Spis treści

Na podstawie:

- *M.J.Rochkind - Programowanie w systemie UNIX dla zaawansowanych (Advanced UNIX Programming)*
- `https://ai.ia.agh.edu.pl/wiki/pl:dydaktyka:so:2017:labs:lab_intro`
- *Graham Glass, King Ables - Linux dla programistów i użytkowników*

1 Wiadomości ogólne potrzebne do zrozumienia tematu

1.1. System plików

Jak wiemy, pliki w Linuksie zorganizowane są w hierarchii etykiet (tzw. struktura katalogowa). Istnieją trzy rodzaje plików określanych przez etykiety

Pliki zwykłe
zawierają sekwencję bajtów będących przeważnie kodem lud danymi. Można z nich korzystać za pomocą standardowych wywołań systemowych wejścia-wyjścia. Operacje na takich plikach omawialiśmy w lab. 1, w tym lab poznamy więcej funkcji systemowych działających na takich plikach.

Katalogi
przechowywane są na dysku w specjalnym formacie i tworzą szkielet systemu plików; można się do nich odwoływać jedynie za pomocą przeznaczonych do tego wywołań systemowych.

Pliki specjalne i systemowe
Odpowiadają urządzeniom peryferyjnym, takim jak drukarki i dyski, i wykorzystywane są do komunikacji międzyprocesowej, takiej jak np. potoki czy gniazda; można się do nich odwołać za pomocą standardowych wywołań systemowych wejścia-wyjścia. W tym lab. rozszerzymy wiedzą o operacje na takich plikach.

1.2. Odczyt danych z dysku, wejście-wyjście

Dane na dysku są przechowywane w 'kawałkach'. Podstawową jednostką pojemności dysku są tzw. bloki. Operacje wejścia-wyjścia są zawsze przeprowadzane na całych, nierozdzielnych blokach. Nawet jeśli odczytamy w programie 1 bajt przy użyciu read, to tak naprawdę żądanie to spowoduje odczytanie całego bloku z dysku, który zostanie przekazany do bufora jądra. Następnie pierwszy bajt zostanie skopiowany do naszego procesu i w ten sposób dostaniemy nasz żądany jeden bajt. Typowe rozmiary bloków to np. 1024 bajty czy 2048 bajtów.

1.3. I-węzły (I-nodes)

W Linuksie do przechowywania informacji o pliku wykorzystywane są I-wezły. Są to struktury danych przechowujące informację o pliku. Każdy plik posiada dokładnie jeden i-node o unikalnym numerze.

- W i-node zwykłego katalogu lub pliku znajdują się wskaźniki do bloków danych pliku
- W i-node plików specjalnych znajdują się informacje pozwalające na identyfikację urządzenia peryferyjnego.

Informacje przechowywane w i-node to większość informacji widocznych po wydaniu polecenia ls -l (UWAGA: i-node NIE przechowuje jednak nazwy pliku). Każdy plik przechowuje informację takie jak:

- typ pliku: zwykły, katalog, specjalny...
- prawa dostępu do pliku
- właściciel, grupa
- licznik dowiązań twardych
- data i godzina ostatniej modyfikacji

Dodatkowo dla plików specjalnych przechowywane są numer główny i poboczny urządzenia, dla dowiązań wartość tego dowiązania, dla plików zwykłych i katalogów położenie bloków danych tego pliku.

1.4. Pliki etc/passwd i etc/group

1.4.1. etc/passwd

Dla każdego użytkownika w pliku haseł etc/passwd znajduje się wpis w postaci:

nazwa_uzytkownikia:haslo:ID_uzytkownika:ID_grupy:dane_osobiste:katalog_domowy:program

Oczywiście hasło nie jest widoczne w tym pliku, jest ono kodowane, a prawdziwe przechowywane jest gdzie indziej i widoczne tylko dla super usera.

1.4.2. etc/group

Dla każdej grupy w systemie znajduje się odpowiedni wpis w pliku grup /etc/group. Ma on postać

nazwa_grupy:hasło_grupy:identyfikator_grupy:uzytkownicy

Również w tym pliku hasło jest zaszyfrowane.

