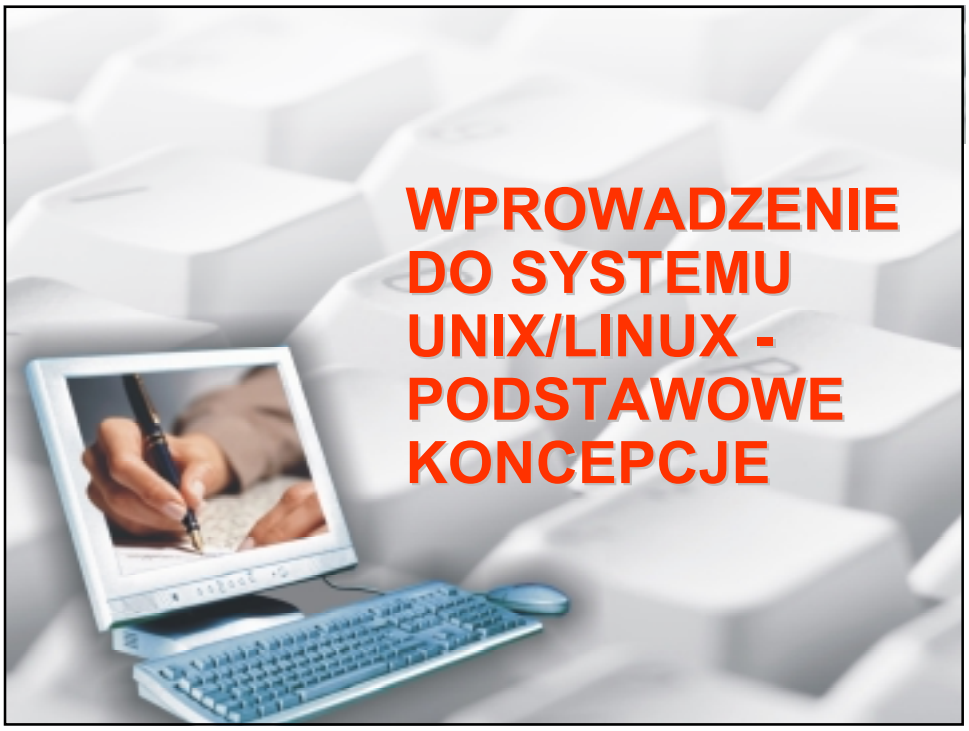
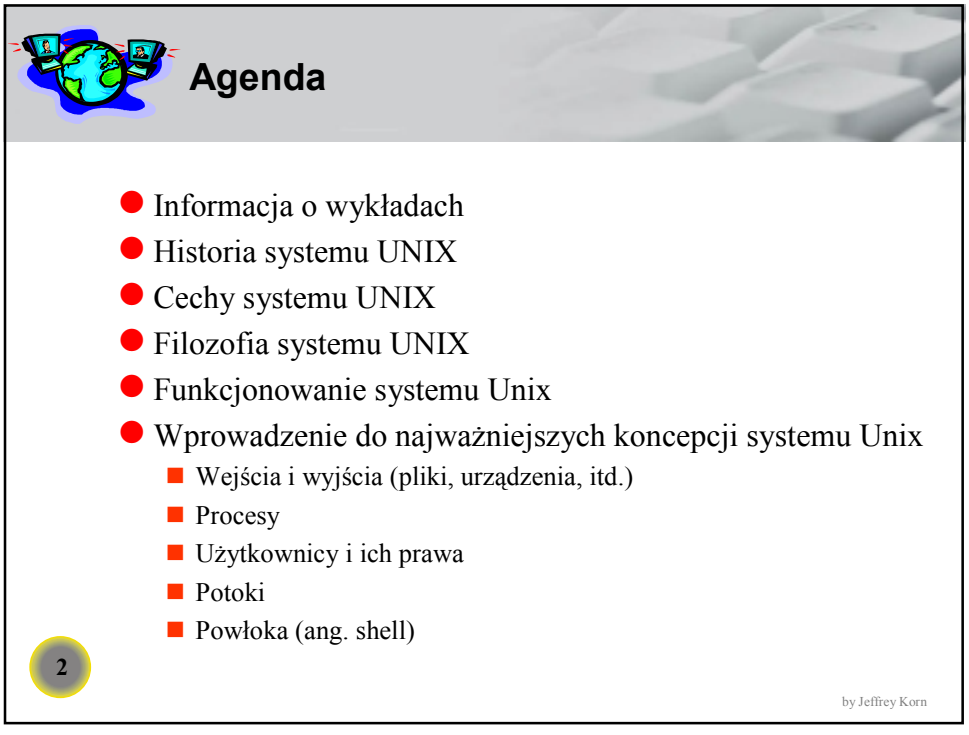

WPROWADZENIE
DO SYSTEMU
UNIX/LINUX -
PODSTAWOWE
KONCEPCJE


Agenda
Informacja o wykładach
Historia systemu UNIX
Cechy systemu UNIX
Filozofia systemu UNIX
Funkcjonowanie systemu Unix
Wprowadzenie do najważniejszych koncepcji systemu Unix
Wejścia i wyjścia (pliki, urządzenia, itd.)
Procesy
Użytkownicy i ich prawa
Potoki
Powłoka (ang. shell)
2
by Jeffrey Korn

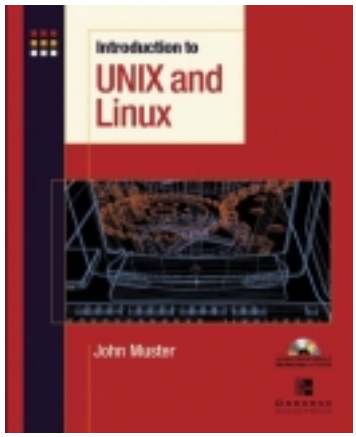
Czego będą dotyczyły wykłady

- Przegląd systemu operacyjnego
- Narzędzia systemu Unix
- Języki skryptowe
- Narzędzia programistyczne
- Administrowanie
- Bezpieczeństwo
- Sieci

