


A decorative graphic on the left side of the slide. It consists of a blue parallelogram and a light green parallelogram, both tilted at an angle. The blue shape is in the foreground, and the green shape is partially behind it. They are set against a dark blue background with faint, lighter blue diagonal stripes.

# Bootloader



# Plan Prezentacji

2. Opis przebiegu bootowania
3. Krótko MBR
4. Krótko windows a,linux
5. Setup boot w raspberry pi,konfiguracja uruchamiania różnych komputerów bios rejestry etc
6. Funkcjonalność dawana przez os i ograniczenia związane z jego brakiem



# BIOS(Basic Input Output System)

BIOS który jest oprogramowaniem sprzętowym(firmware) odpowiedzialnym za start komputera do momentu załadowania programu z MBR w tym POST. Kiedyś pośredniczył między systemem operacyjnym a hardwarem ale dzisiaj systemy operacyjne korzystają bezpośrednio z hardwarem

Obecnie często zastąpiony przez UEFI



# BIOS(Basic Input Output System)

BIOS który jest oprogramowaniem sprzętowym(firmware) odpowiedzialnym za start komputera do momentu załadowania programu z MBR w tym POST. Kiedyś pośredniczył między systemem operacyjnym a hardwarem ale dzisiaj systemy operacyjne korzystają bezpośrednio z hardwarem

Obecnie często zastąpiony przez UEFI



# UEFI(Unified Extensible Firmware Interface)

Specyfikacja definiuje interfejs między firmware a systemem operacyjnym  
Pozwala na obejście standardowych ograniczeń BIOSu:

- Użycie partycji ponad 2 tb z GPT
- Użycie 1mb ramu
- Niezależność od architektury i sterowników procesora

Jest mini systemem operacyjnym Obszerny temat:

[https://en.wikipedia.org/wiki/Unified\\_Extensible\\_Firmware\\_Interface](https://en.wikipedia.org/wiki/Unified_Extensible_Firmware_Interface)

# UEFI(Unified Extensible Firmware Interface)



<https://www.howtogeek.com/56958/htg-explains-how-uefi-will-replace-the-bios/>



## Pojęcia:

CMOS pamięć zasilania baterią na płycie głównej bateria odpowiedzialna kiedyś za zasilanie konfiguracji BIOSu

EEPROM pamięć flash przechowująca konfiguracją BIOSu nie wymagająca zasilania do utrzymania informacji



## Pojęcia:

MBR Pierwszy sektor dysku 512 bajtów zawierający program rozruchowy (bootloader stopnia pierwszego) i informacje o partycjach

POST Power-on self-test Test Przeprowadzany przy włączaniu komputera






# Przebieg BOOTowania PC

- 1.Power on lub reset
- 2.POST(Power on self test)
- 3.BIOS uruchamia program na MBR
- 4.bootloader ładowuje system operacyjny



# 1.Sygnał do komputera

Uruchamiany jest BIOS który kontynuuje włączanie komputera



## 2.BIOS uruchamia post POST(Power-on self-test)

POST jest wykonywany aby sprawdzić działanie:

- rejestrów CPU
- integralność BIOSu
- Podstawowe komponenty np timer
- Pamięć RAM

Błędy zgłaszane są poprzez komunikat dźwiękowy lub pisemny



### 3.BIOS uruchamia program z MBR

Wykonywane są instrukcje z MBR które zazwyczaj wczytują system lub bootloader



## 4.Bootloader wczytuje system operacyjny

Bootloader inicjalizuje siebie następnie pozwala na załadowanie systemu operacyjnego.

Daje nam również wybór który system wybrać



# MBR(Master Boot Record)

Pierwszy sektor dysku 512 bajtów zawierający program rozruchowy (bootloader stopnia pierwszego) i informacje o partycjach

# MBR(Master Boot Record) budowa

512 bajtów							
446 bajtów			64 bajty (4 x 16)			2 bajty	
program rozruchowy			partycja 1	partycja 2	partycja 3	partycja 4	0x55 0xAA
Offset (w bajtach)		Długość pola	Opis				
+0h		1 bajt	Status: 80h: partycja bootowalna, 00h: partycja niebootowalna				
+1h		3 bajty	adres CHS pierwszego sektora partycji. Format jest opisany w następnych wierszach.				
	+1h	1 bajt	numer głowicy				
	+2h	1 bajt	sektor w bitach 5–0; bity 7–6 są najbardziej znaczącymi bitami cylindra				
	+3h	1 bajt	bity 7–0 cylindra				
+4h		1 bajt	typ partycji				
+5h		3 bajty	adres CHS ostatniego sektora partycji. Format opisu identyczny jak pierwszego sektora partycji.				
+8h		4 bajty	adres LBA pierwszego sektora partycji				
+Ch		4 bajty	Liczba sektorów w partycji				
Wszystkie wielobajtowe pola w tabeli są typu <a href="#">little endian</a>							

[https://pl.wikipedia.org/wiki/Master\\_Boot\\_Record](https://pl.wikipedia.org/wiki/Master_Boot_Record)



# Boot ARM Cortex

- 1.Restart/power on
- 2.Czytanie pinu Boot0/Boot1 aby ustalić tryb bootowania
- 3.copy 0x00000000 to MSP Main stack pointer
- 4.copy 0x00000004 to PC Program counter(następna instrukcja)





# Boot ARM Cortex

Wywoływany jest Reset\_Handler który wykonuje inicjalizację hardware po czym wywołuje i przekazuje kontrolę funkcji main.

