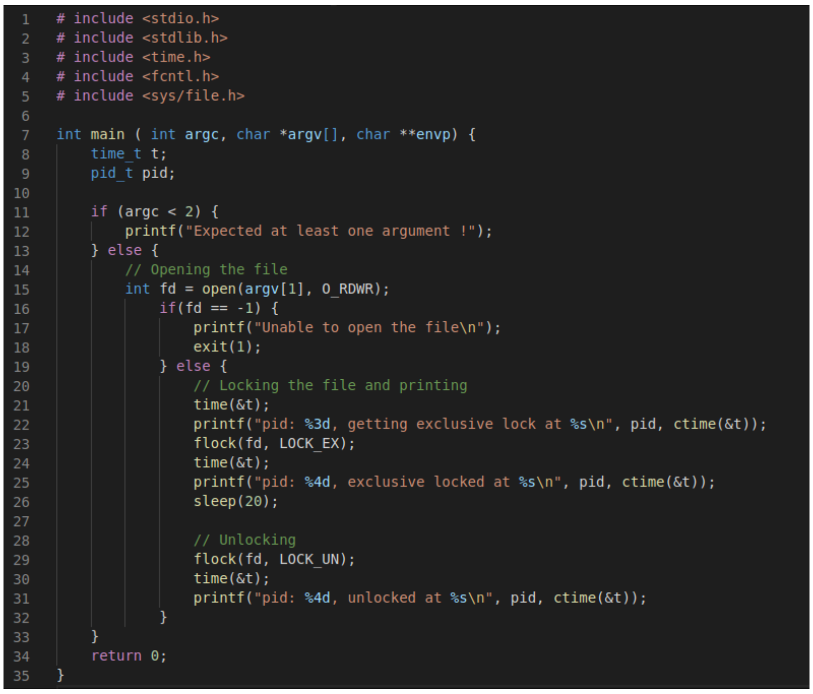
1. File locking

Write a program which acquires a write (exclusive) lock to a file, waits a bit, releases the lock and then acquires a read (shared) lock to the same file, again waits a bit and then quits. Experiment running the program twice simultaneously. Also try to access the file while it is locked with other programs (such as "cat"). Useful functions: *fcntl()* or *flock()*.

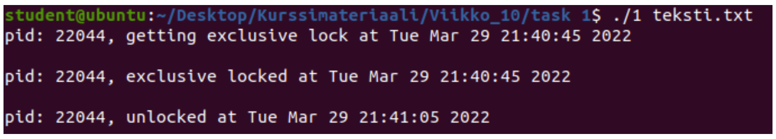
Include suitable prints with a pid and timestamps (using e.g. functions *time* and *ctime*) to help follow what happens in your test runs.

Vastaus:



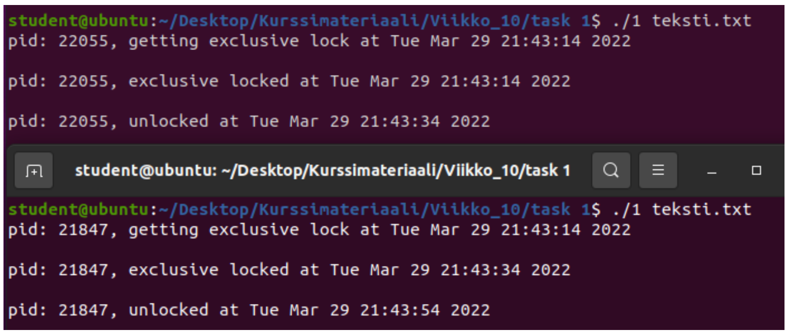
```c
# include <stdio.h>
# include <stdlib.h>
# include <time.h>
# include <fcntl.h>
# include <sys/file.h>

int main ( int argc, char *argv[], char **envp) {
    time_t t;
    pid_t pid;

    if (argc < 2) {
        printf("Expected at least one argument !");
    } else {
        // Opening the file
        int fd = open(argv[1], O_RDWR);
            if(fd == -1) {
                printf("Unable to open the file\n");
                exit(1);
            } else {
                // Locking the file and printing
                time(&t);
                printf("pid: %3d, getting exclusive lock at %s\n", pid, ctime(&t));
                flock(fd, LOCK_EX);
                time(&t);
                printf("pid: %4d, exclusive locked at %s\n", pid, ctime(&t));
                sleep(20);

                // Unlocking
                flock(fd, LOCK_UN);
                time(&t);
                printf("pid: %4d, unlocked at %s\n", pid, ctime(&t));
            }
    }
    return 0;
}
```

