Temat zajęć: Obsługa systemu plików.

Czas realizacji zajęć:	*90 min.*
Zakres materiału, jaki zostanie zrealizowany podczas zajęć:	Tworzenie i otwieranie plików, deskryptory plików, odczyt i zapis danych do plików, implementacja przykładowych programów obsługi plików.

I. Operacje na plikach zwykłych.

Jądro systemu operacyjnego UNIX udostępnia dwie podstawowe operacje na plikach — *odczyt* i *zapis* — realizowane odpowiednio przez funkcje systemowe **read** i **write**. Z punktu widzenia jądra w systemie UNIX plik nie ma żadnej struktury, tzn. nie jest podzielony na przykład na rekordy. Plik jest traktowany jako tablica bajtów, zatem operacje odczytu lub zapisu mogą dotyczyć dowolnego fragmentu pliku, określonego z dokładnością do bajtów.

Wykonanie operacji wymaga wskazania pliku, na którym operacja ma zostać wykonana. Plik w systemie UNIX identyfikowany jest przez nazwę (w szczególności podaną w postaci ścieżki katalogowej), przy czym podawanie nazwy pliku przy każdym odwołaniu do niego wymagałoby każdorazowego przeszukiwania odpowiednich katalogów w celu ostatecznego ustalenia jego lokalizacji. W celu uniknięcia czasochłonnego przeszukiwania katalogów podczas lokalizowania pliku przy każdej operacji na nim, wprowadzona została funkcja systemowa **open**, której zadaniem jest zaalokowanie niezbędnych zasobów w jądrze, umożliwiających wykonywanie dalszych operacji na pliku bez potrzeby przeszukiwania katalogów. Funkcja **open** zwraca *deskryptor*, który jest przekazywany jako parametr aktualny, identyfikujący plik, do funkcji systemowych związanych z operacjami na otwartych plikach. (Standardowo zajęte są deskryptory 0, 1 i 2, odpowiadające standardowemu wejściu, standardowemu wyjściu i standardowemu wyjściu diagnostycznemu. Wszystkie te deskryptory najczęściej związane są z plikiem specjalnym, jakim jest terminal)
Przy otwieraniu pliku przekazywany jest tryb otwarcia, określający dopuszczalne operacje, jakie można wykonać w związku z tym otwarciem, np. *tylko zapis*, *tylko odczyt* lub *zapis i odczyt*. Tryb otwarcia może mieć również wpływ na sposób wykonania tych operacji, np. każda operacja zapisu dopisuje dane na końcu pliku.

Jądro systemu operacyjnego dostarcza też mechanizm tworzenia plików. Mechanizm tworzenia plików zwykłych dostępny jest przez funkcję systemową **creat**, która tworzy plik o nazwie podanej jako parametr aktualny i otwiera utworzony plik w trybie do zapisu, zwracając odpowiedni deskryptor.

Funkcje tworzące pliki i operujące na nich opisane są w części 2 pomocy systemowej.
Tworzenie i otwieranie plików realizowane jest za pomocą funkcji:

- open - otwarcie pliku (uogólniona funkcja open umożliwia również utworzenie pliku),
- creat - utworzenie pliku i otwarcie do zapisu,
- dup - utworzenie kopii deskryptora i nadanie jej pierwszego wolnego numeru z tablicy otwartych plików,
- dup2 - utworzenie kopii deskryptora, umożliwiające określenie jej identyfikatora przez użytkownika,
- close - zamknięcie deskryptora otwartego pliku,
- unlink - usunięcie dowiązania do pliku

Operacje na plikach realizowane są za pomocą funkcji:

- read - odczyt fragmentu pliku,
- write - zapis fragmentu pliku,
- lseek - przesunięcie wskaźnika bieżącej pozycji

Powyższe funkcje zdefiniowane są w pliku `fcntl.h`.