3

by Jeffrey Korn

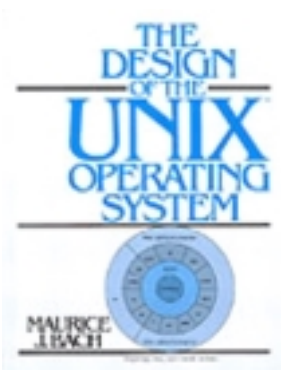
Literatura

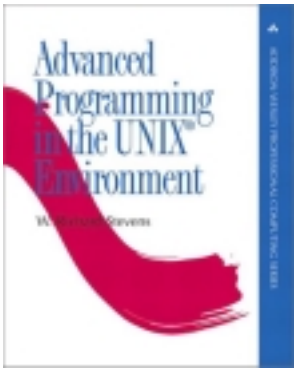


4

by Jeffrey Korn

Literatura

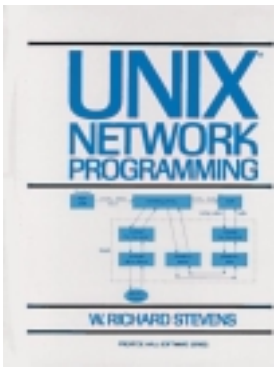


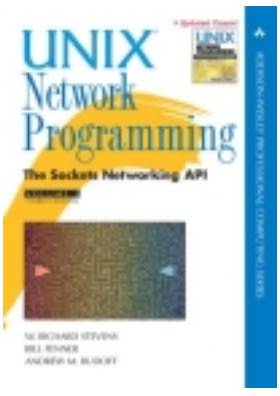


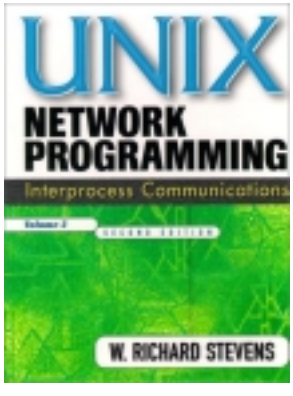
5

by Jeffrey Korn

Literatura



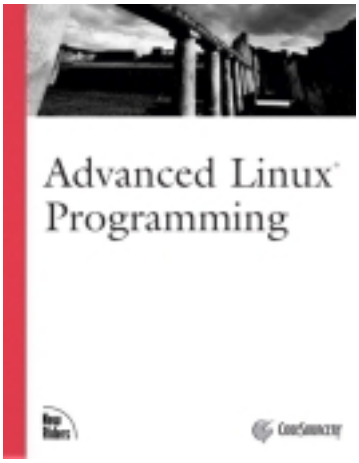


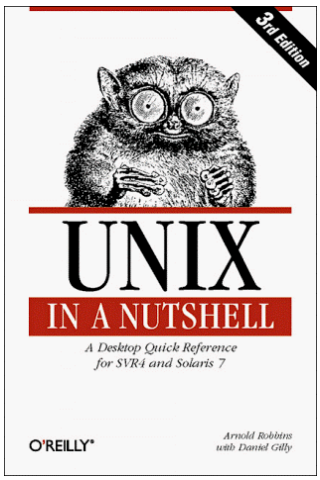


6

by Jeffrey Korn

Literatura





7

by Jeffrey Korn

Twórcy systemu Unix

Ken Thompson

Dennis Ritchie

8

by Jeffrey Korn

Początek systemu Unix

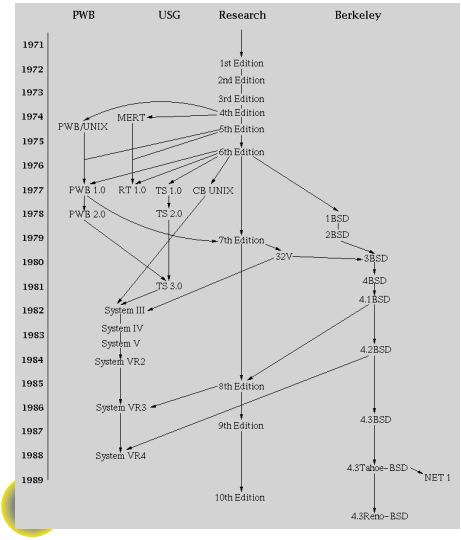
- UNICS: 1969 – minikomputer PDP-7
- PDP-7 został wycofany, system przepisany na PDP-11
- V1: 1971
- V3: 1973 (potoki, język C)
- V6: 1976 (przepisany w jezyky C, punkt wyjścia dla wersji BSD)
- V7: 1979 (licencjonowany, przenośny)

9

PDP-11

by Jeffrey Korn

Kolejne wydania systemu Unix



- BSD: dodano wiele istotnych cech (obsługe sieci, nadzorowanie zadań).
- PWB (Programmer's Workbench), USG (UNIX Support Group), MERT (Multi-Environment Real-Time)
- AT&T wprowadził komputery z System III, V do biznesu

by Jeffrey Korn

Wersje, które odniosły sukces komercyjny

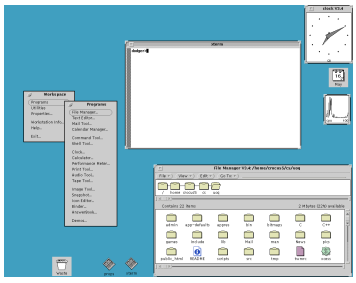
- AIX
- SunOS, Solaris
- Ultrix, Digital Unix
- HP-UX
- Irix
- UnixWare: Novell then SCO now Caldera
- Xenix: SCO,
- Standaryzacja (Posix, X/Open)

11

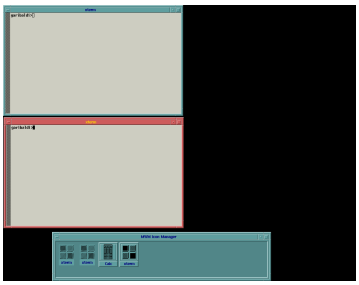
by Jeffrey Korn

…ale potem rozpoczęła się wojna

- Unix International z Open Software Foundation (konkurowanie o rynek komputerów klasy PC desktop)
- Wojna o menadżery okien (Window Managers)



Openlook



Motif

- Zagrożenie ze srronu Microsof Windows NT zakończyło spór o CDE (Common Desktop Environment)

12

by Jeffrey Korn

Powstały różne klony Unixa

- **Linux**
  - Napisany w 1991 r. przez Linusa Torvaldsa
  - Najbardziej popularna odmiana Unixa
  - Rozpowszechniany na zasadach licencji GNU

- **BSD Lite**
  - FreeBSD (1993, dedykowany na PC)
  - NetBSD (1993, zorientowany na sieci komputerowe)
  - OpenBSD (1996, zorientowany na bezpieczeństwo)
  - Wolny od licencyjnych
  - Rozwój mniej scentralizowany

13

by Jeffrey Korn

Źródła sukcesu UNIX

- Mocne podstawy techniczne!
- System projektowany na potrzeby badań, nie zaś na potrzeby rynku komercyjnego
- Komputery PDP-11 były popularne, ale nie posiadały wygodnego systemu operacyjnego
- Uwarunkowania prawne AT&T:
  - Nie pozwalały na sprzedaż komputerów z systemem Unix sferze komercyjnej, ale można było przyłączyć się do pisania oprogramowania, które pracowało pod kontrolą tego systemu
  - Tanie licencje

14

by Jeffrey Korn

Ruch Open Source

- Był i jest siła napędowa rozwoju systemu Unix
  - Nadążał za koniecznością zmian
  - Więcej użytkowników, projektantów
    - Więcej platform, lepsze własności, lepszy kod
- Wielu dostawców sprzętu i usług przerzuciło się na system Linux

15

by Jeffrey Korn

Filozofia systemu UNIX

- Małe jest piękne, bo
  - łatwe do zrozumienia,
  - łatwe w zarządzaniu,
  - bardziej wydajne
  - prostsze do ponownego zastosowania.
- Zrobić tak, aby każdy program robił jedną rzecz, ale za to dobrze
  - Bardziej wyszukane funkcje uzyskuje się jako złożenie programów bardziej prostych
  - Każdy program jest jednocześnie filtrem.

16

by Jeffrey Korn

Filozofia systemu UNIX

..continued

- Przenośność ponad efektywność
  - Najbardziej przenośne implementacje są rzadko przenośne
  - Przenośność jest bardziej pożądana ze względu na częste zmiany sprzętu
- Używanie przede wszystkim plików ASCII
  - Powszechny, prosty format plików (taki wczorajszy XML)
  - Przykład przenośności ponad efektywnością
- Kod wielokrotnego użycia
  - Dobrzy programiści piszą dobry kod; wielcy programiści pożyczają dobry kod

17

by Jeffrey Korn

Filozofia systemu UNIX

- Skrypty polepszyły działanie i przenośność

```
print $(who | awk '{print $1}' | sort | uniq)
                                    | sed 's/ /,/g'
```

Za pomocą polecenia zapisanego w jednej linii można wylistować wszystkich zalogowanych uzytkowników systemu.

who	755
awk	3412
sort	2614
uniq	302
sed	2093

- Szybkie budowanie prototypów (interpretowane języki wysokiego poziomu)

18

by Jeffrey Korn

Filozofia systemu UNIX

- Unikanie wyszukanych interfejsów
  - Użytkownikiem programu nie zawsze jest człowiek
  - Wygląda ładnie, ale kod jest obszerny i źle napisany
  - Problemy skalowania
- Milczenie jest złotem
  - Raportowanie tylko wtedy, gdy dzieje się coś złego
- Myślenie hierarchiczne

19

by Jeffrey Korn

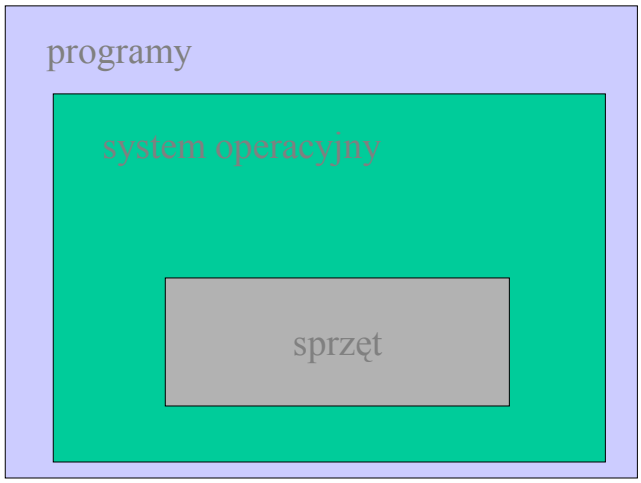
Istotne cechy systemu UNIX

- Przenośność (różne odmiany sprzętu, implementacja w języku C)
- Hierarchiczny system plików, abstrakcja pliku
- Wielozadaniowość i wielożytkownikowość w przypadku minikomputerów
- Komunikacja międzyprocesowa
  - Potoki: wyjście z jednego programu jest wejściem na inny
- Narzędzia programistyczne
- Narzędzia do projektowania
- Języki skryptowe
- Protokół TCP/IP