Juostiin koodi normaalisti



```
student@ubuntu:~/Desktop/Kurssimateriaali/Viikko_10/task 1$ ./1 teksti.txt
pid: 22044, getting exclusive lock at Tue Mar 29 21:40:45 2022

pid: 22044, exclusive locked at Tue Mar 29 21:40:45 2022

pid: 22044, unlocked at Tue Mar 29 21:41:05 2022
```

Juostiin koodi 2 kertaa samaan aikaan

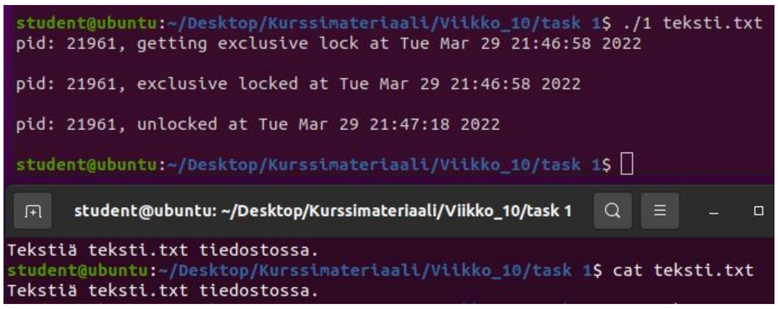


```
student@ubuntu:~/Desktop/Kurssimateriaali/Viikko_10/task 1$ ./1 teksti.txt
pid: 22055, getting exclusive lock at Tue Mar 29 21:43:14 2022

pid: 22055, exclusive locked at Tue Mar 29 21:43:14 2022

pid: 22055, unlocked at Tue Mar 29 21:43:34 2022
```

student@ubuntu: ~/Desktop/Kurssimateriaali/Viikko_10/task 1

```
student@ubuntu:~/Desktop/Kurssimateriaali/Viikko_10/task 1$ ./1 teksti.txt
pid: 21847, getting exclusive lock at Tue Mar 29 21:43:14 2022

pid: 21847, exclusive locked at Tue Mar 29 21:43:34 2022

pid: 21847, unlocked at Tue Mar 29 21:43:54 2022
```

Juostiin koodi ja cat



```
student@ubuntu:~/Desktop/Kurssimateriaali/Viikko_10/task 1$ ./1 teksti.txt
pid: 21961, getting exclusive lock at Tue Mar 29 21:46:58 2022

pid: 21961, exclusive locked at Tue Mar 29 21:46:58 2022

pid: 21961, unlocked at Tue Mar 29 21:47:18 2022

student@ubuntu:~/Desktop/Kurssimateriaali/Viikko_10/task 1$
```

```
Tekstiä teksti.txt tiedostossa.
student@ubuntu:~/Desktop/Kurssimateriaali/Viikko_10/task 1$ cat teksti.txt
Tekstiä teksti.txt tiedostossa.
```

2. Executing a program from a process

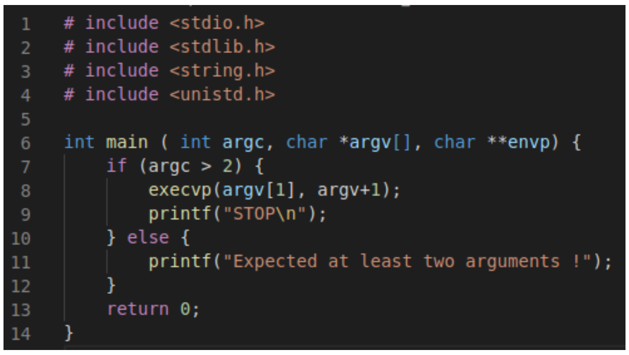
Write a program which uses *system()* and *exec()* system calls to execute a program with parameters (select a suitable system call from the *exec*-family). Test it with the program *"ls"*, e.g. "*ls -l*".

