**LAB 0 AK2**

1. Podstawy kompilacji programów w systemie Linux:

* stworzenie pliku “*makefile*” (bez rozszerzenia) w katalogu “*lab1*”;
* stworzenie plików “*hello.s*”, “*writeText.s*”, “*compareText.s*”;
* stworzenie reguł budowy i kompilacji utworzonych plików “*\*.s*” w pliku “*makefile”*:

**CEL: ŹRÓDŁA**

**KOMENDA\***

* + **\***proszę pamiętać o obowiązkowym znaku tabulacji i o specyficznej kolejności tworzenia poszczególnych składników programu wykonywalnego;
  + do wykorzystania komendy “***as***” oraz “***ld***” (kompilator, linker) z odpowiednimi parametrami;
  + pierwsza linia pliku “*makefile*” ma przyjąć postać:

all: hello writeText compareText

* + dodanie do każdego z plików “*\*.s*” następującego szkieletu programu:

SYSEXIT = 1

EXIT\_SUCCESS = 0

.global \_start

\_start:

mov $SYSEXIT, %eax

mov $EXIT\_SUCCESS, %ebx

int $0x80

* + zapoznanie się ze szkieletem programu (z dyrektywami, rejestrami, przerwaniem i nazwami symbolicznymi) - do opisania przez studenta;
  + uruchomienie pliku “*makefile”* poleceniem **make** (plik musi mieć prawo wykonywania - nadaje się je komendą: **chmod +x makefile**);
  + efektem wykonania polecenia powinny być 3 pliki “*\*.o*” oraz 3 pliki binarne, wykonywalne (do podejrzenia komendą **ls -ali** w katalogu **lab1**);
  + sprawdzenie działania plików binarnych poprzez wywołanie **./nazwaPliku** (efektem wykonania każdego z nich powinno być pojawienie się kolejnej linii komend w konsoli).

2. Program “*hello*”:

* rozszerzenie zawartości pliku *“hello.s”* o następujące nazwy symboliczne:

SYSWRITE = 4

STDOUT = 1

* utworzenie zmiennych:

.text

msg: .ascii "Hello! \n"

msg\_len = . - msg

* algorytm działania programu:
  + rozpocznij wykonywanie w punkcie \_start;
  + z polecenia korzystając mov umieść w rejestrach procesora **eax, ebx, ecx, edx** w kolejności: numer funkcji, argument 1 (strumień wyjściowy), argument 2 (wiadomość), argument 3 (długość wiadomości);
  + wykonaj przerwanie systemowe (interrupt);
  + wykonaj funkcję wyjścia (jak w szkielecie programu z punktu 1);
* po zapisaniu pliku *“hello.s”* należy wykonać polecenie **make** i uruchomić powstały plik wykonywalny “hello” (**./hello**);
* efektem działania programu powinno być wyświetlenie się napisu “Hello! “, po którym pojawi się znak nowej linii, a wraz z nią kolejna pusta linia komend w konsoli.

3. Program “*writeText*”:

* rozszerzenie zawartości pliku *“hello.s”* o następujące nazwy symboliczne:

SYSREAD = 3

STDIN = 0

* utworzenie zmiennych:

.text

msg: .ascii "Write text (5): \n"

msg\_len = . - msg

msg2: .ascii "Written text: "

msg2\_len = . - msg2

newline: .ascii "\n"

newline\_len = . - newline

* utworzenie bufora (5 znakowego):

.data

buf: .ascii " "

buf\_len = . - buf

* program “*writeText”* jest rozszerzeniem programu “*hello”*, algorytm:
  + wyświetl tekst zawarty w msg (analogicznie jak w “*hello”*);
  + korzystając z **SYSREAD** pobierz znaki z klawiatury i umieść je w **buf**;
  + wyświetl tekst zawarty w **buf** poprzedź go tekstem z **msg2**, a na końcu umieść **newline**.

4. Program “*compareText*”:

* bazując na programie z zadania nr 3 należy napisać program porównujący ciąg znaków zadany z klawiatury z ciągiem zapisanym w zmiennej; należy wykorzystać dodatkowe polecenia:

CMP, JNE

* należy prócz tego skorzystać z rejestru **akumulatora**:

%al

* należy określić ograniczenie jakie nakłada na program “*compareText*” wykorzystanie rejestru akumulatora.