Środowisko Linuxa

Program → proces działajacy w środowisku wielozadaniowym

getopt

wybiera znaki opcji z tablicy argumentów

```
#include <unistd.h>
int getopt(int liczbarg, char *const tablarg[], const char *optciag);
extern char *optarg;
extern int optind, opterr, optopc;
```

optciag – okresla opcje zdefiniowane dla programu;

znak : wskazuje, że z opcją związana jest wartość

Np. getopt(liczarg, tablarg, ”if:lr”);

wartość – **optarg**

indeks następnego argumentu - **optind**

Zwraca : opcję

-1 – koniec opcji

? – niezrozumiała opcja - (**optopc** ---?)

: - brak wartości opcji, jeśli jest wymagana

Plik: **opcje.c**

Wywołanie:

opcje –i –lr ‘hej hej’ –f plik.c –q

Efekt:

```
opcja: i
opcja: l
opcja: r
opcja: f  wartośc: plik.c
./opcje: invalid option -- q
nieznana opcja: -q
argument: hej hej
```

```
#include <stdio.h>
#include <unistd.h>
int main(int liczarg, char *tablarg[])
{
	int opt;
	while((opt=getopt(liczarg, tablarg, "if:lr"))!=-1){
		switch(opt){
		case 'i':
		case 'l':
		case 'r':
			printf("opcja: %c\n", opt);
			break;
		case ':':
			printf("opcja: %c wymaga wartości\n",opt);
			break;
		case 'f':
			printf("opcja: %c  wartośc: %s\n",opt, optarg);
			break;
		case '?':
			printf("nieznana opcja: %s\n",tablarg[optind-1]);
			break;
		}
	}
	for(; optind<liczarg; optind++)
	printf("argument: %s\n",tablarg[optind]);
exit(0);
}
```

- wszystkie opcje są przetwarzane bez względu na pozycję na liście argumentów – getopt ponownie zapisuje tablicę tablarg (sprawdzić)

```
for(i=1;i<liczarg;i++)
    printf("argument %d: %s\n",i,tablarg[i]);
```

- getopt – wyświetla swój komunikat o błędach – nieznane opcje

dostęp do zmiennych środowiskowych

putenv getenv

getenv – zwraca wartość związaną z daną
nazwą lub null

putenv – dodaje do środowiska ciąg postaci:
nazwa=wartość

zwraca -1 - brak pamięci; errno=ENOMEM

```
#include <stdlib.h>
char *getenv(const char *nazwa);
int putenv(const char *ciag);
```

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int main(int liczarg, char *tablarg[])
{
	char *zmienna, *wart;
	if(liczarg==1||liczarg>3){
		fprintf(stderr, "blad liczby argumentow\n");
		exit(1);
	};
	zmienna=tablarg[1];
	wart=getenv(zmienna);
	if(wart)
		printf("zmienna %s ma wartosc %s\n",zmienna,wart);
	else
		printf("zmienna %s nie ma wartosci\n",zmienna);
```

```
if(liczarg==3){
        char *string;
        wart=tablarg[2];
        string=malloc(strlen(zmienna)+strlen(wart)+2);
        if(!string)
            {fprintf(stderr,"brak pamieci\n"); exit(1);}
        strcpy(string, zmienna);
        strcat(string,"=");
        strcat(string,wart);
        if(putenv(string)!=0){
            fprintf(stderr,"blad\n");
            free(string);
            exit(1);
        }
        wart=getenv(zmienna);
        if(wart)
            printf("nowa wart zmiennej %s jest %s\n", zmienna,wart);
        else
            printf("nowa wart zmiennej %s jest null\n", zmienna);
        }
        exit(0);
    }
```

```
$ ./srod KO hkflhk
zmienna KO nie ma wartosci
nowa wart zmiennej KO jest hkflhk

$ KO=2 ./srod KO hkflhk
zmienna KO ma wartosc 2
nowa wart zmiennej KO jest hkflhk
```

Zmienna environ

```
#include <stdio.h>
#include <stdlib.h>
extern char**environ;
int main()
{
char **srod = environ;
while(*srod)
        {
                printf("%s\n",*srod);
                srod++;
        }
        exit(0);
}
```

time

liczba sekund od 1.01.1970

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

```
#include<time.h>
#include<stdio.h>
#include<unistd.h>
int main()
{
time_t czas;
time_t *ti;
printf("czas=%ld\n",time(ti));
sleep(5);
czas=time(ti);
printf("czas=%ld\n",*ti);
sleep(2);
czas=time(ti);
printf("czas=%ld\n",czas);
exit(0);
}
```

difftime

```
#include <time.h>
double difftime(time_t czas1, time_t czas2);
#include<time.h>
#include<stdio.h>
#include<unistd.h>
int main()
{
time_t czas;
time_t *ti;
time_t czas2;
czas=time(ti);
printf("czas=%ld\n",*ti);
sleep(2);
czas2=time(ti);
printf("czas=%ld\n",czas2);
printf("roznica difftime=%lf\n",difftime(czas2,czas));
czas2=czas2-czas;
printf("roznica=%ld\n",czas2);
exit(0);
}
```

gmtime mktime

