Systemy komputerowe

Lista zadań nr 7

Na zajęcia 27 – 30 kwietnia 2020

Zadanie 1 (3pkty). Na podstawie rozdziału §7.12 podręcznika „Computer Systems: A Programmer's Perspective" opisz proces **leniwego wiązania** (ang. *lazy binding*) symboli i odpowiedz na następujące pytania:

- Czym charakteryzuje się **kod relokowalny** (ang. *Position Independent Code*)?
- Do czego służą sekcje «`.plt`» i «`.got`» – jakie dane są tam przechowywane?
- Czemu sekcja «`.got`» jest modyfikowalna, a sekcje kodu i «`.plt`» są tylko do odczytu?
- Co znajduje się w sekcji «`.dynamic`»?

Zaprezentuj leniwe wiązanie na podstawie programu:

```
1 /* lazy.c */
2 #include <stdio.h>
3
4 int main(void) {
5   puts("first time");
6   puts("second time");
7   return 0;
8 }
```

skompilowanego poleceniem `gcc -Os -Wall -ggdb -o lazy lazy.c`. Uruchom go pod kontrolą debuggera GDB, ustaw punkty wstrzymań (ang. *breakpoint*) na linię 4 i 5. Po czym wykonując krokowo program (`stepi`) pokaż, że gdy procesor skacze do adresu procedury «`puts`» zapisanego w «`.got`» – za pierwszym wywołaniem jest tam przechowywany inny adres niż za drugim.

Wskazówka: Wykorzystaj rysunek 7.19. Dobrze jest zainstalować sobie nakładkę na debugger `GDB dashboard`[1] lub GDB TUI.

Zadanie 2. Rozważmy program składający się z dwóch plików źródłowych:

```
1  /* str-a.c */
2  #include <stdio.h>
3
4  char *somestr(void);
5
6  int main(void) {
7    char *s = somestr();
8    s[5] = '\0';
9    puts(s);
10   return 0;
11 }
```

```
1  /* str-b.c */
2  char *somestr(void) {
3    return "Hello, world!";
4  }
```

Po uruchomieniu program kończy się z błędem dostępu do pamięci. Dlaczego? Gdzie została umieszczona stała znakowa `"Hello, world!"`? Popraw ten program tak, by się poprawnie zakończył. Gdzie został umieszczony ciąg znaków po poprawce? Nie wolno modyfikować sygnatury procedury «`somestr`» i pliku «`str-a.c`», ani korzystać z dodatkowych procedur.

[1] `https://github.com/cyrus-and/gdb-dashboard`

Zadanie 3 (2pkt). Korzystając z **dyrektyw asemblera** opisanych w GNU as: Assembler Directives[2] stwórz plik źródłowy (z rozszerzeniem `.s`) w którym

1. zdefiniujesz globalną funkcję `foobar`,
2. zdefiniujesz lokalną strukturę podaną niżej:

```
static const struct {
  char a[3]; int b; long c; float pi;
} baz = { "abc", 42, -3, 1.4142 };
```

3. zarezerwujesz miejsce dla tablicy `long array[100]`?

Pamiętaj, że dla każdego zdefiniowanego symbolu należy uzupełnić odpowiednio tablicę `.symtab` o typ symbolu i rozmiar danych, do których odnosi się symbol. Z pliku źródłowego stwórz plik relokowalny. Analizując go przekonaj się o poprawności rozwiązania.

Wskazówka Ktoś, a raczej coś może stworzyć rozwiązanie za Ciebie, a Ty musisz je tylko zrozumieć.

Zadanie 4. Jakie konstrukcje językowe w C są blokadami optymalizacji (ang. *optimization blockers*)? Porównaj funkcje `combine1` i `combine4` ze slajdów i wyjaśnij, dlaczego wydajniejsza wersja musiała zostać stworzona ręcznie.

Zadanie 5. Na podstawie rozdziału §5.7 podręcznika „Computer Systems: A Programmer's Perspective" i rysunku 5.11 wyjaśnij zasadę działania procesora o architekturze **superskalarnej**. Jak działa **spekulatywne** wykonywanie instrukcji? Co to znaczy, że poszczególne jednostki funkcjonalne procesora działają w sposób **potokowy**?

Zadanie 6. Rozpatrujemy procesor o charakterystyce podanej w tabeli 5.12 podręcznika „Computer Systems: A Programmer's Perspective". Zdefiniuj miarę wydajności CPE (ang. *cycles per element/operation*), następnie wyjaśnij pojęcie granicy opóźnienia (ang. *latency bound*) oraz granicy przepustowości (ang. *throughput bound*) procesora. Skąd biorą się wartości w tabelce na stronie 560?

Zadanie 7. Z czego wynika wysoka wydajność funkcji `combine6` (tabelka 5.21, str. 573)? Jak zoptymalizować ten kod by dojść do granicy przepustowości procesora?

Zadanie 8. Rozważmy dysk o następujących parametrach: jeden talerz; jedna głowica; 400 tysięcy ścieżek na powierzchnię; 2500 sektorów na ścieżkę; 7200 obrotów na minutę; czas wyszukiwania: 1ms na przeskoczenie o 50 tysięcy ścieżek.

1. Jaki jest średni czas wyszukiwania?
2. Jaki jest średni czas opóźnienia obrotowego?
3. Jaki jest czas transferu sektora?
4. Jaki jest całkowity średni czas obsługi żądania?

Zadanie 9. Rozważmy dysk o następujących parametrach: 360 obrotów na minutę, 512 bajtów na sektor, 96 sektorów na ścieżkę, 110 ścieżek na powierzchnię. Procesor czyta z dysku całe sektory. Dysk sygnalizuje dostępność danych zgłaszając przerwanie na każdy przeczytany bajt. Jaki procent czasu procesora będzie zużywała obsługa wejścia-wyjścia, jeśli wykonanie procedury przerwania zajmuje 2.5μs? Należy zignorować czas wyszukiwania ścieżki i sektora.

Do systemu dodajemy kontroler DMA. Przerwanie będzie generowane tylko raz po wczytaniu sektora do pamięci. Jak zmieniła się zajętość procesora?

Wskazówka W tym zadaniu procedura obsługi przerwania zajmuje się przetworzeniem przeczytanych danych i będzie wywołana za każdym razem, gdy zgłoszono przerwanie.

Zadanie 10. W przeważającej większości systemów implementujących moduły DMA, procesor ma niższy priorytet dostępu do pamięci głównej niż moduły DMA. Dlaczego?

Wskazówka: Co się może stać, jeśli urządzenia nie mają gwarancji wykonywania transferów w regularnych odstępach czasu?

[2] `https://sourceware.org/binutils/docs-2.26/as/Pseudo-Ops.html`