Użytkownik i zasoby

Użytkownik - identyfikatory

Rzeczywisty UID Rzeczywisty GID	*Kim jestem*? Ustawiane w momencie otwierania sesji przez użytkownika na podstawie pliku `/etc/passwd`. Obowiązują całą sesję. Może je zmienić jedynie superużytkownik.
Efektywny UID Efektywny GID	*Jakie mam prawa dostępu do plików*? Początkowo są takie same, jak rzeczywisty UID i rzeczywisty GID. Można je zmienić podczas sesji.

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

`uid_t getuid(void)`	rzeczywisty identyfikator właściciela procesu (UID)
`gid_t getgid(void)`	rzeczywisty identyfikator grupy procesu (GID)
`uid_t geteuid(void)`	obowiązujący identyfikator właściciela procesu (EUID) –
`gid_t getegid(void)`	obowiązujący identyfikator grupy procesu (EGID)

- Przykład:

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main() {
    printf("Real user ID: %d\n", getuid());
    printf("Effective user ID: %d\n", geteuid());
    printf("Real group ID: %d\n", getgid());
    printf("Effective group ID: %d\n", getegid());
    exit(0);
}
```

```
$ ./ids
Real user ID: 1260
Effective user ID: 1260
Real group ID: 101
Effective group ID: 101
```

Zmieńmy prawa dostępu (trzeba mieć odpowiednie uprawnienia):

```
$ ll ids
-rwsr-xr-x  1 root others 5363 Oct 13 15:41 ids
```

```
$ ./ids
Real user ID: 1260
Effective user ID: 0
Real group ID: 101
Effective group ID: 101
```

Użytkownik – nazwa

```
#include <unistd.h>
char *getlogin(void);
```

- Funkcja zwraca wskaźnik do nazwy właściciela procesu, lub `NULL` jeśli ta informacja nie jest dostępna.

- Przykład:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

int main() {
  char *login;

/* Pobierz nazwe wlasciciela procesu*/
  if((login = getlogin()) == NULL) {
    printf("Not in /var/run/utmp ?\n");
    perror("getlogin");
    exit(1);
  }

  printf("Login name = %s\n",login);
  exit(0);
}

$ ./program
Login name = adam
```

Baza użytkowników

Użytkownicy opisani są w pliku /etc/passwd.

Funkcje związane z bazą użytkowników

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

- Struktura opisująca użytkownika

```
struct passwd {
  char *pw_name;      /* user name */
  char *pw_passwd;    /* user password */
  uid_t pw_uid;        /* user id */
  gid_t pw_gid;        /* group id */
  char *pw_gecos;     /* real name */
  char *pw_dir;       /* home directory */
  char *pw_shell;     /* shell program */
};
```

- Funkcje obsługi

```
struct passwd *getpwnam(const char *name);
struct passwd *getpwuid(uid_t uid);

struct passwd *getpwent(void);
void setpwent(void);
void endpwent(void);
```

- Przykład: wykaz wszystkich kont w systemie

```
#include <stdio.h>
#include <pwd.h>

int main (void)
{
  struct passwd *pwd;
  setpwent ();
  while ((pwd = getpwent()) != NULL) {
    if (*pwd -> pw_gecos != '\0') {
      printf ("\"%s\" is %s (%ld, %ld)\n",
      pwd -> pw_gecos, pwd -> pw_name,
      (long) pwd -> pw_uid, (long) pwd -> pw_gid);
      }
    else {
      printf ("\"%s\" is %s (%ld, %ld)\n",
      pwd -> pw_name, pwd -> pw_name,
      (long) pwd -> pw_uid, (long) pwd -> pw_gid);
    }
  }
  endpwent();

  return 0;
}
```

- Przykład: wyświetlenie informacji o właścicielu wykonywanego programu

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <pwd.h>

int main()
{
  char *login;
  struct passwd *opis;

/* Pobierz nazwe wlasciciela procesu*/
  if((login = getlogin()) == NULL) {
    perror("getlogin");
    exit(EXIT_FAILURE);
  }

