Zadanie 3

Opened: Monday, 20 May 2024, 12:00 AM
Due: Wednesday, 12 June 2024, 11:59 PM

Pliki z dziurami

Pliki w Linuksie mogą być dziurawe. Na potrzeby tego zadania przyjmujemy, że plik z dziurami składa się z ciągłych fragmentów. Na początku fragmentu jest dwubajtowa długość w bajtach danych we fragmencie. Potem są dane. Fragment kończy się czterobajtowym przesunięciem, które mówi, o ile bajtów trzeba się przesunąć od końca tego fragmentu do początku następnego fragmentu. Długość danych w bloku jest 16-bitową liczbą w naturalnym kodzie binarnym. Przesunięcie jest 32-bitową liczbą w kodowaniu uzupełnieniowym do dwójki. Liczby w pliku zapisane są w porządku cienkokońcówkowym (ang. *little-endian*). Pierwszy fragment zaczyna się na początku pliku. Ostatni fragment rozpoznaje się po tym, że jego przesunięcie wskazuje na niego samego. Fragmenty w pliku mogą się stykać i nakładać.

Suma kontrolna pliku

Sumę kontrolną pliku obliczamy za pomocą cyklicznego kodu nadmiarowego (ang. *CRC*, *cyclic redundancy code*), biorąc pod uwagę dane w kolejnych fragmentach pliku. Dane pliku przetwarzamy bajtami. Przyjmujemy, że najbardziej znaczący bit bajtu z danymi i wielomianu (dzielnika) CRC zapisujemy po lewej stronie.

Polecenie

Zaimplementuj w asemblerze program `crc`, który oblicza sumę kontrolną danych zawartych w podanym pliku z dziurami:

```
./crc file crc_poly
```

Parametr `file` to nazwa pliku. Parametr `crc_poly` to ciąg zer i jedynek opisujący wielomian CRC. Nie zapisujemy współczynika przy najwyższej potędze. Maksymalny stopień wielomianu CRC to 64 (maksymalna długość dzielnika CRC to 65). Przykładowo `11010101` oznacza wielomian $x^8 + x^7 + x^6 + x^4 + x^2 + 1$. Wielomian stały uznajemy za niepoprawny.

Program wypisuje na standardowe wyjście obliczoną sumę kontrolną jako napis składający się z zer i jedynek, zakończony znakiem nowego wiersza `\n`. Program sygnalizuje poprawne zakończenie kodem `0`.

Program powinien korzystać z funkcji systemowych Linuksa: `sys_open`, `sys_read`, `sys_write`, `sys_lseek`, `sys_close`, `sys_exit`.

Program powinien sprawdzać poprawność parametrów i poprawność wykonania funkcji systemowych (z wyjątkiem `sys_exit`). Jeśli któryś z parametrów jest niepoprawny lub wywołanie funkcji systemowej zakończy się błędem, to program powinien zakończyć się kodem `1`. W każdej sytuacji program powinien przed zakończeniem jawnie wywołać funkcję `sys_close` dla pliku, który otworzył.

Dla uzyskania zadowalającej szybkości działania programu odczyty należy buforować. Należy dobrać optymalny rozmiar bufora, a informację o tym umieścić w komentarzu.

Oddawanie rozwiązania

Jako rozwiązanie należy wstawić w Moodle plik o nazwie `crc.asm`.

Kompilowanie

Rozwiązanie będzie kompilowane poleceniami:

?

```
nasm -f elf64 -w+all -w+error -o crc.o crc.asm
ld --fatal-warnings -o crc crc.o
```

Rozwiązanie musi się kompilować i działać w laboratorium komputerowym.

Przykłady użycia

Na komputerach w laboratorium i na maszynie `students` rozwiązanie powinno wypisywać poniższe wyniki:

```
$ ./crc /home/students/inf/PUBLIC/SO/zadanie_3_2024/plik1 1; echo $?
1
0
```

```
$ ./crc /home/students/inf/PUBLIC/SO/zadanie_3_2024/plik1 011; echo $?
010
0
```

```
$ ./crc /home/students/inf/PUBLIC/SO/zadanie_3_2024/plik1 11010101; echo $?
01010111
0
```

```
$ ./crc /home/students/inf/PUBLIC/SO/zadanie_3_2024/plik2 1; echo $?
1
0
```

```
$ ./crc /home/students/inf/PUBLIC/SO/zadanie_3_2024/plik2 011; echo $?
111
0
```

```
$ ./crc /home/students/inf/PUBLIC/SO/zadanie_3_2024/plik2 11010101; echo $?
10110011
0
```

```
$ ./crc /home/students/inf/PUBLIC/SO/zadanie_3_2024/plik 1; echo $?
1
```

```
$ ./crc /home/students/inf/PUBLIC/SO/zadanie_3_2024/plik1 ""; echo $?
1
```

Inne przykłady można wygenerować za pomocą kalkulatora CRC.

Ocenianie

Zgodność rozwiązania ze specyfikacją będzie oceniania za pomocą testów automatycznych, za które można otrzymać maksymalnie 7 punktów. Za błędną nazwę pliku odejmiemy jeden punkt.

Za jakość kodu i styl programowania można dostać maksymalnie 3 punkty. Tradycyjny styl programowania w asemblerze polega na rozpoczynaniu etykiet od pierwszej kolumny, a mnemoników rozkazów od wybranej ustalonej kolumny. Nie stosuje się innych wcięć. Taki styl mają przykłady pokazywane na zajęciach. Kod powinien być dobrze skomentowany, co oznacza między innymi, że każdy blok kodu powinien być opatrzony informacją, co robi. Należy opisać przeznaczenie rejestrów. Komentarza wymagają wszystkie kluczowe lub nietrywialne linie kodu. W przypadku asemblera nie jest przesadą komentowanie prawie każdej linii kodu, ale należy unikać komentarzy opisujących to, co widać.

W tym zadaniu priorytetem jest szybkość działania programu, ale oceniane będą też rozmiary sekcji i wykorzystanie stosu.

Wystawienie oceny może zostać uzależnione od osobistego wyjaśnienia szczegółów działania programu prowadzącemu zajęcia.

Rozwiązania należy implementować samodzielnie pod rygorem niezaliczenia przedmiotu. Zarówno korzystanie z cudzego kodu, jak i prywatne lub publiczne udostępnianie własnego kodu jest zabronione.

Contact us

Follow us

Contact site support

You are logged in as Witold Formański (Log out)

Data retention summary

Get the mobile app

Get the mobile app

This theme was developed by

conecti.me

Moodle, 4.1.10 (Build: 20240422) | moodle@mimuw.edu.pl