



Zadanie 1

Otwarto: poniedziałek, 18 marca 2024, 00:00

Wymagane do: sobota, 20 kwietnia 2024, 23:59

Bramki NAND

Zadanie polega na zaimplementowaniu w języku C dynamicznie ładowanej biblioteki obsługującej kombinacyjne układy boolowskie złożone z bramek NAND. Bramka NAND ma całkowitą nieujemną liczbę wejść i jedno wyjście. Bramka NAND bez wejść daje na wyjściu zawsze sygnał o wartości `false`. Bramka NAND o jednym wejściu jest negacją. Dla dodatniego `n` wejścia bramki `n`-wejściowej są numerowane od `0` do `n - 1`. Na wejście bramki można podać sygnał boolowski przyjmujący wartości `false` lub `true`. Na wyjściu bramka daje sygnał `false`, jeśli na wszystkich jej wejściach są sygnały `true`, a `true` w przeciwnym przypadku. Sygnał z wyjścia bramki można podłączyć do wielu wejść bramek. Do wejścia bramki można podłączyć tylko jedno źródło sygnału.

Interfejs biblioteki

Interfejs biblioteki znajduje się w załączonym do treści zadania pliku `nand.h`, który zawiera poniższe deklaracje. Dodatkowe szczegóły działania biblioteki należy wywnioskować z załączonego do treści zadania pliku `nand_example.c`, który jest integralną częścią specyfikacji.

```
typedef struct nand nand_t;
```

Jest to nazwa typu strukturalnego reprezentującego bramkę NAND. Typ ten trzeba zdefiniować (zaimplementować) w ramach tego zadania.

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

Funkcja `nand_new` tworzy nową bramkę NAND o `n` wejściach. Wynik funkcji:

- wskaźnik na strukturę reprezentującą bramkę NAND;
- `NULL` – jeśli wystąpił błąd alokowania pamięci; funkcja ustawia wtedy `errno` na `ENOMEM`.

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

Funkcja `nand_delete` odłącza sygnały wejściowe i wyjściowe wskazanej bramki, a następnie usuwa wskazaną bramkę i zwalnia całą używaną przez nią pamięć. Nic nie robi, jeśli zostanie wywołana ze wskaźnikiem `NULL`. Po wykonaniu tej funkcji przekazany jej wskaźnik staje się nieważny.

Parametr funkcji:

- `g` – wskaźnik na strukturę reprezentującą bramkę NAND.

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

Funkcja `nand_connect_nand` podłącza wyjście bramki `g_out` do wejścia `k` bramki `g_in`, ewentualnie odłączając od tego wejścia sygnał, który był do niego dotychczas podłączony.

Parametry funkcji:

- `g_out` – wskaźnik na strukturę reprezentującą bramkę NAND;
- `g_in` – wskaźnik na strukturę reprezentującą bramkę NAND;
- `k` – numer wejścia bramki `g_in`.

Wynik funkcji:

- `0` – jeśli operacja zakończyła się sukcesem;

- **-1** – jeśli któryś wskaźnik ma wartość **NULL**, parametr **k** ma niepoprawną wartość lub wystąpił błąd alokowania pamięci; funkcja ustawia wtedy **errno** odpowiednio na **EINVAL** lub **ENOMEM**.

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

Funkcja **nand_connect_signal** podłącza sygnał boolowski **s** do wejścia **k** bramki **g**, ewentualnie odłączając od tego wejścia sygnał, który był do niego dotychczas podłączony.

Parametry funkcji:

- **s** – wskaźnik na zmienną typu **bool**;
- **g** – wskaźnik na strukturę reprezentującą bramkę NAND;
- **k** – numer wejścia bramki **g**.

Wynik funkcji:

- **0** – jeśli operacja zakończyła się sukcesem;
- **-1** – jeśli któryś wskaźnik ma wartość **NULL**, parametr **k** ma niepoprawną wartość lub wystąpił błąd alokowania pamięci; funkcja ustawia wtedy **errno** odpowiednio na **EINVAL** lub **ENOMEM**.

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

Funkcja **nand_evaluate** wyznacza wartości sygnałów na wyjściach podanych bramek i oblicza długość ścieżki krytycznej.

