1 Użytkownik

1.1 Użytkownik – identyfikatory

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

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

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.

- 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
```

1.2 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
```

1.3 Baza 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);`
  - zwraca wskaźnik do struktury zawierającej dane użytkownika o wskazanej nazwie

`struct passwd *getpwuid(uid_t uid);`
  - zwraca wskaźnik do struktury zawierającej dane użytkownika o wskazanym identyfikatorze

`struct passwd *getpwent(void);`
  - zwraca wskaźnik do struktury zawierającej opis kolejnego użytkownika w bazie użytkowników (kolejne wywołania zwracają opisy kolejnych użytkowników)

`void setpwent(void);`
  - przewija bazę użytkowników do początku

`void endpwent(void);`
  - zamyka bazę użytkowników po zakończeniu przetwarzania

- 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_name,
                  pwd -> pw_name, (long) pwd -> pw_uid, (long) pwd ->
                  pw_gid);
            }
            else {
                  printf ("\"%s\" is %s (%ld, %ld)\n", pwd -> pw_gecos,
                    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
```

1.4 Baza 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);
```

2 Zasoby

2.1 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!

2.2 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
```

2.3 System plików /proc

- Wirtualny system plików, który zawiera informacje o jądrze, jego strukturach, stanie wykonywanych procesów.
- Polecenia:
  - `procinfo` – ogólne informacje o systemie
  - polecenia czytania plików tekstowych: `cat`, `less`, ...
- Struktura katalogu `/proc`

```
$ ls -F /proc
1/     1222/       18/         22003/      4110/       978/        ide/        partitions
10/    1223/       1809/       22004/      4111/       997/        interrupts  pci
1011/  1224/       19/         22005/      4112/       buddyinfo   iomem       scsi/
1020/  1225/       19714/      22006/      4113/       bus/        ioports     self@
1057/  1226/       2/          22007/      4114/       cmdline     irq/        slabinfo
1060/  13/         20/         22008/      4132/       config.gz   kallsyms    stat
1061/  13682/      21/         23376/      5/          cpuinfo     kcore       swaps
11/    13683/      21217/      3/          6/          crypto      kmsg        sys/
1107/  13715/      21218/      3631/       7/          devices     loadavg     sysrq-
 trigger
1128/  13717/      21222/      3632/       761/        diskstats   locks       sysvipc/
1146/  13720/      21223/      3633/       762/        dma         meminfo     tty/
1154/  13910/      21232/      3634/       8/          dri/        misc        uptime
1174/  14/         21233/      4/          9/          driver/     modules     version
1183/  15/         22000/      4107/       954/        execdomains mounts@     vmstat
12/    16/         22001/      4108/       958/        filesystems mtrr        zoneinfo
1206/  17/         22002/      4109/       968/        fs/         net/
```

- Katalog `self` jest dowiązaniem identyfikatora bieżącego procesu. Alternatywne sposoby uzyskiwania informacji o identyfikatorze PID bieżącego procesu:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
const int size = 20;
int main( ){
   pid_t proc_PID, get_PID;
   char buffer[size];
   get_PID = getpid( );
   readlink("/proc/self", buffer, size);
   proc_PID = atoi(buffer);
   printf("getpid %d\n", get_PID);
   printf("/proc/self : %d\n" , proc_PID );
   return 0;
}

$./program
getpid 1234
/proc/self : 1234
```

- Każdy proces posiada swój katalog, o nazwie odpowiadającej identyfikatorowi procesu.
- Przykład 1:

```
$ ps
      PID   TTY    TIME       CMD
      1061 pts/2   00:00:00   bash
      1247 pts/2   00:00:00   ps

$ ls -F /proc/1061
attr/       cwd@        fd/   mounts      root@       statm wchan
auxv        environ     maps  oom_adj     seccomp     status
cmdline     exe@        mem   oom_score   stat        task/

$ cat /proc/1061/environ | tr "\0" " "
USER=adam LOGNAME=adam HOME=/home/staff/adam
PATH=/usr/local/bin:/bin:/usr/bin MAIL=/var/mail/adam SHELL=/bin/bash
SSH_CLIENT=::ffff:213.135.34.107 2600 22
SSH_CONNECTION=::ffff:213.135.34.107 2600 ::ffff:213.135.44.46 22
SSH_TTY=/dev/pts/2 TERM=xterm

$ ls -l /proc/1061/exe
lrwxrwxrwx 1 adam staff 0 paź 04 10:04 /proc/1061/exe -> /bin/bash
```

- Przykład 2: W pliku `cmdline` znajdują się argumenty przesłane do procesu. Są one pamiętane jako ciąg znaków, w którym każdy argument zakończony jest znakiem \0.

```
$ cat /proc/1459/cmdline | tr "\0" " "
./program plik1 plik2
```