Why doesn't the process continue after *exec()* but does so after *system()*? (Add a *printf* after the call to *exec* to confirm this.)

Now test the program with commands "*ls -l *.c*" and "*ls -l *.c*" and compare the results. Why doesn't the latter work with *exec()* as you might expect? (hint: man glob.)

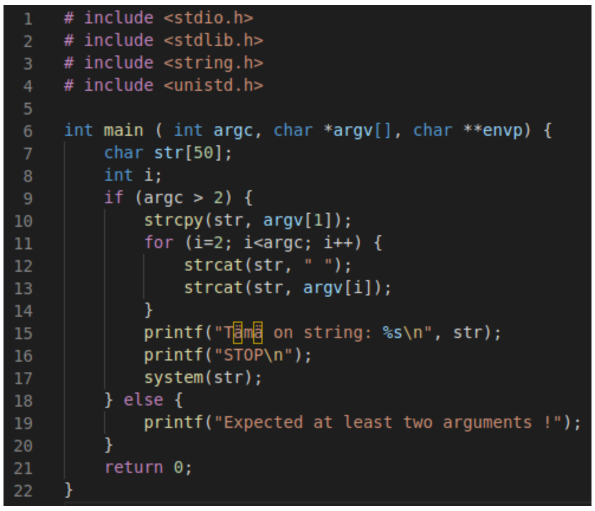
Vastaus:

Execciä käyttävä ohjelma



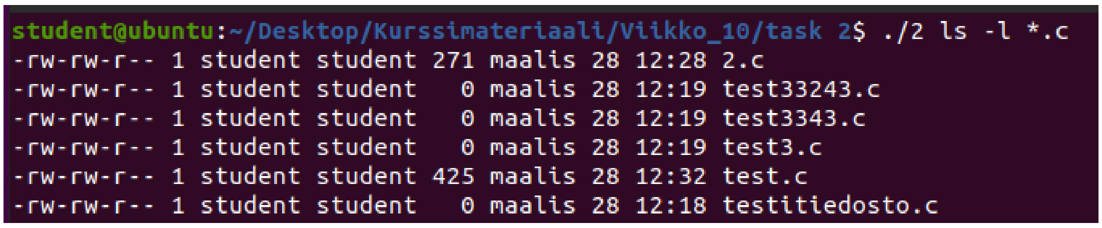
```c
# include <stdio.h>
# include <stdlib.h>
# include <string.h>
# include <unistd.h>

int main ( int argc, char *argv[], char **envp) {
    if (argc > 2) {
        execvp(argv[1], argv+1);
        printf("STOP\n");
    } else {
        printf("Expected at least two arguments !");
    }
    return 0;
}
```

Systemiä käyttävä ohjelma (sisältää yhden debug printin)

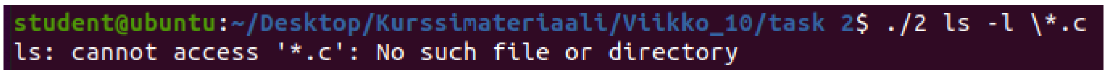


```c
# include <stdio.h>
# include <stdlib.h>
# include <string.h>
# include <unistd.h>

int main ( int argc, char *argv[], char **envp) {
    char str[50];
    int i;
    if (argc > 2) {
        strcpy(str, argv[1]);
        for (i=2; i<argc; i++) {
            strcat(str, " ");
            strcat(str, argv[i]);
        }
        printf("Tämä on string: %s\n", str);
        printf("STOP\n");
        system(str);
    } else {
        printf("Expected at least two arguments !");
    }
    return 0;
}
```