20

by Jeffrey Korn

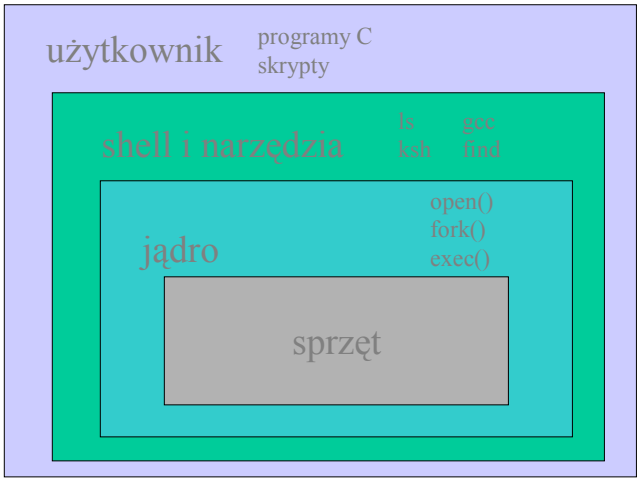
Struktura systemu operacyjnego



21

by Jeffrey Korn

Struktura systemu Unix



22

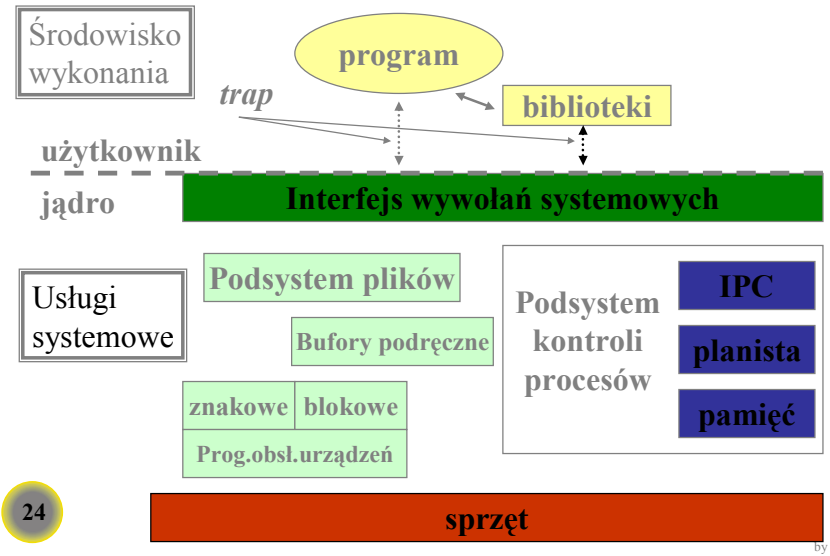
by Jeffrey Korn

Interakcje jądra

23

by Jeffrey Korn

Klasyczna architektura jądra

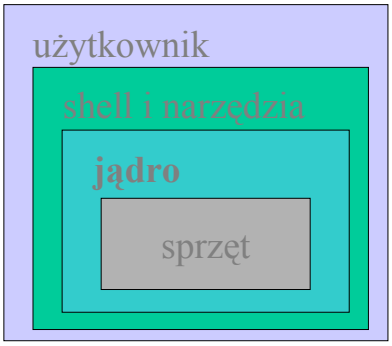


24

by Jeffrey Korn

Jądro

- Zarządza zasobami
  - Nośniki danych
  - Pamięć
  - CPU
  - Ekrany
  - Sieć
- Współdzielenie
  - Użytkowników
  - Zadań
- Komunikacja



25

by Jeffrey Korn

Jądro

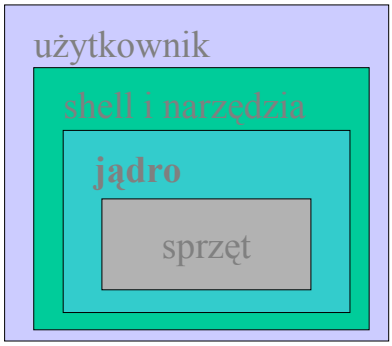
- Odgrywa role warstwy ochronne pomiędzy programami i sprzętem
- Zapobiega wzajemnemu oddziaływaniu programów na siebie.
  - Kontroluje wykonywanie procesów (tworzenie, kończenie, wstrzymywanie, komunikacja)
  - Planuje przydział czasu CPU poszczególnym procesom.
  - Przydziela pamięć procesom.
- Obsługuje urządzenia peryferyjne takie jak terminale, taśmy, urządzenia dyskowe i sieciowe.

26

by Jeffrey Korn

Shell i narzędzia

- Pozostała część systemu operacyjnego
- Będą także przedmiotem tego i dalszych wykładów
- Są przyczyną dyskusji w środowisku użytkowników systemu Linux



27

by Jeffrey Korn

Podsystemy jądra

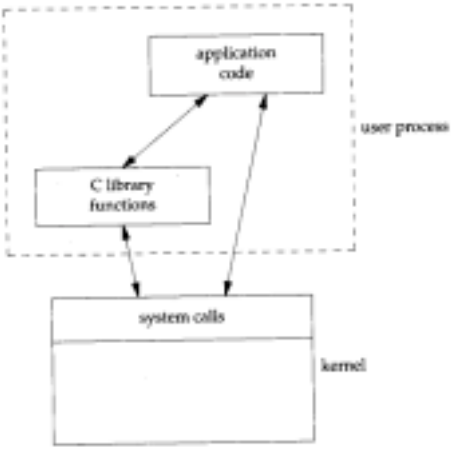
- System plików
  - Hierarchia katalogów, pliki regularne, peryferiały
  - Mnogość systemów plikowych
  - Wejścia/wyjścia (I/O)
    - Sposób dostępu procesów do plików, terminale I/O
- Zarządzanie procesami
  - Sposób współdzielenia przez procesy CPU, pamięci i sygnałów
  - Planowanie
  - Komunikacja międzyprocesowa
  - Zarządzanie pamięcią
- Różne odmiany systemu UNIX mają odmienne implementacje różnych podsystemów.

28

by Jeffrey Korn

Wywołania systemowe

- Na poziomie jadra zaimplementowano szereg specialnych procedur
- Program użytkownika wywołuje procedurę w trybie jądra, wykorzystując do tego celu pułapkę sprzętową.
- Procedura obsługi pułapki przełącza CPU w tryb uprzywilejowany i jądro wykonuje wywołanie systemowe
- CPU przechodzi z powrotem do trybu użytkownika
- Istnieje API języka C do wszystkich wywołań systemowych



29

by Jeffrey Korn

Logowanie (po polsku: rejestrowanie) się

- Należy najpierw posiadać konto i hasło
  - W odpowiedzi na zapytanie naley podać swój identyfikator konta
  - Hasło wprowadzane jest bez echa na ekranie
  - Po pomyślnym zalogowaniu się, użytkownik widzi znak zachęty powłoki (shell'a)
- Wprowadzanie poleceń
  - Po znaku zachęty powłoki wpisuje się polecenie
    - Typowy format: **polecenie** opcje arumenty
    - Przykłady: who, date, ls, cat myfile, ls –l
  - System rozróżnia duże i małe litery
- Polecenie exit oznacza wylogowanie (wyrejestrowanie) się z systemu

30

by Jeffrey Korn

UNIX w systemie Windows

Bardzo popularne są dwa środowiska emulujace system Unix:

- **UWIN (AT&T)**
  - `http://www.research.att.com/sw/tools/uwin`
- **Cygwin (GPL)**
  - `http://sources.redhat.com/cygwin`

31

by Jeffrey Korn

Interfejs a implementacja

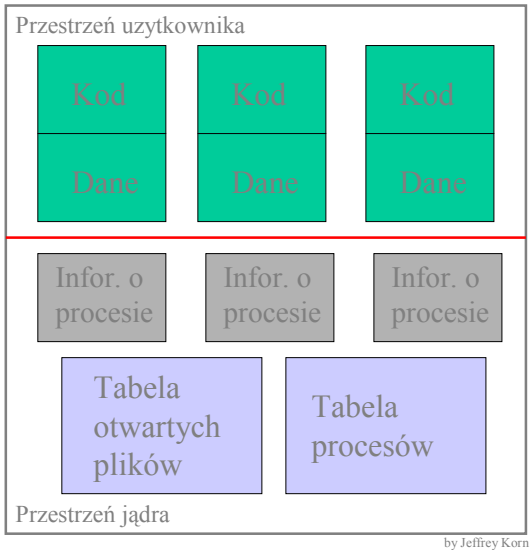
- **Interfejs**: które żądania mogą być zgłaszane i jakiego typu odpowiedzi należy się podziewać?
- **Implementacja**: w jaki sposób udzielane są odpowiedzi na zgłaszane żądania

32

by Jeffrey Korn

Struktury danych jądra

- Informacja o każdym procesie.
- **Tabela procesów**: zawiera wpis dla każdego procesu w systemie.
- **Tabela otwartych plików**: zawiera przynajmniej jeden wpis dla każdego otwartego pliku w systemie.

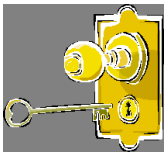


33

by Jeffrey Korn

Podstawy bezpieczeństwa systemu Unix

- Z systemów UNIX może korzystać jeden lub więcej użytkowników, posiadających identyfikator i nazwę.
- Zbiór użytkowników może tworzyc grupę. Użytkownicy mogą być członkami wielu grup.



- Specjalny użytkownik (id: 0, nazwa: root) ma pełn
systemem.
- Każdy użytkownik należy do pierwotnej (domyślnej) gr

34

by Jeffrey Korn

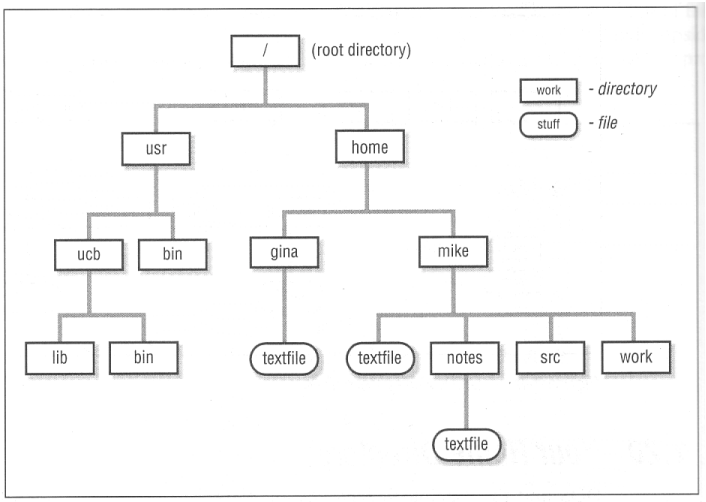
Kontrolowanie praw użytkowników i grup

- Podczas kontroli weryfikowane jest czy:
  - może być wykonana określona operacja dotycząca pliku lub procesu?
  - można czytać lub pisać z/do pliku?
  - można uruchomić program?
  - można użyć określonego sprzętu?
  - można zatrzymać określony proces?

