

Proces bootowania - wstęp

- bootowaniem (ang. *booting*) nazywamy proces *bootstrappingu* prowadzący do uruchomienia systemu operacyjnego po włączeniu komputera
- słowo *bootstrapping* jest nawiązaniem do niemieckich opowieści o baronie Münchhausenie, który po wpadnięciu do bagna był w stanie wyciągnąć z niego sam siebie ciągnąc się za włosy (w późniejszych wersjach tej historii hrabia używa rzemieni ze swoich butów - ang. *boot straps* - do wyciągnięcia się z morza. Z historii tych narodził się termin *bootstrapping*)
- we wszelkim kontekście informatycznym słowo *bootstrapping* odnosi się do sytuacji, w której jakiś mały system uruchamia pewien duży system

Proces bootowania - wstęp - c.d.

- sekwencja bootowania - to ciąg operacji, które muszą zostać wykonane przez komputer od momentu uruchomienia do momentu załadowania systemu operacyjnego
- wszystkie komputery (jako sprzęt) potrafią uruchamiać tylko te programy, które znajdują się w głównej pamięci - a większość programów (łącznie z systemami operacyjnymi) jest przechowywana na rozmaitych nośnikach
- zaraz po uruchomieniu komputer nie ma możliwości dostępnych w systemach operacyjnych - nie może ładować programów z dysku do pamięci
- prowadzi to do pozornie nierozwiązywalnego paradoksu

Proces bootowania - bootloadery

- rozwiązaniem są małe programy zwane *bootloaderami* (programami rozruchowymi), które idealnie realizują ideę bootstrappingu
- bootloader nie posiada pełnej funkcjonalności systemu operacyjnego, ale potrafi załadować taką jego część, która pozwoli na jego całkowite uruchomienie
- program rozruchowy oraz cały system operacyjny może być pobierany także z innych urządzeń takich jak stacja dyskiety, napęd CDROM, dyski USB a nawet spoza komputera - z serwera w sieci lokalnej
- bootloader (a w zasadzie jego pierwszy etap) jest ładowany przez BIOS

Co się dzieje po włączeniu przycisku POWER? - BIOS

- kontrolę nad sprzętem przejmuje BIOS (ang. *Basic Input/Output System*)
- BIOS jest programem zapisanym na stałe w pamięci ROM i jest wykonywany przy każdym włączeniu komputera
- po włączeniu komputera BIOS dekompresuje swój kod z pamięci flash i ładuje się do pamięci RAM i stamtąd rozpoczyna swoje działanie
- opcje użytkownika dla programu BIOSu zapisywane są w pamięci CMOS (ang. *complementary metal-oxide-semiconductor*), której trwałość jest podtrzymywana przez niezależne źródło prądu (baterię)
- kod źródłowy dla BIOSu pochodzący z 80x86 jest dostępny w *IBM Technical Reference Manual*)

Co się dzieje po włączeniu przycisku POWER? - BIOS c.d.

- BIOS jest czasami nazywany *firmware*'m ponieważ jest integralną częścią sprzętu i jego wersja jest charakterystyczna dla produktów konkretnych firm (największe z nich to: American Megatrends Inc. Phoenix Technologies czy Award Software International)
- BIOS jest przechowywany na nośnikach typu EEPROM czy pamięć flash, co umożliwia jego aktualizację
- ponieważ aktualizacja BIOSu jest czynnością niebezpieczną wprowadzono kilka zabezpieczeń:
 - *dual bios* - oryginalny program BIOSu jest nienaruszony w postaci backupu
 - „*boot block*” - nienadpisywalna część BIOSu, która jest uruchamiana w pierwszej kolejności i testuje dalszą część kodu - np. sprawdza sumy kontrolne

Co się dzieje po włączeniu przycisku POWER? - POST

- BIOS zapewnia dostęp do fundamentalnych komponentów komputera
- jego funkcją jest także testowanie sprawności tych komponentów - zajmuje się tym procedura POST (Power On Self Test)
- główne funkcje BIOSU podczas wywoływania procedur POST to:
 - sprawdzenie integralności kodu programu BIOS
 - ustalenie powodu, dla którego POST zostało uruchomione
 - znalezienie, ustalenie rozmiaru i sprawdzenie głównej pamięci
 - znalezienie, zainicjalizowanie i skatalogowanie wszystkich magistrali
 - dostarczenie interfejsu do konfiguracji systemu (BIOS Setup)
 - zidentyfikowanie urządzeń zdolnych do bootowania