Exec ohjelmalla



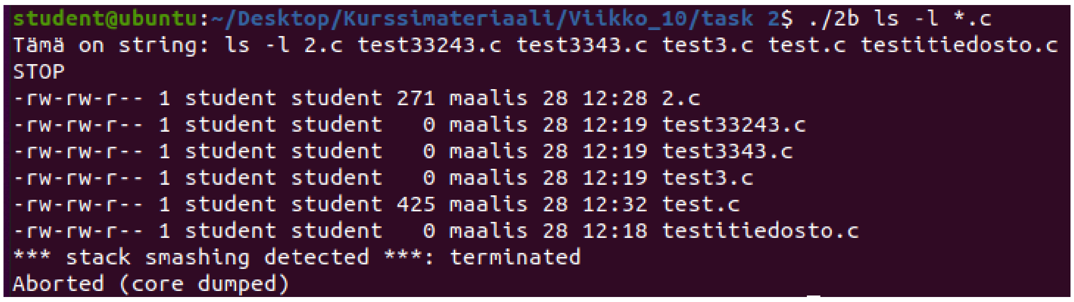
```
student@ubuntu:~/Desktop/Kurssimateriaali/Viikko_10/task 2$ ./2 ls -l *.c
-rw-rw-r-- 1 student student 271 maalis 28 12:28 2.c
-rw-rw-r-- 1 student student   0 maalis 28 12:19 test33243.c
-rw-rw-r-- 1 student student   0 maalis 28 12:19 test3343.c
-rw-rw-r-- 1 student student   0 maalis 28 12:19 test3.c
-rw-rw-r-- 1 student student 425 maalis 28 12:32 test.c
-rw-rw-r-- 1 student student   0 maalis 28 12:18 testitiedosto.c
```



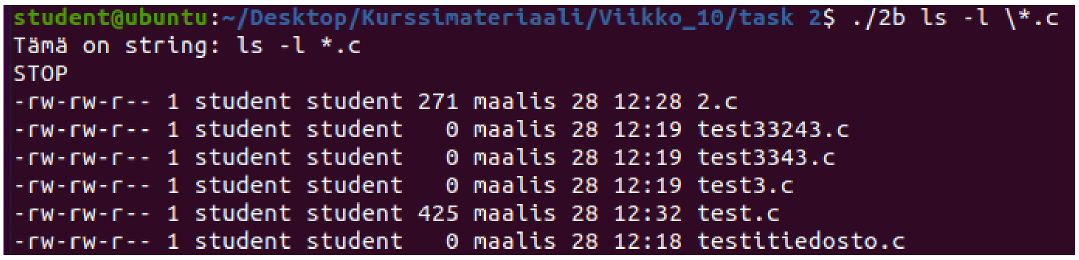
```
student@ubuntu:~/Desktop/Kurssimateriaali/Viikko_10/task 2$ ./2 ls -l \*.c
ls: cannot access '*.c': No such file or directory
```

exec kutsut kirjoittavat prosessi kutsun kuvan päälle (image of the calling process), jonka takia printf ei materialisoidu.

