

Podstawy programowania

Wprowadzenie

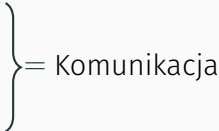
Radosław Roszczyk

7 października 2021

O czym będzie

1. Wprowadzenie
2. Języki programowania
3. Od kodu źródłowego do wykonania programu

Wprowadzenie

- Tworzenie oprogramowania to proces iteracyjny, obejmujący kilka kluczowych etapów:
 - Projektowanie
 - Implementacje
 - Testowanie
 - Wdrożenie= Komunikacja
- Poszczególne fazy wykonywane są cyklicznie,
- Błędy pojawiające się w programie bądź zmiany ze strony użytkownika powodują cofnięcie się do poprzedniej fazy i wprowadzenie zmian lub poprawek,
- Praca zespołowa,
- Mechanical Sympathy.

Mechanical Sympathy

Mechanical Sympathy to pojęcie ukute w branży motoryzacyjnej i jak wiele innych zagościło na naszym programistycznym podwórku.

- Aby być dobrym programistą powinno się wiedzieć co siedzi po spodem oprogramowania, bowiem zrozumienie sprzętu sprawia, że stajemy się jeszcze lepszymi programistą.
- Mechaniczna sympatia polega na korzystaniu z narzędzia lub systemu ze zrozumieniem tego, w jaki sposób działa on najlepiej.

*Nie musisz być inżynierem, aby zostać kierowcą wyścigowym,
ale musisz mieć Wyczucie Sprzętu.*

– Jackie Stewart, kierowca wyścigowy

Architektura komputera

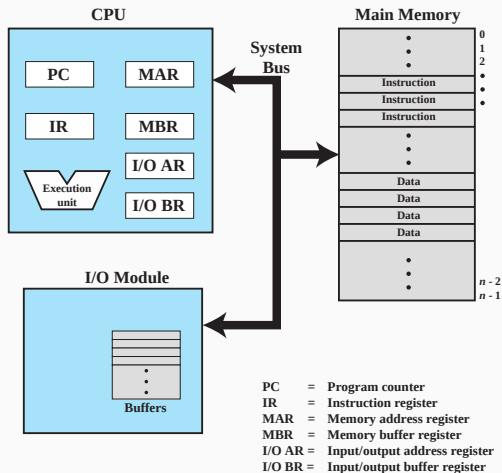
Architektura Von Neumanna

Dane oraz kod programu są przechowywane w tej samej pamięci. Główne komponenty systemu:

- Procesor
- Pamięć
- Urządzenia wejścia / wyjścia

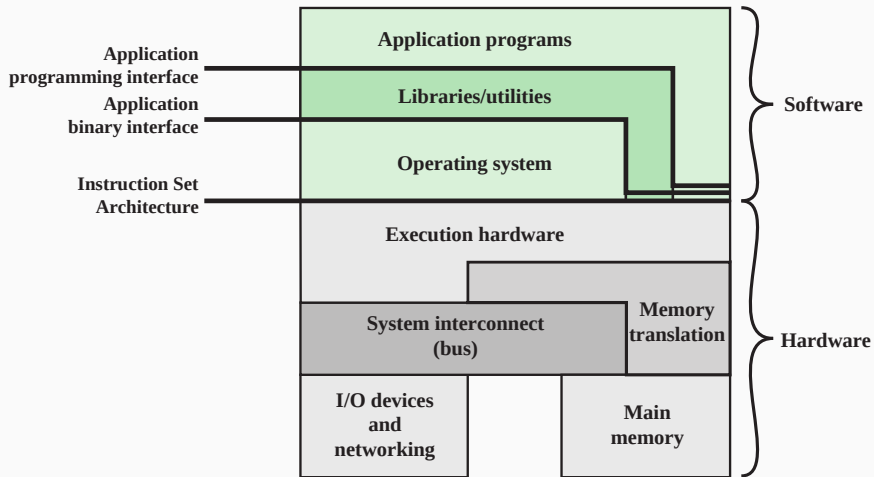
Architektura harwardzka

Kod programu jest przechowywany w innej pamięci niż dane.