BIOS - wzloty i upadki

- w prostych systemach operacyjnych takich jak DOS - BIOS pokrywał prawie wszystkie operacje wejścia/wyjścia oraz bezpośredni dostęp do sprzętu
- wraz z rozwojem bardziej skomplikowanych systemów (Windows, Linux) rola BIOSu została zredukowana tylko do początkowej inicjalizacji i testowania sprzętu
- jednak w ostatnich latach, wraz z pojawieniem się technologii ACPI (*Advanced Configuration and Power Interface*) BIOS przejął część zaawansowanych funkcji takich jak:
 - zarządzanie energią
 - *hotplug devices*
 - kontrola termiczna

Proces bootowania

- po pomyślnym wykonaniu procedur testujących BIOS próbuje załadować pliki bootujące systemu operacyjnego
- w tym celu identyfikuje wszelkie napędy, które są zdolne do bootowania i sprawdza je w kolejności zdefiniowanej przez użytkownika
- możliwości BIOSu dotyczące napędów kończą się na wiedzy jak załadować pierwszy 512 bajtowy sektor z napędu
- BIOS ładuje ze stosownego nośnika pierwsze 512 bajtów do pamięci głównej pod adres 0000:7C00
- 2 ostatnie bajty z 512 bajtowego bloku muszą w przypadku programów bootujących mieć wartość 0x55AA (kolejność bajtów jest na dysku odwrócona) - po napotkaniu tego znacznika, BIOS skacze do adresu 0000:7C00

Struktura sektora bootującego

- bajty: 0..445 (446 bajtów) kod wykonywalny programu ładującego
- bajty 446..509 (64 bajty) tablica partycji (4 wpisy po 16 bajtów każdy - patrz następny slajd nr 11)
- bajty 510 i 511 (2 bajty) - znacznik stale równy: 0xAA55

Struktura Master Boot Recordu



Struktura tablicy partycji

```
// plik: include/linux/genhd.h
struct partition {
    unsigned char boot_ind;           /* 0x80 - aktywna */
    unsigned char head;              /* ścieżka początkowa */
    unsigned char sector;            /* sektor początkowy */
    unsigned char cyl;               /* cylinder początkowy */
    unsigned char sys_ind;           /* Typ partycji */
    unsigned char end_head;          /* ścieżka końcowa */
    unsigned char end_sector;        /* sektor końcowy */
    unsigned char end_cyl;           /* cylinder końcowy */
    unsigned int start_sect;         /* sektor początkowy partycji */
    unsigned int nr_sects;           /* liczba sektorów wchodzących
                                     w skład partycji */
} //razem 16 bajtów
```

Kilka informacji o tablicy partycji

- dysk twardy może posiadać do 4 partycji podstawowych
- zamiast dowolnej z nich może wystąpić partycja rozszerzona zawierająca do 4 partycji zwanych logicznymi
- struktura dysku twardego jest jednoznacznie opisana za pomocą tablicy partycji
- tablica znajdująca się w MBR opisuje partycje podstawowe i rozszerzone znajdujące się bezpośrednio na dysku twardym
- w pierwszym sektorze dowolnej partycji rozszerzonej znajdzie się tablica partycji opisująca partycje logiczne umiejscowione na tej partycji rozszerzonej

Bootstrapping - ciąg dalszy

- w pierwszym sektorze każdej partycji jest zarezerwowane miejsce, w którym jest umieszczony kolejny program ładujący, wczytywany i uruchamiany przez ten z MBR
- zadania programu z MBR często ograniczają się do załadowania tego programu, który znajduje się na partycji oflagowanej znacznikiem: *active flag* (pierwsze pole struktury partition)
- zaawansowane bootloadery (np. LILO), które można umieścić w MBR nie korzystają z tej flagi

Jądro systemu Linux

- obraz jądra powstaje w wyniku kompilacji plików ze źródłami
- współczesne jądra są zwykle większe niż 512KB - dlatego są kompresowane (big kernel bzImage; zImage)
- podczas wczesnego bootowania, jądro ma do dyspozycji tylko niestawne 640KB pamięci - działa w trybie rzeczywistym procesora
- dlatego potrzebne są „tricki” z przerzucaniem części jądra do różnych obszarów pamięci przed włączeniem trybu chronionego
- na samym początku bardzo ważne są części jądra:
 - *bootsect.S* - mały program, który jest zapisany w bootsektorze; jego zadaniem jest wczytanie i wystartowanie głównej części jądra
 - *setup.S* - odpowiada za pobranie z BIOSu danych systemowych i umieszczenie ich w odpowiednich miejscach pamięci (dysk, pamięć i inne parametry)

