

Biblioteki systemowe Unixa

Witold Paluszyński

witold.paluszynski@pwr.wroc.pl

<http://kcir.pwr.wroc.pl/~witold/>

Copyright © 2001–2008 Witold Paluszyński
All rights reserved.

Niniejszy dokument zawiera materiały do wykładu na temat bibliotek i funkcji systemowych Unixa. Jest on udostępniony pod warunkiem wykorzystania wyłącznie do własnych, prywatnych potrzeb i może być kopiowany wyłącznie w całości, razem z niniejszą stroną tytułową.

Biblioteka string

```
#include <string.h>

char *strcpy(char *dst, const char *src);
char *strncpy(char *dst, const char *src, size_t n);
char *strdup(const char *s1);
size_t strlen(const char *s);
char *strcat(char *dst, const char *src);
char *strncat(char *dst, const char *src, size_t n);
char *strchr(const char *s, int c);
char *strrchr(const char *s, int c);
int strcmp(const char *s1, const char *s2);
int strncmp(const char *s1, const char *s2, size_t n);
int strcasecmp(const char *s1, const char *s2);
int strncasecmp(const char *s1, const char *s2, int n);
size_t strcspn(const char *s1, const char *s2);
size_t strspn(const char *s1, const char *s2);
char *strpbrk(const char *s1, const char *s2);
char *strtok(char *s1, const char *s2);
char *strstr(const char *s1, const char *s2);
```


Biblioteka crypt

```
#include <crypt.h>

char *crypt(const char *key, const char *salt);
void setkey(const char *key);
void encrypt(char *block, int edflag);
```


Funkcje regcomp(3C)

Funkcje regcomp(3C) realizują dopasowanie napisów do wzorców (opisanych na stronie manuala regex(5)) typu BRE (*Basic Regular Expressions*) oraz ERE (*Extended Regular Expressions*), tych drugich głównie różniących się od pierwszych obsługą alternatywy |, i brakiem obsługi odwołań wstecznych typu \1.

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

int regcomp(regex_t *preg, const char *pattern, int cflags);
int regexec(const regex_t *preg, const char *string, size_t nmatch,
            regmatch_t pmatch[], int eflags);
size_t regerror(int errcode, const regex_t *preg, char *errbuf,
                size_t errbuf_size);
void regfree(regex_t *preg);
```

Struktura regmatch_t zawiera między innymi pola: rm_so i rm_eo typu regoff_t wypełniane przez funkcję regexec wartościami offsetu od początku bufora dopasowywanego stringu do początku znalezionej dopasowania, oraz do pierwszego znaku po ostatnim znaku dopasowania.

Funkcja sprawdzająca dopasowanie napisu do wzorca (przykład z manuala):

```
#include <regex.h>

int match(const char *string, char *pattern)
/*
 * Match string against the extended regular expression in pattern,
 * treating errors as no match.
 *
 * return 1 for match, 0 for no match
 */
{
    int status;
    regex_t re;
    if (regcomp(&re, pattern, REG_EXTENDED|REG_NOSUB) != 0) {
        return(0);          /* report error */
    }
    status = regexec(&re, string, (size_t) 0, NULL, 0);
    regfree(&re);

    if (status != 0) {
        return(0);          /* report error */
    }
    return(1);
}
```


Kolejny przykład ilustruje użycie flagi REG_NOTBOL do znajdowania kolejnych dopasowań wzorca w danym buforze (pominięto tu sprawdzanie błędów z funkcji regexec):

```
regex_t re;
regmatch_t pm;
int error;

(void) regcomp (&re, pattern, 0);

/* this call to regexec() finds the first match on the line */
error = regexec (&re, &buffer[0], 1, &pm, 0);

while (error == 0) {      /* while matches found */
    /* substring found between pm.rm_so and pm.rm_eo */
    /* This call to regexec() finds the next match */
    error = regexec (&re, buffer + pm.rm_eo, 1, &pm, REG_NOTBOL);
}
```

Uwaga: porównaj ostatni wiersz powyższego przykładu:

```
error = regexec (&re, buffer + pm.rm_eo, 1, &pm, REG_NOTBOL);
z:
error = regexec (&re, buffer + pm.rm_so + 1, 1, &pm, REG_NOTBOL);
```


Przeszukiwanie tablic: liniowe

Liniowe przeszukiwanie tablicy dowolnych elementów:

```
#include <search.h>

void *lsearch(const void *key, void *base,
             size_t *nelp, size_t width,
             int (*compar)(const void *, const void *));

void *lfind(const void *key, const void *base,
           size_t *nelp, size_t width,
           int (*compar)(const void *, const void *));
```

1. funkcja `lsearch` poszukuje elementu w tablicy, i wstawia gdy go nie było, `lfind` tylko poszukuje
2. aktualna liczba elementów `nelp` może być zwiększona przez `lsearch`
3. w tablicy musi być miejsce na dodatkowy element
4. użytkownik musi dostarczyć funkcji porównującej elementy
5. dla użycia bibliotek systemowych niezbędne jest użycie uniwersalnego typu wskaźnikowego (`void *`) i rzutowania

Przeszukiwanie tablic: liniowe — przykład

```
#define TABSIZE 50
#define ELSIZE 120

int porownaj(const void *p1, const void *p2) {
    return strncmp((char *)p1, (char *)p2, ELSIZE);
}

main() {
    char bufor[ELSIZE];          /* do czytania danych*/
    char tab[TABSIZE][ELSIZE];  /* tablica napisow */
    size_t nel = 0;             /* liczba elementow */
    int i;

    while (fgets(bufor, ELSIZE, stdin) != NULL &&
           nel < TABSIZE)
        (void) lsearch(bufor, tab, &nel, ELSIZE, porownaj);

    for( i = 0; i < nel; i++ )
        (void) fputs(tab[i], stdout);

    return 0;
}
```

Przeszukiwanie tablic: binarne

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

```
void *bsearch(const void *key, const void *base, size_t nel, size_t size,  
             int (*compar)(const void *, const void *));
```

```
void qsort(void *base, size_t nel, size_t width,  
          int (*compar)(const void *, const void *));
```

Przeszukiwanie tablic: binarne — przykład

```
#define STRSIZE 20
struct elem {
    char tresc[STRSIZE];
    int info;
};

static int porownaj(const void *elem1, const void *elem2) {
    return strcmp(((const struct elem *)elem1)->tresc,
                 ((const struct elem *)elem2)->tresc);    }

int nelem = 100;                /* liczba elementow tablicy tab */
struct elem tab[100], element, *elem_ptr;

qsort((void *)tab, nelem, sizeof(struct elem), porownaj);

/* poszukujemy elementu o tresci wczytanej z wejscia */
while (scanf("%20s", element.tresc) != EOF) {
    elem_ptr = bsearch(&element, tab, nelem, sizeof(struct elem), porownaj);
    if (elem_ptr != NULL)
        printf("tresc = %20s, info = %d\n", elem_ptr->tresc, elem_ptr->info);
    else
        printf("nie znaleziono: %20s\n", element.tresc);
}
```

Przeszukiwanie tablic: haszowe

```
#include <search.h>
```

```
ENTRY *hsearch(ENTRY item, ACTION action);
```

```
int hcreate(size_t mekments);
```

```
void hdestroy(void);
```

Przeszukiwanie drzew binarnych

```
#include <search.h>
```

```
void *tsearch(const void *key, void **rootp,  
             int (*compar)(const void *, const void *));
```

```
void *tfind(const void *key, void * const *rootp,  
           int (*compar)(const void *, const void *));
```

```
void *tdelete(const void *key, void **rootp,  
             int (*compar)(const void *, const void *));
```

```
void twalk(const void *root, void(*action)(void *, VISIT, int));
```


Przeszukiwanie drzew binarnych — przykład

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

#define MAX_NODE 20

struct node {
    char *string;
    int length;
};

char string_space[10000];
struct node nodes[MAX_NODE];
void *root = NULL;

int node_compare(const void *node1, const void *node2) {
    return strcmp(((const struct node *) node1)->string,
                 ((const struct node *) node2)->string);
}

void print_node(const void *node, VISIT order, int level) {
    if (order == postorder || order == leaf)
        printf("level=%d, string=%-20s\n", level, (*(struct node **)node)->string);
}
```

```
}

main() {
    char *strptr = string_space;
    struct node *nodeptr = nodes;
    int i = 0;

    while (gets(strptr) != NULL && i++ < MAX_NODE) {
        nodeptr->string = strptr;
        nodeptr->length = strlen(strptr);
        (void) tsearch((void *)nodeptr, &root, node_compare);
        strptr += nodeptr->length + 1; /* space for NULL */
        nodeptr++;
    }
    twalk(root, print_node);
}
```

