lab_5 - Instrukcja do ćwiczenia

Teoria (lab_5.pdf):

Pierwsze zadanie polega na utworzeniu pliku dyskowego, zapisie do pliku danych w formie tekstowej oraz zamknięciu pliku. W drugim zadaniu celem jest otwarcie już istniejącego pliku i wczytanie zawartych w nim danych do pamięci, następnie plik jest zamykany a wczytana zawartość jest wyświetlana na ekranie. Realizacja poszczególnych etapów odbywa się poprzez wywoływanie odpowiednich funkcji systemowych:

- int **creat**(const char *pathname, mode_t mode);
- ssize_t **write**(int fd, const void *buf, size_t count);
- int **open**(const char *pathname, int flags, mode_t mode);
- ssize_t **read**(int fd, void *buf, size_t count);
- int **close**(int fd);

Nazwy plików będące argumentami funkcji **creat** i **open** muszą być łańcuchami znaków zakończonymi bajtem o wartości **0** (tak, jak łańcuchy znaków w C) – najłatwiej to uzyskać wykorzystując do definiowania nazw dyrektywy **.asciz** lub **.string** (są tożsame).

Bufor na wczytywane dane można utworzyć przy pomocy dyrektywy **.space** – jej parametrami są wielkość rezerwowanego obszaru pamięci (w bajtach) i ewentualna wartość początkowa.

W programach zawarto najprostszą obsługę błędów – wystąpienie błędów w trakcie tworzenia pliku, jego otwierania lub zamykania, a także podczas zapisywania lub odczytywanie danych są sygnalizowane poprzez wyświetlenie stosownego komunikatu na **stderr**. Źródłem informacji o tym, że wystąpił błąd w trakcie wykonywania funkcji systemowej jest ujemna wartość rejestru **%rax** (traktowana jako liczba ze znakiem) lub wartość **1** najbardziej znaczącego bitu w rejestrze **%rax** (jeśli uznamy, że to liczba bez znaku). W przypadku operacji zapisu i odczytu danych oprócz takiej sygnalizacji (dotyczącej błędów poważnych) celowe jest też uwzględnienie innych przypadków – wykrywane są następujące sytuacje:

- zapis danych:
  - **%rax** < **0** – błąd poważny
  - **%rax** >= **0** (liczba zapisanych bajtów) && liczba zapisanych bajtów jest różna od liczby bajtów, które miały zostać zapisane

  powyższe warunki można połączyć w jeden: liczba zapisanych bajtów jest różna od liczby bajtów, które miały być zapisane
- odczyt danych:
  - **%rax** < **0** – błąd poważny

- **%rax** >= **0** (liczba wczytanych bajtów) && liczba wczytanych bajtów jest większa lub równa od liczby bajtów, które miały zostać wczytane (faktycznie może wystąpić tylko równość – większość ma tylko pokazać, że w pliku być może jest więcej znaków, niż jesteśmy w stanie jednorazowo wczytać (przy zadanej wielkości bufora))

W kodach źródłowych obu programów dokonano podziału wszystkich instrukcji na bloki realizujące konkretne zadania (np. utworzenie pliku, otwarcie pliku, zapis danych, odczyt danych, zamknięcie pliku, wyświetlenie komunikatu, wyświetlenie zawartości bufor, itd.) – elementem rozdzielającym poszczególne fragmenty są komentarze w postaci:

„#----------”

- należy pamiętać aby przy ewentualnych modyfikacjach programów nie rozdzielać instrukcji znajdujących się w konkretnym bloku, a jedynie przestawiać kompletne bloki (zmieniać ich kolejność).

Praktyka (lab_5a.s i lab_5b.s):

Dane:

- **file_n** – łańcuch znaków będący nazwą pliku
- **file_h** – identyfikator pliku (ang. *file handle*) - zwracany przez funkcje **creat** i **open**

Kod:

- **program główny lab_5a**
  - utworzenie pliku
  - zapis danych
  - zamknięcie pliku
  - wyświetlenie komunikatu „All OK”
  - wyświetlenie komunikatu „File error!” (o ile faktycznie wystąpił)
  - zakończenie działania programu

- **program główny lab_5b**
  - otwarcie pliku
  - odczyt danych
  - zamknięcie pliku
  - wyświetlenie komunikatu „File contains …”
  - wyświetlenie wczytanych danych
  - wyświetlenie komunikatu „All OK”
  - wyświetlenie komunikatu „File too big!” (o ile faktycznie wystąpił)
  - wyświetlenie komunikatu „File error!” (o ile faktycznie wystąpił)
  - zakończenie działania programu

Działania:

1. C (Compile) – polecenie: **as –o lab_5a.o lab_5a.s**
2. L (Link) – polecenie: **ld –o lab_5a lab_5a.o**
3. Linker sygnalizuje brak symbolu „**stderr**”.
4. Dodajemy w kodzie żródłowym definicję:

```
.equ stderr, 2
```

5. CLR (Compile, Link, Run) – polecenie: **./lab_5a**
6. Pojawiają się jednocześnie oba komunikaty **„File error!”** i **„All is OK”** oraz trochę „śmieci”. Ponieważ z logiki programu wynika, że nie jest możliwe wyświetlenie obu komunikatów, świadczy to (w połączeniu z pojawieniem się „śmieci”) o tym, że wyświetlony komunikat zawiera zbyt dużą liczbę znaków. Problem polega na niewłaściwym użyciu zmiennych, które przechowują informacje o długościach poszczególnych napisów – chodzi o symbole: **txtlen**, **errlen** i **alloklen**.

`$variable` – adres zmiennej

`variable` – wartość zmiennej

7. Usuwamy $ przy powyższych symbolach w instrukcjach **MOV …, %rdx**, bo chcemy uzyskać odwołania do wartości zmiennych, a nie ich adresów.
8. CLR
9. Pojawia się komunikat „All is OK”.
10. Wyświetlamy zawartość utworzonego pliku poleceniem **cat testfile.txt** lub **more testfile.txt**
11. Powinien pojawić się napis: „A line of text”
12. Sprawdzamy wielkość pliku i prawa dostępu poleceniem **ls –l testfile.txt**
13. Wielkość pliku to **15** bajtów, prawa dostępu mogą być różne w różnych systemach – jedna z możliwych wartości to:

```
-rwx------ 1 buba buba  15 kwi  7 22:29 testfile.txt
```

14. Podejrzewamy, że za prawa dostępu w jakimś stopniu odpowiada wartość stałej **mode**: **0x180**
15. W zapisie bitowym **x0180** to **1 1000 0000**. Jeśli pogrupujemy kolejne bity po **3**, to uzyskamy zapis: **110 000 000** – każda trójka określa uprawnienia do odczytu, zapisu i wykonania (**rwx**) odpowiednio dla użytkownika, grupy i reszty.
16. Sprawdzamy hipotezę zmieniając wartość stałej **mode** na **0x1FF** (**rwx rwx rwx**).
17. Usuwamy plik poleceniem: **rm testfile.txt**
18. CLR
19. Ponownie sprawdzamy prawa dostępu – przykładowy wynik wygląda następująco:

```
-rwxr-xr-x 1 buba buba  15 kwi  7 22:31 testfile.txt
```

20. Widać, że zmiany w programie wpłynęły jakoś na prawa dostępu, ale uzyskany efekt odbiega od zamierzonego – problemem jest nieuwzględnienie wartości **umask**.
21. Sprawdzamy wartość poleceniem **umask** – przykładowy rezultat (wartości mogą być różne) wygląda następująco:

```
buba@buba-pc:~/asm/l5$ umask
0022
```

22. Wynikowe prawa dostępu (takie, jakie plik otrzyma) zależą zarówno od wartości stałej **mode** jak i wartości **umask**: formuła ma postać: **rights = mode & ~umask**, co oznacza, że w powyższym przykładzie możliwe jest ustawienie dowolnych praw dla użytkownika, a dla grupy i reszty można tylko ustawić prawa **rx**. Aby ustawić dowolne prawa dostępu w sposób programowy (poprzez wartość stałej mode), konieczne jest wcześniejsze zmodyfikowanie **umask** na **0**.
23. Usuwamy plik poleceniem: **rm testfile.txt**
24. Zmieniamy wartość **umask** i sprawdzamy efekt poleceniami:

```
buba@buba-pc:~/asm/l5$ umask 0
buba@buba-pc:~/asm/l5$ umask
0000
```

25. R

26. Ponownie sprawdzamy prawa dostępu – przykładowy wynik wygląda następująco:

```
-rwxrwxrwx 1 buba buba  15 kwi  7 22:36 testfile.txt
```

27. Okazuje się, że dla **umask** = **0** mamy pełną kontrolę nad prawami dostępu (ogólnie rzecz biorąc, poprzez zmiany w wartości stałej **mode** możemy wymusić dowolne atrybuty pliku).
28. Przechodzimy do pliku **lab_5b** – ten program ma za zadanie otworzyć już istniejący plik, wczytać jego zawartość do bufora w pamięci, zamknąć plik, wyświetlić zawartość bufora i zakończyć swoje działanie. Dodatkowy efektem jest wyświetlenie dwóch komunikatów: „File contains..” i „All is OK” w trakcie nomalnego działania lub innych komunikatów w przypadku pojawienia się błędów: „File error!” i „File too big!”.
29. C (Compile) – polecenie: **as –o lab_5b.o lab_5b.s**
30. L (Link) – polecenie: **ld –o lab_5b lab_5b.o**
31. Linker sygnalizuje brak symboli „**stdout**” i „**stderr**”.
32. Dodajemy w kodzie źródłowym definicje:

```
.equ stdout, 1

.equ stderr, 2
```