Rysunek 1: Podstawowe komponenty komputera

Struktura systemu komputerowego

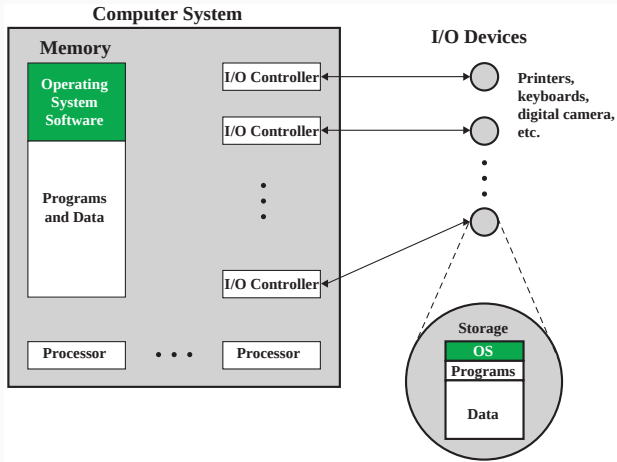


Rysunek 2: Sprzęt komputerowy i struktura oprogramowania

Cele i funkcje systemu operacyjnego

Podstawowe zadania systemu operacyjnego

- Zarządzanie zasobami
- Zarządzanie procesami
- Zarządzanie pamięcią
- Zarządzanie plikami
- Zarządzanie we/wy
- Zarządzanie urządzeniami
- Obsługa błędów
- Interfejs użytkownik – komputer



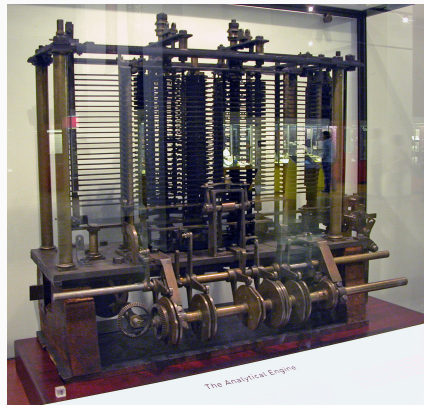
Rysunek 3: System operacyjny jako menadżer zasobów

Języki programowania

Augusta Ada King, hrabina Lovelace

A wszystko zaczęło się od... Ady

- Brytyjska matematyczka i poetka, córka Lorda Byrona
- Pierwsza programistka – stworzyła pierwszy opublikowany algorytm dla maszyny analitycznej Babbage'a 1843 rok



Początek języków programowania

- Short Code – 1949-50 Tonik and Logan – SC for Univac 1952
- FORTRAN – 1954 (Formula Translation)
- LISP – 1958 – (Locator/Identifier Separation Protocol)
- COBOL – 1959 (Common Business Oriented Language)
- **1936 – Rachunek Lambda**
- 1951 – Regional Assembly Language
- **1952 – Autocode**
- 1954 – IPL (forerunner to LISP)
- **1957 – FORTRAN (First compiler)**
- **1958 – LISP (forerunner to Clojure)**
- 1958 – ALGOL 58
- 1959 – FACT (forerunner to COBOL)
- **1959 – COBOL**
- 1962 – Simula (forerunner to C++)
- 1963 – CPL (forerunner to C)
- **1973 – C**
- **1979 – C++**
- 1979 – Ada
- 1987 – Perl
- 1990 – Haskell
- **1991 – Python**
- **1996 – Java**
- 1997 – R
- **2001 – C#**
- 2007 – Clojure
- 2009 – Go

Klasyfikacja języków programowania

- język kompilowany
 - kod programu kompilowany jest do kodu maszynowego,
 - wymaga konkretnej architektury lub maszyny wirtualnej,
 - przykładowe języki: Assembler, C, C++, C#, JAVA, Pascal
- język interpretowany
 - kod programu przechowywany jest w postaci źródłowej,
 - wykonanie przez interpreter w momencie uruchomienia,
 - przykłady: JavaScript, Python, PHP, Perl, R, Ruby
- język skryptowy
 - kod programu przechowywany jest w postaci źródłowej,
 - wykonanie przez środowisko uruchomieniowe,
 - przykłady: AutoCAD, CGI, VBA, Matlab, (PHP, Perl, Python)

Generacje języków programowania

- **języki pierwszej generacji** – języki maszynowe, czyli języki procesorów, instrukcje zapisane są w postaci binarnej,
- **języki drugiej generacji** – języki symboliczne, asemblery. Składniowo tożsame z maszynowymi, używają mnemoników,
- **języki trzeciej generacji** – języki wysokiego poziomu, proceduralne (imperatywne). Jedna instrukcja jest tłumaczona na kilka instrukcji procesora. Przykład – język C.
- **języki czwartej generacji** – języki bardzo wysokiego poziomu, nieproceduralne (deklaratywne). Semantyka wielu języków tej generacji przypomina składnię języka naturalnego. Przykład – język SQL.
- **języki piątej generacji** – języki sztucznej inteligencji, najbardziej zbliżone do języka naturalnego. Przykład – język PROLOG.

Paradygmaty programowania

Paradygmat programowania pozwala klasyfikować język programowania ze względu na jego cechy, między innymi: sposób sterowania, model wykonania, kolejność operacji, grupowanie kodu w określone jednostki, styl składni bądź gramatykę.