Biblioteka ndbm(3)

Biblioteka ndbm to prosty zestaw procedur pozwalający tworzyć programy zakładające własne bazy danych zapewniający szybki dostęp do dużej ilości danych, a także dużą przenośność źródłową programów.

```
#include <ndbm.h>

int dbm_clearerr(DBM *db);
void dbm_close(DBM *db);
int dbm_delete (DBM *db, datum key);
int dbm_error(DBM *db);
datum dbm_fetch(DBM *db, datum key);
datum dbm_firstkey(DBM *db);
datum dbm_nextkey(DBM *db);
DBM *dbm_open(char *file, int flags, int mode);
int dbm_store(DBM *db, datum key, datum content, int flags);

typedef struct {
    char    *dptr;
    int     dsize;
} datum;
```

Przykład zapisania i odczytania rekordu (użycie makra DBM_INSERT powoduje błąd, jeśli w bazie istniałby już rekord z podanym kluczem, natomiast użycie zamiast niego makra DBM_REPLACE powoduje nadpisanie istniejącego rekordu):

```
#define NAME          "Bill"
#define PHONE_NO     "123-4567"
#define DB_NAME      "DB_FILE.DB"

DBM *db;
datum name = {NAME, sizeof (NAME)};
datum obuf = {PHONE_NO, sizeof (PHONE_NO)};
datum ibuf;

db = dbm_open(DB_NAME, O_RDWR | O_CREAT, 0660);
(void) dbm_store(db, name, obuf, DBM_INSERT);
dbuf = dbm_fetch(db, name);
dbm_close(db);
```

Przykład przeszukania całej bazy:

```
datum key;
for (key = dbm_firstkey(db); key.dptr != NULL; key = dbm_nextkey(db))
```

Biblioteka ndbm(3) — uwagi

- ograniczenie wielkości pojedynczej danej: łączna wielkość klucza i danej jest ograniczona do wielkości jednego bloku (1kB na Solarisie)
- brak mechanizmów równoczesnego dostępu
- brak mechanizmu tworzenia wielu kluczy
- operacje tworzenia i modyfikacji bazy są wolniejsze niż w systemach wykorzystujących serwer bazy danych
- jednak dostęp do danych jest szybki
- procedury biblioteki ndbm tworzą pliki z dziurami, dzięki czemu zajmują mniej miejsca na dysku
- powszechność implementacji biblioteki ndbm na systemach Unixowych zapewnia dużą przenośność źródłową programów, i popularność biblioteki

Inne implementacje ndbm(3)

Istnieje podobna do ndbm ale bardziej rozbudowana biblioteka gdbm dostępna w źródle na licencji Gnu. Jeśli jest zainstalowana w systemie, to można z niej korzystać jak z lepszej implementacji ndbm (wczytując inny plik nagłówkowy).

Można również korzystać z pełnej funkcjonalności biblioteki gdbm (np. synchronizacji, blokad, zarządzania pamięcią), ale wtedy trzeba zrezygnować z kompatybilności z biblioteką ndbm, która jest standardowym elementem systemów Unixowych.

Jeszcze inną możliwość korzystania z funkcji ndbm zapewnia inna biblioteka do tworzenia bazy danych zwana „Berkeley db”. Jest ona jeszcze poważniejszą implementacją systemu bazy danych, i w programach które ją wykorzystują oferuje znacznie więcej możliwości niż ndbm oraz gdbm. Budowa tej biblioteki jest już zupełnie inna niż ndbm, jednak szereg jej wersji zapewnia interfejs programowy dla ndbm, tzn. możliwość zlinkowania się i pracy programów napisanych dla ndbm.

