Wybrane funkcje systemowe Unixa

Witold Paluszyński
witold.paluszynski@pwr.wroc.pl
http://sequoia.ict.pwr.wroc.pl/~witold/

Niniejszy dokument zawiera materiały do wykładu na temat wybranych funkcji systemowych Unixa. Jest on udostępniony pod warunkiem wykorzystania wyłącznie do własnych, prywatnych potrzeb i może być kopiowany wyłącznie w całości, razem z niniejszą stroną tytułową.

Informacje o pliku: `inode`

System Unix trzyma informacje o plikach i katalogach w strukturach zwanych *i-node*'ami (i-węzłami???). Można uzyskać do nich dostęp poprzez strukturę `stat`, której elementami program może manipulować.

```
struct  stat {
  dev_t     st_dev;      /* ID of device containing */
                         /* a directory entry for this file */
  dev_t     st_rdev;     /* ID of device */
                         /* This entry is defined only for */
                         /* char special or block special files */
  ino_t     st_ino;      /* Inode number */
  mode_t    st_mode;     /* File mode (see mknod(2)) */
  nlink_t   st_nlink;    /* Number of links to file */
  uid_t     st_uid;      /* User ID of the file's owner */
  gid_t     st_gid;      /* Group ID of the file's group */
  off_t     st_size;     /* File size in bytes */
  time_t    st_atime;    /* Time of last access */
  time_t    st_mtime;    /* Time of last data modification */
  time_t    st_ctime;    /* Time of last file status change */
                         /* Times measured in seconds since */
                         /* 00:00:00 UTC, Jan. 1, 1970 */
  long      st_blksize;  /* Preferred I/O block size */
```



```
  blkcnt_t st_blocks;   /* Number of 512 byte blocks allocated*/
  ...
};


int stat(const char *path, struct stat *buf);
int lstat(const char *path, struct stat *buf);
int fstat(int fildes, struct stat *buf);
```

Inne operacje na plikach

rename — zmienia nazwę pliku

chmod — zmienia prawa dostępu do pliku

umask — ustawia maskę praw dostępu nowo tworzonych plików

access — sprawdza prawa dostępu procesu do pliku

link — tworzy nowy link (twardy) do pliku

symlink — tworzy nowy link symboliczny do pliku

unlink — kasuje plik (lub link symboliczny)

rmdir — kasuje kartotekę (musi być pusta)

remove — kasuje plik lub pustą kartotekę

mkdir — tworzy kartotekę

chdir — przełącza się do innej kartoteki

dup — duplikuje deskryptor otwartego pliku

fcntl — pozwala wykonać wiele różnych operacji na otwartych plikach

Czytanie zawartości kartotek

```
struct  dirent {
        ino_t           d_ino;
        off_t           d_off;
        unsigned short  d_reclen;
        char            d_name[1];
};

DIR *opendir(const char *filename);
struct dirent *readdir(DIR *dirp);
int closedir(DIR *dirp);


#include <stdio.h>
#include <dirent.h>

DIR *dirp;
struct dirent *direntp;

dirp = opendir( "." );
while ( (direntp = readdir( dirp )) != NULL )
  (void)printf( "%s\n", direntp->d_name );
(void)closedir( dirp );
```


Przykład: wyświetlanie struktury kartotek

```
#include <unistd.h>
#include <stdio.h>
#include <dirent.h>
#include <string.h>
#include <sys/stat.h>

void printdir(char *dir, int depth) {
  DIR *dp;
  struct dirent *entry;
  struct stat statbuf;

  if((dp = opendir(dir)) == NULL) {
    fprintf(stderr,"cannot open directory: %s\n", dir);
    return;
  }
  chdir(dir);
  while((entry = readdir(dp)) != NULL) {
    stat(entry->d_name,&statbuf);
    if(S_ISDIR(statbuf.st_mode)) {
      /* Found a directory, but ignore . and .. */
      if(strcmp(".",entry->d_name) == 0 ||
         strcmp("..",entry->d_name) == 0)
           continue;
```



```
      printf("%*s%s/\n",depth,"",entry->d_name);
      /* Recurse at a new indent level */
      printdir(entry->d_name,depth+4);
    }
    else printf("%*s%s\n",depth,"",entry->d_name);
  }
  chdir("..");
  closedir(dp);
}
```

Funkcje czasu zegarowego

Unixowe funkcje czasu zegarowego obsługują czas od godziny 0:00 1 stycznia 1970 do godziny 04:14:07 19 stycznia 2038.[1]

```
#include <sys/types.h>
#include <time.h>

time_t time(time_t *tloc);
```

Funkcja `time` — zwraca czas bieżący w postaci liczby sekund, jakie upłynęły od godziny 0:00 dnia 1 stycznia 1970.

[1]Do odliczania czasu użyta jest wartość long (ze znakiem), co wydaje się marnotrawstwem jednego cennego bitu (użycie liczby bez znaku wydłużyłoby życie Unixa do roku 2106). Zatem w roku 2038 można oczekiwać problemów z czasem w systemach komputerowych na platformie Unixa podobnych do tych, które występowały na innych platformach na początku roku 2000. W rzeczywistości różne problemy z czasem zaczęły się już pojawiać. 10 stycznia 2004 o godzinie 14:37:04 minęła połowa okresu „życia" Unixa, co zresztą właśnie dzięki zastosowaniu liczby ze znakiem pozwoliło uniknąć możliwych błędów, zawinionych przez programistów, którzy by o tym znaku zapomnieli. Ale np. 12 maja 2006 pojawiły się raporty o wielu „zwisach" baz danych, które nastąpiły dokładnie jeden miliard sekund przed feralną datą 2038 roku. Okazało się, że w niektórych serwerach ustawione były tak długie time-outy na transakcje, i programy sprawdzające datę po tym okresie nie mogły sobie poradzić z otrzymanymi wynikami...