- programowanie imperatywne / obiektowe
 - kod programu jest wykonywany sekwencyjnie
 - wykonanie kolejnych instrukcji zmienia stan programu
 - przykładowe języki: C, C#, C++, JAVA, Pascal
- programowanie funkcyjne
 - wykonanie programu polega na obliczaniu funkcji matematycznych
 - funkcje przedstawiają zależności między danymi wejściowymi i wyjściowymi
 - przykładowe języki: LISP, Scala, Clojure, Python, Perl, R

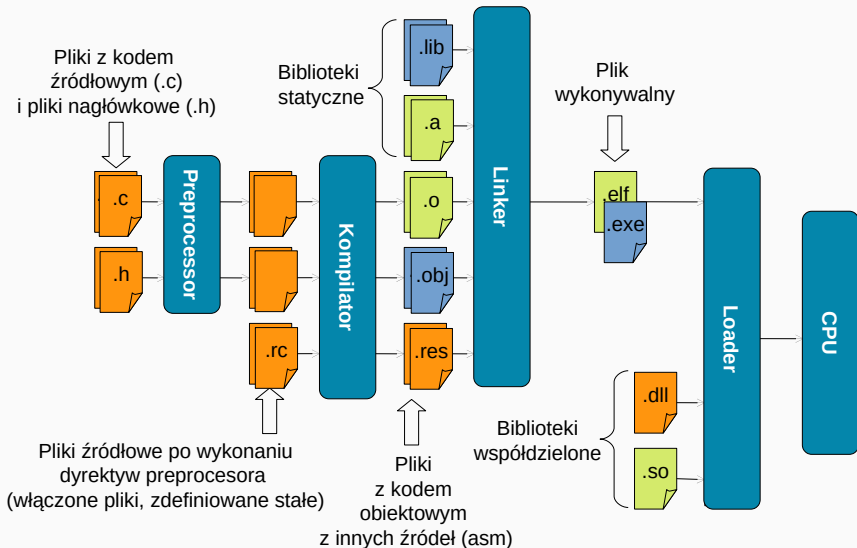
Od kodu źródłowego do wykonania programu

Kod programu należy przetłumaczyć do postaci zrozumiałej przez komputer, tzn. możliwej do wykonania przez procesor. W przypadku języka C proces ten obejmuje kilka etapów:

- Preprocessing kodu źródłowego, podczas którego preprocesor interpretuje dyrektywy (m.in. wstawia zawartość plików włączonych dyrektywą `#include`, usuwa fragmenty objęte klauzulą `#ifdef/#endif`, `#ifndef/#endif` oraz podstawia tekst zdefiniowany dyrektywami `#define`)
- Właściwa kompilacja do postaci tzw. pliku obiektowego
- Zasadniczo jeden plik źródłowy `.c` stanowi pojedynczą jednostkę kompilacji, kompilowaną do osobnego pliku obiektowego
- Łączenie (linkowanie, konsolidacja) wszystkich plików obiektowych do pojedynczego pliku wykonywalnego

- Zamiast pliku wykonywalnego wynikiem linkowania może być również:
 - Biblioteka statyczna
 - Biblioteka dynamiczna
- Biblioteka to plik zawierający wiele plików obiektowych, definiujących funkcjonalność, z której możemy skorzystać w swoich programach
 - Biblioteki statyczne (zwykle pliki z rozszerzeniem .a lub .lib) dołączamy do programu na etapie linkowania
 - Biblioteki dynamiczne (w systemie Windows pliki .dll, w systemie Unix .so) są dołączane do programu dopiero w momencie jego uruchomienia

Proces kompilacji



Jak skompilować program

Cały proces budowania wersji wykonywalnej programu możemy uruchomić z wiersza poleceń, np.: za pomocą kompilatora gcc stanowiącego składnik GNU Compiler Collection

Kompilacja programu z linii komend

```
1 gcc program.c -o program
```

Pierwszy argument, to nazwa pliku z kodem źródłowym Po opcji -o (ang. output) podajemy nazwę pliku wynikowego.

w systemach z rodziny Unix rozszerzenie .exe nie jest konieczne. Programy wykonywalne muszą natomiast mieć ustawiony atrybut wykonalności. W powyższym przykładzie gcc ustawi go automatycznie, ale w razie potrzeby warto pamiętać, że służy do tego polecenie chmod

Jak skompilować program z wielu plików

Poprzedni przykład, to bardzo prosty przypadek (cały program zawarty był w pojedynczym pliku źródłowym) możliwe jest również podanie wielu plików źródłowych.

