULL;
    int i, count = 0;
    size_t size;
    FILE *read = NULL;
    FILE *write = NULL;

    //Määritetään luettava tiedosto ja virheenkäsittely
    read = fopen(readfile, "r");
    if(read == NULL) {
        fprintf(stderr, "Error in opening file");
        exit(1);
    }

    //Määritetään kirjoitettava tiedosto ja virheenkäsittely
    write = fopen(writefile, "w");
    if(write == NULL) {
        fprintf(stderr, "Error in opening file");
        exit(1);
    }

    //Hyödynnetään reverseFromFile-funktion lukemista
    while(getline(&input, &size, read) != -1) {
        strcpy(list[count], input);
        count++;
    }

    //Iteriadaan lista läpi ja kirjoitetaan tiedostoon
    for(i = count-1; i >= 0; i--){
        fprintf(write, "%s", list[i]);
    }

    //Muistin vapautus, tiedoston sulku ja ohjelman lopetus
    free(input);
    fclose(read);
    fclose(write);
    exit(1);
}

```

```

aleksi@ubuntu:~/Desktop/Käyttöjärjestelmät ja systeemiohjelmointi/T1$ ./reverse input.txt output.txt
aleksi@ubuntu:~/Desktop/Käyttöjärjestelmät ja systeemiohjelmointi/T1$

```

Output.txt-tiedostossa tulostus näkyy oikein

```

words
of
order
correct
the
is
This |

```

Lopuksi vielä main, joka ohjaa käskyt oikeisiin aliohjelmiin sekä välittää komentoriviparametrit.

```
int main(int argc, char *argv[]) {  
    //First use case where ./reverse is called  
    if (argc == 1){  
        reverseInTerminal();  
    }  
  
    //Second use case where ./reverse input.txt is called  
    if (argc == 2){  
        reverseFromFile(argv[1]);  
    }  
  
    //Third use case where ./reverse input.txt output.txt is called  
    if (argc == 3){  
        reverseToFile(argv[1], argv[2]);  
    }  
    return 0;  
}
```

2)

Toisen harjoitustehtävän tavoitteena oli luoda Unixin terminaalista tutut komennot:

cat  
grep  
zip  
unzip

Kyseiset komennot ovat varattuja linuxissa, joten toiminnot on nimetty seuraavasti:

my-cat  
my-grep  
my-zip  
my-unzip

Ohjelmat on ohjelmoitaessa käännetty seuraavalla komennolla:

aleksi@ubuntu:~/Desktop/Käyttöjärjestelmät ja systeemiohjelmointi/T2 gcc <my-ohjelma.c> -o <my-ohjelma> -Wall -lm -Werror

Syötä <>-sulkeiden sisään haluamasi c-tiedosto ja ohjelmanimi. Käännettyjen ohjelmien perään tulee liittää tiedosto esim. ./my-cat test1.txt. My-zip.c:lle tulee antaa komentoriviparametriksi tiedosto luettavaksi ja pakattavaksi esimerkiksi: ./my-zip input.txt > output.z. Tämän tiedoston voi avata my-unzip ohjelmalla.

Ohjelmat toimivat palautuksen aikaan halutulla tavalla ja ilman virheitä.

My-cat:

my-grep:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
//Puskurin koko 128 merkkiä
#define buffSize 128

int main(int argc, char *argv[]) {
    int foundWords = 0, i;
    char *findWord = argv[1], buffer[buffSize];
    FILE *file;
    //Komentoriviparametrit puuttuvat
    if (argc == 1) {
        printf("my-grep: searchterm [file ...]\n");
        exit(1);
    }
    //Luetaan stdinistä
    if (argc == 2) {
        printf("Reading stdin\n");
        while(fgets(buffer, buffSize, stdin)) {
            if(strstr(buffer, findWord) != NULL) {
                printf("%s", buffer);
            }
        }
    }
    //Tiedoston avaus ja avauksen virheentarkistus
    else {
        for(i = 2; i < argc; i++) {
            printf("Finding in file: %s\n", argv[i]);
            file = fopen(argv[i], "r");
            if(file == NULL){
                perror("Error: my-grep\n");
                exit(1);
            }
            //Tietojen tulostus
            while(fgets(buffer, buffSize, file) != NULL) {
                if(strstr(buffer, findWord) != NULL) {
                    printf("%s", buffer);
                    foundWords++;
                }
            }
            //Tieto käyttäjälle, jos sanalla ei löydy hakutuloksia
            if(foundWords == 0) {
                printf("No matches found for word: %s\n", findWord);
            }
        }
        //Tiedoston sulkeminen
        fclose(file);
    }
    return 0;
}
```

```

aleksi@ubuntu:~/Desktop/Käyttäjärjestelmät ja systeemiohjelmointi/T1$ gcc my-grep.c -o my-grep -Wall -lm -Werror
aleksi@ubuntu:~/Desktop/Käyttäjärjestelmät ja systeemiohjelmointi/T1$ ./my-grep a testi1.txt
Finding in file: testi1.txt
aaaaasdffjhjyyyyyykllllllllllllllllllggggnnnnnnnnnn
aleksi@ubuntu:~/Desktop/Käyttäjärjestelmät ja systeemiohjelmointi/T1$ ./my-grep b testi1.txt
Finding in file: testi1.txt
No matches found for word: b
aleksi@ubuntu:~/Desktop/Käyttäjärjestelmät ja systeemiohjelmointi/T1$ ./my-grep b testi1.txt
testi2.txt
Finding in file: testi1.txt
No matches found for word: b
Finding in file: testi2.txt
No matches found for word: b
aleksi@ubuntu:~/Desktop/Käyttäjärjestelmät ja systeemiohjelmointi/T1$