Ładowanie jądra linuxa - scenariusz dla nieskompresowanych jąder (<512K)

- bootsector (*bootsector.S* uruchamiany w pamięci w miejscu 0x7c00 w trybie rzeczywistym) przesuwa swój kod w miejsce 0x90000 i wczytuje - korzystając z przerwań BIOSu - sektory znajdujące się tuż za nim
- reszta kernela jest ładowana pod adres 0x10000 (0x90000 - 0x10000 = 512KB) wypełniając maksymalnie do pół megabajta
- kod załadowany w miejscu 0x90200 (0x90000 + 512 bajtów na *bootsector.S*) - zdefiniowany w pliku *setup.S* zajmuje się inicjalizacją sprzętu i pozwala na zmianę trybu wyświetlania (*video.S*)
- później cały kernel jest przesuwany z adresu (0x10000 = 64K) pod adres (0x1000 = 4K) i w ten sposób nadpisuje dane BIOSu trzymane w pamięci RAM - wywołania BIOS-owe nie mogą już mieć miejsca
- w tym punkcie *setup.S* wchodzi w tryb chroniony procesora i skacze pod adres 0x1000, gdzie umieszczony jest kernel. W trybie chronionym cała pamięć jest już dostępna i system może zacząć się uruchamiać (wywołanie funkcji: *start_kernel()*)

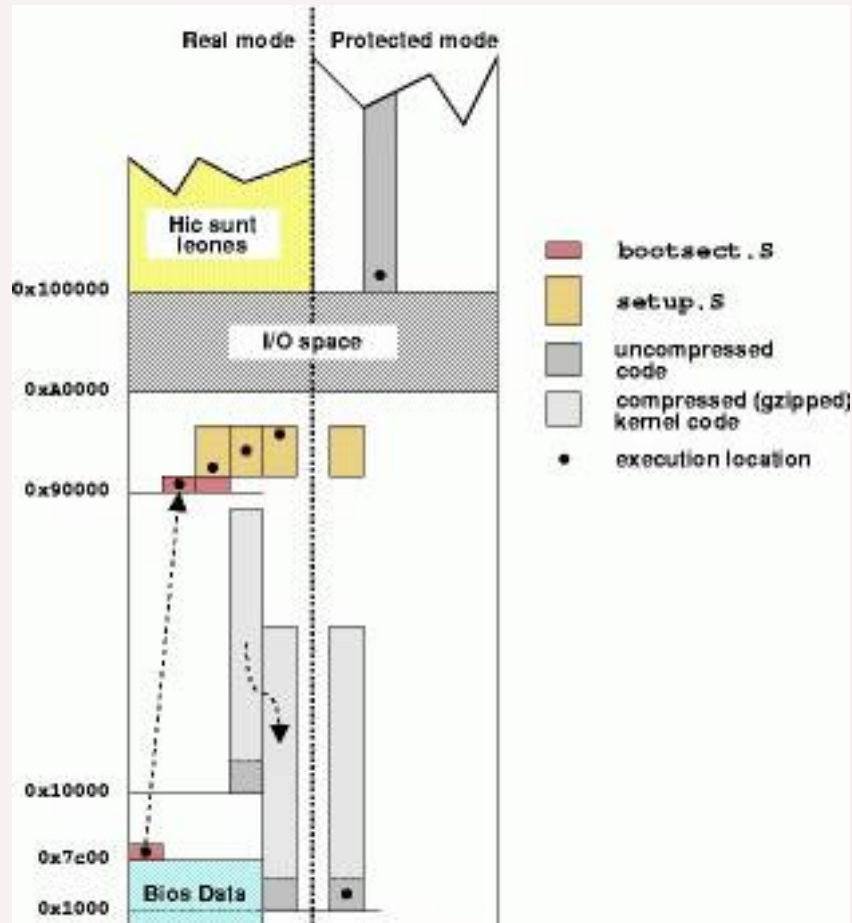
Kompresowane jądra (zImage)

- powyższy scenariusz miał miejsce w czasach, kiedy kernel był mały, na tyle mały, by zmieścić się w 512K
- z czasem do kernela włączano rozmaite ficzury i kernel rozrósł się do gigantycznych rozmiarów (dla wielu wyczynem jest zmieszczenie własnoręcznie skonfigurowanego jądra na dyskietce 1,44M)
- kod większy niż pół megabajta nie może być przesuwany pod adres 0x1000 - nie zmieści się w przydzielonym fragmencie pamięci
- dlatego - w tym przypadku - w miejscu 0x1000 nie leży jądro, tylko część „gunzip” programu **gzip**