Alternatywny sposób pobierania argumentów:

```
#include <sys/types.h>
#include <unistd.h>
#include <string.h>
const int size = 512;
int main( ){
      char name[255];
      char buffer[size];
      FILE* file;
      sprintf(name,"/proc/%d/cmdline", getpid( ));
      printf("--Reading from file: %s--\n",name);

      file=fopen(name,"r");
      fgets(buffer,size,file);

      char *p = &buffer[0];
      do {
            printf( "[%s]\n", p );
            p += strlen(p)+1;
      } while ( *p );
      return 0;
}

$ ./program prog1 prog2 prog3
--Reading from file: /proc/1367/cmdline--
[./program]
[prog1]
[prog2]
[prog3]
```

2.4 Zmienne środowiskowe

- Pobranie 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:

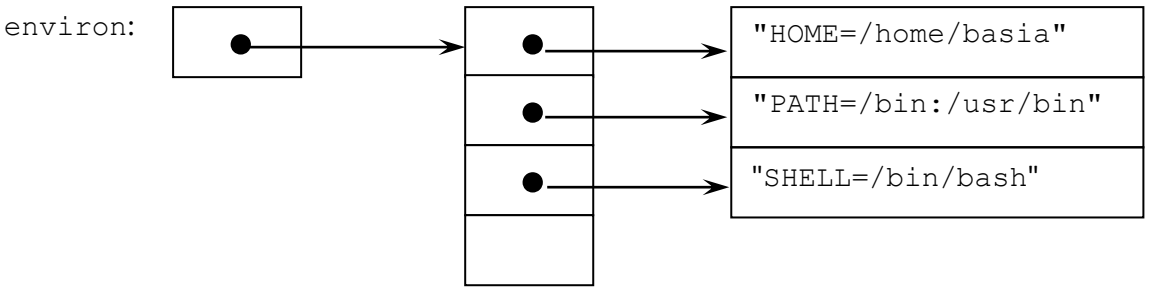
```
#include <stdio.h>
#include <stdlib.h>

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

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

lub

```
int main()
{
      char **env;
      env=environ;
      for (env=environ; *env != NULL; ++ env)
            printf("%s\n", *env++);
      exit(0);
     }
```



- Funkcje dotyczące pojedynczych zmiennych środowiskowych:

```
#include <stdlib.h>
char *getenv(const char *name);
```

  - Zwraca wartość zmiennej środowiskowej o podanej nazwie lub NULL, jeżeli nie ma zmiennej o podanej nazwie.

```
int putenv(const char *str);
```

  - Nadaje wartość zmiennej środowiskowej. `str` jest napisem postaci:

    <nazwa zmiennej>=<wartość>.

    W przypadku sukcesu zwraca 0, przeciwnie nadaje wartość `errno`.

```
int setenv(const char name, const char *value, int overwrite);
```

  - Nadaje wskazanej zmiennej środowiskowej wskazaną wartość. W przypadku sukcesu zwraca 0.

```
int unsetenv(const char *name);
```

  - Usuwa wskazaną zmienną środowiskową ze środowiska danego procesu. W przypadku sukcesu zwraca 0.

- Przykład:

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   char* sciezka = getenv ("PATH");
   if (sciezka == NULL)
         sciezka = "/bin:/usr/bin:/usr/local/bin";
   printf ("sciezka dostepu do plikow %s\n", sciezka);
   /* .. reszta programu ... */
   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
```

