# Σύντομη Περιγραφή Της Bonus Εργασίας Θεωρίας Read/Fread

Ο κώδικας υλοποιείται από δύο διαφορετικά προγράμματα σε γλώσσα C, ένα για τη περίπτωση της read και ένα για την fread και λειτουργούν με παρόμοιο τρόπο. Εξετάζουν ότι έχει δωθεί η σωστή είσοδος μοτίβου (αναφαίρεται ως 'sequential/irregular pattern' εντός του προγράμματος) στο stdin, εκτελούν την ανάγνωση του κατάλληλου αριθμού bytes από το αρχείο(small/large file) που δόθηκε στο stdin και εκτυπώνουν αυτά τα bytes στο stdout μέσω της μεταβλητής 'fd'(file descriptor), η οποία περιέχει τα δεδομένα(χαρακτήρες στη περίπτωσή μας) του φακέλου.

**File size:** smallFile: 1.5K, largeFile: 461M

Η γενική μορφή της εντολής που εκτελώ στο terminal του Linux συστήματός μου είναι:

stdin: *time ./myProgram file pattern*

Στο argv[0] αποθηκεύεται το όνομα του προγράμματος, στο argv[1] το όνομα του φάκελου από τον οποίο θα διαβαστούν τα δεδομένα και στο argv[2] το μοτίβο ανάγνωσης bytes.

Οι υπόλοιπες λεπτομέρειες είναι καταγεγραμμένες ως σχόλια στο κώδικα.

Στο stdout(οθόνη) παρατητείται πως το μέγεθος του φακέλου(small/large) αλλά και το μοτίβο(sequential/irregular) παίζουν σημαντικό ρόλο στο χρόνο που παίρνει η κάθε εκτέλεση [wallclock(real time)/user/kernel(system)] των εντολών:

1. *time ./read smallFile.txt sequential*

```
xrathws@Ubuntu22:~/Documents$ time ./read smallFile.txt sequential


real    0m0.002s
user    0m0.002s
sys     0m0.000s
```

2. *time ./read smallFile.txt irregular*

```
xrathws@Ubuntu22:~/Documents$ time ./read smallFile.txt irregular


real    0m0.003s
user    0m0.000s
sys     0m0.003s
```

3. *time ./read largeFile.txt sequential*

```
xrathws@Ubuntu22:~/Documents$ time ./read largeFile.txt sequential


real    0m0.004s
user    0m0.000s
sys     0m0.005s
```

4. *time ./read largeFile.txt irregular*

```
xrathws@Ubuntu22:~/Documents$ time ./read largeFile.txt irregular


real    0m0.010s
user    0m0.000s
sys     0m0.009s
```

5. *time ./fread smallFile.txt sequential*

```
xrathws@Ubuntu22:~/Documents$ time ./fread smallFile.txt sequential


real    0m0.001s
user    0m0.001s
sys     0m0.000s
```

6. *time ./fread smallFile.txt irregular*

```
xrathws@Ubuntu22:~/Documents$ time ./fread smallFile.txt irregular


real    0m0.002s
user    0m0.002s
sys     0m0.000s
```

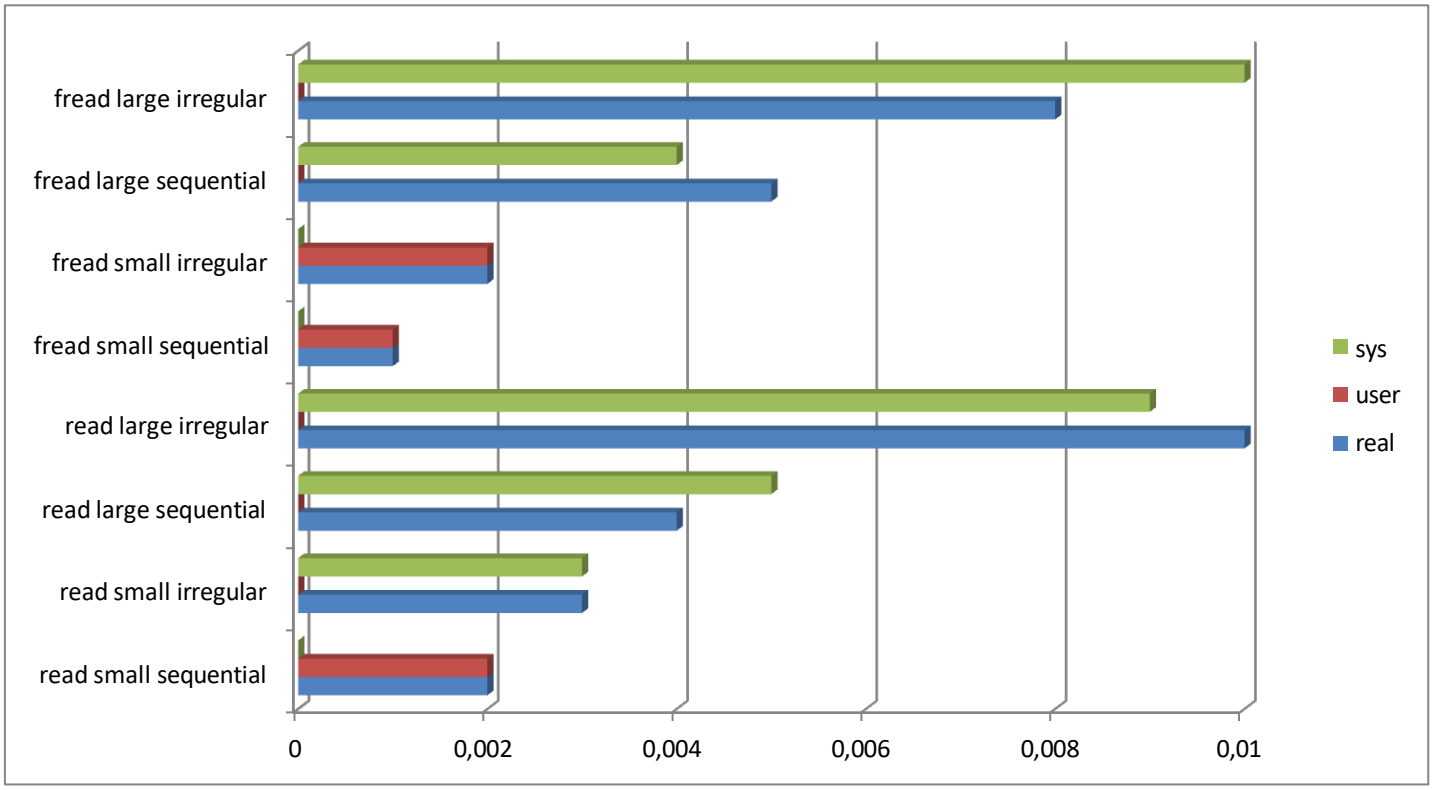
7. *time ./fread largeFile.txt sequential*

```
xrathws@Ubuntu22:~/Documents$ time ./fread largeFile.txt sequential


real    0m0.005s
user    0m0.000s
sys     0m0.004s
```