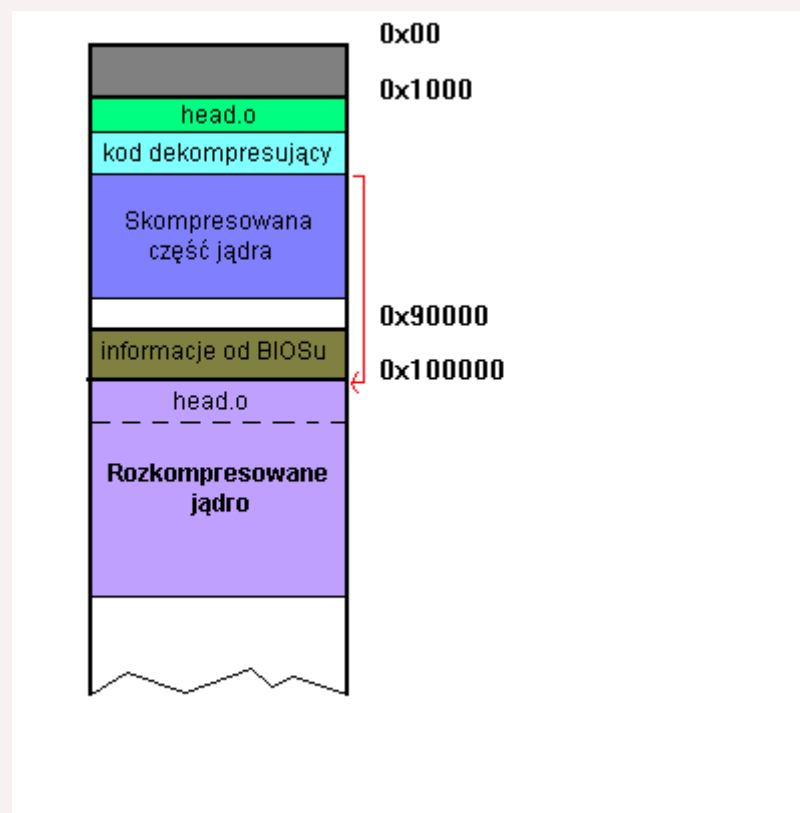
Kompresowane jądra - c.d.

- *head.S* jest umieszczany pod adresem 0x1000
- jego zadaniem jest rozzipowanie kernela - wywołuje on funkcję *decompress_kernel* (*compressed/misc.c*), która z kolei woła funkcję *inflate*
- output tej funkcji jest pisany do pamięci począwszy od adresu 0x100000 (1M)
- dostęp do pamięci powyżej 1M jest zapewniony, ponieważ pracujemy w trybie chronionym
- po dekompresji *head.S* skacze do aktualnego początku jądra
- proces bootowania jest teraz zakończony i kod, który znajduje się w 0x100000 (ten sam kod, który był w 0x1000 w nieskompresowanych jądrach) zajmuje się inicjalizacją i wywołaniem funkcji *start_kernel()*