Terminal i sterownik terminala

Terminal reprezentuje infrastrukturę komunikacji znakowej, przez którą użytkownik może komunikować się z systemem komputerowym. W dawnych systemach komputerowych użytkownicy włączali się do systemu przez rzeczywiste terminale alfanumeryczne i łącza komunikacyjne. W dzisiejszych systemach użytkownicy, włączający się przez sieć lub system okienkowy, mogą nadal posługiwać się emulatorami terminali, dla których system tworzy pseudoterminale symulujące łącza szeregowej komunikacji znakowej.

Programy uruchamiane przez takich użytkowników mogą za pośrednictwem terminala komunikować się z użytkownikami. Dla programu posiadającego dostęp do terminala system tworzy pliki *stdin*, *stdout*, i *stderr*, połączone z klawiaturą i ekranem terminala.

Poza zwykłym przesyłaniem znaków do i od terminala, program może terminalem sterować. Część jądra Unixa odpowiedzialna za komunikację znakową z terminalem nazywa się sterownikiem terminala (*terminal driver*). Jest to podsystem o wielu parametrach konfiguracyjnych. Jego przykładowe funkcje to: buforowanie i edycja wiersza danych, oraz funkcja echa.

Sterowanie terminalem — przykład

```
struct termios org_termios, mod_termios;
int ans;
if (!isatty(fileno(stdin))) {
    fprintf(stderr, "stdin not a terminal, exit.\n");
    exit(1);
}
tcgetattr(fileno(stdin), &org_termios); /* get complete term.settings */
mod_termios = org_termios;             /* make a copy of settings */
mod_termios.c_lflag &= ~ICANON;       /* turn off canonical processing*/
mod_termios.c_cc[VMIN] = 1;           /* return after one char */
mod_termios.c_cc[VTIME] = 0;          /* do not wait for chars */
tcsetattr(fileno(stdin), TCSETS, &mod_termios); /* set new term.modes */
do {
    printf("Select: e(cho), n(oecho), q(uit): ");
    fflush(stdout);
    ans = getchar();
    printf("-- you selected: %c\n", ans);
    if (ans == 'n') mod_termios.c_lflag &= ~ECHO; /* turn off echo bit */
    if (ans == 'e') mod_termios.c_lflag |= ECHO; /* echo bit back on */
    tcsetattr(fileno(stdin), TCSETS, &mod_termios);
    sleep(1);
} while (ans != 'q');
tcsetattr(fileno(stdin), TCSETS, &org_termios); /* restore term.modes */
```


Terminale: termcap/terminfo

Poza przekazywaniem znaków z klawiatury terminala do programu, i z programu na ekran terminala, terminale realizują dodatkowe funkcje. Np. okno terminala może służyć jako wizualny interfejs dla wielu programów, takich jak edytor vi, program top, itp. Rzeczywiste sprzętowe terminale, jak również programowe emulatory terminali, realizują szereg funkcji związanych z wyświetlaniem znaków, jak np. adresowanie kursora, czyli ustawianie kursora znakowego w dowolnej pozycji ekranu. Ponieważ istnieje wiele typów terminali różniących się szczegółami realizacji tych operacji, Unix posiada bazę danych terminali, zwaną terminfo albo termcap, szczegółowo opisujących te funkcje.

```
echo $TERM;          infocmp
tput clear
tput cup 10 10
tput reset           # zapamiętaj te dwa polecenia
tput init
tput bel;           tput flash
tput smul
tput rmul
tput rev;           tput blink
tput bold
tput sgr0           # wyłącza atrybuty znaków
```

Wykorzystanie definicji terminfo — przykład

```
int nrows, ncols;
char *cursor, *clear;
time_t cur_time = time(NULL);

setupterm(NULL, fileno(stdout), (int *)0);
nrows = tigetnum("lines");
ncols = tigetnum("cols");
cursor = tigetstr("cup");
clear = tigetstr("clear");

putp(clear);
printf("Terminal has %d cols and %d rows", ncols, nrows);
putp(tparm(cursor, 0, ncols-31)); /* be careful about last char in line */
printf("ctime=%s", ctime(&cur_time));
putp(tparm(cursor, nrows/2, ncols/2));
printf("X");
putp(tparm(cursor, nrows-2, 0));
printf("____");
putp(tparm(cursor, nrows-1, ncols-4));
printf("END");
putp(tparm(cursor, nrows-1, 0));
```

