Język ANSI C

część 7

systemowe wejście - wyjście

Tylko kilka informacji i przykładów.
Temat ten NIE był omawiany na wykładzie

Jarosław Gramacki
Instytut Informatyki i Elektroniki

systemowe wejście - wyjście

- podstawowe funkcje
- patrz też: Rozdział 8, K&R, ANSI C

- **systemowe** we-wy (non-ANSI, Low Level I/O)
- z poziomu języka C jedynie wywołujemy te (systemowe) funkcje

```
open()
close()
read()
write()
```

```
int open (const char *pathname, int flags);              // istniejący plik
int open (const char *pathname, int flags, mode_t mode); // nowy plik

man 2 open
...
The  parameter  flags is one of O_RDONLY, O_WRONLY or O_RDWR which request
opening the file read-only, write-only or read/write, respectively, bitwise-
or'd with zero or  more  of  the following:
...

int close (int fd);
ssize_t write (int fd, const void *buf, size_t count);
ssize_t read  (int fd, void *buf, size_t count);
```

systemowe wejście - wyjście

– na przykładzie Linux-a
– tzw. niebuforowane wejście-wyjście

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

#include <fcntl.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>

#define MAX_ILOSC 80

void ZapiszDoPliku  (int jakiego, char *co);  // prototypy
void OdczytajZPliku (char *nazwaPliku);

int main(void)
{
   int deskryptor;
   char msg[] = "O zesz ty orzeszku!";

   if ((deskryptor = open ("test.tmp", O_CREAT | O_RDWR,
                                       S_IWRITE | S_IREAD)) == -1) {
     perror("Blad:");
     return 1;
   }
   ZapiszDoPliku (deskryptor, msg);
   close(deskryptor);
   OdczytajZPliku ("test.tmp");
   return 0; }
```

dla UNIX

```
// MSDOS
#include <fcntl.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <io.h>
```

deskryptor pliku, **liczba całkowita**
(w <stdio.h> była zmienna typu FILE *)

systemowe wejście - wyjście

```
void ZapiszDoPliku (int jakiego, char *co)
{
  write (jakiego, co, strlen(co));
}

void OdczytajZPliku (char *nazwaPliku)
{
   char bufor[80];
   int iloscBajtow, deskryptor, i;

   bufor = malloc(80);
   if ((deskryptor = open( nazwaPliku, O_RDONLY )) == -1)
   {
     perror("\nBlad otwarcia pliku\n");
     exit(1);
   }

   if ((iloscBajtow = read( deskryptor, bufor, MAX_ILOSC)) == -1)
   {
     perror("Wystapil blad przy probie odczytu.\n");
     exit(1);
   }
   else
   {
     printf("\nOdczytano: %d bajtow. Oto one:\n", iloscBajtow);
     for(i=0; i<iloscBajtow; i++)
         printf("%c", bufor[i]);
   }
   free(bufor);
}
```

systemowe wejście - wyjście

– inne przykłady

```
// otwarcie tylko do odczytu
deskryptor = open ("test.tmp", O_RDONLY);

// otwarcie w trybie do zapisu
deskryptor = open ("test.tmp", O_CREAT | O_WRONLY, 0644);
```

```
// niebuforowany getchar()
// Uwaga! należy odwołać oryginalne makro getchar() z <stdio.h> przez #undef

int getchar (void)
{
  char c
  return (read(0, &c, 1) == 1) ? (unsigned char)c : EOF;
}
```

gdy ew. problemy z powielaniem bitu znaku

stdin

systemowe wejście - wyjście

– inne przykłady

```
// buforowany getchar()
// wczytuj dużymi porcjami, oddawaj po jednym znaku

int getchar (void)
{
 static char buf[BUFSIZE];
 static char *bufp = buf;
 static int n = 0;

 if (n == 0) // bufor pusty
 {
   read(0, buf, sizeoof(buf))
   bufp = buf;
 }
 return (--n >= 0) ? (unsigned char)*bufp++ : EOF;
}
```

systemowe wejście - wyjście

– kopiuj (szybko) pliki
– zalety zauważymy dla dużej ilości kopiowanych danych
– użyto dużego bufora danych

```
#include <stdio.h>
#ifndef __MSDOS__        /* jesli nie jest to system MS-DOS, */
#define __UNIX__         /* zatem jest to system UNIX */
#endif /* __MSDOS__ */

#include <stdlib.h>

#ifdef __UNIX__
#include <sys/types.h>  /* definicja pliku systemu plikow systemu UNIX */
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#endif /* __UNIX__ */

#ifdef __MSDOS__
#include <fcntl.h>     /* definicja pliku systemu plikow systemu DOS */
#include <sys\stat.h>
#include <io.h>
#endif __MSDOS__

#ifndef O_BINARY
#define O_BINARY 0     /* definicja znacznika, jesli jeszcze nie istnieje */
#endif /* O_BINARY */

#define BUFFER_SIZE (16 * 1024) /* uzycie bufora o rozmiarze 16Kb */
```


systemowe wejście - wyjście

```
int main(int argc, char *argv[]) {
    char  buffer[BUFFER_SIZE];  /* bufor danych */
    int   in_file;              /* deskryptor pliku zrodlowego */
    int   out_file;             /* deskryptor pliku docelowego */
    int   read_size;            /* ilosc bajtow odczytanych w
                                   ostatniej operacji */

    if (argc != 3) {
        fprintf(stderr, "Blad: Nieprawidlowa ilosc argumentow\n");
        fprintf(stderr, "Format funkcji: copy <z> <do>\n");
        exit(8);}
    in_file = open (argv[1], O_RDONLY | O_BINARY);
    if (in_file < 0) {
        fprintf("Blad: Nie mozna bylo otworzyc pliku %s\n", argv[1]);
        exit(8);}
    out_file = open (argv[2], O_WRONLY | O_TRUNC | O_CREAT | O_BINARY, 0666);
    if (out_file < 0) {
        fprintf("Blad: Nie mozna bylo otworzyc pliku %s\n", argv[2]);
        exit(8);}
    while (1) {
        read_size = read (in_file, buffer, sizeof(buffer));

        if (read_size == 0) break;              /* koniec pliku */

        if (read_size < 0) {
            fprintf(stderr, "Blad: Blad odczytu\n");
            exit(8); }
        write (out_file, buffer, (unsigned int) read_size);
    }
    close(in_file);  close(out_file);
    return (0);
}
```