9 Programowanie urządzeń

• Każde urządzenie przyłączone do komputera uniksowego jest reprezentowane przez plik.

9.1 Pliki terminala

```
#include <stdio.h>
  char *ctermid(char *s);
```
Zwraca ścieżkę do bieżącego terminala sterującego (na parametrze `s` i jako wartość funkcji).

```
#include <unistd.h>
  char *ttyname(int fd);
```
Zwraca ścieżkę do terminala sterującego wskazywanego przez `fd`. Przy błędzie zwraca NULL i ustawia `errno`.

```
#include <unistd.h>
  int isatty(int fd);
```
Sprawdza, czy `fd` jest podłączone do terminala. Jeżeli tak, zwraca 1 i 0 przy odpowiedzi negatywnej.

9.2 Właściwości połączenia terminalowego

- Buforowanie (w5p3)

```
#include <stdio.h>

int main()
{
int c, n = 0;
while( ( c = getchar()) != 'q' )
  printf("char %3d is %c code %d\n", n++, c, c );

return 0;
}
```

  - proces nie otrzyma danych dopóki nie zostanie naciśnięty Return
  - użytkownik naciska Return (kod ASCII 13), proces widzi nową linię (ASCII 10)
  - proces wysyła nową linię, terminal otrzymuje parę Return-Nowa linia

- Sterownik terminala

Polecenie `stty` pozwala ustawiać parametry terminala z poziomu shella. Do parametrów, które można w ten sposób ustawiać należą:
- włączanie/wyłączanie wyświetlania wpisywanego znaku na ekranie,
- obsługa znaków nowej linii (NewLine) i powrotu karetki (Carriage Return) ,
- edycja wpisywanego wiersza,
- obsługa klawiszy generujących sygnały,
- i wiele innych.

```
$ stty -a
speed 38400 baud; rows 24; columns 80; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>;
eol2 = <undef>; swtch = <undef>; start = ^Q; stop = ^S; susp = ^Z;
rprnt = ^R; werase = ^W; lnext = ^V; flush = ^O; min = 1; time = 0;
-parenb -parodd cs8 -hupcl -cstopb cread -clocal -crtscts
-ignbrk -brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr icrnl ixon
```

```
-ixoff -iuclc -ixany -imaxbel -iutf8
opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0 cr0 tab0 bs0
vt0 ff0
isig icanon iexten echo echoe echok -echonl -noflsh -xcase -tostop -echoprt
echoctl echoke
```

minus – dana opcja jest wyłączona

`icrnl` – włącz Input: convert Carriage Return to NewLine
`ocrnl` – włącz Output: add to NewLine a CarriageReturn
`echo` – włącz wyświetlanie na ekranie naciśniętego klawisza

9.3 Programowanie sterownika

Grupy operacji:

- wejściowe (input) – co można zrobić ze znakami pochodzącymi z terminala (na przykład zamiana małych liter na wielkie, zamiana CR na LF)

- wyjściowe (output) – co sterownik może zrobić ze znakami przesyłanymi do terminala (zastępowanie znaku TAB sekwencją spacji, zamiana LF na CR, konwersja małych na wielkie litery)

- sterujące (control) – jak reprezentowane są znaki (liczba bitów, kontrola parzystości, bity stopu)

- lokalne (local) – co sterownik może zrobić, gdy znaki są już w sterowniku (przesyłanie znaku z powrotem do użytkownika – echo, buforowanie wejścia do naciśnięcia Return).