Biblioteka curses — założenia

Cele i założenia biblioteki curses:

- poziom abstrakcji ułatwiający pisanie programów z interfejsem ekranowym na poziomie wyższym niż `terminfo`, a zwłaszcza niż programowanie sekwencji sterujących konkretnego terminala,
- przenośność źródłowa programów: niezależność od hardware'u komputera i systemu operacyjnego,
- niezależność od środowiska docelowego — osiągnięta przez wykorzystanie bazy danych `terminfo`,
- optymalizacja wyświetlania, kompensująca prędkość transmisji łączy asynchronicznych, wzorowana na kodzie edytora `vi`.

Podstawowe funkcje biblioteki curses:

- adresowanie kursora, wyświetlanie tekstów, aktualizacja ekranu
- czytanie z klawiatury, czytanie z ekranu, odpytywanie o własności terminala

- opcje przy wyświetlaniu (atrybuty znaków)
- opcje przy wczytywaniu (cbreak, noecho, klawisze funkcyjne)
- tworzenie okienek na ekranie i koordynacja ich wyświetlania na ekranie

Biblioteka curses — teoria

- Biblioteka pozwala tworzyć i wyświetlać na ekranie terminala okna, które są prostokątnymi tablicami znaków. Lewy górny narożnik każdego okna ma współrzędne (0,0).
- W chwili rozpoczęcia pracy z biblioteką tworzone jest okno domyślne, które zajmuje cały ekran terminala. Jeśli funkcja nie określa na jakim oknie operuje, to operuje na tym oknie domyślnym.
- Każde okno posiada kursor, który jest zapamiętaną pozycją w oknie. Kursor można przesuwać, a jeśli się go nie przesuwa to pozostaje w danym oknie w pozycji, w której pozostawiła go ostatnia operacja odczytu lub zapisu. Kursorów okien nie należy mylić ze „sprzętowym” kursorem terminala, który jest jeden i wyświetla się w jakimś miejscu na ekranie.
- Operacje pisania w oknach operują na wewnętrznych strukturach biblioteki curses, i jakiegokolwiek operacje wejścia/wyjścia wykonywane są dopiero w chwili wywołania jednej z funkcji `refresh` lub `update`.
- Do biblioteki curses należą również funkcje, które nie wykonują operacji wejścia/wyjścia, lecz ustawiają parametry pracy drivera terminala, który decyduje o sposobie realizacji niektórych operacji I/O, np. buforowaniu.

Biblioteka curses — wyświetlanie tekstów

```
#include <curses.h>

main()
{
    initscr();

    addstr("To jest pierwszy wiersz okna.");
    mvaddstr(1,0,"Nacisnij ENTER.");
    refresh();
    getch();
    move( LINES - 1, COLS - 30 );
    addstr("To jest ostatni wiersz okna.");
    refresh();
    endwin();
}
```

Biblioteka curses — wyświetlanie tekstów

```
#include <curses.h>

main()
{
    initscr();
    addstr("Press RETURN to delete from here to the end of the line and on.");
    addstr("\nDelete this, too.\nAnd this.");
    move(0,32);
    refresh();
    getch();
    clrtoeol();
    refresh();
    endwin();
}
```

Biblioteka curses — czytanie znaku w trybie natychmiastowym

```
#include <curses.h>

main()
{
    int ch;

    initscr();
    cbreak();
    addstr("Press any character: ");
    refresh();
    ch = getch();
   printw("\n\n\nThe character entered was a '%c'.\n", ch);
    refresh();
    endwin();
}
```


Biblioteka curses — czytanie napisów

```
#include <curses.h>

main()
{
char str[256];

    initscr();
    cbreak();
    addstr("Enter a character string terminated by <Return>:\n\n");
    refresh();
    getstr(str);
   printw("\n\n\nThe string entered was \n'%s'\n", str);
    refresh();
    endwin();
}
```

Biblioteka curses — atrybuty wyświetlania

Biblioteka curses pozwala na włączanie i wyłączanie atrybutów wyświetlania, takich jak: podświetlenie (ang. *reverse video*), gruba czcionka, podkreślenie, tekst pulsujący, itp.