System ohjelmalla



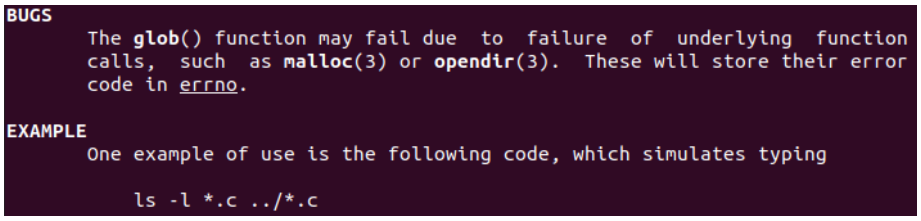
```
student@ubuntu:~/Desktop/Kurssimateriaali/Viikko_10/task 2$ ./2b ls -l *.c
Tämä on string: ls -l 2.c test33243.c test3343.c test3.c test.c testitiedosto.c
STOP
-rw-rw-r-- 1 student student 271 maalis 28 12:28 2.c
-rw-rw-r-- 1 student student   0 maalis 28 12:19 test33243.c
-rw-rw-r-- 1 student student   0 maalis 28 12:19 test3343.c
-rw-rw-r-- 1 student student   0 maalis 28 12:19 test3.c
-rw-rw-r-- 1 student student 425 maalis 28 12:32 test.c
-rw-rw-r-- 1 student student   0 maalis 28 12:18 testitiedosto.c
*** stack smashing detected ***: terminated
Aborted (core dumped)
```



```
student@ubuntu:~/Desktop/Kurssimateriaali/Viikko_10/task 2$ ./2b ls -l \*.c
Tämä on string: ls -l *.c
STOP
-rw-rw-r-- 1 student student 271 maalis 28 12:28 2.c
-rw-rw-r-- 1 student student   0 maalis 28 12:19 test33243.c
-rw-rw-r-- 1 student student   0 maalis 28 12:19 test3343.c
-rw-rw-r-- 1 student student   0 maalis 28 12:19 test3.c
-rw-rw-r-- 1 student student 425 maalis 28 12:32 test.c
-rw-rw-r-- 1 student student   0 maalis 28 12:18 testitiedosto.c
```

Stack smashing aiheutti assarin kanssa ihmetystä, mutta päätettiin olla etsimättä aiheuttajaa, ettei mene koko harkkaryhmä siihen. Epäiltiin johtuvan stringistä, joka menee system funktioon, mutta kuten debug printistä näkyy ei siinä näy mitään poikkeavaa. Kasvatin myös str muuttujan koon 150, joka ei korjannut asiaa.

Manuaalissa luki näin



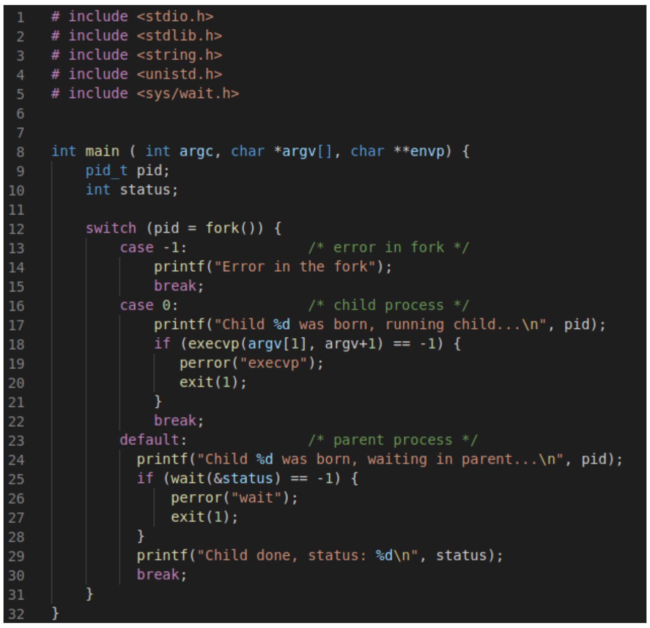
```
BUGS
       The glob() function may fail due  to  failure  of  underlying  function
       calls,  such  as malloc(3) or opendir(3).  These will store their error
       code in errno.

EXAMPLE
       One example of use is the following code, which simulates typing

           ls -l *.c ../*.c
```

3. Passing parameters to the process

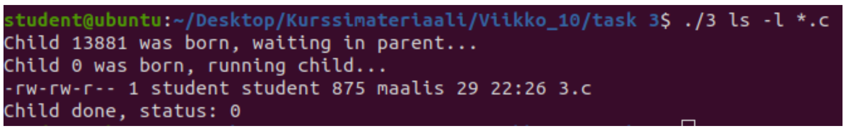
Write a program to execute a command and all its parameters in another process. The main process prints the PID of the child process, waits for it to terminate and then prints its status. Important commands are *fork()*, *exec()* and *wait()*.

Test it, for example, with the same commands as in Task 2.

