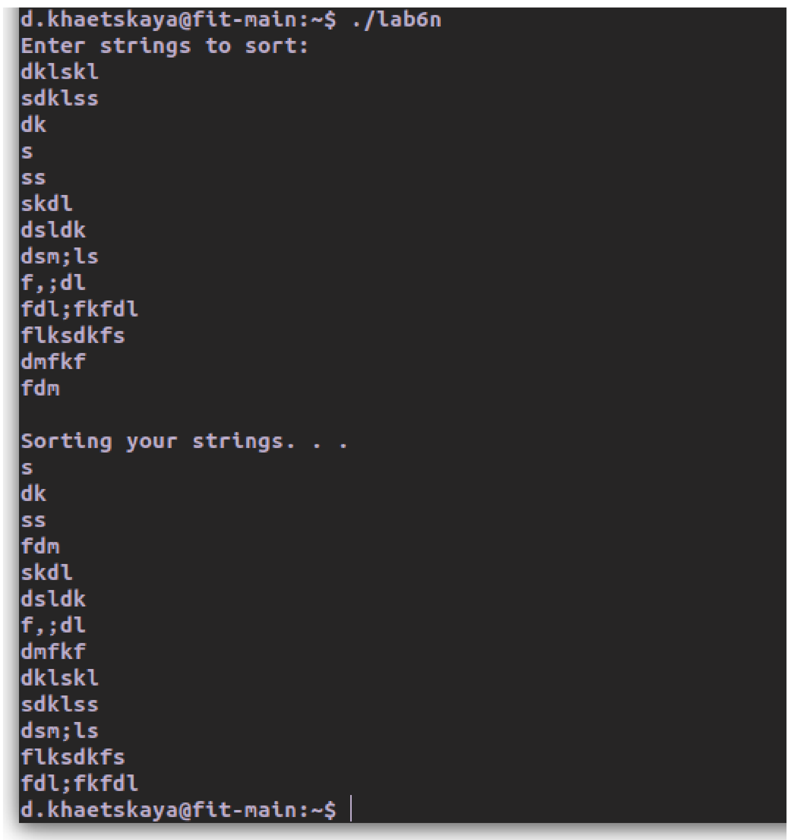
OS2 Lab6

checkbox	✓
due date	@September 15, 2021
class	OS
Status	approved

Task

```
Реализуйте уникальный алгоритм сортировки sleepsort с
 асимптотикой O(N) (по процессорному времени).
На стандартный вход программы подается не более 100 ст
рок различной длины. Вам необходимо вывести эти строк
и, отсортированные по длине. Строки одинаковой длины м
огут выводиться в произвольном порядке.

Для каждой входной строки, создайте нить и передайте е
й эту строку в качестве параметра. Нить должна вызвать
sleep(2) или usleep(2) с параметром, пропорциональным
 длине этой строки. Затем нить выводит строку в станда
ртный поток вывода и завершается. Не следует выбирать
 коэффициент пропорциональности слишком маленьким, вы
 рискуете получить некорректную сортировку.
```



```
d.khaetskaya@fit-main:~$ ./lab6n
Enter strings to sort:
dklskl
sdklss
dk
s
ss
skdl
dsldk
dsm;ls
f,;dl
fdl;fkfdl
flksdkfs
dmfkf
fdm

Sorting your strings. . .
s
dk
ss
fdm
skdl
dsldk
f,;dl
dmfkf
dklskl
sdklss
dsm;ls
flksdkfs
fdl;fkfdl
d.khaetskaya@fit-main:~$
```

```
SYNOPSIS

       cc -mt [ flag... ] file... [ library... ]
       #include <pthread.h>

       int pthread_barrier_wait(pthread_barrier_t *bar
rier);


DESCRIPTION

       The pthread_barrier_wait() function synchronize
s participating threads at
       the barrier referenced by barrier. The calling
 thread blocks until the
       required number of threads have called pthread_
```