Poszczególne atrybuty można włączać i wyłączać selektywnie, bądź ustawiać kompletny wektor atrybutów funkcjami: `attron`, `attroff`, `attrset`.

UWAGA: pomimo iż zdefiniowanych atrybutów wyświetlania jest cały szereg, to nie wszystkie terminale (emulatory terminali) obsługują je wszystkie poprawnie, a dodatkowo w wielu instalacjach Unixa właściwości terminali nie są poprawnie opisane. Dlatego intensywne korzystanie z atrybutów, zwłaszcza gdyby miała od nich zależeć możliwość poprawnego korzystania z programu, jest wysoce ryzykowne. Praktycznie przydatnym atrybutem jest `A_REVERSE`, obsługiwany przez nawet najstarsze terminale tekstowe. W przypadku gdy konieczne jest dodatkowo inne wyróżnienie tekstu, można włączyć atrybut `A_STANDOUT`, co powoduje włączenie najlepiej wyróżniającego tekst atrybutu, wybranego w danej sytuacji.

Biblioteka curses — atrybuty wyświetlania

```
#include <curses.h>

main()
{
char str[256];

    initscr();
    attron(A_REVERSE);
    mvaddstr(0,1,"Wyświetlanie w rewersie");
    mvaddstr(0,40,"w kolumnie 40.");
    attron(A_UNDERLINE);
    mvaddstr(1,1,"Wyświetlanie z podkresleniem");
    mvaddstr(1,40,"w kolumnie 40.");
    attron(A_BLINK);
    mvaddstr(2,1,"Wyświetlanie tekstu migającego");
    mvaddstr(2,40,"w kolumnie 40.");
    attrset(0);
    standout();
    mvaddstr(3,1,"Wyświetlanie z wyróżnieniem");
    mvaddstr(3,40,"w kolumnie 40.");
    refresh();
    endwin();
}
```

Biblioteka curses — użycie kolorów

```
#include <curses.h>

main()
{
char str[256];

    initscr();
    if (has_colors()==FALSE) {
        endwin();
        printf("Sorry, no colors on this terminal!\n");
        exit(1);
    }
    start_color();
    init_pair(1, COLOR_RED, COLOR_BLACK);
    init_pair(2, COLOR_YELLOW, COLOR_BLUE);
    attron(COLOR_PAIR(1));
    mvaddstr(1, 1, "Display in color");
    attroff(COLOR_PAIR(1));
    attron(COLOR_PAIR(2));
    mvaddstr(3, 1, "Display in a different color");
    attroff(COLOR_PAIR(2));
    getch();
}
```

```

attron(A_REVERSE);
attron(COLOR_PAIR(1));
mvaddstr(5, 1, "Display in color in reverse");
attroff(COLOR_PAIR(1));
attron(COLOR_PAIR(2));
mvaddstr(7, 1, "Display in another color in reverse");
attroff(COLOR_PAIR(2));
getch();
if (can_change_color()==FALSE) {
    endwin();
    printf("Sorry, no color definitions on this terminal!\n");
    exit(2);
}
init_color(COLOR_RED, 1000, 753, 796); /* pink */
init_color(COLOR_BLUE, 359, 463, 546); /* navy blue */
init_color(COLOR_YELLOW, 980, 980, 824); /* goldenrod */
getch();
endwin();
}

```

Biblioteka curses — czytanie klawiszy funkcyjnych

Klawisze funkcyjne: strzałki, F1, F2, itp., PageUp/PageDown, Home/End, itp.

Funkcje: keypad, getch

Biblioteka curses — tworzenie okienek

```
#include <curses.h>
```

```
main()
```

```
{
```

```
    WINDOW *w1, *w2;
```

```
    initscr();
```

```
    w1 = newwin (2,6,0,3);
```

```
    w2 = newwin (1,4,5,4);
```

```
    waddstr(w1, "Bulls");
```

```
    wnoutrefresh(w1);
```

```
    waddstr(w2, "Eye");
```

```
    wnoutrefresh(w2);
```

```
    doupdate();
```

```
    endwin();
```

```
}
```

Biblioteka curses — tworzenie okienek (cd.)