```

my-zip:

My-zip -komento ottaa sisäänsä merkkijonon, jonka se pakkaa lyhyempään muotoon.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char** argv){
    FILE* file;
    int c = 0, lastChar = -1, count = 0;
    //Virheentarkastus tyhjän komentoriviparametrin varalle
    if (argc < 2) {
        printf("my-zip: file1 [file2 ...]\n");
        exit(1);
    }
    for (int i = 1; i < argc; i++){
        // Avataan tiedosto ja tarkistetaan virheiden varalta
        file = fopen(argv[i], "r");
        if (file == NULL) {
            printf("Error opening file\n");
            exit(1);
        }
        // Käydään läpi tiedosto
        while ((c = fgetc(file)) != EOF){
            // Katsotaanko onko ensimmäinen merkki ja kasvatetaan muuttujien arvoja
            if (lastChar == -1) {
                lastChar = c;
                count++;
                // Merkki vaihtuu, kirjoitetaan tiedostoon
            } else if (c != lastChar) {
                // Kirjoittaa lukumäärän
                fwrite(&count, sizeof(int), 1, stdout);
                // Kirjoittaa kirjaimen
                fputc(lastChar, stdout);
                count = 1;
                // Sama merkki, laskuri kasvaa
            } else {
                count++;
            }
            lastChar = c;
        }
        fclose(file);
        // Tulostaa rivinvaihdon merkkijonon perään
        if (count > 0){
            fwrite(&count, sizeof(int), 1, stdout);
            fputc(lastChar, stdout);
        }
        return 0;
    }
}

```