Przemieszczanie jądra w pamięci podczas bootowania



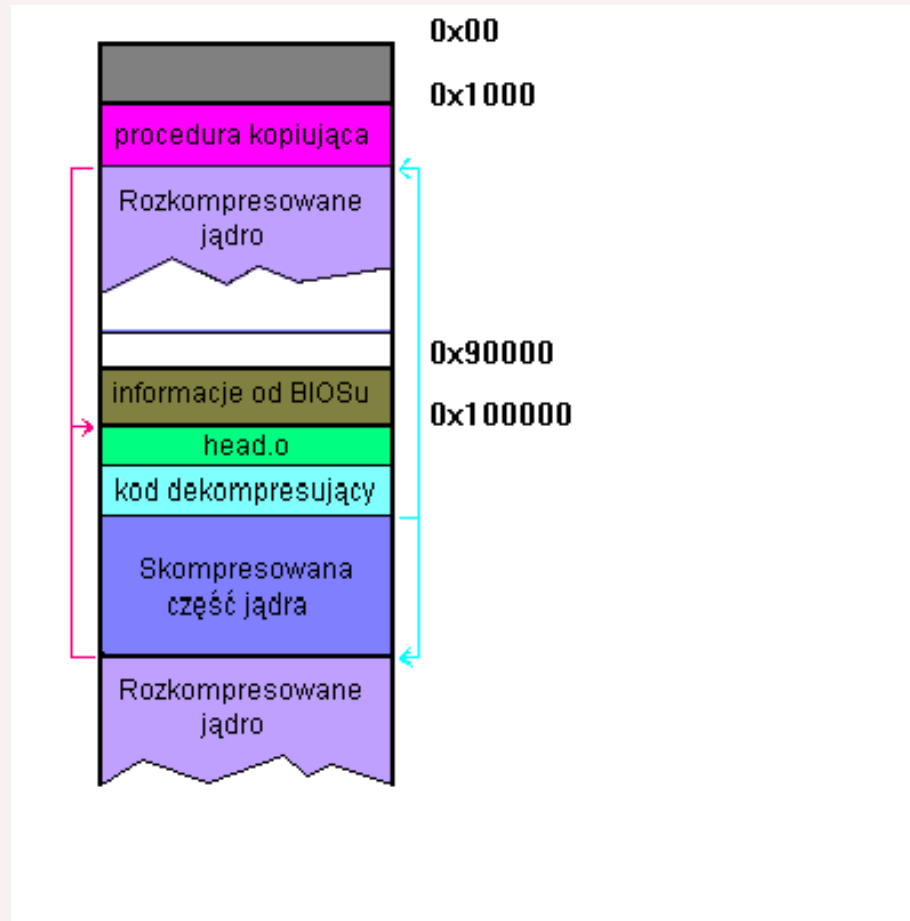
Dekompresja zwykłego jądra



Kompresowane jądra - metoda bzImage

- począwszy od jądra 1.3.73 dostarczono nowe narzędzie - *make bzImage*
- ten rodzaj jądra ładuje się w trochę inny sposób - zachodzą pewne zmiany:
 - kiedy system jest ładowany pod adres 0x10000, zostaje wywołany pewien pomocniczy kod po odczytaniu każdych 64K danych. Ta pomocnicza funkcja przenosi blok danych do pamięci wysokiej korzystając z funkcji biosowych
 - *setup.S* nie przesuwa systemu z powrotem do 0x1000 ale, po wkroczeniu w tryb chroniony skacze bezpośrednio do 1M, gdzie leżą dane przeniesione przez BIOS w poprzednim kroku
 - dekompresor, którego kod znajduje się pod adresem 0x100000 (1M) rozpakowuje kernel do niskiej pamięci, dopóki nie zostanie wyczerpana, a potem do pamięci wysokiej, zaraz za skompresowanym kodem
 - te dwa kawałki kodu są łączone i zapisywane pod adres (0x100000) - aby wykonać to zadanie prawidłowo potrzeba kilka przerzutów kodu w pamięci

Dekompresja dużego jądra



Informacje dodatkowe

- zadania *setup.S*:
 - Sprawdzenie wersji loadera, poprawności załadowania i rodzaju jądra
 - Rozmiar pamięci (BIOS)
 - Rozpoznanie i inicjalizacja karty graficznej
 - Ilość i parametry dysków podłączonych do pierwszego kontrolera
 - Obecność architektury MCA
 - Mysz PS/2
 - BIOS APM (Advanced Power Management) albo ACPI (Advanced Configuration and Power Interface)

Informacje dodatkowe - 2

- przełączenie w tryb chroniony:
 - Wyzerowanie lokalnej tablicy deskryptorów (LDT)
 - Ustawienie w globalnej tablicy deskryptorów (GDT) wskaźników na 4 GB segmenty kodu i danych jądra, zaczynające się od 0
 - Wyłączenie przerw
 - Włączenie linii adresowej A20
 - Reset koprocatora
 - Przesunięcie wektorów przerw programowych pod 0x20
 - Załadowanie słowa stanu procesora z ustawionym bitem trybu chronionego

Inne architektury - SPARC

- bogatszy w funkcje firmware (czyli „BIOS”) - uruchamia program o nazwie SILO (Simple LOader)
- SILO jest w stanie czytać pliki z partycji ext2 albo ufs
- firmware ładuje bootsector zaraz po sprawdzeniu sprzętu i jego inicjalizacji
- kod bootsectorowy znajduje się w pliku /boot/first.b (512B) i jest ładowany pod adres 0x4000
- dalej ładowany jest /boot/second.b i zapisywany pod adres (0x280000 = 2.5M) - adres został wybrany dlatego, że specyfikacja SPARCa zakłada, że przynajmniej 3M zostaną zamapowane podczas bootowania
- kod zapisany w second.b robi wszystkie pozostałe rzeczy (ma dostęp do partycji i może załadować (i zdekompresować) np. jądro systemu linux
- SILO odmawia współpracy z jądrami większymi niż 2.5M
- head.S jest sporo większe niż w przypadku PC