

# IdyllaOS

[www.idyllaos.org](http://www.idyllaos.org)

Prosty, alternatywny system operacyjny

*Autor: Grzegorz Gliński*

*Kontakt: [milyges@gmail.com](mailto:milyges@gmail.com)*

# Dlaczego powstał IdyllaOS?

- Główny cel: edukacja, czyli jak to działa
- Udowodnienie że napisanie OS-a nie musi być trudne
- Chęć stworzenia czegoś bardziej rozbudowanego niż edytor tekstu
- Chwała, sława i władza nad światem ;)

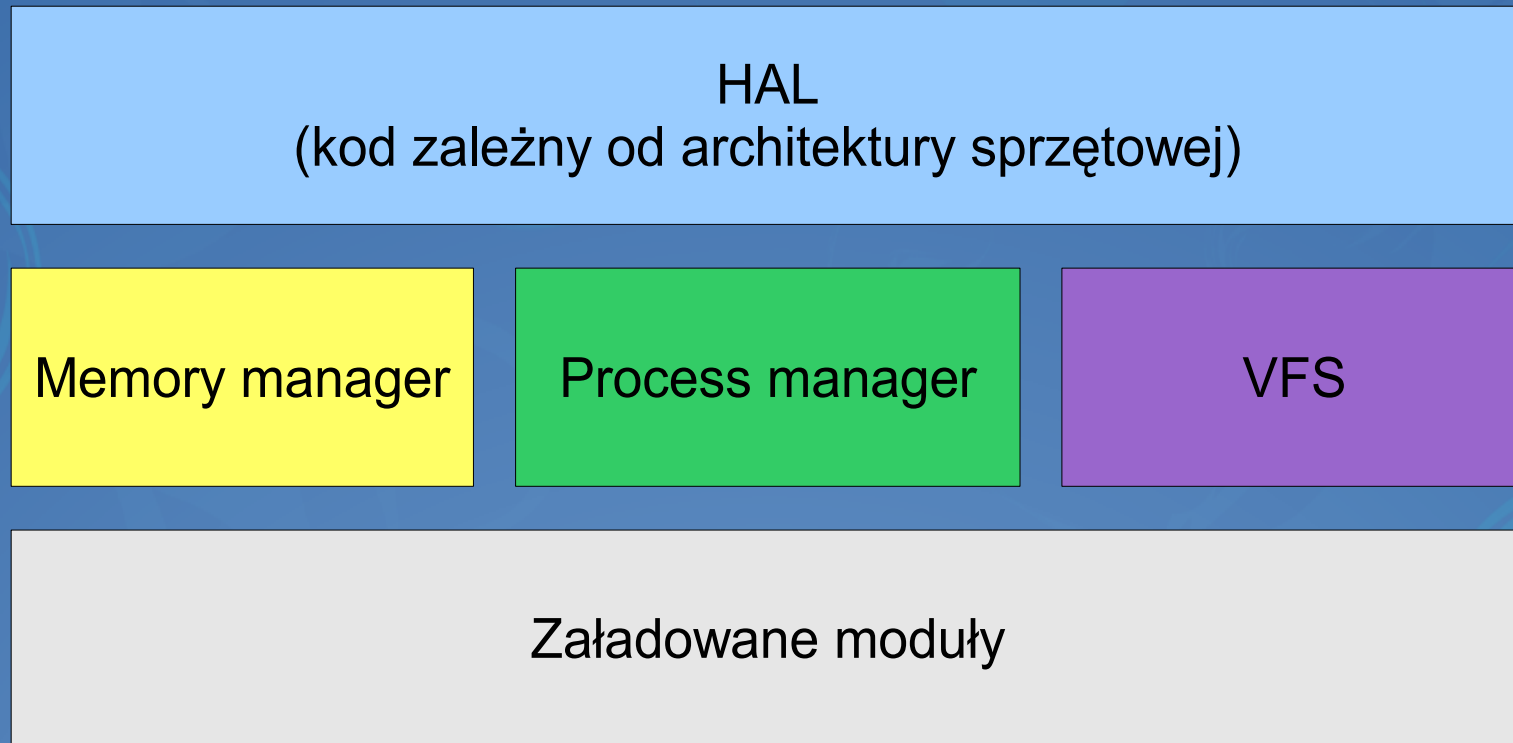
# Cechy systemu:

- Bootloader: Dowolny zgodny ze standardem multiiboot (np. GRUB)
- Kernel: Jądro monolityczne z obsługą dynamicznie dołączanych modułów
- Funkcje systemowe: w większości zgodne ze standardem POSIX
- Biblioteka standardowa: port biblioteki newlib
- Aplikacje: pliki wykonywalne w formacie ELF, planowana możliwość dołączania bibliotek dynamicznych.

