Zadanie 1.9

1. Zarządzanie pamięcią
9. Projekt: Rozszerzenie na ocenę 5,0

Termin	Trudność	Próba	Ocena maszyny	Inspekcja kodu	Plagiat	Raporty
2020-12-20	★★★★★	3				Raport główny
zostało 10 dni	Punkty: 5		Czas testu: 399sek			

Alokator pamięci, wymagania na ocenę 5,0:

Rozbuduj funkcje API zarządzające pamięcią sterty, przygotowane w poprzednich zadaniach, o możliwość poprawnego działania w środowisku wielowątkowym (tzw. *thread-safe*).

W tym celu wybierz i wykorzystaj jeden z mechanizmów synchronizacji, dostępnych w standardzie Posix. Wybrany mechanizm powiniem umożliwiać blokowanie dostępu do zasobów sterty więcej niż jednemu wątkowi w danym czasie. Przez zasób sterty należy rozumieć struktury kontrolne sterty.

Nie ma bowiem możliwości, aby np. aktualizację wskaźników w blokach realizowały równolegle dwa wątki. Sytuacja taka będzie prowadziła do **losowych i nieprzewidywalnych błędów** - błędy wielowątkowości należą do najtrudniejszych w diagnostyce i usuwaniu, głównie ze względu na losowość swojego występowania.

- Funkcje `heap_setup()` oraz `heap_clean()` zmodyfikuj tak, aby odpowiednio tworzyły nowy (`heap_setup`) oraz usuwały/zwalniały istniejący (`heap_clean`) mechanizm synchronizacji.
- Pozostałe funkcje `heap_*` zmodyfikuj tak, aby ich implementacje były chronione przed współbieżnym wykonywaniem.

Pamiętaj, że każdy taki mechanizm posiada wejście/blokadę i wyjście/zdjęcie blokady z chronionego obszaru kodu. Zawsze zdejmuj blokadę przed wyjściem z funkcji. Brak zdjęcia blokady spowoduje, że kolejne jej uruchomienie może zakończyć się zablokowaniem wykonywania programu.

Polecane linki:

- https://pubs.opengroup.org/onlinepubs/9699919799/idx/threads.html
- https://pubs.opengroup.org/onlinepubs/9699919799/functions/V2_chap02.html#tag_15_09
- https://linux.die.net/man/3/pthread_mutex_lock
- https://pl.wikibooks.org/wiki/POSIX_Threads/Synchronizacja_mi%C4%99dzy_w%C4%85tkami/Mutexy

Opis testów jednostkowych:

W zadaniu przewidziano 139 testów jednostkowych.

- Testy 1-136 testują przygotowane funkcje pod kątem zgodności spełnienia wymagań wersji na 3,0 i 4,0 (ok. 2 minuty).
- Testy 137-139 testują zachowanie przygotowanych funkcji w środowisku wielowątkowym (ok. 10 minut).
  - Test 137 testuje zachowanie `heap_malloc`, `heap_calloc`, `heap_free`, `heap_realloc` oraz `heap_validate`.
  - Test 138 testuje zachowanie `heap_aligned_malloc`, `heap_aligned_calloc`, `heap_free`, `heap_aligned_realloc` oraz `heap_validate`.
  - Test 139 testuje zachowanie `heap_malloc_aligned_debug`, `heap_calloc_aligned_debug`, `heap_realloc_aligned_debug`, `heap_malloc_debug`, `heap_calloc_debug`, `heap_realloc_debug`, `heap_free` oraz `heap_validate`.

Testowanie polega na intensywnej alokacji, realokacji i zwalnianiu niewielkich bloków pamięci na stercie, za pomocą wielu równolegle działających wątków. Test ten nie może zagwarantować poprawności przesłanej implementacji alokatora, jest jednak dobrym kompromisem między złożonością istniejących środowisk testowania systemów współbieżnych a całkowitym brakiem testowania.

Tak intensywne alokowanie i zwalnianie pamięci, w przypadku poprawnej implementacji alokatora nie ma prawa uszkodzić sterty. Uszkodzenia sprawdzane są pośrednio, poprzez porównywanie funkcji alokujących z `NULL`, oraz bezpośrednio - za ponocą funkcji `heap_validate`.

Wszystkie funkcje API sterty, wraz z definicjami struktur i typów danych, należy umieścić w pliku nagłówkowym `heap.h`. Natomiast faktyczne implementacje należy umieścić w pliku źródłwym `heap.c`.

Uwagi:

- Zadanie przewiduje jedynie testy jednostkowe, co pozwala przeprowadzać testowanie w całości na swojej maszynie. W tym celu pobierz komplet plików z wygenerowanego raportu i użyj ich do testowania swojego kodu. Korzystaj z wirtualnej maszyny, przygotowanej pod kątem przedmiotu *Systemy Operacyjne 2*.
- Link do pliku CMake dla środowiska CLion: https://pastebin.com/DGr27FLE.