/* Odszukaj opis w /etc/passwd */
  if((opis= getpwnam(login)) == NULL) {
    perror("getpwnam");
    exit(EXIT_FAILURE);
  }

/* Wyswietl informacje o uzytkowniku */
  printf("user name: %s\n", opis->pw_name);
  printf("UID      : %d\n", opis->pw_uid);
  printf("GID      : %d\n", opis->pw_gid);
  printf("gecos    : %s\n", opis->pw_gecos);
  printf("home dir : %s\n", opis->pw_dir);
  printf("shell    : %s\n", opis->pw_shell);
  exit(EXIT_SUCCESS);
}
```

Wynik:

```
user name: adam
UID      : 1260
GID      : 101
gecos    : Adam Malicki
home dir : /home/staff/adam
shell    : /bin/bash
```

- Przykład: wyświetlenie informacji o wybranym użytkowniku

```
#include <stdio.h>
#include <pwd.h>

int main (int argc, char **argv)
{
  struct passwd *pwd;
  int i;

  for (i = 1; i < argc; i++) {
    if ((pwd = getpwnam (argv [i])) == NULL)
      printf ("%s: No such user\n", argv [i]);
    else {
      printf ("User name: %s\n", pwd -> pw_name);
      printf ("User ID: %ld\n", (long) pwd -> pw_uid);
      printf ("Group ID: %ld\n", (long) pwd -> pw_gid);
      printf ("GECOS: %s\n", pwd -> pw_gecos);
      printf ("Home directory: %s\n", pwd -> pw_dir);
      printf ("Login shell: %s\n", pwd -> pw_shell);
      printf ("\n");
    }
  }
  return (0);
}
```

```
$ ./program kowalski nowak

User name: kowalski
User ID: 8359
Group ID: 100
GECOS: Jan Kowalski
Home directory: /home/informatyka/2004/ID/k/kowalski
Login shell: /bin/bash

User name: nowak
User ID: 8647
Group ID: 100
GECOS: Piotr Nowak
Home directory: /home/informatyka/2005/IZ/n/nowak
Login shell: /bin/bash
```

Baza grup

Grupy opisane są w pliku /etc/group.

Funkcje związane z bazą grup

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

- Struktura opisująca grupę

```
struct group {
  char *gr_name;      /* group name */
  char *gr_passwd;    /* group password */
  gid_t gr_gid;       /* group id */
  char **gr_mem;      /* group members */
};
```

- Funkcje obsługi

```
struct group *getgrnam(const char *name);
struct group *getgrgid(gid_t gid);

struct group *getgrent(void);
void setgrent(void);
void endgrent(void);
```

Informacja o systemie

- Informacja o systemie operacyjnym

```
#include <sys/utsname.h>
int uname(struct utsname *buf);

struct utsname {
  char sysname[];
  char nodename[];
  char release[];
  char version[];
  char machine[];
  #ifdef _GNU_SOURCE
    char domainname[];
  #endif
};
```

- Przykład:

```
#include <stdio.h>
#include <sys/utsname.h>

int main ()
{
  struct utsname utsname;

  if (uname (&utsname) == -1)
    err_msg ("uname failed");  // własna funkcja
  printf ("Info from uname:\n");
  printf ("  sysname: %s\n", utsname.sysname);
  printf ("  nodename: %s\n", utsname.nodename);
  printf ("  release: %s\n", utsname.release);
  printf ("  version: %s\n", utsname.version);
  printf ("  machine: %s\n\n", utsname.machine);
  return (0);
}
```

Uzyskane informacje:

```
Info from uname:
  sysname: Linux
  nodename: gift.wsisiz.edu.pl
  release: 2.6.13-rc3
  version: #2 SMP Tue Aug 23 11:51:29 CEST 2005
  machine: i686
```

To samo z użyciem polecenia `uname -a`:

```
$ uname -a
Linux gift.wsisiz.edu.pl 2.6.13-rc3 #2 SMP Tue Aug 23 11:51:29 CEST
2005 i686 i686 i386 GNU/Linux
```

- Nazwa hosta

```
#include <unistd.h>
int gethostname(char *name, size_t len);

char name[20];
gethostname(name, sizeof(name));
printf("%s\n",name);
```

- Identyfikator maszyny

```
#include <unistd.h>
long gethostid(void);
```

Funkcja zwraca 32 bitowy identyfikator hosta.

- Informacje statystyczne o systemie

```
#include <sys/systeminfo.h>
int sysinfo(struct sysinfo *info);

struct sysinfo {
  long uptime;             /* ilość sekund od startu systemu */
  unsigned long loads[3];  /* średnie obciążenie w ciągu 1, 5 i
                              15min.*/
  unsigned long totalram;  /* ilość pamięci */
  unsigned long freeram;   /* ilość wolnej pamięci */
  unsigned long sharedram; /* ilość pamięci wspólnej */
  unsigned long bufferram; /* pamięć wykorzystywana przez bufory */
  unsigned long totalswap; /* ilość pamięci wymiany */
  unsigned long freeswap;  /* ilość wolnej pamięci wymiany */
  unsigned short procs;    /* ilość procesów */
  unsigned long totalhigh; /* ilość pamięci wysokiej */
  unsigned long freehigh;  /* ilość wolnej pamięci wysokiej */
  unsigned int mem_unit;   /* wielkość jednostki pamięci w bajtach */
};
```

Funkcja kopiuje informacje o systemie do struktury `info`.
UWAGA: funkcja nie jest kompatybilna z innymi Uniksami!

Średnie obciążenie systemu

```
#include <stdlib.h>
int getloadavg(double loadavg[], int nelem);
```

Średnie obciążenie jest to liczba procesów czekających na uruchomienie i uruchomionych (obciążenie) w określonych odcinkach czasu (1, 5, 15 minut).

```
#include <stdio.h>
#include <stdlib.h>
int main (void)
{
  double load_av [3];

  if (getloadavg (load_av, 3) == -1){
    perror("getloadavg failed");
    exit(1);
  }

  printf ("Load averaged over 1 minute: %.2f\n", load_av [0]);
  printf ("Load averaged over 5 minutes: %.2f\n", load_av [1]);
  printf ("Load averaged over 15 minutes: %.2f\n", load_av [2]);

  return 0;
}
```

```
$ ./program
Load averaged over 1 minute: 3.62
Load averaged over 5 minutes: 3.78
Load averaged over 15 minutes: 3.83

$ uptime
21:26:18 up 18 days,  4:16, 16 users,  load average: 3.65, 3.78, 3.83
```

Zmienne środowiskowe

Wyświetlanie wszystkich zmiennych środowiskowych

Tradycyjne podejście:

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[], char *envp[])
{
  while(*envp)
    printf("%s\n", *envp++);
  exit(0);
}
```

Zalecane przez POSIX.1:

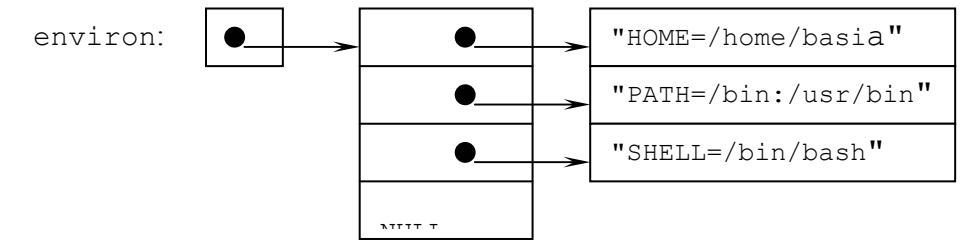
```
#include <stdio.h>
#include <stdlib.h>

extern char **environ;  // Nie modyfikuj tej zmiennej!

int main()
{
  char **envp;
  envp=environ;
  while(*envp)
    printf("%s\n", *envp++);
  exit(0);
}
```