Wartości wskaźników na miejscu 0x00000000 i 0x00000004 możemy ustalić w pliku startup\_\*\*.s



# Boot Mode

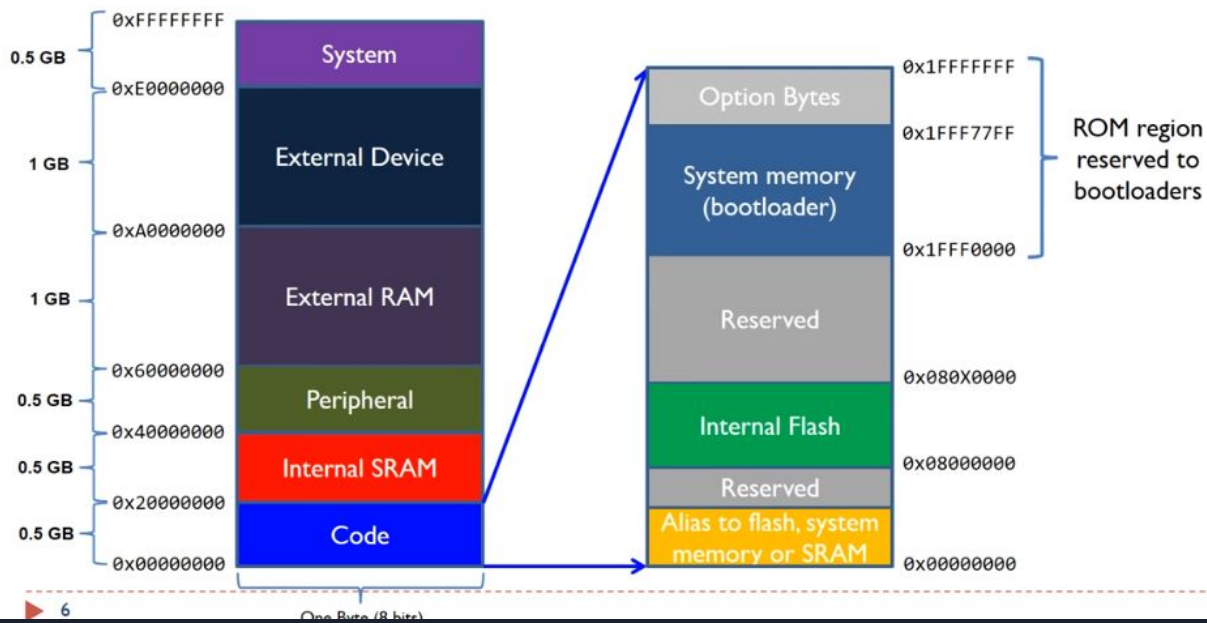
Boot0 = 0 boot odbywa się z głównej pamięci

Boot1 = 0 Boot0 = 1 pamięć systemowa (bootlader) pozwala na upgrade firmwaru potencjalnie custom bootloader jeśli chcemy zaszyfrować firmware

Boot1 = 1 Boot0 = 1 Boot from embedded SRAM

# Mapa pamięci

## Boot Mode





# Windows vs linux

Linuxowe bootloadery:

- GRUB

- LILO

- SYSLINUX

Zachowanie:

- Wykrywa zainstalowany windowsowy system

Windowsowy bootloader:

- NT OS Loader(NTLDR od New Technology Loader)

Zachowanie:

- Nie wykrywa zainstalowanych linuxów na systemie



# Konfiguracja Uruchomienia PC

Większość tradycyjnych komputerów i laptopów jest wyposażona w BIOS w którym można przełączać wiele funkcjonalności komputera jeszcze przed uruchomieniem systemu operacyjnego  
najpopularniejsze ustawienia które możemy zmienić to przeglądając menu:

- domyślny medium z którego ma bootować się komputer
- ustawienie quick boot(zazwyczaj przestaje pokazywać menu POST)



# Konfiguracja Uruchomienia Embedded:

Większość komputerów bez systemów operacyjnych pozwala na konfigurację działania komputera poprzez zmiany w rejestrach zgodnie z dokumentacją. Takiej konfiguracji uruchamiania dokonywaliśmy na laboratoriach w poprzednim semestrze. Jak wiemy jest to zazwyczaj proces trudny ale bardzo proste systemy embedded często nie mają żadnej innej możliwości.

Często stosowane funkcje to:

- wybór zastosowania pinów np: input lub output bądź obsługa interfejsu (np UART) lub układu specjalistycznego np DMA
- wybór sposobu pracy procesora, włączenie funkcji na przykład watchdog



# Konfiguracja Uruchomienia Raspberry Pi:

Raspberry Pi zamiast BIOSu posiada konfigurację przechowywaną w plikach config.txt oraz konfiguracje cmdline.txt która zawiera polecenia dla jądra systemu podczas startu błędy.