```
#include <time.h>
struct tm *gmtime(const time_t *czas);
```

tm:
int tm_sec
int tm_min
int tm_hour
int tm_mday
int tm_mon
int tm_year
int tm_wday
int tm_yday
int tm isdst

localtime

```
time_t mktime(struct tm *wskczas);
```

```
#include <time.h>
#include <stdio.h>
int main()
{
struct tm *wczas;
time_t czas;
(void) time(&czas);
wczas=gmtime(&czas);
printf("czas=%ld\n",czas);
printf("data:%04d/%02d/%02d\n",wczas->tm_year, wczas->tm_mon+1, wczas->tm_mday);
printf("godzina: %02d:%02d:%02d\n", wczas->tm_hour, wczas->tm_min,wczas->tm_sec);
exit(0);
}
```

asctime ctime

```
#include <time.h>
char *asctime(const struct tm *wskczas);
char *ctime(const time_t *wartczas);
```

ctime ≡ asctime(localtime(wartczasu))

```
czas=time(ti);
printf("ctime: %s\n",ctime(ti));
```

ctime: Wed Feb 11 23:50:31 2004

Formatowanie godzin i dat

strftime

```
#include <time.h>
size_t strftime(char *s, size_t maxrozmiar, const char *format,
    struct tm *wskczas);
```

wynik – w s

Specyfikatory formatu:

%a, %A	skrócona, pełna nazwa dnia tygodnia
%b, %B	skrócona, pełna nazwa miesiąca
%c	data i godzina
%H	godzina
%Y	dwie ostatnie cyfry roku
…..	
%p	a.m lub p.m

strptime

```
#include <time.h>
char *strptime(const char *bufor, const char *format, struct tm
    *wskczas);
```

Wczytuje datę - wypełnia strukturę tm wg. formatu analizując bufor

```
#include <time.h>
#include <stdio.h>
int main()
{
	struct tm *wskczas;
	time_t czas;
	char bufor[256];
	char *rezultat;

	(void) time(&czas);
	wskczas=localtime(&czas);
	strftime(bufor, 256,"%Y %A %d  %B, %I:%M:%S %p",wskczas);
	printf("data:%s\n", bufor);
	rezultat=strptime(bufor,"%Y %a %d %B %I %M %S %p",wskczas);
	printf("data1: %02d/%02d/%02d\n",wskczas->tm_year,
	wskczas->tm_mon+1,wskczas->tm_mday);
	strcpy(bufor,"Mon 5 January 2004, 22:10");
	rezultat=strptime(bufor,"%a %d %B %Y %R",wskczas);
	printf("data2: %02d/%02d/%02d\n",wskczas->tm_year,
	wskczas->tm_mon+1,wskczas->tm_mday);
	exit(0);
}
```

```
data:2004 Monday 16  February, 02:20:17 AM
data1: 104/02/16
data2: 104/01/05
```

Pliki tymczasowe

Generacja nazwy:

#include <stdio.h>

char *tmpnam(char *s);

Generacja nazwy i otwarcie:

#include <stdio.h>

char *tmpfile(void); fopen w+

mktemp; mkstemp

użytkownicy

```
#include <sys/types.h>
#include <unistd.h>
uid_t getuid(void);
char *getlogin(void);
uid_t geteuid(void);
uid_t getgid(void);
uid_t getegid(void);

#include <sys/types.h>
#include <pwd.h>
struct passwd *getpwuid(uid_t uid);
struct passwd *getpwnam(constchar *nazwa);
```

struktura passwd - plik pwd.h

- char *pw_name
- uid_t pw_uid
- gid_t pw_gid
- char *pw_dir
- char *pw_shell

Plik haseł

```
#include <pwd.h>
void endpwent(void);
struct passwd *getpwent(void);
void setpwent(void);
```

Informacje o komputerze

```
#include <unistd.h>
int gethostname(char *nazwa, size_t dl);
int gethostid(void);

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

utsname:

```
char sysname[]
char nodename[]
char release[]
char version[]
char machine[]
```

limity

#include <sys/resource.h>
int getpriority(int **który**, id_t kto);
int setpriority(int **który**, id_t kto, int priorytet);

który – PRIO_PROCESS; PRIO_PGRP; PRIO_USER
getpriority – zwraca priorytet lub -1 (to też może być wartość prt!!)
pri=int getpriority(PRIO_PROCESS, getpid());

int getrlimit(int zasoby, struct rlimit *r_limit);
int setrlimit(int **zasoby**, const struct rlimit *r_limit);

zasoby: RLIMIT_CORE, RLIMIT_CPU, RLIMIT_DATA, RLIMIT_FSIZE, RLIMIT_NOFILE..
struktura rlimit: rlim_t rlim_cur, rlim_max

int getrusage(int **kto**, struct rusage *r_usage);
wykorzystanie czasu procesora (użytkownik, system)
kto – RUSAGE_SELF; RUSAGE_CHILDREN
struktura rusage: struct timeval ru_utime, ru_stime
timeval -- tv_sec; tv-usec (sekundy, mikrosek)