# Jądro systemowe

- Jądro wielozadaniowe, wieloużytkownikowe
- Obsługa pamięci wirtualnej
- Przełączanie zadań przez użycie stosów
- Jądro jest wywłaszczalne
- Moduły w formacie plików ELF-Reloc
- Obsługa urządzeń znakowych i blokowych
- System plików oparty o architekturę i-węzłów
- Wbudowany monitor trybu v86

# Jądro systemowe: Architektura



# Jądro systemowe: HAL

- Odpowiada za komunikację z podstawowymi urządzeniami jak procesor czy pamięć
- Zawiera fragmenty kodu specyficzne dla danej architektury (np. włączenie stronicowania, obsługa przerwań, etc.)
- Kod inicjalizacji pisany w assemblerze
- Duża część kodu wywoływana tylko podczas inicjalizacji jądra

# Jądro systemowe: Manager pamięci

- Operuje na abstrakcyjnych strukturach jak przestrzeń adresowa czy region pamięci
- Zleca operacje niskopoziomowe modułowi HAL (jak np. zamapowanie pamięci, stworzenie przestrzeni adresowej)
- Umożliwia łatwą edycję przestrzeni adresowej dowolnego procesu
- W planach: umożliwienie tzw. swapa czyli trzymanie części nieużywanych danych na dysku.

# Jądro systemowe: Process manager

- Możliwość tworzenia zarówno procesów jądra jak i użytkownika
- Kolejowanie zadań: round-robin z możliwością ustawienia priorytetu zadania
- Obsługa stanów zadań
- Synchronizacja procesów (semafory, blokady typu rw oraz spinlocki)
- Obsługa sygnałów



# Jądro systemowe: VFS

- Architektura VFS oparta o interfejs i-węzłów
- Wsparcie dla wielu systemów plików
- Obsługa uniksowych praw dostępu
- Obsługa urządzeń blokowych i znakowych
- Obsługa dowiązań symbolicznych
- Obsługa cache i-węzłów dla danego punktu montowania
- Planowany cache dla bloków danych na poziomie sektorów dla urządzeń blokowych

# Jądro systemowe: Moduły

- Moduły w formacie relokowalnym ELF
- Ładowane bezpośrednio do przestrzeni jądra
- Moduły mogą importować i wywoływać niektóre funkcje jądra
- Sterowniki wszystkich do wszystkich urządzeń, systemów plików, etc. jako moduły, brak sterowników w jądrze
- Moduły pisane przeważnie w C

# Struktura katalogów

- / Główny katalog
- /boot Bootloader, kernel, initrd
- /system Aplikacje, pliki konfiguracyjne, etc.
- /dev Pliki urządzeń
- /users Katalogi domowe użytkowników

# Katalog /system

- bin pliki wykonywalne
- etc pliki konfiguracyjne
- include pliki nagłówkowe
- lib pliki bibliotek
- share dane dzielone aplikacji
- tmp dane tymczasowe

# Biblioteki

- Na chwilę obecną aplikacje linkowane statycznie z bibliotekami
- W planach: biblioteki dołączane dynamicznie w formacie ELF
- Implementacja biblioteki standardowej: newlib
- Wsparcie dla bibliotek takich jak: ncurses, zlib, libpng i inne

# Aplikacje

- Aplikacje w formacie ELF
- Każda aplikacja uruchamiana we własnej przestrzeni adresowej
- Synchronizacja aplikacji: spinlock'i, semafony, blokady typu czytelnicy-pisarze
- Komunikacja międzyprocesowa: sygnały, w planach pamięć dzielona oraz wiadomości
- Przeniesione popularne aplikacje z systemów UNIX: bash, less, nasm, binutils, nano i inne

# Proces bootowania systemu

- Bootloader ładuje do pamięci kernela (pod adres liniowy 0x00100000)
- Bootloader ładuje obraz initrd (initial ram-disk)
- Rozpoczyna się wykonywanie kernela (procedura `_start` w pliku `x86-loader/start.S`)
- Następuje inicjalizacja stronicowania i zamapowanie kernel'a pod adres 0xC0000000
- Dalej następuje inicjalizacja przerwań i innych modułów zarządzających sprzętem i pamięcią (jak np. DMA)

# Proces bootowania systemu cd.

- Dalej kontrola przekazywana jest procedurze kinit (plik kernel/init.c)
- Zainicjowane zostają pozostałe moduły kernela (jak np. manager procesów czy VFS)
- Kernel ładuje do pamięci program /bin/init z initrd
- Tworzony jest proces dla programu init, jest on dodawany do listy uruchomionych aplikacji.
- Kernel przełącza się w tzw. IDLE



# Proces bootowania systemu cd.

- Program `init` na początku parsuje parametry jądra, ładuje niezbędne moduły a następnie montuje główny system plików jako `/root`
- Następnie następuje wywołanie funkcji `chroot()` która ustawia jako katalog `/` dla `inita` katalog `/root`
- Następnie montowany jest system plików `devfs`
- Dalej ładowane są pozostałe moduły jądra, montowane pozostałe systemy plików
- Na końcu uruchamiany jest program `getty`.