Ponieważ konfiguracja odbywa się poprzez pliki textowe a nie menu biosu zdecydowanie łatwiej jest o pomyłkę o ile pomyłka w pliku config nie będzie miała poważnych konsekwencji a jego konstrukcja to wyrażenia klucz=wartość to zmiany w pliku cmdline.txt mogą doprowadzić do nie bootowania się systemu. dodatkowo istnieje narzędzie raspi-config które pozwala na edycję ustawień z graficznym interfejsem

Często stosowane funkcje to:

- włączanie interfejsów na pinach np spi
- konfiguracja pracy podzespołów np overclocking

<https://www.raspberrypi.org/documentation/configuration/config-txt/README.md>

<https://www.raspberrypi.org/documentation/configuration/raspi-config.md>



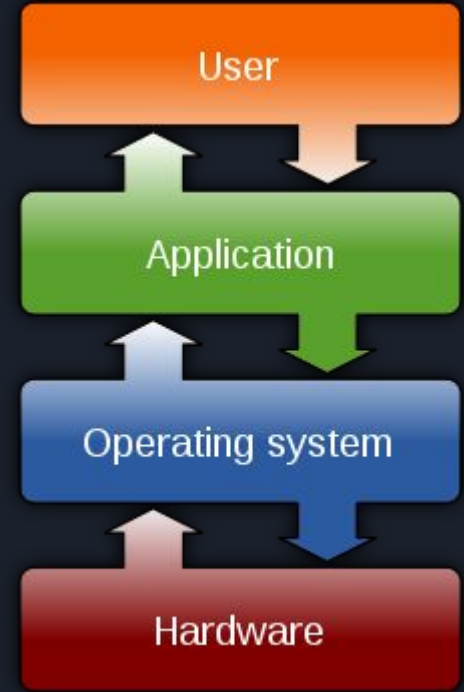
# Rodzaje systemów operacyjnych

- Jedno i Wielozadaniowe(single-tasking ,multi-tasking)
- z jednym lub wieloma użytkownikami(single multi user)
- rozproszone(distributed)
- Szablonowe(Templated)
- Osadzone(Embedded)
- czasu rzeczywistego(Real-time)
- Biblioteka(Library)



# Rola systemu operacyjnego

- Przydział zasobów dla poszczególnych zadań
- Mechanizmy synchronizacji zadań i komunikacji między nimi
- Pozwolenie na równoległe wykonywanie wielu zadań





# Typowe zadania systemu operacyjnego

- Stworzenie środowiska dla aplikacji



# Typowe zadania systemu operacyjnego

- Obsługiwanie przerw aby móc szybko reagować na środowisko,programowanie oparte o przerwania jest wspierana przez obecne procesory
- Obsługa trybów pracy tak aby móc ograniczyć zasoby przydzielane procesowi
- Zarządzanie pamięcią operacyjną przydzielanie zadaniom pamięci RAM
- Tworzenie wirtualnej pamięci
- Dostęp do dysku i system plików
- Sterowniki urządzeń
- Funkcje sieciowe
- Bezpieczeństwo



# Typowe zadania systemu operacyjnego

- Zarządzanie zasobami sprzętowymi między zadaniami
- Organizacja przestrzeni dyskowej dla zadań



# Zarządzanie zasobami

- Zarządzanie pamięcią RAM procesy dostają izolowaną przestrzeń adresową



# Dlaczego własny os?

- Kontrola nad sprzętem

- Ciekawe i trudne

**WAŻNE!:**

Robienie systemu operacyjnego jest wyjątkowo trudnym zadaniem i próby nie należy się go podejmować bez lat doświadczenia w programowaniu niskopoziomym.



## Problemy które spotkaliśmy związane z brakiem os

- Brak biblioteki standardowej
- Brak prostego pomiaru czasu
- Brak możliwości prostego wykonywania wielu zadań naraz
- Brak prostej obsługi peryferii
- Brak interfejsu komunikacji z komputerem



# Inne Problemy związane z brakiem os

- duża trudność używania aplikacji nie będących głównym programem w sposób bezpieczny
- trudność implementacji komunikacji sieciowej
- brak możliwości skorzystania z programów
- brak podziału zasobów między zadania





# Tworzenie systemu operacyjnego

Aby skompilować system operacyjny potrzebujemy cross kompilatora ponieważ nasz program ma działać bezpośrednio procesorze ani w typowym środowisku systemowym.

Dodatkowo potrzebować będziemy bootloadera który spowoduje że nasz program będzie bootowalny



# Tworzenie systemu operacyjnego

Na laboratoriach skorzystamy z kompilatora i686-elf dla asemblera i c

Oraz Gruba który pozwoli nam z naszego programu zrobić bootowalny obraz.

Aby sprawdzić działanie naszego systemu korzystać będziemy z qemu które pozwoli nam zasymulować komputer i móc przetestować działanie naszego programu