lub

```
for (env=environ; *env != NULL;
++ env)
  printf("%s\n", *envp++);
```



Wynik działania programu:

```
HOSTNAME=gift.wsisiz.edu.pl
TERM=xterm
SHELL=/bin/bash
HISTSIZE=1000
PATH=/usr/kerberos/bin:/usr/java/jdk1.5.0_01/bin:/usr/local/bin:/bin:/
usr/bin:/usr/X11R6/bin:/home/oracle/ORCL/bin:/home/staff/bozena/bin
...
```

Działania na pojedynczych zmiennych środowiskowych

```
#include <stdlib.h>
// pobierz wartość zmiennej środowiskowej name
char *getenv(const char *name);
// zmodyfikuj wartość zmiennej środowiskowej
int putenv(const char *str);
// zmodyfikuj wartość zmiennej środowiskowej
int setenv(const char name, const char *value, int overwrite);
// usuń definicję zmiennej środowiskowej
void unsetenv(const char *name);
```

- Przykład:

```
#include <stdio.h>
#include <stdlib.h>
int main () {
  char* sciezka = getenv("PATH");
  if (sciezka != NULL)
    printf ("sciezka dostepu do plikow %s\n", sciezka);
  return 0;
}
```

Wykonanie:

```
$ ./sciezka
sciezka dostepu do plikow
/usr/kerberos/bin:/usr/java/jdk1.5.0_01/bin:/usr/local/bin:/bin:/usr/
bin:/usr/X11R6/bin:/home/oracle/ORCL/bin:/home/staff/bozena/bin

$ echo $PATH
/usr/kerberos/bin:/usr/java/jdk1.5.0_01/bin:/usr/local/bin:
/bin:/usr/bin:/usr/X11R6/bin:/home/oracle/ORCL/bin
```

Przykłady:

```
putenv("TERM=dumb");

setenv("TERM","dumb",0); // tylko o ile zmienna nie istnieje
setenv("TERM","dumb",1); // utwórz lub zmień

unsetenv("MOJA");
```

Daty i czas

- Rodzaje czasu:
  - czas kalendarzowy
  - czas wykonywania

Czas kalendarzowy

- Uzyskanie czasu w systemie

```
#include <time.h>

time_t time(time_t *t);
```

Funkcja **`time`** - zwraca czas w postaci liczby sekund od początku epoki Uniksa czyli od 1 stycznia 1970 roku UTC oraz jeśli argument jest różny od NULL zapisuje go w argumencie.

```
int gettimeofday(struct timeval *tv, struct timezone *tz);
struct timeval {
  int tv_sec;  /* sekundy */
  int tv_usec; /* mikrosekundy */
};
struct timezone {
  int tz_minutewest; /* minuty na zachód od Greenwich */
  int tz_dsttime;    /* typ poprawki dla czasu letniego */
};
```

Funkcja **`gettimeofday`** - zwraca czas w postaci liczby sekund i mikrosekund od początku epoki w strukturze `timeval` i `timezone`.

```
double difftime(time_t time1, time_t time0);
```

Funkcja `difftime` zwraca różnicę między dwoma czasami

- Konwersja na postać czytelną dla użytkownika
  - Funkcje formatujące na podstawie liczby sekund

```
#include <time.h>
char *ctime(const time_t *timep);
```

Funkcja pobiera jako argument czas w sekundach od 1 stycznia 1970 i zwraca wskaźnik do bufora zawierającego czas podany w postaci napisu zakończonego znakiem nowego wiersza (bufor statyczny).

Inne funkcje konwersji tej grupy:

```
#include <time.h>
struct tm *localtime(const time_t *t);  // czas lokalny
struct tm *gmtime(const time_t *t);     // czas UTC
```

`struct tm` – czas w latach, miesiącach, dniach, godzinach, ..