Procedura

1. Pobierz atrybuty terminala
2. Zmodyfikuj potrzebne atrybuty
3. Prześlij aktualne wartości atrybutów do sterownika

```
#include <termios.h>

struct termios {
   tcflag_t c_iflag; /* przetwarzanie wejścia – flagi */
   tcflag_t c_oflag; /* przetwarzanie wyjścia */
   tcflag_t c_cflag; /* tryb sterowania */
   tcflag_t c_lflag; /* tryb lokalny */
   cc_t c_line; /* organizacja wiersza */
   cc_t c_cc[NCCS]; /* znaki sterujące */
};
```

gdzie NCCS to stała.

```
// czytaj atrybuty sterownika terminala
      int tcgetattr(int fd, struct termios *t);

// ustaw atrybuty sterownika terminala
      int tcsetattr(int fd, int akcja, struct termios *t);
```

gdzie parametr `akcja` może przyjmować wartości:

- `TCSETNOW` - zmiany mają skutek natychmiastowy
- `TCSADRAIN` - zmiany mają miejsc dopiero po przesłaniu wszystkich danych wyjściowych do terminala
- `TCAFLUSH` - zmiany mają miejsc dopiero po przesłaniu wszystkich danych wyjściowych do terminala, zawartość kolejki wejściowej (dane odebrane, ale jeszcze nie przetworzone) odrzucana

- Przykład: operacje bitowe

Czynność	Kod
testowanie bitu	`if (flagset & MASK)`
ustawienie bitu	`flagset |= MASK`
czyszczenie bitu	`flagset &= ~MASK`

- Przykład: sprawdzanie ustawienia bitu

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

int main()
{
    struct termios info;
    int rv;

    rv = tcgetattr( 0, &info );
    if ( rv == -1 ){
        perror( "tcgetattr");
        exit(1);
    }
    if ( info.c_lflag & ECHO )
        printf(" echo is on , its bit is 1\n");
    else
        printf(" echo if OFF, its bit is 0\n");
}
```

- Przykład: ustawianie bitu

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

#define oops(s,x) { perror(s); exit(x); }

int main(int argc, char *argv[])
{
struct termios info;

if ( argc == 1 )
      exit(0);

if ( tcgetattr(0,&info) == -1 )
      oops("tcgettattr", 1);

if ( argv[1][0] == 'y' )
      info.c_lflag |= ECHO ;
else
      info.c_lflag &= ~ECHO ;

if ( tcsetattr(0,TCSANOW,&info) == -1 )
      oops("tcsetattr",2);
}
```

• Przykład

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

int main()
{
char line[BUFSIZ];
sigset_t sig, savesig;
struct termios modes, savemodes;

sigemptyset(&sig);
sigaddset(&sig, SIGINT);
sigaddset(&sig, SIGQUIT);
sigaddset(&sig, SIGTSTP);
sigprocmask(SIG_BLOCK, &sig, &savesig);

if (tcgetattr(0, &modes) < 0) {
  perror("tcgetattr");
  exit(1);
}
savemodes = modes;
modes.c_lflag &= ~(ECHO | ECHOE | ECHOK | ECHOKE);

if (tcsetattr(0, TCSAFLUSH, &modes) < 0) {
  perror("tcsetattr");
  exit(1);
}

printf("Wpisz wiersz (nie bedzie wyswietlony na ekranie: ");
fgets(line, sizeof(line), stdin);
line[strlen(line)-1] = '\0';
putchar('\n');

if (tcsetattr(0, TCSAFLUSH, &savemodes) < 0) {
   perror("tcsetattr");
   exit(1);
}

sigprocmask(SIG_SETMASK, &savesig, (sigset_t *) 0);
printf("Wpisales \"%s\"\n", line);
exit(0);
}
```

9.4 Tryby pracy terminala

- tryb kanoniczny – buforowanie i edycja
- tryb niekanoniczny – wyłączenie bufowania i edycji

- Przykład (w5p5.c)

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

int main()
{
int c;
while ((c=getchar()) != EOF) {
   if (c=='z')
    c='a';
   else
    if (islower(c))
       c++;
   putchar(c);
}
}
```

- Program uruchomiony w trybie kanonicznym:

```
$ ./rotate
abc
bcd
```

Program uruchomiony w trybie niekanonicznym:

```
$ ./rotate
ab

cd
```

Program uruchomiony w trybie niekanonicznym z wyłączonym echem:

```
$ stty -icanon -echo
$./rotate (tego nie będzie widać)
bde
```