- Przykład:

```
/* M. Johnson, E. Troan: Oprogramowanie użytkowe w systemie Linux */
/* id.c - Displays uid, gid, and supplemental group information */
#include <grp.h>
#include <pwd.h>
#include <sys/types.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

void usage(int die, char *error) {
      fprintf(stderr, "Usage: id [<username>]\n");
      if (error) fprintf(stderr, "%s\n", error);
      if (die) exit(die);
}

void die(char *error) {
      if (error) fprintf(stderr, "%s\n", error);
      exit(3);
}

int main(int argc, const char *argv[]) {
      struct passwd *pw;
      struct group *gp;
      int current_user = 0;
      uid_t id;
      int i;
      if (argc > 2)
            usage(1, NULL);

      if (argc == 1) {
            id = getuid();
            current_user = 1;
            if (!(pw = getpwuid(id)))
                  usage(1, "Username does not exist");
      }
      else {
            if (!(pw = getpwnam(argv[1])))
                  usage(1, "Username does not exist");
            id = pw->pw_uid;
      }

      printf("uid=%d(%s)", id, pw->pw_name);
      if ((gp = getgrgid(pw->pw_gid)))
            printf(" gid=%d(%s)", pw->pw_gid, gp->gr_name);
      if (current_user) {
            gid_t *gid_list;
            int gid_size;

            if (getuid() != geteuid()) {
                  id = geteuid();
                  if (!(pw = getpwuid(id)))
                        usage(1, "Username does not exist");
                  printf(" euid=%d(%s)", id, pw->pw_name);
            }
```

```
            if (getgid() != getegid()) {
                  id = getegid();
                  if (!(gp = getgrgid(id)))
                        usage(1, "Group does not exist");
                  printf(" egid=%d(%s)", id, gp->gr_name);
            }

            /* use getgroups interface to get current groups */
            gid_size = getgroups(0, NULL);
            if (gid_size) {
                  gid_list = malloc(gid_size * sizeof(gid_t));
                  getgroups(gid_size, gid_list);
                  for (i = 0; i < gid_size; i++) {
                        if (!(gp = getgrgid(gid_list[i])))
                              die("Group does not exist");
                        printf("%s%d(%s)", (i == 0) ? " groups=" : ",",
                        gp->gr_gid, gp->gr_name);
                  } /* koniec for */

            free(gid_list);
            }
      }
      else {
      /* get list of groups from group database */
      i = 0;
      while ((gp = getgrent())) {
            char *c = *(gp->gr_mem);
            while (c && *c) {
                  if (!strncmp(c, pw->pw_name, 16)) {
                        printf("%s%d(%s)", (i++ == 0) ? " groups=" : ",",
                              gp->gr_gid, gp->gr_name);
                        c = NULL;
                  }
                  else {
                        c++;
                  }
            } /* koniec while */
      } /* koniec while */
      endgrent();
}
      printf("\n");
      exit(0);
}
```

Wynik działania programu:

```
$ ./info
uid=1260(adam) gid=101(staff) groups=101(staff)

$ id # Polecenie systemowe
uid=1260(adma) gid=101(staff) grupy=101(staff)
```

2.5 Wykorzystanie zasobów

- Jądro Uniksa śledzi ile zasobów używa każdy proces.
- Proces może sprawdzać swoje zużycie zasobów, jak i zużycie zasobów przez wszystkich swoich potomków.

```
int getrusage(int who, struct rusage *usage);
```

gdzie:

`who` – określa, który z dostępnych liczników nas interesuje
- `RUSAGE_SELF` – bieżący proces
- `RUSAGE_CHILDREN` – całkowite zużycie przez procesy potomne
- `RUSAGE_BOTH` – proces bieżący i procesy potomne

`usage` jest strukturą, w której umieszczona będzie informacja o zasobach

```
struct rusage {
      struct timeval ru_utime; /* user time used */
      struct timeval ru_stime; /* system time used */
      long ru_maxrss; /* maximum resident set size */
      long ru_ixrss; /* integral shared memory size */
      long ru_idrss; /* integral unshared data size */
      long ru_isrss; /* integral unshared stack size */
      long ru_minflt; /* page reclaims */
      long ru_majflt; /* page faults */
      long ru_nswap; /* swaps */
      long ru_inblock; /* block input operations */
      long ru_oublock; /* block output operations */
      long ru_msgsnd; /* messages sent */
      long ru_msgrcv; /* messages received */
      long ru_nsignals; /* signals received */
      long ru_nvcsw; /* voluntary context switches */
      long ru_nivcsw; /* involuntary context switches */
};
```

- Przykład: Wyświetlenie czasu użytkownika i czasu systemowego dla procesu

Czas użytkownika: czas procesora wykorzystany na wykonanie programu użytkownika
Czas systemowy: czas procesora wykorzystany na wykonanie wywołań systemowych

```
void print_cpu_time()
{
 struct rusage usage;
 getrusage (RUSAGE_SELF, &usage);
 printf ("CPU time: %ld.%06ld sec user, %ld.%06ld sec system\n",
       usage.ru_utime.tv_sec, usage.ru_utime.tv_usec,
       usage.ru_stime.tv_sec, usage.ru_stime.tv_usec);
}
```

2.6 Ograniczenia zasobów

- Są dwa typy ograniczeń: miękkie (soft limit) i twarde (hard limit).
- Tylko administrator może definiować ograniczenie twarde.
- Ograniczenie miękkie może być zmienione przez dowolny proces na wartość mniejszą lub równą ograniczeniu twardemu.
- Każdy proces może zmniejszyć ograniczenie twarde do wartości większej lub równej ograniczeniu miękkiemu. Użytkownik zwykły nie może przywrócić jednak wartości poprzedniej ograniczenia twardego.
- Funkcje **`getrlimit`** i **`setrlimit`**

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

struct rlimit {
      rlim_t rlim_cur; /* soft limit: bieżący limit */
      rlim_t rlim_max; /* hard ilmit: max dla rlim_cur */
};
/* pobierz wartości limitów */
int getrlimit(int resource, struct rlimit *rlim);