35

by Jeffrey Korn

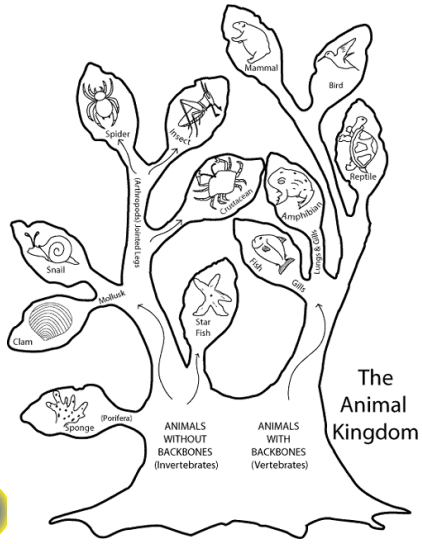
Hierarchia plików systemu UNIX

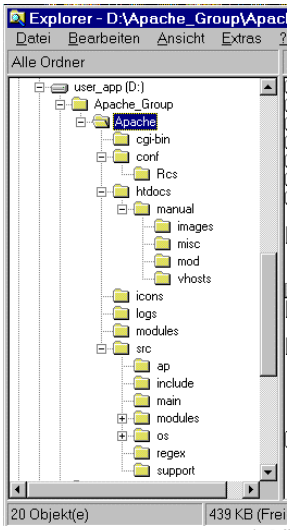


36

by Jeffrey Korn

Hierarchia jest wszechobecna!



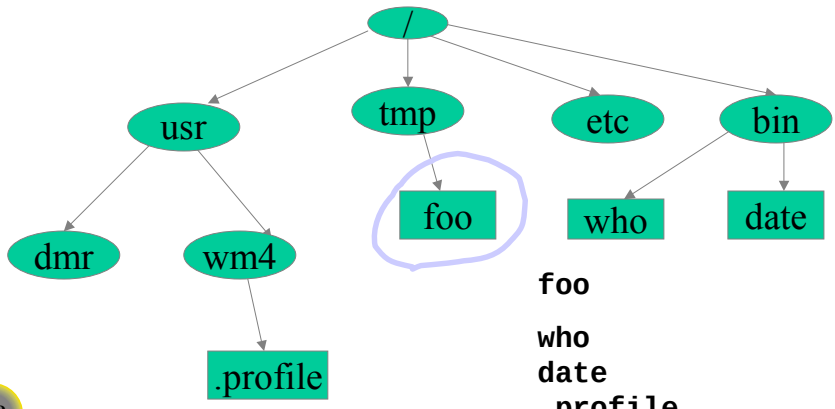


37

by Jeffrey Korn

Definicja: Nazwa pliku

Ciąg znaków różnych niż ukośnik (ang. slash). Nazwa jest wrażliwa na duże i małe litery.



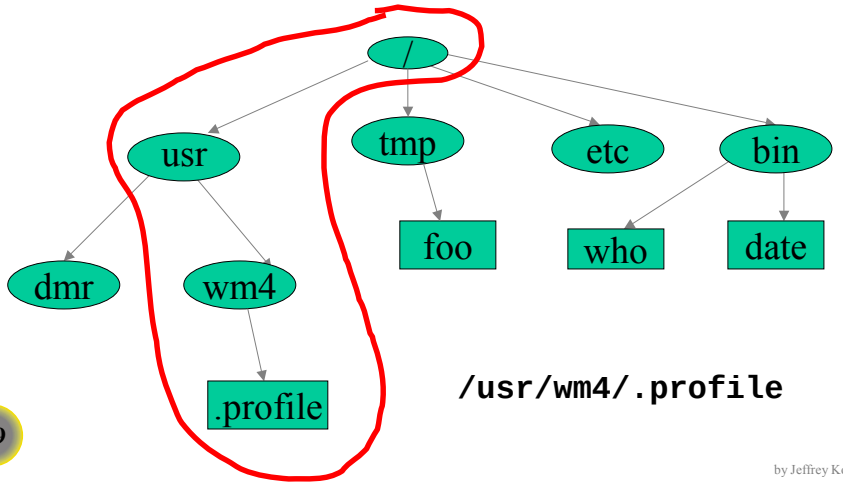

```
foo

who
date
.profile
```

38

by Jeffrey Korn

Definicja: Nazwa ścieżki

Ciąg zera lub więcej nazw plików, oddzielonych od siebie znakiem ukośnika / i opcjonalnie rozpoczynająca się znakiem /.



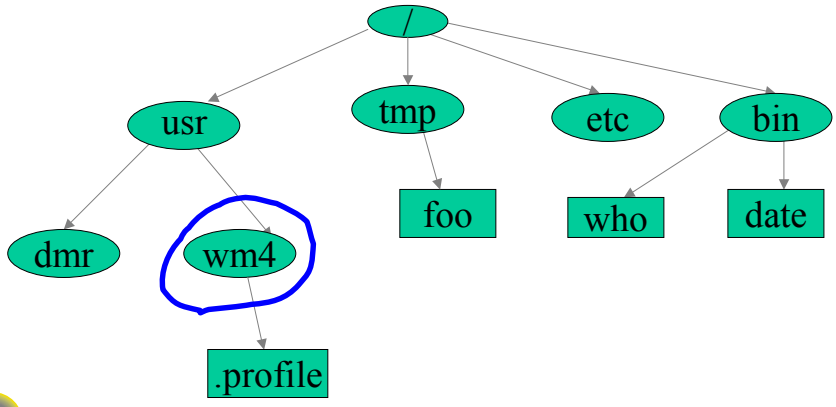
`/usr/wm4/.profile`

39

by Jeffrey Korn

Definicja: Katalog roboczy

Katalog, na który domyślnie wskazuje nazwa pliku.

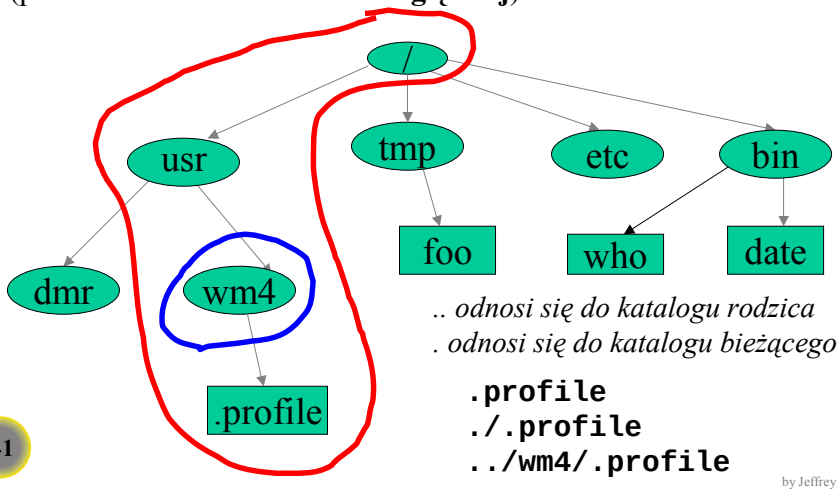


40

by Jeffrey Korn

Definicja: Względna nazwa ścieżki

Nazwa ścieżki zbudowana względem katalogu roboczego (przeciwieństwo **ścieżki bezwzględnej**)



.. odnosi się do katalogu rodzica
. odnosi się do katalogu bieżącego

```
.profile
./.profile
../wm4/.profile
```

41

by Jeffrey Korn

Pliki i katalogi

- Plik jest po prostu ciagiem bajtów
  - Nie występuje typ pliku (dane, czy też plik wykonywalny)
  - Brak sekcji
  - Przykład filozofii systemu Unix
- Katalogi są listami plików i ich statusu:
  - daty utworzenia,
  - itp.

