

Exercise 1

Opened: Monday, March 18, 2024, 12:00 AM

Due: Saturday, 20 April 2024, 11:59 PM

NAND frames

The task involves implementing a dynamically loaded library in C that supports combinational Boolean circuits composed of AND gates. A NAND gate has an integer non-negative number of inputs and one output. A NAND frame without inputs always gives a signal at the output `false`. a single-input NAND frame is a negation. For positive `n` gate input - input are numbered from `0` down `n - 1`. A Boolean signal accepting values can be given to the gate input `false` or `true`. At the exit, the gate gives a signal `false`, if there are signals on all its inputs `true`, `at true` otherwise. The signal from the gate output can be connected to multiple gate inputs. Only one signal source can be connected to the gateway input.

library interface

The library interface is located in the file attached to the task content `nand.h`, which contains the following declarations. Additional details of the library's operation should be obtained from the file attached to the task `nand_example.c`, which is an integral part of the specification.

```
typedef struct nand nand_t;
```

`nand_t` is the name of the structural type representing a NAND gate. This type must be defined (implemented) as part of this task.

```
nand_t * nand_new(unsigned n);
```

`nand_new` creates a new NAND gate on `n` entrances. Function result:

- pointer to a structure representing a NAND gate;
- `NULL`—if a memory allocation error occurred; the function sets then `errno` on `ENOMEM`.

```
void nand_delete(nand_t *g);
```

`nand_delete` disconnects the input and output signals of the specified gate, then removes the specified gate and frees all memory used by it. it does nothing if called with a pointer `NULL`. After executing this function, the pointer passed to it becomes invalid.

function parameter:

- `g`—pointer to a structure representing a NAND gate.

```
int nand_connect_nand(nand_t *g_out, nand_t *g_in, unsigned k);
```

`nand_connect_nand` connects the gate output `g_out` to the entrance `k` goals `g_in`, possibly disconnecting from this input the signal that was previously connected to it.

function parameters:

- `g_out`—pointer to a structure representing a NAND gate;
- `g_in`—pointer to a structure representing a NAND gate;
- `k`—gate input number `g_in`.

function result:

- 0—if the operation was successful;
- -1—if any pointer has a value `NULL`, parameter `k` has an invalid value or a memory allocation error occurred; the function sets then `errno` accordingly on `EINVAL` or `ENOMEM`.

```
int nand_connect_signal(bool const *s, nand_t *g, unsigned k);
```

unction `nand_connect_signal` connects a boolean signal `p` to the entrance `k` gate, possibly disconnecting from this input the signal that was previously disconnected to it.

function parameters:

- `p`—pointer to a type variable `bool`;
- `g`—pointer to a structure representing a NAND gate;
- `k`—gate input number `g`.

function result:

- 0—if the operation was successful;
- -1—if any pointer has a value `NULL`, parameter `k` has an invalid value or a memory allocation error occurred; the function sets then `errno` accordingly on `EINVAL` or `ENOMEM`.

```
size_t nand_evaluate(nand_t **g, bool *s, size_t m);
```

unction `nand_evaluate` determines the signal values at the outputs of given frames and calculates the length of the critical path.

the length of the critical path for the Boolean signal and for the gate without inputs is zero. The length of the critical path at the gate output is $+ \max(S_0, S_1, S_2, \dots, S_{n-1})$, Where S_n is the length of the critical path on `ej` and this entrance. The critical path length of a gate system is the maximum of the lengths of the critical paths at the outputs of all given gates.

function parameters:

- `g`—pointer to an array of pointers to structures representing NAND gates;
- `p`—a pointer to the table where the designated signal values are to be placed;
- `m`—size of arrays pointed to by `g` and `p`.

function result:

- the length of the critical path if the operation was successful; board `p` it then contains the designated signal values at the gate outputs; `s[i]` contains the value of the signal at the output of the gate pointed to by `g[i]`;
- -1—if any pointer has a value `NULL`, `m` is zero, the operation failed or memory allocation failed; the function sets then `errno` accordingly on `EINVAL`, `ECANCELED` or `ENOMEM`, and the contents of the array `p` is undefined.

```
size_t nand_fan_out(nand_t const *g);
```

unction `nand_fan_out` determines the number of gate inputs connected to the output of a given gate.

function parameter:

- `g`—pointer to a structure representing a NAND gate.

function result:

- the number of gate inputs connected to a given gate's output if the operation was successful;
- -1—if the pointer has a value `NULL`; the function sets then `errno` on `EINVAL`.

```
oid* nand_input(nand_t const *g, unsigned k);
```

unction `nand_input` returns a pointer to a boolean signal or gate connected to the input `k` gateway pointed to by `g` or `NULL`, if nothing is connected to this input.