8. *time ./fread largeFile.txt irregular*

```
xrathws@Ubuntu22:~/Documents$ time ./fread largeFile.txt irregular


real    0m0.008s
user    0m0.000s
sys     0m0.010s
```

## Γράφημα:




Στο γράφημα γίνονται εμφανείς οι παρατηρήσεις που έγιναν παραπάνω, όπως και το ότι η read χρησιμοποιεί περισσότερο χρόνο από την fread.

Screenshots του κώδικα της fread:

```c
//it2022134 EXARCHOU ATHANASIOS

//included libraries
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

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

    int i; //counter for the for-loops
    FILE* fd; //file descriptor
    char *c = NULL; //represents the buffer (initialized at NULL)

    fd = fopen(argv[1], "r"); //opens the file

    //checks for errors
    if (fd == NULL) {
        //informs the user via appropriate message
        printf("There was an error during the process of opening the requested file. Exiting...\n");
        abort(); //exits the program
    }

    c = (char *) malloc (1000 * sizeof(char)); //allocates needed memory

    //checks for errors
    if (c == NULL) {
        //informs the user via appropriate message
        printf("There was an error during the process of memory allocation. Exiting...\n");
        fclose(fd); //closes the file before exiting
        abort(); //exits the program
    }

    //sequential pattern:

    if (strcmp(argv[2], "sequential") == 0) {

    //1000 repetitions(reads 1000 bytes from the file descriptor)
    for (i = 1; i < 1000; i++) {
        fread(c, sizeof(char), 1, fd);
        //printf("%s", c);
    }
    printf("\n");
```

```c
    }

    //irregular pattern:

    else if (strcmp(argv[2], "irregular") == 0) {

    for (i = 0; i < 500; i++) { //500 repetitions

        fseek(fd, i, SEEK_SET); //seeks to the start of the file descriptor
        fread(c, sizeof(char), 1, fd);
        //printf("%c", *c);
        fseek(fd, -i-2, SEEK_END); //seeks to the end of the file descriptor
        fread(c, sizeof(char), 1, fd);
        //printf("%c", *c);
    }
    printf("\n");
    }
    //case where the pattern is invalid:
    else {
        printf("Invalid pattern! Exiting...\n"); //informs the user via appropriate message
        abort(); //exits the program
    }

    fclose(fd); //closes the file
    free(c); //deallocates the previously allocated space

    return 0; //exits once the program has finished executing succesfully
}
```

Screenshots του κώδικα της read:

```c
//it2022134 EXARCHOU ATHANASIOS

//included libraries
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <errno.h>
#include <dirent.h>
#include <string.h>
#include <unistd.h>

extern int errno; //indicates whether something went wrong

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

    int i; //counter for the for-loops
    char *c = NULL; //represents the buffer (initialized at NULL)
    int fd;

    fd = open(argv[1], O_RDONLY, 00400); //opens the file

    //checks for errors
    if (fd == -1) {
        //informs the user via appropriate message
        printf("There was an error during the process of opening the requested file. Exiting...\n");
        abort(); //exits the program
    }

    c = (char *) malloc (1000 * sizeof(char)); //allocates needed memory

    //checks for errors
    if (c == NULL) {
        //informs the user via appropriate message
        printf("There was an error during the process of memory allocation. Exiting...\n");
        close(fd); //closes the file before exiting
        abort(); //exits the program
    }

    //sequential pattern:

    if (strcmp(argv[2], "sequential") == 0) {
```

```c
    //1000 repetitions(reads 1000 bytes from the file descriptor)
    for (i = 1; i < 1000; i++) {
        read(fd, c, 1);
        //printf("%s", c);
    }
    printf("\n");
    }
    //irregular pattern:
    else if (strcmp(argv[2], "irregular") == 0) {

    for (i = 0; i < 500; i++) { //500 repetitions

        lseek(fd, i, SEEK_SET); //seeks to the start of the file descriptor
        read(fd, c, 1);
        //printf("%c", *c);
        lseek(fd, -i-2, SEEK_END); //seeks to the end of the file descriptor
        read(fd, c, 1);
        //printf("%c", *c);
    }
    printf("\n");
    }
    //case where the pattern is invalid:
    else {
        printf("Invalid pattern! Exiting...\n"); //informs the user via appropriate message
        abort(); //exits the program
    }

    close(fd); //closes the file
    free(c); //deallocates the previously allocated space

    return 0; //exits once the program has finished executing succesfully
}
```