Funkcje czasu — obliczenia kalendarzowe

```
struct tm *localtime(const time_t *clock);
time_t mktime(struct tm *timeptr);

struct  tm {
   int  tm_sec;    /* seconds after the minute - [0, 61] */
                   /* for leap seconds */
   int  tm_min;    /* minutes after the hour - [0, 59] */
   int  tm_hour;   /* hour since midnight - [0, 23] */
   int  tm_mday;   /* day of the month - [1, 31] */
   int  tm_mon;    /* months since January - [0, 11] */
   int  tm_year;   /* years since 1900 */
   int  tm_wday;   /* days since Sunday - [0, 6] */
   int  tm_yday;   /* days since January 1 - [0, 365] */
   int  tm_isdst;  /* flag for alternate daylight savings time */
};
```

Funkcja `localtime` tworzy i wypełnia strukturę `tm`, która daje dostęp do elementów aktualnego czasu. Brana jest pod uwagę lokalna strefa czasowa, czas letni/zimowy, lata przestępne, a nawet sekundy przestępne.[2]

Funkcja `mktime` zamienia strukturę czasową `tm` na liczbę sekund jak w funkcji `time`, dodatkowo kompletując i normalizując pola w strukturze, które mogą być wypełnione częściowo, lub poza zakresem (np. $\texttt{tm\_hour} < 0$ lub > 23).

[2]Ostatnie sekundy przestępne, określane przez organizację IERS (International Earth Rotation Service), wystąpiły (równocześnie na całym świecie): 1998-12-31 23:59:60Z, 2005-12-31 23:59:60Z, 2008-12-31 23:59:60Z...

Funkcje czasu — napisy sformatowane

```
char *ctime(const time_t *clock);
```

Funkcja `ctime` — tworzy zapis daty i czasu w postaci napisu o ustalonym 26-znakowym formacie: `"Thu Nov 23 11:04:20 2000\n\0"`. Wyświetlany jest czas lokalny, i napis ten nie podlega żadnym, lokalizacjom, modyfikacjom, ani konwencjom.

Istnieją również funkcje do tworzenia dowolnie sformatowanych napisów czasowych (patrz `strftime`).

Funkcje czasu — przykład

```
#include <stdio.h> /* Przyklad: obliczanie numeru dnia w roku*/
#include <time.h>  /* dzien, miesiac, rok bierzemy z argv[]  */
#include <errno.h> /* jesli brak to bierzemy z dnia biezacego*/

int wiek21 = 0;    time_t sekczas;    struct tm strczas;
char *const wday[] = {"Nie","Pon","Wto","Sro","Czw","Pia","Sob"};

sekczas= time(NULL);           /* w sekundach od 1970-01-01  */
strczas= *localtime(&sekczas);

if (argc > 1) sscanf(argv[1], "%d", &strczas.tm_mday);
if (argc > 2) {
  sscanf(argv[2], "%d", &strczas.tm_mon);
  strczas.tm_mon--;            /* musi byc w przedziale[0,11]*/
}
if (strczas.tm_year >= 100)    /* mamy XXI wiek, fajnie jest */
  wiek21 = 1;
if (argc > 3) {
  sscanf(argv[3], "%d", &strczas.tm_year);
  if (strczas.tm_year < 100)   /* mamy rok wzgledem stulecia */
    if (wiek21==1)
      strczas.tm_year += 100;
  if (strczas.tm_year >= 1900) /* uzytkownik podal pelny rok */
    strczas.tm_year -= 1900;
}
mktime(&strczas);
printf("Numer dnia %.4d-%.2d-%.2d ---> %3d (%s)%s\n",
    strczas.tm_year + 1900, strczas.tm_mon + 1,
    strczas.tm_mday, strczas.tm_yday + 1,
    wday[strczas.tm_wday],
    (strczas.tm_isdst) > 0 ? " CZAS LETNI" : "");
```

Funkcje czasu procesów

```
#include <sys/times.h>
#include <limits.h>

clock_t times(struct tms *buf);

struct tms {
   clock_t tms_utime;     /* user time */
   clock_t tms_stime;     /* system time */
   clock_t tms_cutime;    /* user time, children */
   clock_t tms_cstime;    /* system time, children */
};
```

- funkcja `times` zwraca czas zegarowy, jaki upłynął od arbitralnie ustalonego momentu w czasie (może to być np. moment startu systemu); jednostką jest tzw. *tick*, którego liczbę na sekundę określa makro clk_tck (przykładowo 50, 60, albo 100)

- podana struktura jest wypełniana przez funkcję `times` wartościami czasu procesora zużytego przez proces i jego podprocesy, które już się zakończyły i

zostały poprawnie obsłużone funkcją `wait`; wartości czasu są podobnie liczone od arbitralnego momentu i podane w tych samych jednostkach

- czas procesora zużyty przez proces liczony jest w rozbiciu na tzw. czas użytkownika, czyli instrukcje programu, i czas systemu, tzn. operacje jądra Unixa: funkcje systemowe i operacje pomocnicze