1.5. Linki

Linki dzielimy na

- twarde - wskazują na i-node, każdy osobny link twardy jest osobnym wskaźnikiem na ten sam i-node, tworzone są komendą *ln blah1 blah1-hard*
- miękkie - wskazują na twardy link, który wskazuje do i-node, a nie bezpośrednio na i-node, tworzone są komendą *ln -s blah2 blah2-soft*

2 Funkcje systemowe

2.1. stat, fstat, lstat

Często w programach przydatne byłoby uzyskanie dostępu do informacji o plikach takich jak data modyfikacji, prawa dostępu itp. Jest to możliwe przy użyciu funkcji systemowych stat, fstat oraz lstat. Funkcje te służą do uzyskiwania danych przechowywanych w i-nodes dla konkretnych plików. Funkcje te różnią się pomiędzy sobą nieznacznie w sposobie działania i wywoływania. Do wywoływania każdej z funkcji potrzebny jest nagłówek sys/stat.h.

```c
#include <sys/stat.h>
int stat(
  const char *path, /* pathname */
  struct stat *buf /* returned information */
);
/* Returns 0 on success or -1 on error (sets errno) */
```

```c
#include <sys/stat.h>
int lstat(
  const char *path, /* pathname */
  struct stat *buf /* returned information */
); /* Returns 0 on success or -1 on error (sets errno) */
```

```c
#include <sys/stat.h>
int fstat(
  int fd, /* file descriptor */
  struct stat *buf /* returned information */
); /* Returns 0 on success or -1 on error (sets errno) */
```

- Funkcja *stat* przyjmuje ścieżkę jako argument i szuka i-node podążając tą ścieżką.
- Funkcja *lstat* jest identyczna jak *stat*, ale w przypadku gdy ścieżka prowadzi do linku symbolicznego, wyświetla metadane powiązane z tym linkiem, a nie z tym do czego on wskazuje, jak w przypadku *stat*.
- Funkcja *fstat* przyjmuje deskryptor, a nie ścieżkę do pliku, znajduje i-node w tablicy aktywnych i-node w jądrze systemu.

Rzeczą, którą zauważamy jest to, że każda z tych funkcji zwraca strukturę tego samego typu - stat. Strukturę przekazujemy z zewnątrz przez wskaźnik aby funkcja mogła zmodyfikować przekazaną strukturę. Naturalnie nasuwa się pytanie, czym jest ta struktura.

3 Struktura *stat*

3.1. Informacje ogólne

Struktura ta jest zdefiniowana w sys.stat.h, a jej pola są dostosowane do przechowywania informacji o pliku. Struktura ta jest postaci:

```
struct stat {
  dev_t st_dev;      /* device ID of file system */
  ino_t st_ino;      /* i-number */
  mode_t st_mode;    /* mode (see below) */
  nlink_t st_nlink;   /* number of hard links */
  uid_t st_uid;    /* user ID */
  gid_t st_gid;      /* group ID */
  dev_t st_rdev;     /* device ID (if special file) */
  off_t st_size;     /* size in bytes */
  time_t st_atime;  /* last access */
  time_t st_mtime;  /* last data modification */
  time_t st_ctime;  /* last i-node modification */
  blksize_t st_blksize; /* optimal I/O size */
  blkcnt_t st_blocks; /* allocated 512-byte blocks */
};
```

Poszczególne implementacje tej struktury mogą różnić się dla konkretnego systemu.
Znaczenie pól struktury:

- dev_t st_dev - identyfikuje urządzenie, które zawiera i-node pliku
- ino_t st_ino - numer i-node
- **mode_t st_mode** - typ pliku, prawa dostępu i kilka innych
- nlink_t st_nlink - liczba dowiązań twardych
- uid_t st_uid - identyfikator użytkownika
- gid_t st_gid - identyfikator grupy
- dev_t st_rdev - tylko dla plików specjalnych, to urządzenie które reprezentuje plik
- off_t st_size - rozmiar pliku
- time_t st_atime - czas ostatniego dostępu
- time_t st_mtime - czas ostatniej modyfikacji
- time_t st_ctime - czas ostatniej zmiany statusu

3.2. Pole mode_t st_mode struktury stat

st_mode - szczególnie uważnie przyjrzymy się temu polu ze struktury stat. Pole to zawiera informacje o typie pliku, pozwoleniach do pliku i kilka innych. Informacje te są zapisane przy pomocy bitów, które są reprezentowane przez flagi. Odpowiedzmy sobie na pytanie jak odczytać te informacje z tego pola.