Vastaus:



```c
# include <stdio.h>
# include <stdlib.h>
# include <string.h>
# include <unistd.h>
# include <sys/wait.h>


int main ( int argc, char *argv[], char **envp) {
    pid_t pid;
    int status;

    switch (pid = fork()) {
        case -1:             /* error in fork */
            printf("Error in the fork");
            break;
        case 0:              /* child process */
            printf("Child %d was born, running child...\n", pid);
            if (execvp(argv[1], argv+1) == -1) {
               perror("execvp");
               exit(1);
            }
            break;
        default:             /* parent process */
          printf("Child %d was born, waiting in parent...\n", pid);
          if (wait(&status) == -1) {
              perror("wait");
              exit(1);
          }
          printf("Child done, status: %d\n", status);
          break;
    }
}
```



```
student@ubuntu:~/Desktop/Kurssimateriaali/Viikko_10/task 3$ ./3 ls -l *.c
Child 13881 was born, waiting in parent...
Child 0 was born, running child...
-rw-rw-r-- 1 student student 875 maalis 29 22:26 3.c
Child done, status: 0
```

Koska tämäkin ohjelma käyttää exec funktiota ls -l *.c ei toimi halutulla tavalla.

4. Input/output streams for processes

Modify the previous program so that you can specify files to be used as input and/or output for a selected program (also commands possible here). Using symbol "-" stdin/stdout should still be used in the program. Hint: *open()* (or *fopen()*) file, then *dup2()* the file descriptor to stdin or stdout if a file is given in the command line. Running (using) the program would be like

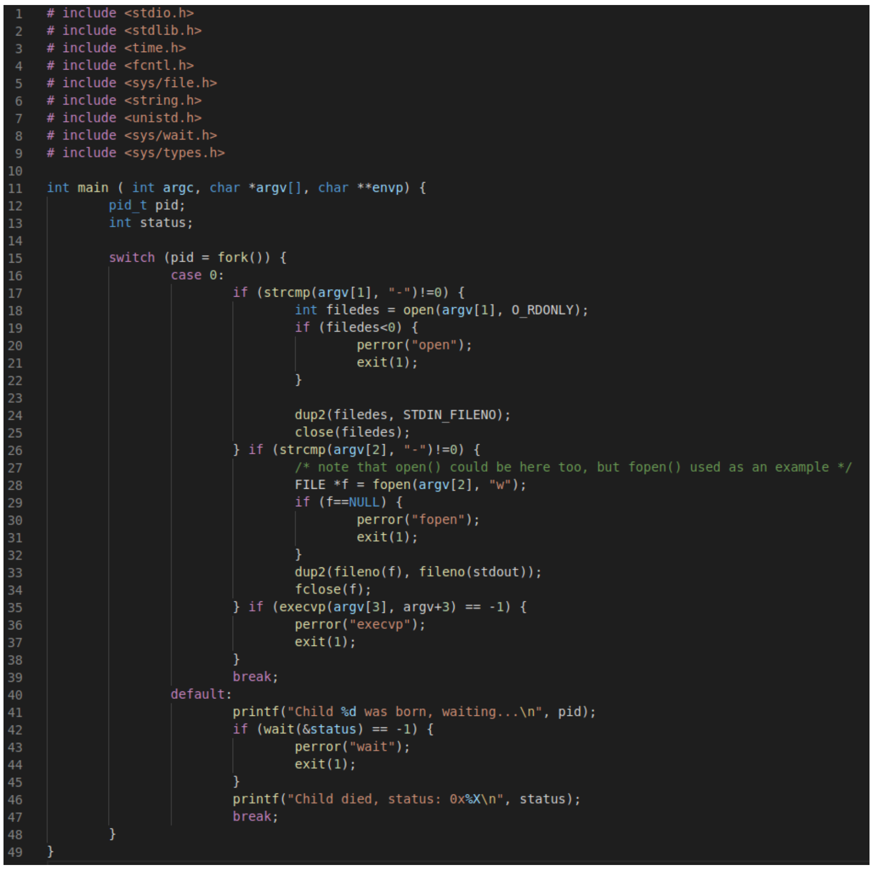
Usage: $ ./ex4 <file-for-stdin> <file-for-stdout> <command> [arguments]