```
aleksi@ubuntu:~/Desktop/Käyttöjärjestelmät ja systeemiohjelmointi/T1$ gcc my-zip.c -o my-zip -Wall -Werror
aleksi@ubuntu:~/Desktop/Käyttöjärjestelmät ja systeemiohjelmointi/T1$ ./my-zip testi1.txt > file.z
aleksi@ubuntu:~/Desktop/Käyttöjärjestelmät ja systeemiohjelmointi/T1$
```

My-unzip:

My-unzip -komento purkaa aiemmin pakatut tiedostot. Se lukee tiedostoja merkki kerrallaan ja tulostaa terminaaliin.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char** argv){
    FILE* file;
    int c, count;

    //Virheentarkastus tyhjän komentoriviparametrin varalle
    if (argc == 1){
        printf("my-unzip: file1 [file2...]\n");
        exit(1);
    }
    for (int i = 1; i < argc; i++){
        //Tiedoston avaus ja avauksen virheentarkistus
        file = fopen(argv[i], "r");
        if (file == NULL) {
            printf("Error opening file\n");
            exit(1);
        }

        // Käydään tiedosto läpi
        while (fread(&count, sizeof(int), 1, file) == 1){
            c = fgetc(file);
            // Tulostetaan j-määrä samaa merkkiä
            for (int j = 0; j < count; j++){
                printf("%c", c);
            }
        }
        fclose(file);
    }
    return 0;
}
```

```
aleksi@ubuntu:~/Desktop/Käyttöjärjestelmät ja systeemiohjelmointi/T1$ gcc my-unzip.c -o my-unzip -Wall -Werror
aleksi@ubuntu:~/Desktop/Käyttöjärjestelmät ja systeemiohjelmointi/T1$ ./my-unzip file.z
aaaaasdffjhjyyyyyykllllllllllllllllllggggnnnnnnnnnnn
aleksi@ubuntu:~/Desktop/Käyttöjärjestelmät ja systeemiohjelmointi/T1$
```

4)

Neljännessä tehtävänä oli harjoitella käyttöjärjestelmän järjestelmäkutsuja, luomalla oma järjestelmäkutsu `getreadcount()`, joka laskee ja palauttaa luvun, kuinka monesti `read()`-kutsua kutsutaan. Järjestelmäkutsua olisi voinut laajentaa myös muihin kutsuihin ja nollaamaan laskurin tarvittaessa. Harjoitustyössä jouduin luomaan uuden tiedoston `getreadcount.c`. Tämän lisäksi jouduin muokkaamaan jo valmiita tiedostoja: `syscall.c`, `syscall.h`, `user.h`, `sysfile.c`, `sysproc.c`, `usys.S` ja `Makefileä`. Itse tuotettu koodi on kommentoitu //Omaa koodia -tekstillä

`getreadcount.c`

`getreadcount` on hyvin yksinkertainen ohjelma jonka tarkoituksena on kutsua kyseistä järjestelmäkutsua. Se toimiikin vain käyttäjän ja käyttöjärjestelmän välisenä viestinviejänä. `Getreadcount` tulostaa käyttäjälle tiedon, kuinka monta kertaa `read`-kutsua on kutsuttu.

```
// Omaa koodia
#include "fcntl.h"
#include "types.h"
#include "stat.h"
#include "user.h"

int main(void){
    printf(1, "read -system call was called %d times\n", getreadcount());
    exit();
}
```

`syscall.c`

`syscall`-tiedostossa on määritelty järjestelmäkutsu.

```

// Oma koodia
extern int sys_getreadcount(void);

static int (*syscalls[])(void) = {
[SYS_fork]      sys_fork,
[SYS_exit]      sys_exit,
[SYS_wait]      sys_wait,
[SYS_pipe]      sys_pipe,
[SYS_read]      sys_read,
[SYS_kill]      sys_kill,
[SYS_exec]      sys_exec,
[SYS_fstat]     sys_fstat,
[SYS_chdir]     sys_chdir,
[SYS_dup]       sys_dup,
[SYS_getpid]    sys_getpid,
[SYS_sbrk]      sys_sbrk,
[SYS_sleep]     sys_sleep,
[SYS_uptime]    sys_uptime,
[SYS_open]      sys_open,
[SYS_write]     sys_write,
[SYS_mknod]     sys_mknod,
[SYS_unlink]    sys_unlink,
[SYS_link]      sys_link,
[SYS_mkdir]     sys_mkdir,
[SYS_close]     sys_close,
// Oma koodia
[SYS_getreadcount] sys_getreadcount,
};

```

## syscall.h

syscallin header-tiedostossa on myös määritelty järjestelmäkutsu. Valitsin oman järjestelmäkutsulleni seuraavan indeksin, eli 22.

```

// System call numbers
#define SYS_fork    1
#define SYS_exit    2
#define SYS_wait    3
#define SYS_pipe    4
#define SYS_read    5
#define SYS_kill    6
#define SYS_exec    7
#define SYS_fstat   8
#define SYS_chdir   9
#define SYS_dup     10
#define SYS_getpid  11
#define SYS_sbrk    12
#define SYS_sleep   13
#define SYS_uptime  14
#define SYS_open    15
#define SYS_write   16
#define SYS_mknod   17
#define SYS_unlink  18
#define SYS_link    19
#define SYS_mkdir   20
#define SYS_close   21
// Omaa koodia
#define SYS_getreadcount  22

```

user.h

user.h-tiedostoon piti määritellä lisätty järjestelmäkutsu.