3.2.1. Odczytywanie typu pliku

Typ pliku można odczytać na dwa sposoby, oba wykorzystują predefiniowane makra reprezentujące ustawione w polu st_mode bity:

- przy użyciu S_IFMT
- przy użyciu makr testujących, które zwracają 0 dla fałszu lub inną liczbę dla prawdy

3.2.2. Odczytywanie typu pliku przy użyciu S_IFMT

Jest to makro reprezentujące maskę dzięki której można sprawdzić typ pliku. Aby przy pomocy tego makra uzyskać typ pliku należy wykonać logiczny AND z polem st_mode struktury stat w taki sposób:

```
buf.st_mode & S_IFMT
```

Wynikiem wykonania takiej instrukcji jest powstanie nowej sekwencji bitów, które reprezentują konkretny typ pliku. Bity te można sprawdzić przyrównując wynik do któregoś z makr reprezentujących typ pliku:

```
S_IFBLK /* block special file */
S_IFCHR /* character special file */
S_IFDIR /* directory */
S_IFIFO /* named or un-named pipe */6
S_IFLNK /* symbolic link */
S_IFREG /* regular file */
S_IFSOCK /* socket */
```

Przykładowe sprawdzenie czy plik jest typu Socket miałoby więc postać:

```
if ((buf.st_mode & S_IFMT) == S_IFSOCK)
```

Niepoprawne jest natomiast:

```
if ((buf.st_mode & S_IFSOCK) == S_IFSOCK) /* wrong */
```

3.2.3. Odczytywanie typu pliku przy użyciu makr testujących

Drugim ze sposobów jest użycie makr testujących, które jako argument przyjmują pole st_mode. Makra te zwracają 0 w przypadku fałszu oraz inną liczbę dla prawdy. Lista możliwych do użycia makr:

```
S_ISBLK(mode) /* is a block special file */
S_ISCHR(mode) /* is a character special file */
S_ISDIR(mode) /* is a directory */
S_ISFIFO(mode) /* is a named or un-named pipe */
S_ISLNK(mode) /* is a symbolic link */
S_ISREG(mode) /* is a regular file */
S_ISSOCK(mode) /* is a socket */
```

Przykład sprawdzenia czy plik jest typu Socket:

```
if (S_ISSOCK(buf.st_mode))
```

3.2.4. Sprawdzanie praw dostępu do pliku

W polu st_mode struktury stat przechowywane są również prawa dostępu do pliku, reprezentowane przez 9 bitów z tego pola. Sprawdzenia praw, podobnie jak dla typu pliku, dokonujemy przy użyciu makr. Makra określające prawa dostępu omówione zostały w teorii do Laboratorium 1. Mają one postać

S_Ipwww

Praw w takiej postaci używa się również jako maski dla pola st_mode. Przykład poniżej prezentuje sprawdzenie czy grupa ma prawo do zapisu **i** odczytu:

```
if ((buf.st_mode & (S_IRGRP | S_IWGRP)) == (S_IRGRP | S_IWGRP))
```

Natomiast kolejny przykład prezentuje sprawdzenie ma grupa ma prawo do zapisu **lub** odczytu.

```
if ((buf.st_mode & S_IRGRP) == S_IRGRP ||  (buf.st_mode & S_IWGRP) == S_IWGRP)
```

3.2.5. Inne bity pola st_mode

Pole st_mode zawiera także inne bity zawierające informacje o pliku, takie jak

```
S_ISUID /* set-user-ID on execution */
S_ISGID /* set-group-ID on execution */
S_ISVTX /* directory restricted-deletion */
```

4 Funkcje c.d.

4.1. getpwuid, getgrgid

Jak zobaczyliśmy, struktura stat zawiera numery grupy i użytkownika do których należy plik, są to pola:

```
struct stat {
  ...
  uid_t st_uid;   /* user ID */
  gid_t st_gid;     /* group ID */
  ...
}
```