Długość ścieżki krytycznej dla sygnału boolowskiego i dla bramki bez wejść jest równa zeru. Długość ścieżki krytycznej na wyjściu bramki wynosi $1 + \max(S_0, S_1, S_2, \dots, S_{n-1})$, gdzie S_i jest długością ścieżki krytycznej na jej i -tym wejściu. Długość ścieżki krytycznej układu bramek jest to maksimum z długości ścieżek krytycznych na wyjściach wszystkich podanych bramek.

Parametry funkcji:

- **g** – wskaźnik na tablicę wskaźników do struktur reprezentujących bramki NAND;
- **s** – wskaźnik na tablicę, w której mają być umieszczone wyznaczone wartości sygnałów;
- **m** – rozmiar tablic wskazywanych przez **g** i **s**.

Wynik funkcji:

- długość ścieżki krytycznej, jeśli operacja zakończyła się sukcesem; tablica **s** zawiera wtedy wyznaczone wartości sygnałów na wyjściach bramek; **s[i]** zawiera wartość sygnału na wyjściu bramki wskazywanej przez **g[i]**;
- **-1** – jeśli któryś wskaźnik ma wartość **NULL**, **m** jest równe zeru, operacja nie powiodła się lub nie udało się alokować pamięci; funkcja ustawia wtedy **errno** odpowiednio na **EINVAL**, **ECANCELED** lub **ENOMEM**, a zawartość tablicy **s** jest nieokreślona.

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

Funkcja **nand_fan_out** wyznacza liczbę wejść bramek podłączonych do wyjścia danej bramki.

Parametr funkcji:

- **g** – wskaźnik na strukturę reprezentującą bramkę NAND.

Wynik funkcji:

- liczba wejść bramek podłączonych do wyjścia danej bramki, jeśli operacja zakończyła się sukcesem;
- **-1** – jeśli wskaźnik ma wartość **NULL**; funkcja ustawia wtedy **errno** na **EINVAL**.

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

Funkcja **nand_input** zwraca wskaźnik do sygnału boolowskiego lub bramki podłączonej do wejścia **k** bramki wskazywanej przez **g** albo **NULL**, jeśli nic nie jest podłączone do tego wejścia.

Parametry funkcji:

- **g** – wskaźnik na strukturę reprezentującą bramkę NAND;
- **k** – numer wejścia.

Wynik funkcji:

- wskaźnik typu `bool*` lub `nand_t*`;
- `NULL` – jeśli nic nie jest podłączone do podanego wejścia; funkcja ustawia wtedy `errno` na `0`;
- `NULL` – jeśli wskaźnik `g` ma wartość `NULL` lub wartość `k` jest niepoprawna; funkcja ustawia wtedy `errno` na `EINVAL`.

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

Funkcja `nand_output` pozwala iterować po bramkach podłączonych do wyjścia podanej bramki. Wynik tej funkcji jest nieokreślony, jeśli jej parametry są niepoprawne. Jeśli wyjście bramki `g` jest podłączone do wielu wejść tej samej bramki, to ta bramka pojawia się wielokrotnie w wyniku iterowania.

Parametry funkcji:

- `g` – wskaźnik na strukturę reprezentującą bramkę NAND;
- `k` – indeks z zakresu od zera do wartości o jeden mniejszej niż zwrócona przez funkcję `nand_fan_out`.

Wynik funkcji:

- wskaźnik typu `nand_t*` na bramkę podłączoną do wyjścia bramki `g`.

Wymagania funkcjonalne

Wyznaczenie wartości sygnału i długości ścieżki krytycznej na wyjściu bramki wymaga rekurencyjnego wyznaczenia tych wartości na jej wejściach (choć nie wymagamy rekurencyjnej implementacji), chyba że bramka nie ma wejść. Wyznaczenie tych wartości może się nie udać, jeśli do któregoś wejścia nie jest podłączony żaden sygnał, procedura się zapętla (bramki nie tworzą układu kombinacyjnego) lub wystąpi błąd alokowania pamięci.

Należy zadbać, aby wyznaczenie wartości sygnału i długości ścieżki krytycznej na wyjściu bramki było wykonane tylko raz.