42

by Jeffrey Korn

Prawa związane z plikami

- W systemie UNIX dostępny jest sposób ochrony plików oparty na pojęciu użytkownika (właściciela) i grupy użytkowników.
- Występują trzy typy praw:
  - **read** - proces może czytać zawartość pliku
  - **write** - proces może zapisywać dane do pliku
  - **execute** - proces może wykonać zawartość pliku
- Trzy zbiory uprawnień:
  - uprawnienia dla właściciela
  - uprawnienia dla grupy
  - uprawnienia dla pozostałych

43

by Jeffrey Korn

Prawa związane z katalogiem

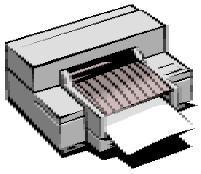
- Dostępne są te same typy i zbiory praw jak w przypadku plików
  - **read**: proces może czytać zawartość katalogu (tj. listować jej zawartość)
  - **write**: proces może dodawać/usuwać pliki do/z katalogu
  - **execute**: proces może otwierać pliki w katalogu lub podkatalogu

44

by Jeffrey Korn

Urządzenia

- Oprócz plików, wejścia/wyjścia mogą być związane także z różnymi urzadzeniami


- Innowacja systemu UNIX: traktuj te urządzenia tak samo jak pliki.
  - `/dev/tty, /dev/lpr, /dev/modem`

45

by Jeffrey Korn

Tablica otwartych plików

- Wykonywanie operacji I/O na pliku wymaga najpierw jego otwarcia, potem czytania/zapisywania, i wreszcie zamknięcia.
- Jądro zarządza globalna tablicą, zawierająca informacje o każdym otwartym pliku.

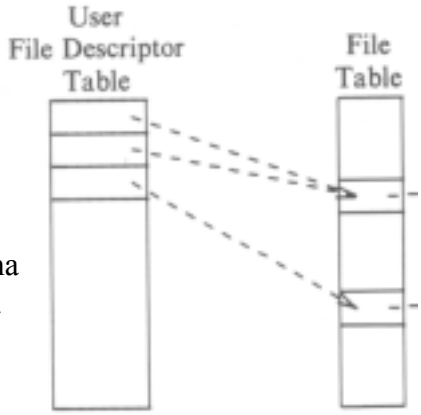
Inode	Tryb	Licznik	Pozycja
1023	read	1	0
1331	read/write	2	50
		…	

46

by Jeffrey Korn

Tablica deskryptorów pliku

- Każdy proces zawiera własną tabelę plików, które otworzył.
- Każdy otwarty plik jest skojarzony z **numerem** lub „**uchwytem**”, nazywanym **deskryptorem pliku** (fd).
- Każdy wpis w tabeli wskazuje na wpis w tabeli otwartych plików.
- Tablica indeksowana jest od numeru 0.



47

by Jeffrey Korn

Dlaczego nie stosuje się bezpośrednich odwołań do tablicy otwartych plików?

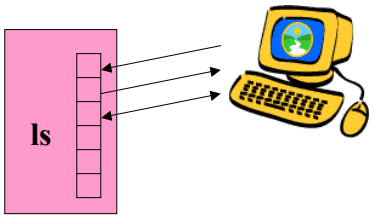
- Przechowywana jest dodatkowa informacja:
  - Na przykład, czy potomek powinien dziedziczyć otwarte pliki (tzw. close-on-exec flag)
- Wygoda z punktu widzenia jądra
  - Przekierowanie (pośredni szczebel w dostępie do otwartych plików) ułatwia zarządzanie bezpieczeństwem
- Schemat numerowania otwartych plików może być lokalny dla danego procesu (0 .. 128).

48

by Jeffrey Korn

Standardowe wejście/wyjście/diagnostyka

- Trzy pierwsze wpisy w tabeli deskryptorów są zarezerwowane i oznaczają:



- Wpis 0 oznacza standardowe wejście
- Wpis 1 oznacza standardowe wyjście
- Wpis 2 standardowe wyjście komunikatów o błędach

- Domyślnie standardowe wejścia/wyjścia dotyczą terminala (/dev/tty)

49

by Jeffrey Korn

Montowanie systemu plików

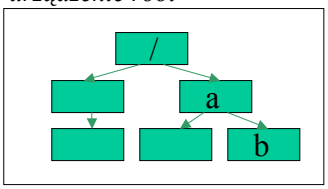
- Gdy startuje UNIX, hierarchiczny katalog odpowiada systemowi plików ulokowanemu na pojedynczym dysku nazywanym *urządzeniem root*.
- *Montowanie* pozwala na połączenie katalogu **root** z istniejącą hierarchia katalogów.
- Za pomocą mechanizmu montowania systemy plikowe utworzone na innych urządzeniach mogą być dołaczone do pierwotnej hierarchi katalogów.

50

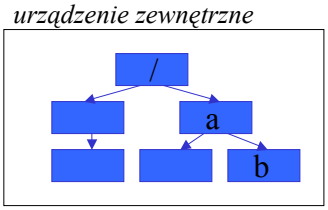
by Jeffrey Korn

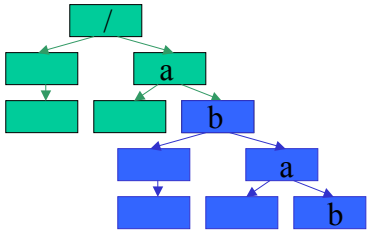
Montowanie systemów plików

urządzenie root



urządzenie zewnętrzne





Urządzenie	Punkt montowania
	/
	/a/b

Tablica montowania

51

by Jeffrey Korn

Dowiązania (ang.links)

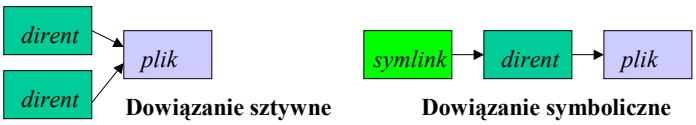
- Katalogi zawierają listy plików i katalogów (dokładnie: podkatalogów).
  - Każdy wpis w katalogu jest dowiązaniem do pliku na dysku
  - Dwa różne wpisy w katalogu mogą być dowiązaniami do tego samego pliku
    - W tym samym katalogu lub pomiędzy różnymi katalogami
  - Przemieszczenie pliku z katalogu do katalogu nie oznacza przemieszczenia danych w inne miejsce.
    - Tworzone jest nowe dowiązanie w nowym miejscu
    - Usuwane jest to w starym miejscu

52

by Jeffrey Korn

Dowiązania symboliczne

- **Dowiązania symboliczne** są inne niż dowiązania regularne (nazywane często **dowiązaniami sztywnymi**).
- Można je interpretować jako pliki, które zawierają nazwę innego pliku
- Nie zmieniają liczby dowiązań do pliku
  - usunięcie oryginalnego pliku pozostawia bez zmian dowiązanie symboliczne
- Dowiązania sztywne istnieją ponieważ:
  - nie można ich używać pomiędzy systemami róznymi plikowymi
  - stosuje się je do plików regularnych, nie zaś do katalogów



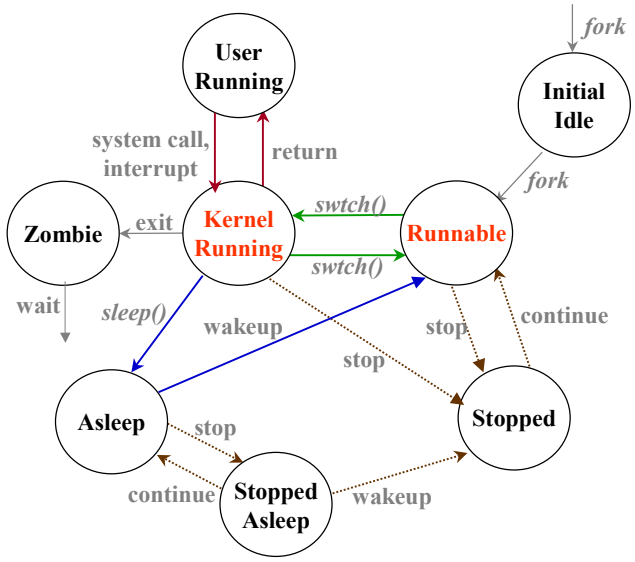
53

Procesy

- Zwykle tworzone za pomocą *fork* lub *vfork*
- Dobrze zdefiniowana hierarchia procesów: jeden rodzic i zero lub więcej procesów potomnych. Proces **init** jest korzeniem tego drzewa.
- Podczas życia procesu za pomocą wywołania funkcji systemowej **exec** mogą być tworzone inne procesy inne programy
- Procesy kończą się zwykle po wywołaniu **exit**

54

Stanu procesów w systemie Unix



55

by Jeffrey Korn

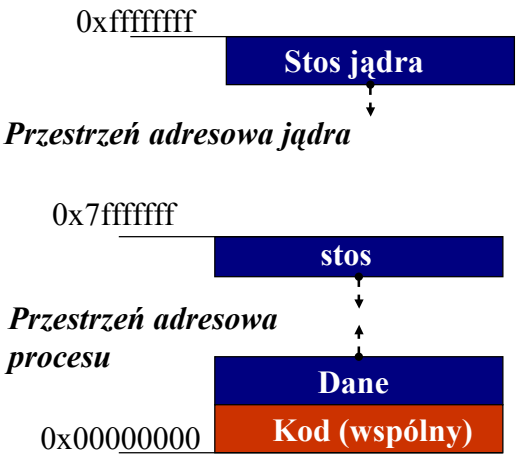
Kontekst procesu

- Przestrzeń adresowa
  - kod, dane, stos, pamięć współdzielona ...
- Informacje kontrolne (u-obszar, tablica procesów **proc**)
  - u-obszar, struktura procesów, odwzorowania
  - stos jądra
  - Odwzorowania translacji adresów
- Dane uwierzytelniające (ang. credentials)
  - ID użytkownika i grupy
- Zmienne środowiskowe
  - Zmienna=wartość
  - Zwykle przechowywane na spodzie stosu