II. Funkcje systemowe i ich argumenty.

- **`int creat(const char *pathname, mode_t mode)`**

  Wartości zwracane:
  poprawne wykonanie funkcji: deskryptor otwartego pliku
  zakończenie błędne: -1

  Możliwe kody błędów (*errno*) w przypadku błędnego zakończenie funkcji:
  EEXIST – plik o podanej nazwie już istnieje, a użyto flag O_CREAT i O_EXCL
  EFAULT – nazwa *pathname* wskazuje poza dostępną przestrzeń adresową
  EACCES – żądany dostęp do pliku nie jest dozwolony
  ENFILE – osiągnięto limit otwartych plików w systemie
  EMFILE – proces już otworzył dozwoloną maksymalną liczbę plików
  EROFS – żądane jest otwarcia w trybie zapisu pliku będącego plikiem tylko do odczytu

  Argumenty funkcji:
  pathname – wskaźnik do napisu zawierającego nazwę ścieżki pliku, który ma być otwarty (nazwa bezwzględna lub względna)
  mode – prawa dostępu (np. 0640)

  UWAGI:
  Funkcja tworzy plik, którego lokalizację wskazuje parametr *pathname.* Prawa dostępu do utworzonego pliku ustawiane są zgodnie z parametrem *mode*. Jeśli plik o takiej nazwie już istnieje a proces wywołujący funkcję **creat** ma prawo do zapisu tego pliku, to jego zawartość jest usuwana (następuje obcięcie pliku). Plik wskazywany przez *pathname* otwierany jest w trybie do zapisu.

- **`int open(const char *pathname, int flags[, mode_t mode])`**

  Wartości zwracane:
  poprawne wykonanie funkcji: deskryptor otwartego pliku
  zakończenie błędne: -1

  Możliwe kody błędów (*errno*) w przypadku błędnego zakończenie funkcji – analogicznie do funkcji **creat**

  Argumenty funkcji:
  pathname – wskaźnik do napisu zawierającego nazwę ścieżki pliku, który ma być otwarty (nazwa bezwzględna lub względna)
  flags – metoda dostępu
  - O_RDONLY – otwarcie w trybie tylko do odczytu
  - O_WRONLY – otwarcie w trybie tylko do zapisu
  - O_RDWR –- otwarcie w trybie do odczytu i do zapisu
    Argument flags może być połączony bitowym OR z jedną (lub więcej) z następujących wartości:
  - O_CREAT – utworzenie pliku, jeśli plik jeszcze nie istnieje,
  - O_TRUNC – obcięcie pliku, jeśli plik istnieje i otwierany jest w trybie O_WRONLY lub O_RDWR,
  - O_EXCL – powoduje zgłoszenie błędu jeśli plik już istnieje i otwierany jest z flagą O_CREAT
  - O_APPEND – operacje pisania odbywają się na końcu pliku.

mode – prawa dostępu, jest to argument opcjonalny

UWAGI:
Parametr wejściowy *pathname* jest nazwą (w szczególności pełną nazwą ścieżkową) pliku, parametr wejściowy *flags* oznacza tryb otwarcia pliku pliku i może mieć następujące wartości: O_RDONLY, O_WRONLY, O_RDWR .
Dodatkowo w trybie zapisu możliwe jest użycie flagi O_APPEND, która jest sumowana bitowo z O_WRONLY lub O_RDWR i powoduje, że zapis wykonywany jest zawsze na końcu pliku. Dane są więc dopisywane do pliku i system gwarantuje, że nie nastąpi nadpisanie danych zapisanych wcześniej.
Poza funkcjami **open** i **creat** istnieje uogólniona, trzyparametrowa wersja funkcji **open**, która łączy cechy obu tych funkcji. Dodatkowy parametr prawa określa prawa dostępu do pliku (podobnie jak dla funkcji **creat**) i wykorzystywany jest wówczas, gdy tryb otwarcia wymusza tworzenie pliku. Przydatne są wówczas dodatkowe flagi umieszczane w trybie otwarcia: O_CREAT, O_TRUNC, O_EXCL.

Funkcja **creat** jest równoważna uogólnionej funkcji **open** z parametrem tryb równym O_WRONLY| O_CREAT|O_TRUNC, czyli poniższe wywołania są równoważne:

```
creat( nazwa_pliku, prawa );
open( nazwa_pliku, O_WRONLY|O_CREAT|O_TRUNC, prawa );
```

- **`int close(int fd)`**

Wartości zwracane:
poprawne wykonanie funkcji: 0
zakończenie błędne: -1

Możliwe kody błędów (*errno*) w przypadku błędnego zakończenie funkcji:
EBADF – wartość *fd* nie jest prawidłowym deskryptorem otwartego pliku