Wymagania formalne

Jako rozwiązanie zadania należy wstawić w Moodle archiwum zawierające plik `nand.c` oraz opcjonalnie inne pliki `*.h` i `*.c` z implementacją biblioteki, oraz plik `makefile` lub `Makefile`. Archiwum nie powinno zawierać innych plików ani podkatalogów, w szczególności nie powinno zawierać plików binarnych. Archiwum powinno być skompresowane programem `zip`, `7z` lub `rar`, lub parą programów `tar` i `gzip`. Po rozpakowaniu archiwum wszystkie pliki powinny się znaleźć we wspólnym podkatalogu.

Dostarczony w rozwiązaniu plik `makefile` lub `Makefile` powinien zawierać cel `libnand.so`, tak aby polecenie `make libnand.so` uruchomiło kompilowanie biblioteki i aby w bieżącym katalogu powstał plik `libnand.so`. Polecenie to powinno również kompilować i dołączać do biblioteki załączony do treści zadania plik `memory_tests.c`. Należy opisać zależności między plikami i zapewnić, że kompilowane są tylko pliki, które zostały zmienione lub pliki, które od nich zależą. Wywołanie `make clean` powinno usuwać wszystkie pliki utworzone przez polecenie `make`. Plik `makefile` lub `Makefile` powinien zawierać pseudocel `.PHONY`. Może zawierać też inne cele, na przykład cel kompilujący i linkujący z biblioteką przykład jej użycia zawarty w załączonym do treści zadania pliku `nand_example.c` bądź cel uruchamiający testy.

Do kompilowania należy użyć `gcc`. Biblioteka powinna się kompilować w laboratorium komputerowym pod Linuksem. Pliki z implementacją biblioteki należy kompilować z opcjami:

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

Pliki z implementacją biblioteki należy linkować z opcjami:

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

Opcje `-Wl,--wrap=` sprawiają, że wywołania funkcji `malloc`, `calloc` itd. są przechwytywane odpowiednio przez funkcje `__wrap_malloc`, `__wrap_calloc` itd. Funkcje przechwytywane są zaimplementowane w załączonym do treści zadania pliku `memory_tests.c`.

Implementacja biblioteki nie może gubić pamięci ani pozostawiać struktury danych w niespójnym stanie, również wtedy gdy wystąpił błąd alokowania pamięci. Poprawność implementacji będzie sprawdzana za pomocą programu `valgrind`.

Implementacja nie może zawierać sztucznych ograniczeń na rozmiar przechowywanych danych – jedynymi ograniczeniami są rozmiar pamięci dostępnej w komputerze i rozmiar słowa maszynowego użytego komputera.

Ocena

Za w pełni poprawne rozwiązanie zadania implementujące wszystkie wymagania można zdobyć 20 punktów, z tego 14 punktów zostanie wystawionych na podstawie testów automatycznych, a 6 punktów to ocena jakości kodu. Za problemy ze skompilowaniem rozwiązania lub niespełnienie wymogów formalnych można stracić wszystkie punkty. Za ostrzeżenia wypisywane przez kompilator może być odjęte do 2 punktów.

Rozwiązania należy implementować samodzielnie pod rygorem niezaliczenia przedmiotu. Zarówno korzystanie z cudzego kodu, jak i prywatne lub publiczne udostępnianie własnego kodu jest zabronione.

Załączniki

Załącznikami do treści zadania są następujące pliki:

- `memory_tests.c` – implementacja modułu biblioteki służącego do testowania reakcji implementacji na niepowodzenie alokowania pamięci;
- `memory_tests.h` – deklaracja interfejsu modułu biblioteki służącego do testowania reakcji implementacji na niepowodzenie alokowania pamięci;
- `nand_example.c` – przykładowe testy biblioteki;
- `nand.h` – deklaracja interfejsu biblioteki.

Nie wolno modyfikować pliku `nand.h`. Zastrzegamy sobie możliwość zmiany testów oraz zawartości plików `memory_tests.h` i `memory_tests.c` podczas testowania rozwiązań.



[memory_tests.c](#)

27 lutego 2024, 17:12



[memory_tests.h](#)

27 lutego 2024, 17:12



[nand_example.c](#)

15 marca 2024, 19:48



[nand.h](#)

19 marca 2024, 11:16