---

```

struct stat;
struct rtcdate;

// system calls
int fork(void);
int exit(void) __attribute__((noreturn));
int wait(void);
int pipe(int*);
int write(int, const void*, int);
int read(int, void*, int);
int close(int);
int kill(int);
int exec(char*, char**);
int open(const char*, int);
int mknod(const char*, short, short);
int unlink(const char*);
int fstat(int fd, struct stat*);
int link(const char*, const char*);
int mkdir(const char*);
int chdir(const char*);
int dup(int);
int getpid(void);
char* sbrk(int);
int sleep(int);
int uptime(void);
// Omaa koodia
int getreadcount(void);

```

## sysfile.c

Sysfile-tiedostoon lisäsin muuttujan callCount, joka pitää yllä lukua kuinka monta kertaa kutsua on kutsuttu. Read-funktion sisälle lisäsin palan koodia, joka kasvattaa laskuria joka kerta kun funktiota kutsutaan.

```
// Oma koodia
int callCount = 0;

int
sys_read(void)
{
    struct file *f;
    int n;
    char *p;

    // Oma koodia
    callCount++;

    if(argfd(0, 0, &f) < 0 || argint(2, &n) < 0 || argptr(1, &p, n) < 0)
        return -1;
    return fileread(f, p, n);
}
```

## sysproc.c

Sysproc-tiedostoon kirjoitin funtion, joka kuuntelee getreadcount-järjestelmäkutsua ja palauttaa callCountin ylläpitämän lukeman. CallCount tuli myös määritellä kyseisessä tiedostossa, jotta ohjelma osasi käyttää sitä.

```
// Oma koodia
extern int callCount;

int
sys_getreadcount(void)
{
    return callCount;
}
```

---

## usys.S

usys-tiedostoon tuli määritellä makro getreadcountia varten.

```
SYSCALL(unlink)
SYSCALL(fstat)
SYSCALL(link)
SYSCALL(mkdir)
SYSCALL(chdir)
SYSCALL(dup)
SYSCALL(getpid)
SYSCALL(sbrk)
SYSCALL(sleep)
SYSCALL(uptime)
// Omaa koodia
SYSCALL(getreadcount)
```

## Makefile

Makefileen tuli lisätä mainita luomasta tiedostostani kahteen eri paikkaan, jotta xv6 kääntyy oikein.

```
UPROGS=\
    _cat\
    _echo\
    _forktest\
    _grep\
    _init\
    _kill\
    _ln\
    _ls\
    _mkdir\
    _rm\
    _sh\
    _stressfs\
    _usertests\
    _wc\
    _zombie\
    _getreadcount\
```

```
EXTRA=\
mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
ln.c ls.c mkdir.c rm.c stressfs.c usertests.c getreadcount.c wc.c zombie.c\
printf.c umalloc.c\
README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
.gdbinit.tmpl gdbutil\
```

Kun ohjelma ajetaan komennolla `make qemu` ja sille annetaan käsky `getreadcount`, ohjelma palauttaa järjestelmäkutsujen määrän ja ilmoittaa sen käyttäjälle.

QEMU



SeaBIOS (version 1.10.2-1ubuntu1)

iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+1FF8DDD0+1FECDDD0 C980

Booting from Hard Disk...

cpu1: starting 1

cpu0: starting 0

sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58

init: starting sh

\$ getreadcount

read -system call was called 13 times

\$ getreadcount

read -system call was called 26 times

\$ getreadcount

read -system call was called 39 times

\$