Argumenty funkcji:
fd – deskryptor zamykanego pliku

UWAGI:
Zamknięcie deskryptora pliku. Funkcja zamyka deskryptor pliku przekazany przez parametr *fd*. Po zamknięciu pliku zwalniana jest pozycja w tablicy deskryptorów i może ona zostać ponownie wykorzystana przy otwarciu kolejnego pliku, czyli nowo otwarty plik może otrzymać ten sam deskryptor, który miał plik wcześniej zamknięty. Ponadto zmniejszany jest o 1 licznik deskryptorów w tablicy otwartych plików. Jeśli *fd* jest ostatnią kopią deskryptora pliku, to zasoby z nim związane zostają zwolnione, natomiast jeśli deskryptor był ostatnia referencją do pliku, który usunięto komendą **unlink**, plik jest kasowany.

- **`int dup(int oldfd)`**

Wartości zwracane:
poprawne wykonanie funkcji: nowy deskryptor
zakończenie błędne: -1

Możliwe kody błędów (*errno*) w przypadku błędnego zakończenie funkcji:
EBADF – *oldfd* nie jest deskryptorem otwartego pliku lub *newfd* jest poza dozwolonym zasięgiem deskryptorów plików

EMFILE – proces już osiągnął maksymalną liczbę otwartych deskryptorów plików

Argumenty funkcji:
oldfd – deskryptor zamykanego pliku

UWAGI:
Funkcja tworzy kopię pozycji w tablicy deskryptorów na innej, wolnej pozycji o najniższym indeksie. W ten sposób otrzymujemy nowy deskryptor związany z tym samym otwartym plikiem. Nowa pozycja w tablicy deskryptorów wskazuje na tą samą pozycję w tablicy otwartych plików, stąd stary i nowy deskryptor mogą być używane zamiennie. Deskryptory dzielą pozycję pliku i flagi, np. jeśli pozycja pliku zmieniła się po użyciu funkcji **lseek** na jednym z deskryptorów, zmieniła się ona także na drugim.

- **`int dup2(int oldfd, int newfd)`**

Wartości zwracane:
poprawne wykonanie funkcji: nowy deskryptor
zakończenie błędne: -1

Możliwe kody błędów (*errno*) w przypadku błędnego zakończenie funkcji – analogicznie do funkcji **dup**

Argumenty funkcji:
oldfd – deskryptor zamykanego pliku
newfd – nowy deskryptor

UWAGI:
Utworzenie kopii deskryptora. Podobnie jak w przypadku funkcji **dup** tworzony jest nowy deskryptor otwartego pliku identyfikowanego przez *oldfd*. *Newfd* staje się nowym, dodatkowym deskryptorem, a jeśli przed wywołaniem **dup2** identyfikował on inny plik, następuje zamknięcie tego deskryptora przed powieleniem *oldfd*. Funkcja zwraca wartość nowego deskryptora.
Wykonanie operacji **close(1)**; **dup(fd)**; jest równoważne operacji **dup(1,fd)**

- **`int unlink(const char *pathname)`**

Wartości zwracane:
poprawne wykonanie funkcji: 0
zakończenie błędne: -1

Możliwe kody błędów (*errno*) w przypadku błędnego zakończenie funkcji:
EFAULT – nazwa *pathname* wskazuje poza dostępną przestrzeń adresową
EACCES – żądany dostęp do pliku nie jest dozwolony

Argumenty funkcji:
pathname – wskaźnik do napisu zawierającego nazwę ścieżki pliku, który ma być otwarty (nazwa bezwzględna lub względna)

UWAGI:
Funkcja powoduje usunięcie dowiązania do pliku. Wskazana przez parametr *pathname* nazwa pliku jest usuwana, a dodatkowo - jeśli było to jedyne dowiązanie tego pliku następuje usunięcie pliku z systemu

- **`int read(inf fd, void *buf, size_t count)`**

  Wartości zwracane:
  poprawne wykonanie funkcji: rzeczywista liczba bajtów, jaką udało się odczytać
  zakończenie błędne: -1

