PDCurses

PDCurses Implementor's Guide

- Version 1.6 - 2019/09/?? - added PDC_doupdate(); removed argc, argv, lines, cols and SP allocation from PDC_scr_open(); removed PDC_init_pair(), PDC_pair_content()
- Version 1.5 - 2019/09/06 - PDC_has_mouse(), removed PDC_get_input_fd()
- Version 1.4 - 2018/12/31 - PDCurses.md -> USERS.md, MANUAL.md; new dir
- Version 1.3 - 2018/01/12 - notes about official ports, new indentation style; markdown
- Version 1.2 - 2007/07/11 - added PDC_init_pair(), PDC_pair_content(), version history; removed pdc_atrtab
- Version 1.1 - 2007/06/06 - minor cosmetic change
- Version 1.0 - 2007/04/01 - initial revision

This document is for those wishing to port PDCurses to a new platform, or just wanting to better understand how it works. Nothing here should be needed for application programming; for that, refer to USERS.md and MANUAL.md, in man/ . This document assumes that you've read the user- level documentation and are very familiar with application-level curses programming.

If you want to submit your port for possible inclusion into the main PDCurses distribution, please follow these guidelines:

- Don't modify anything in the pdcurses directory or in other port directories. Don't modify curses.h or curspriv.h unless absolutely necessary. (And prefer modifying curspriv.h over curses.h.)
- Use the same indentation style, naming and scope conventions as the existing code.
- Release all your code to the public domain – no copyright. Code under GPL, BSD, etc. will not be accepted.

Data Structures

A port of PDCurses must provide acs_map[], a 128-element array of chtypes, with values laid out based on the Alternate Character Set of the VT100 (see curses.h). PDC_transform_line() must use this table; when it encounters a chtype with the A_ALTCHARSET flag set, and an A_CHARTEXT value in the range 0-127, it must render it using the A_CHARTEXT portion of the corresponding value from this table, instead of the original value. Also, values may be read from this table by apps, and passed through functions such as waddch(), which does no special processing on control characters (0-31 and 127) when the A_ALTCHARSET flag is set. Thus,

any control characters used in acs_map[] should also have the A_ALTCHARSET flag set. Implementations should provide suitable values for all the ACS_ macros defined in curses.h; other values in the table should be filled with their own indices (e.g., acs_map['E'] == 'E'). The table can be either hardwired, or filled by PDC_scr_open(). Existing ports define it in pdcdisp.c, but this is not required.

Functions

A port of PDCurses must implement the following functions, with extern scope. These functions are traditionally divided into several modules, as indicated below; this division is not required (only the functions are), but may make it easier to follow for someone familiar with the existing ports.

Any other functions you create as part of your implementation should have static scope, if possible. If they can't be static, they should be named with the "PDC_" prefix. This minimizes the risk of collision with an application's choices.

Current PDCurses style also uses a single leading underscore with the name of any static function; and modified BSD/Allman-style indentation, approximately equivalent to "indent -kr -nut -bl -bli0", with adjustments to keep every line under 80 columns.

pdcdisp.c:

void PDC_doupdate(void);

Called at the end of doupdate(), this function finalizes the update of the physical screen to match the virtual screen, if necessary, i.e. if updates were deferred in PDC_transform_line().

void PDC_gotoyx(int y, int x);

Move the physical cursor (as opposed to the logical cursor affected by wmove()) to the given location. This is called mainly from doupdate(). In general, this function need not compare the old location with the new one, and should just move the cursor unconditionally.

void PDC_transform_line(int lineno, int x, int len, const chtype *srcp);

The core output routine. It takes len chtype entities from srcp (a pointer into curscr) and renders them to the physical screen at line lineno, column x. It must also translate characters 0-127 via acs_map[], if they're flagged with A_ALTCHARSET in the attribute portion of the chtype. Actual screen updates may be deferred until PDC_doupdate() if desired (currently done with SDL and X11).

pdcgetsc.c:

int PDC_get_columns(void);

Returns the size of the screen in columns. It's used in initscr() and resize_term() to set the value of COLS.

int PDC_get_cursor_mode(void);

Returns the size/shape of the cursor. The format of the result is unspecified, except that it must be returned as an int. This function is called from initscr(), and the result is stored in SP->orig_cursor, which is used by PDC_curs_set() to determine the size/shape of the cursor in normal visibility mode (curs_set(1)).

int PDC_get_rows(void);

Returns the size of the screen in rows. It's used in initscr() and resize_term() to set the value of LINES.

pdckbd.c:

bool PDC_check_key(void);