Notes

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

#define COEF 2
#define MAXNUMSTR 100

pthread_barrier_t b;

void* printString(void *args){
    int res = pthread_barrier_wait(&b);
    if (res != 0 && res != PTHREAD_BARRIER_SERIAL_THREAD){
        perror("pthread_barrier_wait:");
    }
    sleep (COEF * strlen((char*)args));
    printf("%s", (char*)args);
}

void freeStrings(char **strings, int strCount){
    for (int i = 0; i < strCount; i++){
        free(strings[i]);
    }
    free(strings);
}

int main() {

    char** strings = malloc(sizeof(char *) * MAXNUMSTR);
    if (strings == NULL){
        perror("malloc failed:");
    }

    int readCount = 2;
    int idx = 0;
    size_t strlen = 200;

    printf("Enter strings to sort:\n");

    while (readCount > 1){
        strings[idx] = malloc(sizeof(char) * strlen);
        readCount = getline(&strings[idx], &strlen, stdin);
        idx++;

        if (readCount == 1){
            free(strings[idx]);
            idx--;
        }
    }

    int strCount = idx;

    printf("Sorting your strings. . .\n");

    // create barrier
    if (pthread_barrier_init(&b, NULL, strCount) != 0){
        perror("pthread_barrier_init:");
    }

    pthread_t* threads = malloc(sizeof(pthread_t) * strCount);
    if (threads == NULL){
```

```
barrier_wait() specifying
       the barrier.


       When the required number of threads have called
pthread_barrier_wait()
       specifying the barrier, the constant PTHREAD_BA
RRIER_SERIAL_THREAD is
       returned to one unspecified thread and 0 is ret
urned to each of the
       remaining threads. At this point, the barrier i
s reset to the state it
       had as a result of the most recent pthread_barr
ier_init(3C) function that
       referenced it.


       The constant PTHREAD_BARRIER_SERIAL_THREAD is d
efined in <pthread.h> and
       its value is distinct from any other value retu
rned by
       pthread_barrier_wait().


       The results are undefined if this function is c
alled with an
       uninitialized barrier.


       If a signal is delivered to a thread blocked on
a barrier, upon return
       from the signal handler the thread resumes wait
ing at the barrier if the
       barrier wait has not completed (that is, if the
required number of
       threads have not arrived at the barrier during
 the execution of the
       signal handler); otherwise, the thread continue
s as normal from the
       completed barrier wait. Until the thread in the
signal handler returns
       from it, it is unspecified whether other thread
s may proceed past the
       barrier once they have all reached it.


       A thread that has blocked on a barrier does not
prevent any unblocked
       thread that is eligible to use the same process
ing resources from
       eventually making forward progress in its execu
tion.


       Eligibility for processing resources is determi
ned by the scheduling
       policy.
```

illumos: manual page: pthread_barrier_wait.3c

PTHREAD_BARRIER_WAIT(3C) Standard C Library Functions PTHREAD_BARRIER_WAIT(3C) pthread_barrier_wait - synchronize at a barrier int pthread_barrier_wait(pthread_barrier_t

https://illumos.org/man/3C/pthread_barrier_wait

illumos: manual page: pthread_barrier_init.3c

PTHREAD_BARRIER_DESTROY(3C) Standard C Library Functions pthread_barrier_destroy, pthread_barrier_init - destroy and initialize a barrier object int pthread_barrier_destroy(

https://illumos.org/man/3c/pthread_barrier_init

```
        perror("malloc failed:");
    }

    // start threads
    for (int i = 0; i < strCount; i++){
        if (pthread_create(&threads[i], NULL, printString, strings[i]) !
= 0){
            perror("failed to create new thread:");
            freeStrings(strings, strCount);
            free(threads);
            exit(-1);
        }
    }

    // finish threads
    for (int i = 0; i < strCount; i++){
        if (pthread_join(threads[i], NULL) != 0){
            perror("failed to join thread:");
            freeStrings(strings, strCount);
            pthread_barrier_destroy(&b);
            free(threads);
            exit(-1);
        }
        free(strings[i]);
    }

    free(strings);
    // free barrier
    if (pthread_barrier_destroy(&b) != 0){
        perror("pthread_barrier_destroy:");
    }
    free(threads);
    return 0;
}
```

Reading list

☐