Z naszego punktu widzenia ważne są nie tylko numery, ale również nazwy, numer niewiele nam mówi. Aby zobaczyć nazwy grup i użytkownika o danym numerze możemy użyć funkcji getgrgid i getpwuid. Nie są to funkcje systemowe, bo czytają one dane z plików /etc/passwd i /etc/group z których każdy proces może czytać. Aby z nich korzystać należy zawrzeć nagłówki grp.h dla grupy i pwd.h dla użytkownika.

```
#include <grp.h>
struct group *getgrgid(
  gid_t gid /* group ID */
); /* Returns pointer to structure or NULL on error (sets errno) */
```

```
#include <pwd.h>
struct passwd *getpwuid(
  uid_t uid /* user ID */
); /* Returns pointer to structure or NULL on error (sets errno) */
```

4.2. Struktury group i passwd

Podobnie jak w przypadku funkcji stat, dane z funkcji getgrgid i getpwuid są nam zwracane w postaci struktur.

```
struct group {
  char *gr_name; /* group name */
  gid_t gr_gid; /* group ID */
  char **gr_mem; /* member-name array (NULL terminated */
  };
```

```
struct passwd {
  char *pw_name; /* login name */
  uid_t pw_uid; /* user ID */
  gid_t pw_gid; /* group ID */
  char *pw_dir; /* login directory */
  char *pw_shell; /* login shell */
};
```

Przykładowy kod wypisujący użytkowników i grupy, na podstawie danych ze struktury stat:

```
static void print_owner(const struct stat *statp) {
  struct passwd *pwd = getpwuid(statp->st_uid);
  if (pwd == NULL)
    printf(" %-8ld", (long)statp->st_uid);
  else
    printf(" %-8s", pwd->pw_name);
}
static void print_group(const struct stat *statp) {
  struct group *grp = getgrgid(statp->st_gid);
    if (grp == NULL)
      printf(" %-8ld", (long)statp->st_gid);
    else
      printf(" %-8s", grp->gr_name); }
```

4.3. getlogin

To po prostu funkcja zwracająca login pod którym zalogował się użytkownik

```
#include <unistd.h>
char *getlogin(void); /* Returns name or NULL on error (sets errno) */
```

5 Czas

W Unix są używane są dwa typy czasu:

- czas kalendarzowy - używany do modyfikacji dat modyfikacji plików, dostępów itp.
- czas wykonania - używany do pomiaru czasu wykonania instrukcji

5.1. Czas kalendarzowy

Jest reprezentowany w jednej z postaci:

- jako zmienna time_t reprezentująca ilość sekund od powstania UNIX 1970 1 January.
- struktura timeval - przechowuje sekundy i mikrosekundy

```
struct timeval {
  time_t tv_sec; /* seconds */
  suseconds_t tv_usec; /* microseconds */
}
```

- struktura tm - przechowuje rok, miesiąc, dzień, godzinę, minutę

```
struct tm {
  int tm_sec; /* second [0,61] (up to 2 leap seconds) */
  int tm_min; /* minute [0,59] */
  int tm_hour; /* hour [0,23] */
  int tm_mday; /* day of month [1,31] */
  int tm_mon; /* month  [0,11] */
  int tm_year; /* years since 1900 */
  int tm_wday; /* day of week [0,6] (0 = Sunday) */
  int tm_yday; /* day of year [0,365] */
  int tm_isdst; /* daylight-savings flag */
};
```

- string, np. Tue Jul 23 09:44:17 2002.

5.2. time

To funkcja zwracająca liczbę sekund od 1970-01-01 00:00:00 +0000 (UTC) jako zmienną time_t.

```
#include <time.h>
time_t time(time_t *tloc);
```

Jeśli parametr jest ustawiony, to wartość zwracana jest zapisywana w przekazanej zmiennej.