Kompilacja programu z linii komend

```
1 gcc pr1.c pr2.c pr3.c -o program
```

UWAGA: Wszystkie etapy budowania pliku wykonywalnego (preprocessing, kompilacja, linkowanie) realizowane są tym jednym poleceniem. Możemy jednak zobaczyć wyniki pośrednie używając odpowiednich opcji narzędzia gcc

- E (wynik preprocessingu – w postaci tekstowej)
- S (wynik kompilacji – podgląd kodu assemblera)
- c (wynik kompilacji – w postaci plików obiektowych .o)

Kompilator oprócz parametrów przedstawionych wcześniej może

Kompilacja programu z linii komend

```
1 gcc pr1.c pr2.c pr3.c -o program
```

UWAGA: Wszystkie etapy budowania pliku wykonywalnego (preprocessing, kompilacja, linkowanie) realizowane są tym jednym poleceniem. Możemy jednak zobaczyć wyniki pośrednie używając odpowiednich opcji narzędzia gcc

- E (wynik preprocessingu – w postaci tekstowej)
- S (wynik kompilacji – podgląd kodu assemblera)
- c (wynik kompilacji – w postaci plików obiektowych .o)

Automatyzacja procesu kompilacji

W przypadku większych programów proces kompilacji jest dosyć złożony ze względu na dużą liczbę plików źródłowych oraz wykorzystywanych bibliotek.

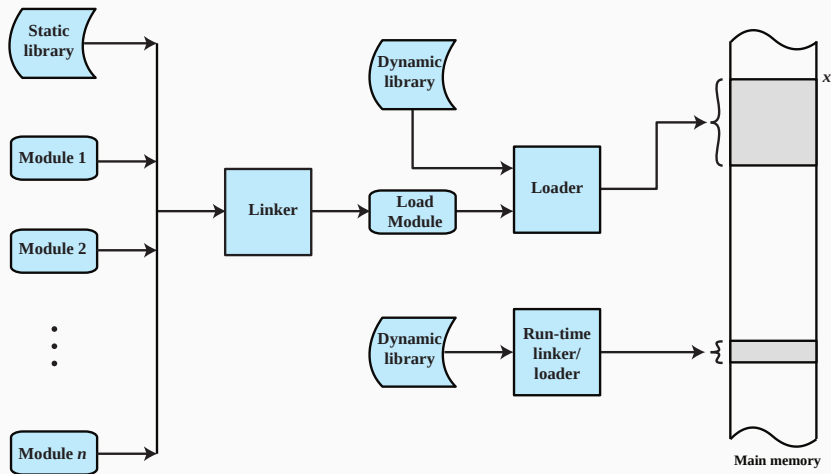
- Osobną grupę narzędzi wspomagających kompilację i budowanie wersji wykonywalnej programu (zwłaszcza w przypadku rozbudowanych aplikacji) stanowią rozwiązania automatyzujące ten proces
- Jednym z najpopularniejszych narzędzi tego typu jest make
- Autor aplikacji oprócz plików z kodem źródłowym przygotowuje specjalny plik Makefile, zawierający instrukcje co i w jakiej kolejności należy kompilować i linkować
- W pliku Makefile można podać m.in. opcje kompilatora i linkera, ścieżki do katalogów, nazwy bibliotek itd.

Kod źródłowy prezentowanych aplikacji oraz materiały pomocnicze będą dostępne na stronie

https://github.com/rroszczyk/1DM1103_2021

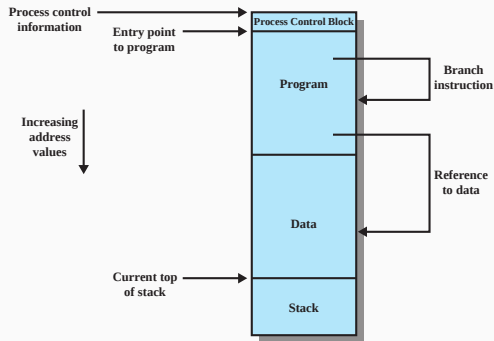
Pytania?

Scenariusz linkowania i ładowania programu

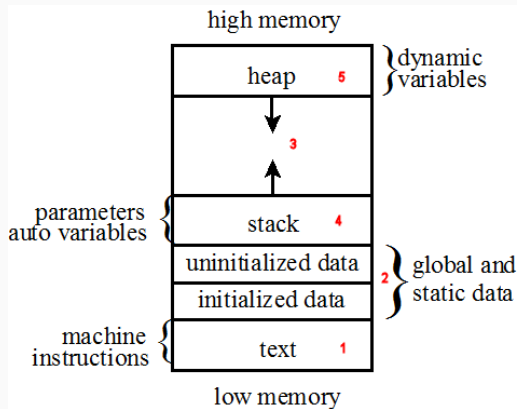


Rysunek 4: Scenariusz linkowania i ładowania programu

Organizacja pamięci procesu



Rysunek 5: Schemat adresacji pamięci dla procesu



Rysunek 6: Organizacja pamięci programu