  Możliwe kody błędów (*errno*) w przypadku błędnego zakończenie funkcji:
  EINTR – wywołanie zostało przerwane sygnałem przed odczytaniem danych
  EAGAIN – przy użyciu O_NONBLOCK wybrano nieblokujące I/O, a nie ma akurat danych dostępnych do odczytania natychmiast
  EIO – błąd I/O. Zdarza się to np. jeśli proces jest w grupie procesów tła próbuje czytać z kontrolującego tty, lub ignoruje sygnał SIGTIN, lub jego grupa procesów jest osierocona.
  EISDIR – *fd* odnosi się do katalogu
  EBADF – *fd* nie jest prawidłowym deskryptorem pliku, lub nie jest otwarty dla odczytu
  EINVAL – *fd* wskazuje na obiekt nieodpowiedni do odczytu
  EFAULT – *buf* wskazuje poza dostępną przestrzeń adresową

  Argumenty funkcji:
  fd – deskryptor pliku z którego mają zostać odczytane dane
  buf – adres bufora znajdującego się w segmencie danych procesu, do którego zostaną przekazane dane odczytane z pliku w wyniku wywołania funkcji **read**
  count – ilość bajtów do odczytania

  UWAGI:
  Odczyt danych z pliku. Funkcja powoduje odczyt *count* bajtów z otwartego pliku, identyfikowanego przez deskryptor *fd*, począwszy od bieżącej pozycji wskaźnika do pliku i umieszczenie ich pod adresem *buf* w przestrzeni adresowej procesu. Funkcja zwraca liczbę bajtów na której udało się wykonać operację (zero oznacza koniec pliku).
  Odczyt powoduje zmianę wskaźnika bieżącej pozycji w pliku. Po otwarciu pliku wskaźnik ten ustawiony jest na 0, czyli na początek pliku, a po kolejnych operacjach przesuwa się w kierunku końca pliku o tyle bajtów ile udało się odczytać

- **`int write(inf fd, void *buf, size_t count)`**

  Wartości zwracane:
  poprawne wykonanie funkcji: rzeczywista liczba bajtów, jaką udało się zapisać
  zakończenie błędne: -1

  Możliwe kody błędów (*errno*) w przypadku błędnego zakończenie funkcji:
  EBADF – deskryptor *fd* nie jest prawidłowym deskryptorem pliku, lub nie jest otwarty dla odczytu
  EINVAL – deskryptor *fd* wskazuje na obiekt nieodpowiedni do zapisu
  EFAULT – deskryptor *buf* jest poza dostępną przestrzenią adresową
  EPIPE – *fd* jest podłączony do potoku, lub gniazda, którego drugi koniec jest zamknięty. Gdy zdarzy się taka sytuacja, proces otrzyma sygnał SIGPIPE; jeśli jednak go przechwytuje, blokuje lub ignoruje, zwrócony zostanie błąd EPIPE
  EAGAIN – wybrano nieblokujący I/O (przy użyciu O_NONBLOCK) a do potoku lub gniazda o deskryptorze *fd* nie można natychmiast zapisać danych
  EINTR – wywołanie zostało przerwane sygnałem przed zapisaniem danych
  ENOSPC – urządzenie, zawierające plik o deskryptorze *fd* nie ma miejsca na dane

  Argumenty funkcji:

fd – deskryptor pliku do którego mają zostać zapisane dane
buf – adres bufora znajdującego się w segmencie danych procesu, z którego zostaną pobrane dane zapisane przez funkcję **write**
count – ilość bajtów do zapisania

UWAGI:
Zapis danych do pliku. Funkcja powoduje zapis *count* bajtów do otwartego pliku, identyfikowanego przez deskryptor *fd*, począwszy od bieżącej pozycji wskaźnika do pliku i umieszczenie ich pod adresem *buf* w przestrzeni adresowej procesu. Funkcja zwraca liczbę bajtów na której udało się wykonać operację.
Podobnie jak dla funkcji **read** zapis powoduje zmianę wskaźnika bieżącej pozycji w pliku.