33. CLR (Compile, Link, Run) – polecenie: **./lab_5b**
34. Podobnie jak wcześniej dla **lab_5a** pojawiają się jednocześnie różne komunikaty oraz trochę „śmieci”. Problem znowu polega na niewłaściwym użyciu zmiennych, które przechowują informacje o długościach poszczególnych napisów – chodzi o symbole: **txtlen**, **errlen**, **alloklen** oraz **bufsize** i **b_read**.

```
$variable       – adres zmiennej

variable        – wartość zmiennej
```

35. Usuwamy $ przy powyższych symbolach w wszystkich instrukcjach gdzie zostały użyte, bo chcemy uzyskać odwołania do wartości zmiennych, a nie ich adresów.
36. CLR
37. Pojawia się komunikat „File contains …”, następnie zawartość pliku i komunikat „All is OK”, co świadczy o prawidłowym działaniu.
38. Kiedy pojawi się komunikat „File too big!”?
39. Komunikat taki zostanie wyświetlony jeśli **b_read >= bufsize** tzn. liczba wczytanych bajtów będzie większa lub równa od rozmiaru bufora (i jednocześnie liczby znaków, które chcieliśmy wczytać) – tak naprawdę większa nie będzie nigdy, jedyna możliwość to taka, że będzie równa – chodzi o pokazanie, że plik zawiera być może więcej danych niż można wczytać za jednym razem. Bufor ma rozmiar **128** bajtów, więc jeśli plik będzie zawierać np. **150** znaków, to przy próbie odczytu powinien pojawić się komunikat „File too big!”.
40. Wracamy do programu **lab_5a** aby zwiększyć ilość zapisywanych danych.
41. Tworzymy pętlę w której dane zapisane zostaną **10** razy – pojedynczy zapis daje **15** znaków, więc uzyskamy łącznie **150** znaków. Lokalizujemy blok odpowiedzialny za zapis

danych (pojawia się w kodzie jako drugi) i dodajemy przed i po nim następujące instrukcje:

```
    MOV  $10, %r15
more:

    write to file block

    DEC  %r15
    JNZ more
```

42. CLR – (**lab_5a**)
43. Program powinien wyświetlić komunikat „All is OK” a dysku powinien pojawić się plik o wielkości 150 bajtów:

```
buba@buba-pc:~/asm/l5$ ./lab_5a

All is OK - too hard to believe!
buba@buba-pc:~/asm/l5$ ls -l *.txt
-rwxrwxrwx 1 buba buba 150 kwi  7 22:45 testfile.txt
```

44. Wracamy do pliku **lab_5b** aby sprawdzić jego działanie na powiększonym pliku.
45. R (**lab_5b**)
46. Pojawia się komunikat „File too big!”:

```
buba@buba-pc:~/asm/l5$ ./lab_5b
File too big!
```

47. Teraz modyfikujemy działanie programu **lab_5b** tak, aby był w stanie wyświetlić zawartość pliku o dowolnej wielkości. Oznacza to konieczność wczytywania danych i wyświetlania wczytanej zawartości w pętli typu **do … while** – warunkiem zakończenia będzie wyczerpanie danych (wczytanych zostanie mniej danych niż chcemy, **b_read<bufsize** – czyli warunek wykorzystany wcześniej w instrukcji skoku warunkowego do etykiety **toobig**, może teraz być wykorzystany do skoku na początek pętli). Wprowadzenie pętli wiąże się z koniecznością zmiany kolejności wykonywanych operacji (poprzestawiania bloków), Kolejność bloków w wersji pierwotnej (po lewej) i po przestawieniu (po prawej) jest następująca:

open		open
read		show message
close	more:	
check toobig		read
show message		show content
show content		check more
show OK		close
		show OK

48. Lokalizujemy odpowiednie bloki w kodzie programu i zmieniamy ich kolejność pamiętając, aby nie rozdzielać instrukcji wchodzących w skład danego bloku.
49. CLR (**lab_5b**)
50. Po poprawnej realizacji zadania efekt powinien być następujący:

```
buba@buba-pc:~/asm/l5$ ./lab_5b
File contains following characters:
A line of text
A line of text
A line of text
A line of text
A line of text
A line of text
A line of text
A line of text
A line of text
A line of text
All is OK - too hard to believe!
```

51. Radujemy się w dwójnasób:
    a. bo znowu się udało!
    b. bo Wielkanoc!