function parameters:

- `g`—pointer to a structure representing a NAND gate;
- `k`—entry number.

function result:

- type pointer `bool*` or `nand_t*`;
- `NULL`— if nothing is connected to the specified input; the function sets then `errno` on `0`;
- `NULL`— if the indicator `g` has value `NULL` or value `k` is incorrect; the function sets then `errno` on `EINVAL`.

```
and_t* nand_output(nand_t const *g, ssize_t k);
```

unction `nand_output` allows you to iterate over the gates connected to the output of the given gate. The result of this function is indeterminate if its parameters are invalid. If the gate output `g` is connected to multiple inputs of the same frame, then this gate appears multiple times as a result of iteration.

function parameters:

- `g`—pointer to a structure representing a NAND gate;
- `k`—an index ranging from zero to a value one less than returned by the function `nand_fan_out`.

function result:

- type pointer `nand_t*` to a gate connected to the gate output `g`.

functional requirements

Determining the signal value and the critical path length at the gate's output requires recursive determination of these values at its inputs (although we do not require recursive implementation), unless the gate has no inputs.

etermination of these values may fail if no signal is connected to any of the inputs, the procedure loops (the gates do not form an combinatory circuit) or a memory allocation error occurs.

It is important to ensure that the determination of the signal value and the length of the critical path at the gate output is performed only once.

formal requirements

As a solution to the task, you should insert an archive containing the file in Moodle `and.c` and optionally other files `*.hand*.c` with the library implementation, and the file `akephiles` or `Makefiles`. The archive should not contain other files or directories, in particular it should not contain binary files. the archive should be compressed with the program `zip`, `7z` or `rar`, or a pair of program `star` and `gzip`. After unpacking the archive, all files should be in a common subdirectory.

file provided in the solution `makefiles` or `Makefiles` should contain a goal `ibnand.so`, so that the command `make libnand.so` started compiling the library and a file was created in the current directory `libnand.so`. This command should also compile and include the file attached to the content of the request to the library `memory_tests.c`. You must describe file dependencies and ensure that only files that have changed or files that depend on them are compiled. Call `make clean` should remove all files created by the command `make`. File `makefiles` or `Makefiles` should contain seudocel `PHONY`. It may also contain other goals, for example a goal compiling and linking with the library, an example of its use included in the link attached to the task content `nand_example.c` or the goal that triggers the tests.

o compilation should be usedgcc. The library should compile in the Linux computer laboratory. Files with the library implementation should be compiled with the following options:

```
Wall -Wextra -Wno-implicit-fallthrough -std=gnu17 -fPIC -O2
```

Links with the library implementation should be linked with the following options:

```
shared -Wl,--wrap=malloc -Wl,--wrap=calloc -Wl,--wrap=realloc -l,--wrap=reallocarray -Wl,--wrap=free -Wl,--wrap=strdup -Wl,--wrap=strndup
```

options-Wl,--wrap=make function callsmalloc,calloctc. are captured by functions respectively__wrap_malloc,__wrap_calloctc. the intercepting functions are implemented in the task attached to the content countmemory_tests.c.

Library implementation must not lose memory or leave the data structure in an inconsistent state, even when a memory allocation error occurs. The correctness of the implementation will be checked using the program algrind.

implementation cannot contain artificial limits on the size of stored data - the only limits are the size of memory available in the computer and the size of the machine word of the computer used.

price

A fully correct solution to the task that implements all the requirements can earn 20 points, of which 14 points will be awarded on the basis of automatic tests, and 6 points will be awarded for the assessment of code quality. You may lose all points for failure to compile the solution or failure to meet the legal requirements. Up to 2 points may be deducted for warnings written by the compiler.



solutions must be implemented independently under pain of failing the course. Both the use of someone else's code and the private or public disclosure of your own code are prohibited.

attachments

The following files are attached to the task content:

- memory_tests.c-implementation of a library module used to test the implementation's response to memory allocation failure;
- memory_tests.h-declaration of a library module interface used to test the implementation's response to memory allocation failure;
- nand_example.c-sample library tests;
- nand.h-library interface declaration.

You are not allowed to modify the file nand.h. We reserve the right to change files and contents memory_tests.h and memory_tests.c when testing solutions.

 memory_tests.c	February 27, 2024, 5:12 PM
 memory_tests.h	February 27, 2024, 5:12 PM
 nand_example.c	March 15, 2024, 7:48 PM
 nand.h	March 19, 2024, 11:16 AM

Submission status

Submission status	Submitted for grading
Grading status	Not graded