Tworzenie okienek będących „podokienkami” innych okienek:

```
#include <curses.h>

main()
{
    WINDOW *sub;

    initscr();
    box(stdscr, 'w', 'w'); /* See the curses(3X) manual page for box() */
    mvwaddstr(stdscr, 7, 10, "-----this is 10,10");
    mvwaddch(stdscr, 8, 10, '|');
    mvwaddch(stdscr, 9, 10, 'v');
    sub = subwin(stdscr, 10, 20, 10, 10);
    box(sub, 's', 's');
    wnoutrefresh(sub);
    wrefresh(sub);
    endwin();
}
```


Biblioteka curses — interakcje między okienkami

```
#include <curses.h>

WINDOW *cmdwin;

main() {
    int i, c;
    char buf[120];
    void exit();

    initscr();
    nonl();
    noecho();
    cbreak();

    cmdwin = newwin(3, COLS, 0, 0);    /* top 3 lines */
    for (i = 0; i < LINES; i++)
        mvprintw(i, 0, "This is line %d of stdscr", i);

    for (;;) {
        refresh();
        c = getch();
        switch (c) {
```

```

case 'c':          /* Enter command from keyboard */
    werase(cmdwin);
    wprintw(cmdwin, "Enter command:");
    wmove(cmdwin, 2, 0);
    for (i = 0; i < COLS; i++)
        waddch(cmdwin, '-');
    wmove(cmdwin, 1, 0);
    touchwin(cmdwin);
    wrefresh(cmdwin);
    echo();
    wgetstr(cmdwin, buf);
    noecho();
    touchwin(stdscr);
    break;

    /*
     * The command is now in buf.
     * It should be processed here.
     */
case 'q':
    endwin();
    exit(0);
}
}
}

```

Biblioteka curses — aktualizacja ekranu

Aktualizacja wyświetlanych na ekranie okienek: wrefresh, wnoutrefresh, douppdate

Biblioteka curses — operacje na zawartości ekranu

Odczytywanie znaków z okienek: `inch`, `instr`, `innstr`

Kasowanie znaków w okienku (z przesunięciem tekstu w lewo): `delch`

Biblioteka curses — sprawdzanie parametrów okien

Odczytywanie parametrów okienek i cursorów

Funkcje: `getyx`, `getparyx` (położenie kursora okna, i okna nadrzędnego), `getbegyx`, `getmaxyx` (położenie i rozmiar okna)

Biblioteka curses — programowanie drivera terminala

Programowanie parametrów drivera terminala: echo/noecho, nl/nonl, raw/noraw, cbreak/nocbreak, halfdelay, nodelay

Biblioteka curses — praca w systemie okienkowym

Wykrywanie zmian rozmiarów terminala w systemie okienkowym, i reinicjalizacja struktur biblioteki curses

Biblioteka curses — program highlight

```
/*
 * highlight: a program to turn \U, \B, and
 * \N sequences into highlighted
 * output, allowing words to be displayed
 * underlined or bolded.
 */

#include <stdio.h>
#include <curses.h>

main(int argc, char **argv)
{
    FILE *fd;
    int c, c2;
    void exit(), perror();

    if (argc != 2) {
        fprintf(stderr, "Usage: highlight file\n");
        exit(1);
    }

    fd = fopen(argv[1], "r");
```



```
if (fd == NULL) {
    perror(argv[1]);
    exit(2);
}

initscr();
scrollok(stdscr, TRUE);
nonl();
while ((c = getc(fd)) != EOF) {
    if (c == '\\') {
        c2 = getc(fd);
        switch (c2) {
            case 'B':
                attrset(A_BOLD);
                continue;
            case 'U':
                attrset(A_UNDERLINE);
                continue;
            case 'N':
                attrset(0);
                continue;
        }
        addch(c);
        addch(c2);
    }
}
```

```
    else
        addch(c);
}
fclose(fd);
refresh();
endwin();
exit(0);
}
```