9.5 Sterowanie funkcją `read()`

```
struct termios {
...
cc_t c_cc[NCCS]; /* znaki sterujące */
};
```

`VMIN, VTIME` – makra używane jako indeksy tablicy `c_cc` w trybie niekanonicznym

VMIN	VTIME	
≠ 0	= 0	VMIN określa liczbę bajtów, na które należy poczekać
= 0	≠ 0	VTIME określa liczbę dziesiątek sekund, przez które `read()` będzie czekać przed powrotem z wywołania
≠ 0	≠ 0	VTIME określa liczbę dziesiątek sekund, przez które `read()` będzie czekać, gdy przynajmniej jeden bajt stanie się dostępny; powrót następuje gdy spełniony jest jeden z warunków
= 0	= 0	powrót natychmiastowy

Przykład

```
#include <termios.h>
#include <signal.h>
#include <stdlib.h>
#include <ctype.h>
#include <stdio.h>

int main()
{
      char c, lastc;
      sigset_t sig, savesig;
      struct termios modes, savemodes;

      sigemptyset(&sig);
      sigaddset(&sig, SIGINT);
      sigaddset(&sig, SIGQUIT);
      sigaddset(&sig, SIGTSTP);
      sigprocmask(SIG_BLOCK, &sig, &savesig);

      if (tcgetattr(0, &modes) < 0) {
            perror("tcgetattr");
            exit(1);
      }

      savemodes = modes;
      modes.c_cc[VMIN] = 1;
      modes.c_cc[VTIME] = 0;
      modes.c_lflag &= ~ICANON;
      modes.c_lflag &= ~(ECHO | ECHOE | ECHOK | ECHOKE);

      if (tcsetattr(0, TCSAFLUSH, &modes) < 0) {
            perror("tcsetattr");
            exit(1);
      }

      while (read(0, &c, 1) > 0) {
            if (isupper(c))
                  c = tolower(c);
            else if (islower(c))
                  c = toupper(c);
            if (c == savemodes.c_cc[VEOF] && lastc == '\n')
                  break;

            write(1, &c, 1);
            lastc = c;
            }

            if (tcsetattr(0, TCSAFLUSH, &savemodes) < 0) {
                  perror("tcsetattr");
                  exit(1);
            }

      sigprocmask(SIG_SETMASK, &savesig, (sigset_t *) 0);
      exit(0);
}
```

9.6 Urządzenia specjalne

```
/dev/null
/dev/zero
/dev/full
/dev/random i /dev/urandom
```

Przykład:

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

int random_number (int min, int max);

int main()
{
      int i;
      for (i=0; i< 5; i++)
      printf("%d\n",random_number(0,10));
      return 0;
}

int random_number (int min, int max)
{
  static int dev_random_fd = -1;

  char* next_random_byte;
  int bytes_to_read;
  unsigned random_value;

  assert (max > min);

  if (dev_random_fd == -1) {
        dev_random_fd = open ("/dev/random", O_RDONLY);
        assert (dev_random_fd != -1);
  }

  next_random_byte = (char*) &random_value;
  bytes_to_read = sizeof (random_value);

  do {
        int bytes_read;
        bytes_read = read (dev_random_fd, next_random_byte, bytes_to_read);
        bytes_to_read -= bytes_read;
        next_random_byte += bytes_read;
  } while (bytes_to_read > 0);

  return min + (random_value % (max - min + 1));
}
```

9.7 Ogólny interfejs do sterowania urządzeniami

```
#include <sys/ioctl.h>
int ioctl(int d, int request, ...);
```

Przykład:

```
#include <fcntl.h>
#include <linux/cdrom.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main (int argc, char* argv[])
{
      int fd = open (argv[1], O_RDONLY);
      ioctl (fd, CDROMEJECT); /* wysuń CDROM */
      close (fd);

      return 0;
}
```