- **`long lseek(inf fd, off_t offset, int whence)`**

Wartości zwracane:
poprawne wykonanie funkcji: nowa bieżąca pozycja wskaźnika bieżącej pozycje, liczona w bajtach względem początku pliku
zakończenie błędne: -1

Możliwe kody błędów (*errno*) w przypadku błędnego zakończenie funkcji:
EBADF – deskryptor *fd* nie jest prawidłowym deskryptorem pliku
ESPIPE – *fd* jest związany z potokiem, gniazdem, lub FIFO
EINVAL – parametr *whence* ma nieprawidłową wartością

Argumenty funkcji:
fd – deskryptor pliku do którego mają zostać zapisane dane
offset – liczba bajtów, o jaką należy przesunąć wskaźnik
whence – parametr określający pozycję względem której jest przesuwany wskaźnik

UWAGI:
Przesunięcie wskaźnika bieżącej pozycji. Funkcja powoduje zmianę wskaźnika bieżącej pozycji w otwartym pliku o deskryptorze *fd*. Nowa pozycja jest bajtem o numerze *offset* liczonym odpowiednio względem :

- początku pliku, gdy *whence* = SEEK_SET
- końca pliku , gdy *whence* = SEEK_END
- bieżącej pozycji, gdy *whence* = SEEK_CUR

Wartość parametru *offset* < 0 oznacza przesunięcie w kierunku początku pliku (niższych indeksów), a wartość *offset* >0 oznacza przesunięcie w kierunku końca pliku (wyższych indeksów). Funkcja zwraca aktualną wartość wskaźnika bieżącej pozycji (po przesunięciu) liczoną względem początku pliku. Działanie funkcji sprowadza się do modyfikacji zawartości odpowiedniego pola w tablicy otwartych plików. Nie można przesunąć wskaźnika na pozycję przed początkiem pliku, można za to wyszczególnić pozycję poza końcem pliku. Jeśli później w tym miejscu są zapisane jakieś dane, to kolejne odczyty danych z luki zwrócą bajty zerowe (aż dane nie zostaną rzeczywiście zapisane w luce). Definicja funkcji **lseek** znajduje się w plikach `unistd.h` oraz `sys/types.h`.

III. Przykłady zastosowania operacji plikowych.

Listing 1 przedstawia program do kopiowania pliku. W programie wykorzystano funkcje systemowe **open**, **creat**, **read**, **write** i **close**. Nazwy plików przykazywane są jako argumenty w linii poleceń przy uruchamianiu programu. Jako pierwszy argument przekazywana jest nazwa istniejącego pliku

źródłowego, a jako drugi argument przekazywana jest nazwa pliku docelowego, który może zostać dopiero utworzony:

```
   #include <fcntl.h>
   #include <stdio.h>
3  #define MAX 512

   int main(int argc, char* argv[]){
6     char buf[MAX];
      int desc_zrod, desc_cel;
      int lbajt;
9
      if (argc<3){
            fprintf(stderr, "Za malo argumentow. Uzyj:\n");
12          fprintf(stderr, "%s <plik zrodlowy> <plik docelowy>\n",
            argv[0]);
            exit(1);
      }
15
      desc_zrod = open(argv[1], O_RDONLY);
      if (desc_zrod == -1){
18           perror("Blad otwarcia pliku zrodlowego");
            exit(1);
      }
21
      desc_cel = creat(argv[2], 0640);
      if (desc_cel == -1){
24          perror("Blad utworzenia pliku docelowego");
            exit(1);
       }
27
      while((lbajt = read(desc_zrod, buf, MAX)) > 0){
            if (write(desc_cel, buf, lbajt) == -1){
30                perror("Blad zapisu pliku docelowego");
                 exit(1);
            }
33    }

      if (lbajt == -1){
            perror("Blad odczytu pliku zrodlowego");
36          exit(1);
      }

39    if (close(desc_zrod) == -1 || close(desc_cel) == -1){
            perror("Blad zamkniecia pliku");
            exit(1);
42    }
      exit(0);
45 }
```

Listing 1: Kopiowanie pliku

Opis programu: W liniach 10–14 następuje sprawdzenie poprawności przekazania argumentów z linii poleceń. Następnie otwierany jest w trybie tylko do odczytu plik źródłowy i sprawdzana jest poprawność wykonania tej operacji (linie 16–20). Podobnie tworzony jest i otwierany w trybie tylko do zapisu plik docelowy (linie 22–26). Właściwe kopiowanie zawartości pliku źródłowego do pliku docelowego następuje w pętli w linich 28–33. Wyjście z pętli **while** następuje w wyniku zwrócenia przez funkcję **read** wartości 0 lub -1. Wartość -1 oznacza błąd, co sprawdzane jest zaraz po