  - Inne funkcje formatujące

```
char *asctime(const struct tm *tm);
```

Funkcja działa podobnie jak `ctime`, ale dla struktury tm

```
size_t strftime(char *s, size_t max, const char *format,
                const struct tm *tm);
```

Funkcja formatuje czas podany w postaci struktury `tm` zgodnie z podanym formatem

- Przykład 1:

```
#include <stdio.h>
#include <time.h>
int main()
{
  time_t czas;
  time(&czas);  // lub czas=time(NULL);
  printf("Data: %s",ctime(&czas));
  return 0;
}

$ ./program
Data: Tue Oct 4 19:16:09 2005
```

- Przykład 2:

```
#include <stdio.h>
#include <sys/time.h>
#include <time.h>
#include <unistd.h>

int main ()
{
  struct timeval tv;
  struct tm* ptm;
  char time_string[40];
  long milliseconds;

  gettimeofday (&tv, NULL);
  czas = localtime (&tv.tv_sec);
  strftime(time_string, sizeof(time_string), "%Y-%m-%d %H:%M:%S", czas);
  milliseconds = tv.tv_usec / 1000;
  printf ("%s.%03ld\n", time_string, milliseconds);
}

$ ./program
2005-10-04 19:20:28.996
```

- Przykład 3:

```
#include <stdio.h>
#include <time.h>
int main (void) {
  struct tm *tp;
  time_t login;
  time_t logout;
  time_t session_length;

  login = 100000;
  logout = 200000;
  session_length = (time_t) difftime (logout, login);
  tp = gmtime (&session_length);
  printf ("Session length is %d days, %d hours, %d minutes, "
          "and %d seconds\n", tp -> tm_yday, tp -> tm_hour,
           tp -> tm_min,
          tp -> tm_sec);
  return (0);
}
```

Czas wykonywania

- Funkcja zwraca czas zegarowy (ang. *wall clock time*), który upłynął od pewnego momentu w przeszłości (najczęściej podniesienia systemu). Jest on liczony w *taktach zegara* (ang. *ticks*). Funkcja w przypadku błędu zwraca -1. Dodatkowo wypełnia strukturę `tms` z czasami związanymi z bieżącym procesem.

```
#include <sys/times.h>
clock_t times(struct tms *buf)

struct tms {
  clock_t tms_utime;  /* user time – czas użytkownika*/
  clock_t tms_stime;  /* system time – czas systemowy*/
  clock_t tms_cutime; /* user time of children */
  clock_t tms_cstime; /* system time of children */
};
```

- Liczba taktów na sekundę:

```
#include <stdio.h>
#include <unistd.h>
int main() {
  long tps = sysconf(_SC_CLK_TCK);
  printf("%s: %ld\n", "liczba taktow na sek",tps);
  return 0;
}

$ ./program
liczba taktow na sek: 100
```

- Przykład

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/times.h>
#include <time.h>
#include <unistd.h>

void doit(char *, clock_t);
int main(void) {
  clock_t start, end;
  struct tms t_start, t_end;

  start = times(&t_start);
  system("grep errno /usr/include/*/* > /dev/null 2> /dev/null");
  end = times(&t_end);

  doit("elapsed", end - start);

  printf("parent times:\n");
  doit("\tuser CPU", t_end.tms_utime);
  doit("\tsys  CPU", t_end.tms_stime);
  printf("child times:\n");
  doit("\tuser CPU", t_end.tms_cutime);
  doit("\tsys  CPU", t_end.tms_cstime);

  exit(EXIT_SUCCESS);
}
void doit(char *str, clock_t time)
{
  long tps = sysconf(_SC_CLK_TCK);
  printf("%s: %6.2f secs\n", str, (double)time/tps);
}
```

Wykonanie:

```
elapsed:   5.74 secs
parent times:
        user CPU:   0.00 secs
        sys  CPU:   0.00 secs
child times:
        user CPU:   0.15 secs
        sys  CPU:   0.23 secs
```