# Pliki konfiguracyjne

- Pliki konfiguracyjne w /system/etc (odpowiednik /etc z systemu uniksowego)
- Pliki konfiguracyjne są to zwykłe pliki tekstowe
- Specyfikacja plików nie jest jeszcze gotowa, najważniejszy jest na razie rozwój jądra
- Formaty kilku plików (np. passwd) zapożyczone z systemu UNIX

# Praca w systemie

```
-----  
IdyllaOS                                     Welcome to IdyllaOS!  
-----  
This is a 0.1-alpha version, thank you for downloading and testing!  
  
IdyllaOS is still in the early stages of development, so please  
expect bugs, and lots of them.  
  
If you spot any bugs, please report it at http://bugs.idyllaos.org/  
Thanks again.  
  
If you have any questions, suggestions, etc. visit our  
IRC channel: #idyllaos on FreeNode network.  
  
Have fun!  
-----  
[root@localhost ~]#
```

*Po zalogowaniu do systemu*

# Praca w systemie cd.

```
[root@localhost ~]# echo Test > plik.txt
[root@localhost ~]# ls
plik.txt
[root@localhost ~]# cat plik.txt
Test
[root@localhost ~]# cd /
[root@localhost /]# ls
boot  dev  mnt  system
[root@localhost /]# cd dev/
[root@localhost /dev]# ls
sound  storage  terminal  video  klog  null  rtc  zero
[root@localhost /dev]# cd storage/
[root@localhost /dev/storage]# ls
cdrom0
[root@localhost /dev/storage]# lsmod
Name                Size      Address      Used by
null                1722      0xC04250A0
pci                  4335      0xC0647E58  ata_generic
tty                  8051      0xC0648F50
ata_generic         12016     0xC064FE58
iso9660fs            5003      0xC0653230
rtc                  2804      0xC042B108
tmpfs                5560      0xC06545C0
ext2fs               6471      0xC0655B80
[root@localhost /dev/storage]# _
```

*Polecenia systemowe, przeglądanie systemu plików*

# Praca w systemie cd.

```
GNU nano 2.0.7          File: syscall.h

#define SYS_SELECT      76
#define SYS_MAP_PMEM    77
#define SYS_STRACE      78
#define SYS_SETPGRP     79

#define SYS_SETEUID     80
#define SYS_SETEGID    81

#define SYSCALL0(num,res) \
    __asm__ __volatile__ ("int $0x80" \
        : "=a"(res) \
        : "a"(num) \
        )

#define SYSCALL1(num,res,p1) \
    __asm__ __volatile__ ("int $0x80" \
        : "=a"(res) \
        : "a"(num), "b"(p1) \
        )

[ Read 153 lines ]
^G Get Help      ^O WriteOut     ^R Read File    ^Y Prev Page   ^K Cut Text     ^C Cur Pos
^X Exit          ^J Justify      ^W Where Is    ^U Next Page   ^U UnCut Text  ^T To Spell
```

*Edycja tekstu edytorem nano*

# Praca w systemie cd.

```
[root@localhost /system/boot]# objdump -fph kernel.sys

kernel.sys:      file format elf32-i386
architecture: i386, flags 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
start address 0xc0105bd0

Program Header:
  LOAD off      0x00001000 vaddr 0xc0100000 paddr 0x00100000 align 2**12
             filesz 0x00016a1f memsz 0x00016a1f flags r-x
  LOAD off      0x00018000 vaddr 0xc0117000 paddr 0x00117000 align 2**12
             filesz 0x000004fc memsz 0x00007000 flags rw-

Sections:
Idx Name          Size      UMA      LMA      File off  Algn
  0 .text          00016a1f c0100000 00100000 00001000 2**5
             CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data          000004fc c0117000 00117000 00018000 2**5
             CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00006000 c0118000 00118000 000184fc 2**5
             ALLOC
  3 .comment       000004b9 00000000 00000000 000184fc 2**0
             CONTENTS, READONLY

[root@localhost /system/boot]# cd
[root@localhost ~]#
```

*Przeglądanie pliku kernela narzędziem objdump (pakiet GNU binutils)*

# Plany na przyszłość

- Poprawa wszelkich zauważonych błędów
- Optymalizacja całości kodu
- Implementacja nowych funkcji jądra
- Implementacja nowych sterowników i aplikacji
- Obsługa sieci
- Graficzny interfejs użytkownika
- W odległej przyszłości wydanie wersji 0.1