Optymalizacja programów Open-Source

Pamięć
część 3

Krzysztof Lichota
lichota@mimuw.edu.pl

Alokacja pamięci

Dlaczego alokacja jest ważna dla wydajności

- Programy w języku wysokiego poziomu wykonują setki tysięcy alokacji, więc jest się o co bić
- Niefrasobliwa alokacja powoduje nadmierne zużycie pamięci, co przekłada się na wolniejsze działanie całego systemu
- Programy współbieżne często zatykają się właśnie w alokatorze, który staje się wąskim gardłem

Problemy z alokacją pamięci

- Standardowe alokatory alokują zazwyczaj nagłówek na każdy blok zaalokowanej pamięci (zwykle 8 bajtów) – dla alokacji małych bloków daje to znaczący narzut
- Standardowe alokatory zaokrąglają alokację do równej granicy bajtów (np. do 8)
- Pola struktur w C są wyrównywane i zostają w strukturze puste miejsca (padding)
- Fragmentacja pamięci
- Niektóre alokatory nie zwracają pamięci do systemu operacyjnego nawet, gdy jej nie używają

Fragmentacja pamięci

- Alokowanie na przemian małych i dużych bloków, a potem zwolnienie małych powoduje powstanie „dziur” między dużymi, w których nie można zaalokować odrobinę większego bloku
- Taka pamięć jest marnowana, nie może zostać użyta, chociaż jest wolna, więc zużycie pamięci przez proces wzrasta

Rozmieszczenie pamięci na stronach

- Pamięć w systemie operacyjnym jest zarządzana stronami (na architekturze o wielkości 4KB)
- Jeśli chociaż 1 bajt na stronie jest używany, cała strona musi siedzieć w pamięci
- Dlatego używane dane nie powinny być porozrzucane po wielu stronach, tylko skupione na jednej

Problemy z wydajnością alokacji pamięci

- Alokator ogólnego przeznaczenia niekoniecznie pasuje do naszych zastosowań
- Standardowe alokatory mają na przykład globalny lock na wszystkie wątki, więc działają kiepsko przy dużym stopniu współbieżności
- Nawet alokator z lokalną pulą na wątek działa źle, kiedy jeden wątek alokuje struktury, a inny je zwalnia

Sposoby radzenia sobie z problemami z alokacją

- Zastąpienie standardowego alokatora C innym
- Zmiana parametrów zwykłego alokatora malloc()
- Użycie nowszej biblioteki C/C++
- Zastąpienie operatora new w C++ (globalnie dla całego programu lub dla klasy)
- Niektóre biblioteki szablonów (np. STL) pozwalają podać alokator przy instancjacji szablonu
- Własne zarządzanie pamięcią
  - Alokacja z puli
  - Własny alokator dla zmiennych rozmiarów
  - Alokacja bezpośrednio z systemu operacyjnego (mmap)

Uwagi

- Biblioteki wysokiego poziomu (np. glib) często mają własne alokatory lub funkcje do alokacji pamięci, więc przy pracy z takimi programami musimy zdawać sobie sprawę, że zarządzanie pamięcią odbywa się na innym poziomie
- Pakowanie struktur żeby zajmowały mniej pamięci może powodować spadek wydajności, bo struktury nie są wyrównane w pamięci

Padding i reorganizacja struktur

- Pola struktur są wyrównywane do odpowiedniej granicy bajtów, żeby przyspieszyć dostęp do pola (np. pola int są wyrównywane do 4 bajtów)
- Można zmniejszyć rozmiar struktury usuwając padding
  - zmiana kolejności pól
  - packed structs
- Najczęściej używane pola powinny się mieścić w jednej linii cache (16 bajtów) w celu przyspieszenia dostępu do pamięci

Pakowanie struktur

- Pakowanie struktur włącza się w GCC za pomocą atrybutu packed, który można zastosować do pola lub całej struktury, np:

```
struct test_t {
  int  a;
  char b;
  int  c;
} __attribute__((__packed__));
```

- Można też zmienić wyrównanie poszczególnych pól za pomocą atrybutu „aligned”

Reorganizacja struktur

- Struktury są tworzone w takiej kolejności jak jest podana w deklaracji pól
- Zmieniając kolejność na taką, by pola były od razu wyrównane do swojej właściwej granicy można zmniejszyć rozmiar struktury
- Rozmiar struktury nadal jest wyrównywany!

```
struct test_1_t {
        char b;
        int  c;
        char d;
        char e;
}; //rozmiar = 12
```

```
struct test_2_t {
        int  c;
        char b;
        char d;
        char e;
}; //rozmiar = 8
```

Wykrywanie problemów alokacji

Wykrywanie problemów alokacji

- Ogólna liczba alokacji – porównać pamięć raportowaną np. przez exmap/top z tym co pokazują profilery malloc()
- Ogólne statystyki - mallinfo/malloc_stats
- Problemy z wydajnością – zwykłe profilery
- Śledzenie liczby alokacji, rozmiarów i położenia (fragmentacji)
  - kmtrace (trzeba sobie samemu napisać)
  - Google heap profiler (ile alokujemy i skąd)
- Rozlokowanie pamięci - memprof z łatą pokazującą położenie zaalokowanych bloków
- Brak kompleksowego narzędzia – może warto napisać?