/* ustaw wartości limitów */
int setrlimit(int resource, struct rlimit *rlim);
```

RLIMIT_CPU	Maksymalny czas procesora (CPU) w sekundach dostępny dla programu
RLIMIT_FSIZE	Maksymalny rozmiar tworzonego pliku w bajtach
RLIMIT_DATA	Maksymalny rozmiar obszaru danych programu (dane+sterta)
RLIMIT_STACK	Maksymalny rozmiar obszaru stosu programu
RLIMIT_CORE	Maksymalny rozmiar pliku ze zrzutem pamięci
RLIMIT_RSS	Maksymalna ilość używanej pamięci RAM
RLIMIT_NPROC	Maksymalna liczba procesów potomnych dla rzeczywistego UID
RLIMIT_NOFILE	Maksymalna liczba otwartych plików przypadająca na proces

Brak ograniczenia jest oznaczany stałą RLIM_INFINITY.

- Przykład:

```
#include <unistd.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <stdio.h>
#include <stdlib.h>
void pr_limits(char *name, int resource);
int main() {
 fprintf(stdout, "%-14s %10s %10s\n","Name","Soft","Hard");
 pr_limits("RLIMIT_CORE",RLIMIT_CORE);
 pr_limits("RLIMIT_CPU",RLIMIT_CPU);
 pr_limits("RLIMIT_DATA",RLIMIT_DATA);
 pr_limits("RLIMIT_FSIZE",RLIMIT_FSIZE);
 pr_limits("RLIMIT_NOFILE",RLIMIT_NOFILE);
 pr_limits("RLIMIT_NPROC",RLIMIT_NPROC);
 pr_limits("RLIMIT_RSS",RLIMIT_RSS);
 pr_limits("RLIMIT_STACK",RLIMIT_STACK);
 exit(0);
}
void pr_limits(char *name, int resource) {
struct rlimit limit;
if (getrlimit(resource, &limit) < 0) {
 fprintf(stderr,"getrlimit: blad dla %s", name);
 exit(1);
 }
printf("%-14s ", name);
if (limit.rlim_cur == RLIM_INFINITY) printf("(infinite) ");
else printf("%10ld ", limit.rlim_cur);
if (limit.rlim_max == RLIM_INFINITY) printf("(infinite)\n");
else printf("%10ld\n", limit.rlim_max);
}
```

Wynik działania programu:

```
$ ./program
Name              Soft        Hard
RLIMIT_CORE          0  (infinite)
RLIMIT_CPU        3600        3600
RLIMIT_DATA (infinite)  (infinite)
RLIMIT_FSIZE (infinite) (infinite)
RLIMIT_NOFILE      164         164
RLIMIT_NPROC        30          30
RLIMIT_RSS   (infinite) (infinite)
RLIMIT_STACK    8388608  (infinite)
```

- Przykład 2: ustawianie limitu

```
#include <sys/resource.h>
#include <sys/time.h>
#include <unistd.h>
int main () {
 struct rlimit rl;
 getrlimit (RLIMIT_CPU, &rl);
       rl.rlim_cur = 1; /* Ustaw limit CPU na jedną sekundę */
 setrlimit (RLIMIT_CPU, &rl);
 while (1)
 ;
 return 0;
}
$ ./program
Przekroczony limit czasu procesora
```

- Funkcja **`sysconf`**

```
#include <unistd.h>
long sysconf(int name);
```

Pozwala uzyskać wartość wskazanego parametru w czasie wykonania. Odpowiednie stałe są zdefiniowane w pliku `unistd.h`. Zwraca żądaną wartość lub -1 w przypadku braku takiego parametru lub błędu.

- Przykładowe parametry, których wartości można uzyskać:

```
#include <stdio.h>
#include <unistd.h>
int
main( ){
   char *limits[ ]={"Max size of argv + envp",
                    "Max # of child processes",
                    "Max # of open files",
                    "Job control supported?",
                    "Saved IDs supported?",
                    "Version of POSIX supported",
                    0};
   int constant[ ]={ _SC_ARG_MAX, _SC_CHILD_MAX,
                    _SC_OPEN_MAX, _SC_JOB_CONTROL,
                    _SC_SAVED_IDS, _SC_VERSION };
   int i;
   for (i=0; limits[i]; ++i) {
      printf("%s %ld\n", limits[i], sysconf(constant[i]));
   }
   return 0;
}
```

Wynik działania:

```
$ ./program
Max size of argv + envp 131072
Max # of child processes 999
Max # of open files 164
Job control supported? 1
Saved IDs supported? 1
Version of POSIX supported 200112
```

2.7 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);
   printf("Data: %s",ctime(&czas));
   return 0;
}

$ ./program
Data: Tue Oct 4 19:16:09 2014
```

- 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
2014-10-04 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
```