zakończeniu pętli w liniach 34–37.
Po każdym błędzie funkcji systemowej wyświetlany jest odpowiedni komunikat i następuje zakończenie procesu przez wywołanie funkcji systemowej exit. Jeśli wywołania funkcji systemowych zakończą się bezbłędnie, sterowanie dochodzi do linii 39, gdzie następuje zamknięcie plików.
Listing 2 przedstawia program do wyświetlania rozmiaru pliku. W programie wykorzystano funkcje systemowe **open**, **lseek** i **close**. Nazwa pliku przykazywana jest jako argument w linii poleceń przy uruchamianiu programu.

```
   #include <fcntl.h>
   #include <stdio.h>
3
   int main(int argc, char* argv[]){
      int desc;
6     long rozm;

      if (argc < 2){
9           fprintf(stderr, "Za malo argumentow. Uzyj:\n");
            fprintf(stderr, "%s <nazwa pliku>\n", argv[0]);
            exit(1);
12    }

      desc = open(argv[1], O_RDONLY);
15    if (desc == -1){
            perror("Blad otwarcia pliku");
            exit(1);
18    }

      rozm = lseek(desc, 0, SEEK_END);
21    if (rozm == -1){
            perror("Blad w pozycjonowaniu");
            exit(1);
24    }

      printf("Rozmiar pliku %s: %ld\n", argv[1], rozm);
27
      if (close(desc) == -1){
            perror("Blad zamkniecia pliku");
30          exit(1);
      }
33    exit(0);
   }
```

Listing 2: Wyprowadzanie rozmiaru pliku

Opis programu: W liniach 8–12 następuje sprawdzenie poprawności przekazania argumentów z linii poleceń. Następnie otwierany jest w trybie tylko do odczytu plik o nazwie podanej jako argument w linii poleceń i sprawdzana jest poprawność wykonania tej operacji (linie 14–18). Po otwarciu pliku następuje przesunięcie wskaźnika bieżącej pozycji za pomocą funkcji **lseek** na koniec pliku i zarazem odczyt położenia tego wskaźnika względem początku pliku (linia 20). Uzyskany wynik działania funkcji **lseek**, jeżeli nie jest to wartość -1, jest rozmiarem pliku w bajtach. Wartość ta jest wyświetlana na standardowym wyjściu (linia 26), po czym plik jest zamykany (linia 28).

IV. Zadania do samodzielnego wykonania.

1) Napisz program kopiujący zawartość pliku o nazwie podanej jako pierwszy parametr do pliku którego nazwa podana jest jako drugi parametr
2) Napisz program zmieniający kolejność znaków w każdej linii pliku o nazwie podanej jako

parametr.

3) Napisz procedurę kopiowania ostatnich 10 znaków, słów i ostatnich 10 linii jednego pliku do innego.
4) Napisz program do rozpoznawania czy plik o podanej nazwie jest plikiem tekstowym (plik tekstowy zawiera znaki o kodach 0-127 – można w tym celu użyć funkcji isascii)
5) Napisz program, który w pliku o nazwie podanej jako ostatni argument zapisze połączoną zawartość wszystkich plików których nazwy zostały podane w linii poleceń przed ostatnim argumentem.
6) Napisz program liczący ile jest słów w pliku (zakładamy, że słowa składają się z małych i dużych liter alfabetu oraz cyfr i znaku podkreślenia, a wszystkie pozostałe znaki są separatorami słów)
7) Napisz program do porównywania plików o nazwach przekazanych jako argumenty. Wynikiem działania programu ma być komunikat że *pliki są identyczne*, *pliki różnią się od znaku nr<nr znaku> w linii<nr znaku linii>* lub – gdy gdy jeden z plików zawiera treść drugiego uzupełnioną o jakieś dodatkowe znaki – *plik <nazwa> zawiera<liczba>znaków więcej niż zawartość pliku <nazwa>*

V. Literatura.

[HGS99] Havilland K., Gray D., Salama B., *Unix - programowanie systemowe*, ReadMe, 1999

[Roch97] Rochkind M.J., *Programowanie w systemie UNIX dla zaawansowanych*, VI. WNT, 1997

[NS99] Neil M., Stones R, *Linux. Programowanie*, ReadMe, 1999

[MOS02] Mitchell M., Oldham J., Samuel A., *Linux. Programowanie dla zaawansowanych*, ReadMe,2002

[St02] Stevens R.W., *Programowanie w środowisku systemu UNIX*, WNT, 2002