mallinfo/malloc_stats

- Funkcje pozwalające się z grubsza zorientować w tym jak alokator malloc() przydziela pamięć
- Należy zaincludować malloc.h i w odpowiednim momencie wywołać funkcję z kodu
  - malloc_stats() wypisuje podstawowe informacje na wyjście programu
  - mallinfo() podaje te informacje w postaci zmiennych

Sterowanie zachowaniem standardowego alokatora

- Można sterować zwykłym alokatorem malloc() w pewnym stopniu
  - Można ustawić od jakiego rozmiaru pamięć alokowana jest za pomocą mmap()
  - Można ustawić kiedy pamięć jest zwracana do systemu operacyjnego za pomocą sbrk()
  - Można wyłączyć używanie mmap() do alokacji

Więcej: http://www.linuxjournal.com/article/6390

Alokacja z puli

Alokacja z puli

- Przydzielamy duży blok normalnym alokatorem
- Dzielimy ten blok na fragmenty o stałej wielkości
- Zarządzamy tym blokiem własnym algorytmem (mapa bitowa, lista wolnych obszarów, itp.)
- Można inicjować każdy fragment tylko raz przy inicjalizacji puli (np. slab allocator w jądrze Linuksa)
- Przykład: boost::pool
  - alokator ogólnego przeznaczenia
  - pule dla obiektów

Memory compaction

Memory compaction

- Jeśli nasz sposób alokacji powoduje rozrzucanie zaalokowanej pamięci po znacznym obszarze, można się zastanowić nad sensownością relokacji używanych bloków do jednego obszaru (analogicznie do defragmentacji dysku)
- Wymaga wprowadzenia jednego poziomu przekierowania (wskaźnik do wskaźnika, tablica, słownik)
- Nie można trzymać bezpośrednich wskaźników do obszarów

tcmalloc

tcmalloc

- Alokator będący częścią pakietu Google perftools
- Działa szybciej niż zwykły alokator, ale zużywa trochę więcej pamięci
- Działa szybciej dla programów wielowątkowych, bo używa puli pamięci lokalnej dla wątku (unika częstej synchronizacji)
- W przeciwieństwie do niektórych wielowątkowych alokatorów pamięć z puli dla wątku wraca do puli globalnej i może być użyta w innych wątkach

Użycie tcmalloc

- Zastępuje malloc/free
- Można użyć za pomocą LD_PRELOAD: LD_PRELOAD=libtcmalloc.so program
- Można wlinkować bezpośrednio do programu za pomocą „-ltcmalloc”
- Jeśli nie potrzebujemy Google heap profilera, to można wlinkować libtcmalloc_minimal zamiast libtcmalloc

Google heap profiler

Google heap profiler

- Przechwytuje odwołania do malloc(), free() i pokrewnych
- Wymaga użycia tcmalloc (alokatora z pakietu Google perftools)
- Może pokazać w których funkcjach jest alokowana pamięć (rozmiar, liczba obiektów)
- Wyniki pokazuje jako graf, analogicznie do Google profilera, za pomocą tego samego narzędzia (pprof)

Użycie Google heap profilera

- LD_PRELOAD="/usr/lib/libtcmalloc.so" HEAPPROFILE=heapprofile.out <polecenie>
- Stan jest zrzucany co określoną liczbę zaalokowanych bajtów do plików heapprofile.out.XXXX.heap
- Procesy potomne są śledzone, ich stan jest zrzucany do plików z końcówką równą pidowi
- Można zmienić co ile jest generowany zrzut za pomocą zmiennych środowiska HEAP_PROFILE_ALLOCATION_INTERVAL i HEAP_PROFILE_INUSE_INTERVAL

Generowanie wyników

- pprof <opcje> program_wykonywalny ślad
- Wynik jest wysyłany na stdout
- Przykład:
  - pprof --gif /usr/bin/gnuplot heapprofile.out.0001.heap >googleheap-gnuplot.gif
- Przydatne opcje (oprócz tych ogólnych dla pprof):
  - inuse_space – pokazuje używaną pamięć w MB
  - inuse_objects – pokazuje liczbę używanych obiektów
  - alloc_space – pokazuje zaalokowaną w czasie całego życia programu pamięć (włączając już zwolnione obszary)
  - alloc_objects – analogicznie dla obiektów
  - base – porównuje ślad z podanym śladem

Uwagi

- Wszystkie uwagi dotyczące zwykłego Google profilera odnoszą sie do heap profilera (zgodność bibliotek, itp.)
- Opcja porównywania generuje bardzo duże obrazki – trzeba uważać przy przeglądaniu, najlepiej generować do formatu wektorowego albo tekstowego

Bibliografia

- http://en.wikipedia.org/wiki/Slab_allocator
- http://www.boost.org/libs/pool/doc/concepts.html
- http://www.boost.org/libs/pool/doc/interfaces.html
- http://www.sgi.com/tech/stl/Allocators.html
- http://www.delorie.com/gnu/docs/glibc/libc_35.html
- http://www.delorie.com/gnu/docs/glibc/libc_32.html
- http://www.linuxjournal.com/article/6390
- http://goog-perftools.sourceforge.net/doc/heap_profiler.
- http://goog-perftools.sourceforge.net/doc/tcmalloc.html
- http://sig9.com/articles/gcc-packed-structures
- http://gcc.gnu.org/onlinedocs/gcc-3.3.3/gcc/Variable-Att