Systemy operacyjne

Lista zadań nr 9

Na zajęcia 10 i 18 grudnia 2019

Należy przygotować się do zajęć czytając następujące rozdziały książek lub publikacji:

- Dynamic Storage Allocation: A Survey and Critical Review[1]: 1 – 3

Zadanie 1. Algorytm przydziału pamięci udostępnia funkcje o sygnaturach «`alloc: words -> @id`» i «`free: @id -> void`» i ma do dyspozycji obszar 50 słów maszynowych. Implementacja używa dwukierunkowej listy wolnych bloków oraz boundary tags. Wyszukiwanie wolnych bloków działa zgodnie z polityką **best-fit**. Operacja zwalniania **gorliwie złącza** bloki i wstawia wolne bloki na początek listy. Posługując się diagramem z wykładu wykonaj krokową symulację algorytmu przydziału pamięci dla poniższego ciągu żądań. Należy wziąć pod uwagę miejsce zajmowane przez struktury danych algorytmu przydziału oraz nieużytki.

```
alloc(5) alloc(12) alloc(15) alloc(8) free(@2) free(@1) free(@3) alloc(10)
```

Uwaga: Funkcja «`alloc`» zwraca bloki o kolejnych identyfikatorach począwszy od `@1`. Adresy są wyrównane do długości słowa.

Zadanie 2 (bonus). Rozważmy algorytm przydziału pamięci z zadania pierwszego. Załóżmy, że implementujemy go na architekturze 64-bitowej – zatem słowo maszynowe ma 8 bajtów. Zarządzane areny będą nie większe niż 512KiB. Wskaźnik zwracany przez operację «`malloc`» będzie podzielny przez 16. Zaproponuj metodę efektywnego kodowania metadanych, aby zminimalizować narzut pamięciowy, oraz strategię wykrywania uszkodzenia danych algorytmu (tj. lista dwukierunkowa, boundary tags).

Wskazówki: Rozważ właściwości wskaźników wynikające z rozmiaru areny.

Zadanie 3. Rozważmy **algorytm kubełkowy** (§3.6) (ang. *segregated-fit*) przydziału pamięci z **gorliwym złączaniem** wolnych bloków. Jak przebiegają operacje «`malloc`» i «`free`»? Co robi «`malloc`», gdy na danej liście nie ma wolnego bloku żądanego rozmiaru? Jak poradzić sobie w trakcie złączania wolnych bloków w procedurze «`free`», jeśli chcemy usunąć ostatni element z listy? Rozważ zastosowanie **leniwego złączania** wolnych bloków w algorytmie kubełkowym przydziału pamięci – jakie problemy zauważasz?

Ściągnij ze strony przedmiotu archiwum «`so19_lista_9.tar.gz`», następnie rozpakuj je i zapoznaj się z dostarczonymi plikami.

Zadanie 4 (P). Programy «`bad-*.c`» posiadają kilka ukrytych błędów związanych z używaniem bloków pamięci przydzielonych dynamicznie. Zauważ, że kompilator nie zgłosił żadnych błędów ani ostrzeżeń, a uruchomienie tych programów nie kończy się błędem. Wytypuj błędy i zapisz je w swoich notatkach.

Następnie skompiluj programy z instrumentacją kodu dodaną przez address sanitizer[2] – wystarczy odkomentować odpowiednią linię w pliku «`Makefile`» i ponownie zbudować pliki wykonywalne. Po uruchomieniu programów otrzymasz komunikaty błędów – wyjaśnij je. Porównaj znalezione błędy z zanotowanymi kandydatami, a następnie popraw kod źródłowy.

Zadanie 5 (bonus). Zapoznaj się z artykułem AddressSanitizer: A Fast Address Sanity Checker[3]. Jaki jest koszt używania tego narzędzia (§4)? Przedstaw zasadę działania tego narzędzia – w szczególności opowiedz o instrumentacji kodu (§3.1) i shadow map (§3.2). W poprzednim zadaniu nie zaprogramowano wszystkich błędów, z których wykryciem radzi sobie address sanitizer – podaj kilka przykładów.

[1] `https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.47.275`
[2] `https://en.wikipedia.org/wiki/AddressSanitizer`
[3] `https://www.usenix.org/system/files/conference/atc12/atc12-final39.pdf`