5.3. difftime

Częstą potrzebą jest aby odjąć od siebie dwie reprezentacje czasu w postaci time_t, służy do tego funkcja difftime.

```
#include <time.h>
double difftime(
  time_t time1, /* time */
  time_t time0 /* time */
); /* Returns time1 - time0 in seconds (no error return) */
```

5.4. strftime

Konwertuje reprezentację czasu ze struktury tm na postać string.

```
#include <time.h>
size_t strftime(
  char *buf, /* output buffer */
  size_t bufsize, /* size of buffer */
  const char *format, /* format */
  const struct tm *tmbuf /* broken-down time */
); /* Returns byte count or 0 on error (errno not defined) */
```

5.5. Czas systemowy (system) vs czas użytkownika (user)

Czasami mamy do czynienie z pomiarem czasu wykonywania jakiegoś programu/polecenia. Wówczas w wyniku dostajemy czas podzielony na

- czas usera - czas spędzony na wykonywaniu instrukcji w procesie użytkownika
- czas systemowy - czas spędzony w jądrze systemu na wykonanie zadań tego procesu

6 Funkcje c.d.

6.1. readlink

Zapisuje zawartość linku symbolicznego o podanej ścieżce do przekazanego bufora.

```
#include <unistd.h>
ssize_t readlink(
  const char *path,   /* pathname */
  char *buf,      /* returned text */
  size_t bufsize    /* buffer size */
); /* Returns byte count or -1 on error (sets errno) */
```

7 Wywołanie zsynchronizowane vs synchroniczne

W systemie UNIX są wywołanie zsynchronizowane i synchroniczne to dwa różne typy wywołań funkcji systemowych.

- wywołanie zsynchronizowane - występuje gdy funkcje zapisujące takie jak write, pwrite itp. nie zwracają wartości dopóki dane nie są zapisane na fizycznym urządzeniu. Jak wiemy z laboratorium 1, defaultowe działanie write jest takie, że dane są trzymane w buforze po zapisaniu, a zapisywane na urządzeniu w dogodnej chwili. Gdyby w tym czasie system przestał działać, to dane byłyby stracone.
- wywołanie synchroniczne - jest to defaultowe działanie wywołań systemowych takich funkcji jak write, read itp. Polega na tym, że funkcja nie zawraca dopóki dane nie są dostępne lub przynajmniej zapisane w buforze.

8 Wywołania niezsynchronizowane vs asynchroniczne

Poprzez analogię do poprzedniej sekcji możemy powiedzieć, że wywołania:

- niezsynchronizowane - to takie dla których buffer cache jest wystarczające.
- asynchroniczne - to takie, które tylko inicjują wykonanie operacji i wracają zaraz po inicjacji, powodzenia wykonania jest natomiast testowane przez inne funkcje systemowe.

9 Asynchroniczne wejście-wyjście - AIO

Asynchroniczne wejście-wyjście może polepszyć wydajność funkcji read, zsynchronizowanej lub nie, a także polepszyć wydajność funkcji zsynchronizowanej write. Dzieje się tak dlatego, że możemy zainicjować działanie tych funkcji na początku, zrobić coś użytecznego, a potem odczytać gotowe już dane.

9.1. Struktura aiocb

Jest to struktura danych używana przez funkcje realizujące operacje asynchronicznego wejścia-wyjścia. Ma ona postać:

```
struct aiocb {
  int aio_fildes; /* file descriptor */
  off_t aio_offset; /* file offset */
  volatile void *aio_buf; /* buffer */
  size_t aio_nbytes; /* size of transfer */
  int aio_reqprio; /* request priority offset */
  struct sigevent aio_sigevent; /* signal information */
  int aio_lio_opcode; /* operation to be performed */
};
```

9.2. aioread, aiowrite

```
#include <aio.h>
int aio_read(
  struct aiocb *aiocbp /* control block */
); /* Returns 0 on success or -1 on error (sets errno) */
```

```
#include <aio.h>
int aio_write(
  struct aiocb *aiocbp /* control block */
); /* Returns 0 on success or -1 on error (sets errno) */
```

Wartość 0 oznacza dla tych funkcji jedynie, że procesy czytania/zapisu zostały zainicjowane, nie, że zostały pomyślnie przeprowadzone. Aby sprawdzić wykonanie funkcji należy użyć aio_error.