56

by Jeffrey Korn

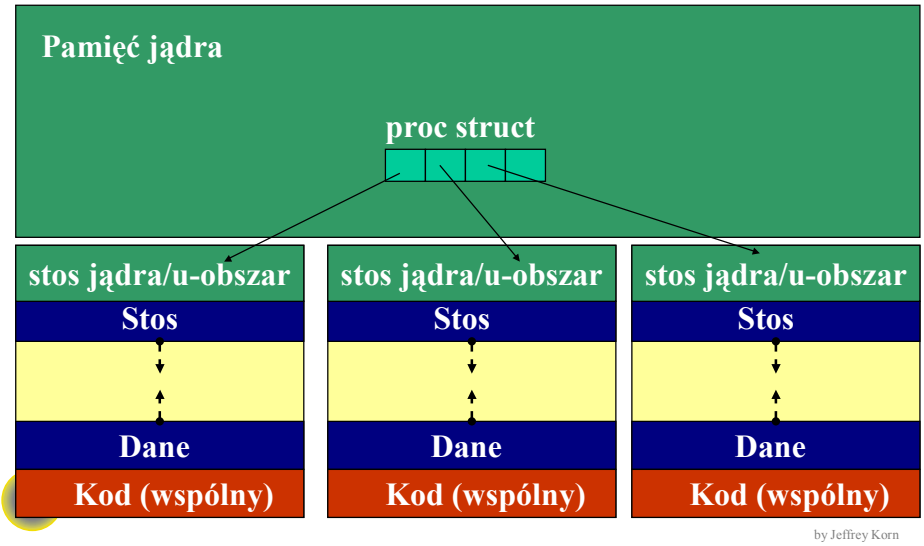
Przestrzeń adresowa procesu (jedno z podejść)



57

by Jeffrey Korn

Inne spojrzenie



by Jeffrey Korn

Informacje kontrolne procesu

- U-obszar
  - Część przestrzeni użytkownika (powyżej stosu).
  - Zwykle odwzorowywany w ustalony adres.
  - Zawiera informacje niezbędne podczas wykonywania procesu.
  - Może być wymieniany (ang. Swapped)
- Tablica procesów **Proc**
  - Zawiera informacje konieczne wtedy, gdy proces nie wykonuje się.
  - Nie może być wymieniany (ang. swapped).
  - Tradycyjna tabela o ustalonym rozmiarze.

59

by Jeffrey Korn

U-obszar i tablica Proc

- U-obszar
  - Kontekst procesu
  - Wskaźnik do pozycji w tablicy Proc
  - Rzeczywisty/efektywny ID
  - argumenty, zwracane wartości lub błędy bieżącego wywołania funkcji systemowej.
  - Info o sygnałach
  - Tabela deskryptorów plików
  - Bieżący katalog i bieżący korzeń.
- Tablica Proc
  - ID procesu i grupy
  - Wskaźnik na U-obszar
  - Stan procesu
  - Wskaźniki na kolejki – planowania, uśpione, etc.
  - Priorytet
  - Informacja zarządzania pamięcią
  - Flagi (znaczniki).

60

by Jeffrey Korn

Dane uwierzytelniające użytkownika

- Każdy użytkownik posiada przypisany unikalny ID użytkownika (uid) oraz ID grupy (gid).
- Superużytkownik (root) ma uid == 0 i gid == 0
- Każdy proces posiada zarówno rzeczywisty jak i efektywny ID.
  - Efektywny ID => tworzenie plików i dostęp,
  - Rzeczywisty ID => rzeczywisty właściciel procesu. Stosowany podczas wysyłania sygnałów.
    - Efektywny lub rzeczywisty ID nadawcy musi być równy rzeczywistemu ID odbiorcy.

61

by Jeffrey Korn

Synchronizacja

- Jądro jest reentrantne.
- W danym czasie może być wykonywany tylko jeden aktywny proces (pozostałe sa blokowane).
- Niewywłaszczeniowa.
- Blokowanie operacji
- Maskowanie przerwań

62

by Jeffrey Korn

Operacje blokowania

- W przypadku, gdy zasoby są niedostępne (być może zablokowane), proces ustawia flagę i wywołuje *sleep()*
- *sleep* umieszcza proces w kolejce zablokowanych procesów, ustawia stan *asleep* i wywołuje *swtch()*
- Gdy zasób jest zwalniany, wywoływana jest funkcja *wakeup()*
- Wszystkie uśpione procesy są budzone, a ich stan ustawiany na **gotowy** (procesy umieszczane sa w kolejce procesów gotowych).
- W stanie wykonywania proces musi sprawdzić, czy zasób jest dostępny.

63

by Jeffrey Korn

Szeregowanie procesów

- Algorytm planowania round-robin z wywłaszczeniami
- Każdy proces posiada przydzielony, ustalony kwant czasu
- Priorytet jest dostrajany wartość wygody i współczynnik stosowania.
- Procesy w trybie jądra mają przypisane priorytet jądra (priorytet uśpiony), który jest wyższy niż priorytety użytkownika.
- Priorytety jadra 0-49, uzytkownika 50-127.

64

by Jeffrey Korn

Sygnały

- Zdarzenia asynchroniczne i wyjątki
- Sygnały generowane są za pomocą funkcji systemowej kill()
- Operacje domyślne lub specyficzne programy obsługi napisane przez użytkownika
- Ustawiają bit w masce sygnału w tablicy procesów
- Wszystkie sygnały są obsługiwane przed procesu do normalnego przetwarzania
- Wywołania systemowe mogą być restartowane.

65

by Jeffrey Korn

Procesy w systemie Unix

Proces: wykonywana jednostka

- *Definicje*
  - **program**: zbiór bajtów przechowywanych w pliku, który może być wykonany
  - **obraz**: komputerowe środowisko wykonywania programu
  - **proces**: wykonanie obrazu
- Unix może jednocześnie (współbieżnie) wykonywać wiele procesów.

66

by Jeffrey Korn

Tworzenie procesu

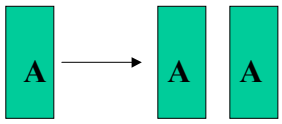
- **fork**
  - Tworzy nowy proces
  - Kopiuje pamięć wirtualna rodzica
  - Kopiuje katalog roboczy i deskryptory otwartych plików
  - Zwraca procesowi rodzicielskiemu wartość PID potomka
  - Zwraca procesowi potomnemu wartość 0
- **exec**
  - Kopiowane parametry mogą być zminione przed wywołaniem exec
  - Nadpisuje proces wywołujący ta funkcje nowym procesem

67

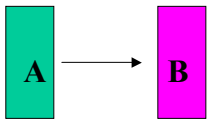
by Jeffrey Korn

Tworzenie procesu

- Interesująca cecha systemu UNIX
- Wywołanie systemowe fork klonuje aktualny proces



- Wywołanie systemowe exec zastępuje bieżący proces



- Zwykle fork jest wywoływany przed exec

68

by Jeffrey Korn

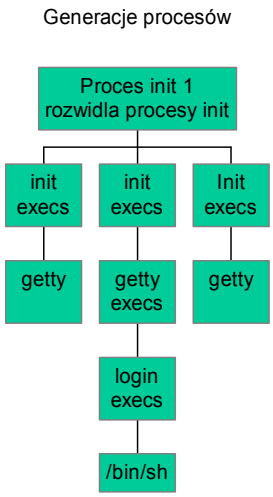
Kończenie procesu

- Wywoływana jest funkcja *exit()*
  - Zamyka otwarte pliki
  - Zwalnia inne zasoby
  - Zapisuje statystyki użytkowania zasobów i status powrotu w tablicy procesów
  - Budzi rodzica (jeśli czeka)
  - Wywołuje **swtch**
- Proces jest w stanie **zombie**
- Rodzic gromadzi, via funkcje *wait()*, status zakończenia procesu i statystyki użytkowania zasobów.

69

by Jeffrey Korn

Drzewo genealogiczne procesów Unix



70

by Jeffrey Korn

Argumenty uruchamianego programu

- Gdy uruchamiany jest proces, to przesyłana jest do niego lista napisów
  - `argv`, `argc`
- Proces może wykorzystywać je zgodnie z ich zaprojektowanym przeznaczeniem.