Biblioteka curses — program scatter

```
/*
 *   The scatter program.
 *
#include <curses.h>
#include <sys/types.h>

extern time_t time();

#define MAXLINES 120
#define MAXCOLS 160
char s[MAXLINES] [MAXCOLS];    /* Screen Array */
int T[MAXLINES] [MAXCOLS];    /* Tag Array -- Keeps track of */
                                /* the number of characters */
                                /* printed and their positions. */

main()
{
    register int row = 0,col = 0;
    register int c;
    int char_count = 0;
    time_t t;
    void exit(), srand();
```

```

initscr();
for(row = 0,row < MAXLINES;row++)
    for(col = 0;col < MAXCOLS;col++)
        s[row][col]=' ';
col = row = 0;
/* Read screen in */
while ((c=getchar()) != EOF && row < LINES ) {

    if(c != '\n') {
        /* Place char in screen array */
        s[row][col++] = c;
        if(c != ' ')
            char_count++;
    }
    else {
        col = 0;
        row++;
    }
}

time(&t); /* Seed the random number generator */
srand((unsigned)t);

while (char_count) {
    row = rand() % LINES;

```

```
col = (rand() >> 2) % COLS;
if (T[row][col] != 1 && s[row][col] != ' ') {
    move(row, col);
    addch(s[row][col]);
    T[row][col] = 1;
    char_count--;
    refresh();
}
}
endwin();
exit(0);
}
```


Biblioteka curses — pełnoekranowa wyświetlarka plików tekstowych

```
#include <curses.h>
#include <signal.h>

main(argc, argv)
    int argc;
    char *argv[];
{
    FILE *fd;
    char linebuf[BUFSIZ];
    int line;
    void done(), perror(), exit();

    if (argc != 2) {
        fprintf(stderr, "usage: %s file\n", argv[0]);
        exit(1);
    }

    if ((fd=fopen(argv[1], "r")) == NULL) {
        perror(argv[1]);
        exit(2);
    }
}
```



```
signal(SIGINT, done);

initscr();
noecho();
cbreak();
nonl();
idlok(stdscr, TRUE);

while(1) {
    move(0,0);
    for (line = 0; line < LINES; line++) {

        if (!fgets(linebuf, sizeof linebuf, fd)) {
            clrtoeol();
            done();
        }
        move(line, 0);
        printw("%s", linebuf);
    }
    refresh();
    if (getch() = 'q')
        done();
}
}
```

```
void done()
{
    move(LINES - 1, 0);
    clrtoeol();
    refresh();
    endwin();
    exit(0);
}
```


Biblioteka curses — inna wersja programu highlight

Operacje konieczne w programie wykorzystującym struktury terminfo:

```
#include <curses.h>
#include <term.h>
...
    setupterm( (char*)0, 1, (int*)0 );
...
    putp(clear_screen);
...
    reset_shell_mode();
exit(0);

/*
 *      A terminfo-level version of the highlight program.
 */

#include <curses.h>
#include <term.h>

int ulmode = 0;                                /* Currently underlining */

main(argc, argv)
    int argc;
    char **argv;
```

```

{
FILE *fd;
int c, c2;
int outch();

if (argc > 2) {
    fprintf(stderr, "Usage: termhl [file]\n");
    exit(1);
}

if (argc == 2) {
    fd = fopen(argv[1], "r");
    if (fd == NULL) {
        perror(argv[1]);
        exit(2);
    }
}
else {
    fd = stdin;
}
setupterm((char*)0,1, (int*)0);

for (;;) {
    c = getc(fd);
    if (c == EOF)

```

```

    break;
if (c == '\\') {
    c2 = getc(fd);
    switch(c2) {
    case 'B':
        tputs(enter_bold_mode, 1, outch);
        continue;
    case 'U':
        tputs(enter_underline_mode, 1, outch);
        ulmode = 1;
        continue;
    case 'N':
        tputs(exit_attribute_mode, 1, outch);
        ulmode = 0;
        continue;
    }
    putch(c);
    putch(c2);
}
else
    putch(c);
}
fclose(fd);
fflush(stdout);
resetterm();

```

```

    exit(0);
}

/*
 *   This function is like putchar, but it checks for underlining.
 */
putch(c)
    int c;
{
    outch(c);
    if (ulmode && underline_char) {
        outch('\b');
        tputs(underline_char, 1, outch);
    }
}

/*
 *   Outchar is a function version of putchar that can be passed to
 *   tputs as a routine to call.
 */
outch(c)
    int c;
{
    putchar(c);
}

```