9.3. aio_error, aio_return

```
#include <aio.h>
int aio_error(
  const struct aiocb *aiocbp /* control block */
); /* Returns 0, errno value, or EINPROGRESS (does not set errno) */
```

aio_error zwraca 0 gdy pomyślnie zakończono wywołanie funkcji asynchronicznej, errno równoważne temu, które zwróciłaby funkcja wołana w przypadku błędu lub EINPROGRESS gdy funkcja jeszcze się nie zakończyła.
Gdy funkcja się zakończy możemy chcieć dostać wartość która byłaby zwrócona przez wywołanie(np. przeczytaną liczbę bajtów), służy do tego funkcja aio_return.

```
#include <aio.h>
ssize_t aio_return(
  struct aiocb *aiocbp /* control block */
); /* Returns operation return value or -1 on error (sets errno) */
```

9.4. aio_suspend

Funkcja ta pozwala zaczekać aż wywołania asynchroniczne dobiegnie końca. Np. gdy już zrobiliśmy inne rzeczy w programie i koniecznie potrzebujemy teraz zakończyć wywołanie asynchroniczne.

```
#include <aio.h>
int aio_suspend(
  const struct aiocb *const list[], /* array of control blocks */
  int cbcnt, /* number of elements in array */
  const struct timespec *timeout /* max time to wait */
); /* Returns 0 on success or -1 on error (sets errno) */
```

9.5. Flagi O_SYNC i O_DSYNC

- O_SYNC - Flaga powodująca, że po każdym wywołaniu write następuje wymuszenie zapisania danych na urządzenie fizyczne lub przynajmniej zaplanowania takiego zapisu
- O_DSYNC - Flaga działająca jak O_SYNC, ale jest trochę szybsza, ponieważ nie wymusza ona zmiany informacji związanych z plikiem od razu (takich jak np. czas modyfikacji pliku).

10 Auto-Test

- Czym są pliki zwykłe ?
- Czym są katalogi ?
- Czym są pliki specjalne i systemowe ?
- Jak realizowany jest odczyt danych z dysku ?
- Czym są I-węzły ? Jakie dane przechowują ?
- Co znajduje się w plikach etc/passwd i etc/group ?
- Czym są linki twarde, a czym linki miękkie ?
- Jak stworzyć link twardy, a jak link miękki ?
- Do czego służą funkcje stat, fstat, lstat ? Czym się różnią ?
- Czym jest struktura stat ?
- Czym jest pole mode_t st_mode struktury stat ?
- Jakie są dwa możliwe sposoby do sprawdzenia typu pliku przy użyciu pola mode_t struktury stat ?
- Jak odczytać typ pliku przy uzyciu S_IFMT ?
- Jak odczytać typ pliku przy użyciu makr testujących ?
- Jak sprawdzić prawa do pliku przy użyciu struktury stat ?
- Co robią funkcje getpwuid, getgrgid ?
- Czym są struktury group i passwd ?
- Czym jest czas kalendarzowy, a czym czas wykonania ?
- Jak można reprezentować czas kalendarzowy ?
- Co robi funkcja time/difftime/strftime ?
- Czym jest czas systemowy, a czym czas użytkownika ?
- Co robi funkcja readlink ?
- Czym są wywołania asynchroniczne funkcji systemowyh ?
- Czym jest struktura aiocb ?
- Do czego służą funkcje aioread, aiowrite ?
- Do czego służą funkcję aio_error, aio_return ?
- Do czego służy funkcja aio_suspend ?

11 Inne pytania z wejściówek

- Jak za pomocą lseek odczytać aktualny wskaźnik w pliku?
- Czy można utworzyć dowiązanie twarde do pliku znajdującego się na innym urządzeniu Odp Nie
- Jakie typy plików nie pozwalają na utworzenie wskaźnika(?), wymienić jeden Odp. gniazda, kolejki, potoki.
- Jak sprawdzić czy dwa dowiązania twarde wskazują na dokładnie ten sam plik? Odp. ls -i
- Jak za pomocą lseek odczytać aktualny wskaźnik w pliku? - Odp. Wywołać z flagą SEEK_CUR, lseek(fd, 0, SEEK_CUR)
- Odczytać za pomocą lseek wielkość pliku. int size = lseek(fd, 0, SEEK_END);
- Odczytać wielkość pliku przy pomocy stat struct stat stbuf; stat(av[1], stbuf); printf ("%lld", stbuf.st_size);