71

by Jeffrey Korn

Utrzymywana informacja o procesie

- Katalog roboczy
- Tablica deskryptorów plików
- ID procesu
  - Liczba używana do identyfikacji procesu
- ID grupy procesów
  - Liczba używana do identyfikacji zbioru procesów
- ID procesu rodzica
  - ID procesu, który utworzył proces

72

by Jeffrey Korn

Utrzymywana informacja o procesie

- Efektywny ID użytkownika i grupy
  - Użytkownik i grupa, którzy mają prawo uruchomienia procesu
- Rzeczywisty ID użytkownika i grupy
  - Użytkownik i grupa, którzy wywołali proces
- **umask**
  - Domyślne prawa dostępu do nowo tworzonego pliku
- Zmienne środowiskowe

73

by Jeffrey Korn

Środowisko procesu

- Zbiór par nazwa-wartość związanych z procesem
- Klucze i wartości są napisami
- Przekazywane procesom potomnym
- Nie mogą być zwracane z powrotem
- Typowe przykłady:
  - **PATH**: gdzie szukać programów
  - **TERM**: typ terminala

74

by Jeffrey Korn

Komunikacja międzyprocesowa

Sposób w jaki komunikują się procesy:

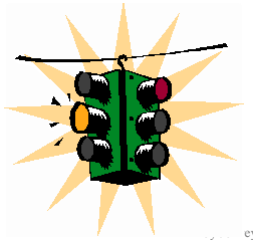
- Przekazywanie argumentów i środowiska
- Zapis/odczyt regularnych plików
- Status wyjścia
- Sygnały
- Łącza

75

by Jeffrey Korn

Sygnały

- **Sygnał**: wiadomość, którą proces może przesłać do procesu lub grupy procesów, jeśli tylko ma odpowiednie uprawnienia.
- Typ wiadomości reprezentowany jest przez nazwę symboliczną
- Dla każdego sygnału otrzymujący go proces może:
  - Jawnie zignorować sygnał
  - Realizować specjalne działania za pomocą tzw. signal handler
  - W przeciwnym przypadku realizowane jest działanie domyślne (zwykle proces jest usuwany)
- Typowe sygnały:
  - SIGKILL, SIGTERM, SIGINT
  - SIGSTOP, SIGCONT
  - SIGSEGV, SIGBUS

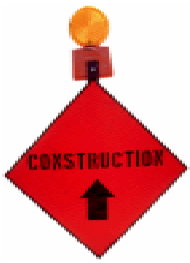


76

ey Korn

Przykład sygnałów

- Jeśli istnieje potomek, to wysyła sygnał SIGCHLD do rodzica.
- Jeśli rodzic chce czekać na zakończenie potomka, to powiadamia system, że chce przechwycić sygnał SIGCHLD
- Jeśli nie zasygnalizuje oczekiwania, to sygnał SIGCHLD jest przez niego ignorowany

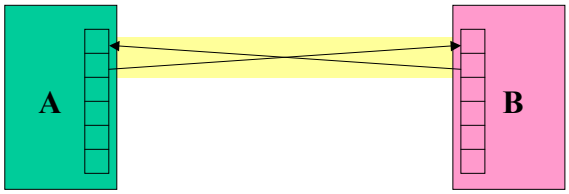


77

by Jeffrey Korn

Łącza

- Ogólna idea: wyjście z jednego programu jest wejściem na następny i vice versa



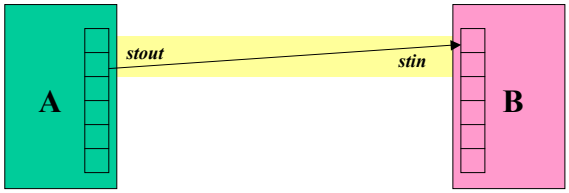
- Oba programu uruchamiane są jednocześnie

78

by Jeffrey Korn

Łącza

- Często używana jest tylko jedna ze stron łącza

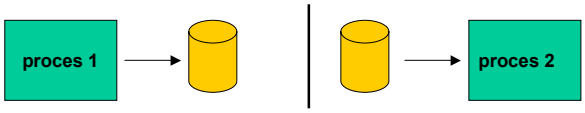


- Czy można tak postępować także z plikami?

79

Podejście z wykorzystaniem plików

- Uruchom pierwszy program, jego wyjście zapisz do pliku
- Uruchom drugi program, używając poprzedniego pliku jako jako wejścia



- Zbędne używanie dysku
  - Wolne operacje
  - Może zająć dużo przestrzeni dyskowej
- Niemożliwe zastosowanie wielozdaniowości

80

Jeszcze o łączach

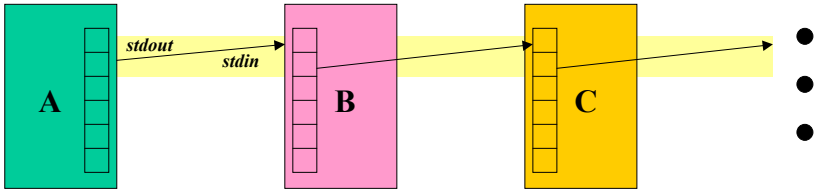
- Co się dzieje jeśli proces próbuje czytać dane, ale żadne nie sa dostępne?
  - UNIX przeprowadza proces w stan uśpienia do czasu nadejścia danych
- Co się dzieje jeśli proces nie może przeczytać wszystkich danych zapisywanych przez proces piszący?
  - UNIX przechowuje bufor nieprzeczytanych danych
    - Zależy to od rozmiaru łącza.
  - Jeśli łącze jest pełne, to UNIX usypia proces piszący do momentu zwolnienia miejsca w łączu przez proces czytający
- Możliwa jest jednoczesna współpraca wielu czytelników i pisarzy z łączem.

81

by Jeffrey Korn

Jeszcze o łączach

- Łącza są często tworzą łańcuch
  - Wywołanie filtrów

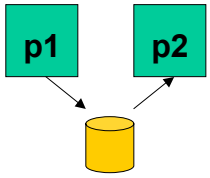


82

by Jeffrey Korn

Komunikacja miedzyprocesowa w przypadków procesów niepowiązanych

- FIFO (łącza nazwane)
  - Specjalny plik, który p otwarciu reprezentuje łącze
- System V IPC
  - Kolejki komunikatów
  - Semafory
  - Pamięć współdzielona
- Gniazda (ang. sockets) – model klient/serwer



by Jeffrey Korn

Mechanizmy setuid i setgid

- Jądro może może ustawić efektywny ID użytkownika i grupy przypisany procesowi na inny niż rzeczywisty ID użytkownika i grupy
  - Pliki wykonywane z ustawioną flagą **setuid** lub **setgid** umożliwają zmianę wartości tych ID
- Umożliwia wykonaie zadań uprzywilejowanych:
  - Np., zmiana hasła użytkownika
- Są źródłem zagrożeń w przypadku wykrycia błędów w systemie

by Jeffrey Korn

Trochę o powłoce (ang. shell)

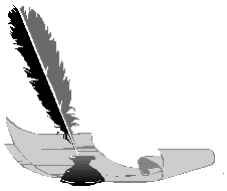
- Interfejs użytkownika, pozwalający na kontakt z systemem operacyjnym
- Funkcjonalność:
  - Wykonywanie innych programów
  - Zarządzanie plikami
  - Zarzdzqnie procesami
- Program jak każdy inny
- Uruchamiany automatycznie po zalogow użytkownika

85

by Jeffrey Korn

Sposoby korzystania z powłoki

- **Interakcyjny**
  - Po zalogowaniu się do systemu, użytkowanik interakcyjnie prowadzi dialog z powłoką
- **Skryptowy**
  - Zbiór poleceń powłoki, które tworza *program* wykonywany



86

by Jeffrey Korn

Praca z powłoka

- Powłoka uruchamiana jest automatycznie podczas sesji logowania się uzytkownika lub ręcznie przez uzytkownika z linii poleceń
  - 1. Czyta specjlany inicjalizujący plik startowy
  - 2. Wyświetla znak zachety i czeka na poleenie uzytkownika
  - 3. Wykonuje polecenia użytkownika i wraca do kroku 2, chyba, że użytkownik wciśnie kombinacje klawisz Control-D lub wpisze polecenie **exit**; wtedy wykonywanie powłoki kończy się

87

by Jeffrey Korn

Najczęściej używane powłoki

- **/bin/csh** C shell
- **/bin/tcsh** rozbudowany (ang. enhanced) C shell
- **/bin/sh** Bourne Shell / POSIX shell
- **/bin/ksh** Korn shell
- **/bin/bash** darmowy klon ksh

Podstawowa postać powłoki:

```
while (read command) {
  parse command
  execute command
}
```

88

by Jeffrey Korn

Proste polecenia

- ***Proste polecenie***: ciąg niepustych argumnetów oddzielonych spacjami lub znakiem tabulacji.
- Pierwszy argument (numerowany jako zero) określa zwykle nazwę wykonywanego polecenia.
- Pozostałe argumenty:
  - Przekazywane są jako argumenty do polecenia.
  - Argumenty mogą być nazwami plików, katalogów lub specjalnymi opcjami
  - Znaki specjalne interpretowane sa przez powłokę

89

by Jeffrey Korn

Prosty przykład

```
$ ls -l /bin
-rwxr-xr-x  1 root   sys    43234 Sep 26  2001 date
$
```

znak zachęty *polecenie* *argumenty*

- Wykonuje podstawowe polecenie
- Parsowanie argumentów na potrzeby polecenia nazywane jest podziałem argumentów (ang. *splitting*)

90

by Jeffrey Korn

Typy argumentów

```
$ tar –c –v –f archive.tar main.c main.h
```

- Flagi
  - Konwencja: *-X* or *--długanazwa*
- Napisy opcji
- Parametry
  - Mogą być plikami, napisami, itp..May be files, may be strings
  - Zależą od polecenia

91

by Jeffrey Korn

Pomoc w systemie UNIX

- **man**: wyświetla opisy z dokumentacji UNIX dostępnej w trybie *online*
- **whatis**, **apropos**
- Organizacja wpisów w podręczniku:
  - 1. polecenia
  - 2. wywołania systemowe
  - 3. podprogramy
  - 4. pliki specjalne
  - 5. formaty plików i konwencje
  - 6. gry

92

by Jeffrey Korn

Przykład strony z podręcznika

ls (1) **USER COMMANDS** **ls (1)**

NAME

ls - list files and/or directories

SYNOPSIS

ls [*options*] [*file ...*]

DESCRIPTION

For each directory argument **ls** lists the contents; for each file argument the name and requested information are listed. The current directory is listed if no file arguments appear. The listing is sorted by file name by default, except that file arguments are listed before directories.

