Systemy operacyjne 2016

Projekt nr 1

Termin oddawania 22 grudnia 2016

Zaimplementuj w języku C odpowiednik bibliotecznego algorytmu zarządzania pamięcią. Należy udostępnić procedury o poniższych sygnaturach – ich znaczenie jest dokładnie opisane w podręcznikach systemowych `malloc(3)` i `posix_memalign(3)`.

```
1 void *malloc(size_t size);
2 void *calloc(size_t count, size_t size);
3 void *realloc(void *ptr, size_t size);
4 int   posix_memalign(void **memptr, size_t alignment, size_t size);
5 void  free(void *ptr);
```

Projekt musi kompilować się bez błędów i ostrzeżeń (opcje `-std=gnu11 -Wall -Wextra`) kompilatorem `gcc` lub `clang` pod systemem `Linux`. Należy dostarczyć plik `Makefile`, tak by po wywołaniu polecenia «`make`» otrzymać pliki binarne, a po «`make clean`» pozostawić w katalogu tylko pliki źródłowe. Rozwiązanie należy zamieścić w systemie oddawania zadań na stronie zajęć.

Przygotowanie

Zanim przystąpisz do programowania dokładnie przemyśl i rozpisz na kartce organizację struktur danych w pamięci oraz działanie poszczególnych operacji – zaoszczędzisz sobie w ten sposób czasu na odpluskwianie, które nie będzie zbyt przyjemne. Mimo wszystko zapewne będziecie zmuszeni do użycia `GNU debugger`. Rozpoczęcie znajomości z tym bardzo potężnym, lecz niezbyt przyjaznym, narzędziem należy zacząć od przeczytania samouczka Richard Stallman's gdb Debugger Tutorial[1].

Należy zapoznać się z implementacją różnych rodzajów list opisanych w podręczniku `queue(3)`[2] i użyć pliku nagłówkowego queue.h[3] do zaprogramowania menadżera pamięci. Ze względu na specyfikę implementacji list pamiętaj, aby nie przekazywać wyrażeń mających efekty uboczne do makr preprocesora! Można również użyć implementacji bitmap oraz drzew splay i drzew czerwono-czarnych opisanych odpowiednio w `bitstring(3)`[4] i `tree(3)`[5].

Nie zapomnij skrupulatnie przeczytać specyfikacji procedur bibliotecznego menadżera pamięci! Lepiej dobrze zrozumieć co się implementuje przed rozpoczęciem prac programistycznych.

[1] `http://web.archive.org/web/20161117202458/http://www.unknownroad.com/rtfm/gdbtut/`
[2] `https://www.freebsd.org/cgi/man.cgi?query=queue&sektion=3`
[3] `http://bxr.su/FreeBSD/sys/sys/queue.h`
[4] `https://www.freebsd.org/cgi/man.cgi?query=bitstring&sektion=3`
[5] `https://www.freebsd.org/cgi/man.cgi?query=tree&sektion=3`

Specyfikacja

Niech **obszar** będzie przedziałem adresów wirtualnych, pod które podczepiono strony z użyciem wywołania systemowego `mmap(2)` z flagą `MAP_ANONYMOUS`. Rozmiar strony można pobrać procedurą `getpagesize(2)`. Obszar należy zwrócić do systemu z użyciem `munmap(2)`, jeśli zawiera wyłącznie jeden wolny **blok**, a ilość wolnego miejsca w pozostałych obszarach przekracza ustalony próg (np. kilka stron). Pamięcią dostępną w obrębie obszaru należy zarządzać algorytmem przydziału dla bloków, który opisano poniżej. Jeśli użytkownik zażądał bloku o rozmiarze większym niż kilka stron, to należy poświęcić na to cały jeden obszar. Poniżej podano propozycję rekordu obszaru:

```
typedef struct mem_arena {
  /* all arenas list entry */
  LIST_ENTRY(mem_arena) ma_list;
  /* list of all free blocks in the arena */
  LIST_HEAD(, mem_block) ma_freeblks;
  /* arena size minus sizeof(mem_arena_t) */
  size_t size;
  /* first block in the arena */
  mem_block_t ma_first;
} mem_arena_t;

LIST_HEAD(, mem_arena) arena_list;
```

Zarządzanie blokami ma bazować na liście dwukierunkowej posortowanej względem adresów. Do przydziału bloku używaj strategii **first-fit**. Przy zwalnianiu bloków gorliwie wykonuj operację scalania. Pamiętaj, że adresy zwracane przez procedurę `malloc` muszą być podzielne przez rozmiar największego słowa maszynowego. Poniżej podano propozycję rekordu bloku:

```
typedef struct mem_block {
  /* all blocks list entry */
  LIST_ENTRY(mem_block) mb_list;
  /* mb_size > 0 => block free, mb_size < 0 => block allocated */
  ssize_t mb_size;
  union {
    /* free blocks list entry, valid if block is free */
    LIST_ENTRY(mem_block) mb_free_list;
    /* user data pointer, valid if block is allocated */
    uint64_t mb_data[0];
  };
} mem_block_t;
```

Algorytm zachowywać się poprawnie w programach wielowątkowych – należy zadbać o zakładanie blokad `pthread_mutex_lock(3)` w odpowiednich miejscach. Udostępnij procedurę do drukowania struktur danych menadżera pamięci – tj. listy wszystkich obszarów oraz bloków wraz z ich stanem. Pamiętaj, że operacja `realloc` przy zmniejszaniu długości bloku nie powinna przenosić go w inne miejsce. Natomiast przy zwiększaniu długości bloku nie powinna go przenosić, jeśli jest za nim wystarczająca ilość wolnego miejsca na jego powiększenie.

Testowanie

Skonsoliduj swoje rozwiązanie jako bibliotekę współdzieloną. Nadając odpowiednią wartość zmiennej środowiskowej `LD_PRELOAD` opisanej w podręczniku `ld.so(8)` przesłoń symbole menadżera pamięci z biblioteki standardowej. Zmusisz w ten sposób wybrane programy do korzystania z Twojego algorytmu. Przed doborem programów testowych upewnij się poleceniem `ltrace(1)`, że korzystają one z procedur bibliotecznego algorytmu zarządzania pamięcią.

Bonus

Punkty dodatkowe będą przyznawane za:

- wykrywanie uszkodzeń struktur danych menadżera pamięci
- identyfikację błędnych adresów przekazywanych do `free`
- szybsze znajdowanie wolnych bloków ustalonego rozmiaru (np. metodą **quick-fit**)
- zmniejszenie objętości struktur danych do przechowywania informacji o małych blokach
- ograniczenie współzawodnictwa między wątkami korzystającymi z procedur menadżera pamięci