For example: (input from file.txt, output to stdout, run command cat)
$ ./ex4 file.txt - cat

or as the second example (input from file.txt, output to checksum.txt, run command md5sum)

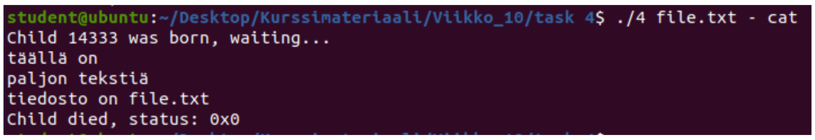
$ ./ex4 file.txt checksum.txt md5sum

Vastaus:



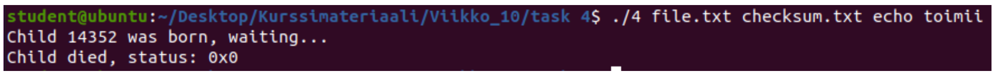
```c
# include <stdio.h>
# include <stdlib.h>
# include <time.h>
# include <fcntl.h>
# include <sys/file.h>
# include <string.h>
# include <unistd.h>
# include <sys/wait.h>
# include <sys/types.h>

int main ( int argc, char *argv[], char **envp) {
        pid_t pid;
        int status;

        switch (pid = fork()) {
                case 0:
                        if (strcmp(argv[1], "-")!=0) {
                                int filedes = open(argv[1], O_RDONLY);
                                if (filedes<0) {
                                        perror("open");
                                        exit(1);
                                }

                                dup2(filedes, STDIN_FILENO);
                                close(filedes);
                        } if (strcmp(argv[2], "-")!=0) {
                                /* note that open() could be here too, but fopen() used as an example */
                                FILE *f = fopen(argv[2], "w");
                                if (f==NULL) {
                                        perror("fopen");
                                        exit(1);
                                }
                                dup2(fileno(f), fileno(stdout));
                                fclose(f);
                        } if (execvp(argv[3], argv+3) == -1) {
                                perror("execvp");
                                exit(1);
                        }
                        break;
                default:
                        printf("Child %d was born, waiting...\n", pid);
                        if (wait(&status) == -1) {
                                perror("wait");
                                exit(1);
                        }
                        printf("Child died, status: 0x%X\n", status);
                        break;
        }
}
```

Juostaan cat kanssa

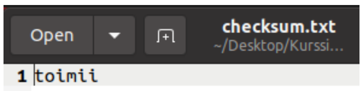


```
student@ubuntu:~/Desktop/Kurssimateriaali/Viikko_10/task 4$ ./4 file.txt - cat
Child 14333 was born, waiting...
täällä on
paljon tekstiä
tiedosto on file.txt
Child died, status: 0x0
```

Juostaan echon kanssa, että nähdään toimiiko.

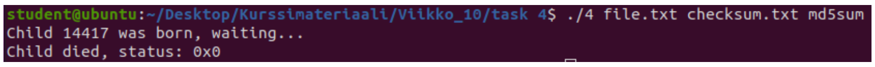


```
student@ubuntu:~/Desktop/Kurssimateriaali/Viikko_10/task 4$ ./4 file.txt checksum.txt echo toimii
Child 14352 was born, waiting...
Child died, status: 0x0
```

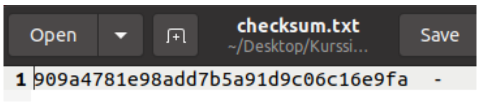


```
1 toimii
```

Toimii myös md5sum kanssa



```
student@ubuntu:~/Desktop/Kurssimateriaali/Viikko_10/task 4$ ./4 file.txt checksum.txt md5sum
Child 14417 was born, waiting...
Child died, status: 0x0
```



```
1 909a4781e98add7b5a91d9c06c16e9fa  -
```