.

OPTIONS

-**a**, **--all**
List entries starting with .; turns off **--almost-all**.
-**F**, **--classify**
Append a character for typing each entry.
-**l**, **--long|verbose**
Use a long listing format.
-**r**, **--reverse**
Reverse order while sorting.
-**R**, **--recursive**
List subdirectories recursively.

SEE ALSO

chmod(1), find(1), getconf(1), tw(1)

93

by Jeffrey Korn

Zadania drugoplanowe

- Domyślnie powłoka wykonująca polecenie będzie oczekiwać na jego zakończenie zanim wyświetli kolejny znak zachęty
- Kończąc polecenie znakiem **&** żądamy jednoczesnego wykonywania się polecenia i powłoki

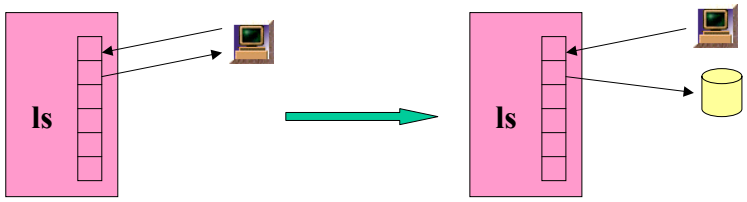
```
$ /bin/sleep 10 &
[1] 3424
$
```

94

by Jeffrey Korn

Przekierowanie

- Zanim polecenie zostanie wykonane, istnieje mozliwość zmiany wejść/wyjść z domyślnych (terminal) na plik
  - Powłoka modyfikuje deskryptory pliku procesu potomnego
  - Program potomny nic nie wie o tej podmianie



95

Przekierowanie wejśc/wyjść

- Przekierowanie wyjścia: >
  - Przykład: **$ ls -l > my_files**
- Przekierowanie wejścia: <
  - Przykład: **$ cat <input.data**
- Dopisywanie do wyjścia: >>
  - Przykład: **$ date >> logfile**
- Przekierowania charakterystyczne dla Bourne Shell : **fd>**
  - Przykład: **$ ls –l 2> error_log**

96

Korzystanie z urządzeń

- Przekierowanie działa także w przypadku urządzeń (tak samo jak w przypadku plików)
- Pliki specjalne w katalogu /dev
  - Przykład: **/dev/tty**
  - Przykład: **/dev/lp**
  - Przykład: **/dev/null**
    - **cat big_file > /dev/lp**
    - **cat big_file > /dev/null**

97

by Jeffrey Korn

Rozwijanie nazwy pliku

- „Dzikie karty” (ang. *wildcards*)
  - ***** - dopasowuje dowolny ciag znaków
  - **?** - dopasowuje dowolny pojedynczy znak
  - **[*list*]** dopasowuje dowolny znak z ***list***
  - **[*lower*-*upper*]** dopasowuje dowolny znak z zakresu od ***lower*** do ***upper*** włącznie

98

by Jeffrey Korn

Rozwijanie nazw plików

- Jeśli występuje dopasowanie wielokrotne, to zwracane są wszystkie nazwy i traktowane jako oddzielne argumenty:

```
$ /bin/ls
file1 file2
$ cat file1
a
$ cat file2
b
$ cat file*
a
b
```

- Obsługiwane przez powłokę (program nigdy nie widzi znaków dzikiej karty)
  - argv[0]: /bin/cat
  - argv[1]: file1
  - argv[2]: file2

a nie

- argv[0]: /bin/cat
- argv[1]: file*

99

Rozwinięcia znaku tyldy

- Cecha dostępna tylko w niektórych powłokach (ksh, csh, bash)
  - **~** rozwija się do nmazwy katalogu domowego użytkownika
    - `~/myfile` → `/home/kornj/myfile`
  - `~user` rozwijane jest w nazwe katalogu domowego użytkownika
    - `~unixtool/file2` → `/home/unixtool/file2`
- Cecha ta jest bardzo przydatna, ponieważ rozmieszczenie katalogu domowego może różnić się na różnych maszynach

100

Zmienna środowiskowa PATH

- Lista katalogów oddzielona znakiem dwukropka.
- Programy wykonywalne bez poprzedzających ich nazwy ścieżek bezwzględnych są wykpnywane tylko wtedy, gdy ścieżka ta występuje na liście.
  - Przeszukiwanie od lewej ku prawej
- Przykład:

```
$ myprogram
sh: myprogram not found
$ PATH=/bin:/usr/bin:/home/kornj/bin
$ myprogram
hello!
```

101

by Jeffrey Korn

Moja ścieżka

```
$ ls
foo
$ foo
sh: foo: not found
```

```
$ ./foo
Hello, foo.
```

- Dlaczego nie tak:

```
$ PATH=.:/bin
$ ls
foo
$ cd /usr/badguy
$ ls
Gratulacje, Twój plik został właśnie usunięty z
Twojego katalogu.
```

102

by Jeffrey Korn

Zmienne powłoki

- Powłoka posiada klika mechanizmów tworzenia zmiennych. Zmienną jest nazwa reprezentująca wartość napisu.
  - Zmienne powłoki mogą oszczędzić czas i zredukować wyświetlanie błędów, zmiennych
- Pozwalaja na przechowywanie i obrabianie informacji
- Dwa typy: lokalne i żrodowiskowe
  - Zmienne lokalne definiowane są przez użytkownika samej powłoki
  - Zmienne środowiskowe poczodzą od systemu operacyjnego i przekazywane sa do procesów potomnych

103

by Jeffrey Korn

Zmienne

- Składnia zmienia się wraz typem powłoki
  - `name=value` `# sh, ksh, bash`
  - `set name = value` `# csh`
- Dostęp do wartości zmiennej: `$var`
- Przekształcenie lokalnej zmiennej w środowiskową:

```
export variable
```

104

by Jeffrey Korn

Zmienne środowiskowe

Nazwa	Znaczenie
$HOME	Bezwzględna nazwa ścieżki katalogu domowego
$PATH	Lista katalogów do przeszukania
$MAIL	Bezwzględna nazwa ścieżki do mailbox
$USER	ID aktualnego użytkownika
$SHELL	Bezwgędna nazwa scieżki powłoki logowania
$TERM	Typ terminala
$PS1	Postać znaków zachęty

105

by Jeffrey Korn

Cytowanie

- Cytowanie przypisuje znakom literalne znaczenie znakom, które są w sposób specjalny przetwarzane przez powłokę. Literalne cytowanie nie sosuje się do poleceń
- Znak apostrofu (') zabrania zastępowania znaku dzikiej karty, podstawiania zmiennej i polecenia
- Znak czydzysłowia (") zabrania zastępowania jedynie znaku dzikiej karty.
- Znak *backslash* (\) może być stosowany podczas cytowania literalnego
- W przpadku zagniezdżania cytowań, znaczenie mają jedynie zewnętrzne znaki cytowania.

106

by Jeffrey Korn

Przykład cytowania

```
$ cat file*
a
b

$ cat “file*”
cat: file* not found

$ FILES=“file1 file2”
$ cat $FILES
a
b

$ cat “$FILES”
cat: file1 file2 not found
```

107

by Jeffrey Korn

Polecenia złożone

- Polecenia wielokrotne
  - Odzielone średnikiem
- Opertory logiczne
- Grupowanie poleceń
  - potoki
- Zastępowanie poleceń
- Struktury sterujące

108

by Jeffrey Korn

Operatory logiczne

- Wartościa wyjściowa z programu jest liczba
  - 0 oznacza sukces
  - cokolwiek innego oznacza kod błędu
- cmd1 && cmd2
  - cmd2 zostanie wykonane wtedy, gdy cmd1 zakończy się sukcesem
- cmd1 || cmd2
  - cmd2 zostanie wykonane wtedy, gdy cmd1 nie zakończy się sukcesem

```
$ ls bad_file > /dev/null && date
$ ls bad_file > /dev/null || date
Wed Sep 26 07:43:23 2001
```

109

by Jeffrey Korn

Potoki

- Wyjście jenego programu jest wejściem na nastepny
- Stosowane są łącza systemu UNIX
- Przykład: `$ cat file | wc -l`
  - Zlicza liczbe linii w pliku
- Mogą być złożone
  - `grep string | sort | uniq | wc`

110

by Jeffrey Korn