Keyboard/mouse event check, called from wgetch(). Returns TRUE if there's an event ready to process. This function must be non-blocking.

void PDC_flushinp(void);

This is the core of flushinp(). It discards any pending key or mouse events, removing them from any internal queue and from the OS queue, if applicable.

int PDC_get_key(void);

Get the next available key, or mouse event (indicated by a return of KEY_MOUSE), and remove it from the OS' input queue, if applicable. This function is called from wgetch(). This function may be blocking, and traditionally is; but it need not be. If a valid key or mouse event cannot be returned, for any reason, this function returns -1. Valid keys are those that fall within the appropriate character set, or are in the list of special keys found in curses.h (KEY_MIN through KEY_MAX). When returning a special key code, this routine must also set SP->key_code to TRUE; otherwise it must set it to FALSE. If SP->return_key_modifiers is TRUE, this function may return modifier keys (shift, control, alt), pressed alone, as special key codes; if SP->return_key_modifiers is FALSE, it must not. If modifier keys are returned, it should only happen if no other keys were pressed in the meantime; i.e., the return should happen on key up. But if this is not possible, it may return the modifier keys on key down (if and only if SP->return_key_modifiers is TRUE).

bool PDC_has_mouse(void);

Called from has_mouse(). Reports whether mouse support is available. Can be a static TRUE or FALSE, or dependent on conditions. Note: Activating mouse support should depend only on PDC_mouse_set(); don't expect the user to call has_mouse() first.

int PDC_modifiers_set(void);

Called from PDC_return_key_modifiers(). If your platform needs to do anything in response to a change in SP->return_key_modifiers, do it here. Returns OK or ERR, which is passed on by the caller.

int PDC_mouse_set(void);

Called by mouse_set(), mouse_on(), and mouse_off() – all the functions that modify SP->_trap_mbe. If your platform needs to do anything in response to a change in SP->_trap_mbe (for example, turning the mouse cursor on or off), do it here. Returns OK or ERR, which is passed on by the caller.

void PDC_set_keyboard_binary(bool on);

Set keyboard input to "binary" mode. If you need to do something to keep the OS from processing ^C, etc. on your platform, do it here. TRUE turns the mode on; FALSE reverts it. This function is called from raw() and noraw().

pdcscrn.c:

bool PDC_can_change_color(void);

Returns TRUE if init_color() and color_content() give meaningful results, FALSE otherwise. Called from can_change_color().

int PDC_color_content(short color, short *red, short *green, short *blue);

The core of color_content(). This does all the work of that function, except checking for values out of range and null pointers.

int PDC_init_color(short color, short red, short green, short blue);

The core of init_color(). This does all the work of that function, except checking for values out of range.

void PDC_reset_prog_mode(void);

The non-portable functionality of reset_prog_mode() is handled here – whatever's not done in _restore_mode(). In current ports: In OS/2, this sets the keyboard to binary mode; in Windows console, it enables or disables the mouse pointer to match the saved mode; in others it does nothing.

void PDC_reset_shell_mode(void);

The same thing, for reset_shell_mode(). In OS/2 and Windows console, it restores the default console mode; in others it does nothing.

int PDC_resize_screen(int nlines, int ncols);

This does the main work of resize_term(). It may respond to non-zero parameters, by setting the screen to the specified size; to zero parameters, by setting the screen to a size chosen by the user at runtime, in an unspecified way (e.g., by dragging the edges of the window); or both. It may also do nothing, if there's no appropriate action for the platform.

void PDC_restore_screen_mode(int i);

Called from _restore_mode() in kernel.c, this function does the actual mode changing, if applicable. Currently used only in DOS and OS/2.

void PDC_save_screen_mode(int i);

Called from _save_mode() in kernel.c, this function saves the actual screen mode, if applicable. Currently used only in DOS and OS/2.

void PDC_scr_close(void);

The platform-specific part of endwin(). It may restore the image of the original screen saved by PDC_scr_open(), if the PDC_RESTORE_SCREEN environment variable is set; either way, if using an existing terminal, this function should restore it to the mode it had at startup, and move the cursor to the lower left corner. (The X11 port does nothing.)

void PDC_scr_free(void);

Free any memory allocated by PDC_scr_open(). Called by delscreen().

int PDC_scr_open(void);

The platform-specific part of initscr(). It must initialize acs_map[] (unless it's preset) and several members of SP, including mouse_wait, orig_attr (and if orig_attr is TRUE, orig_fore and orig_back), mono, _restore and _preserve. If using an existing terminal, and the environment variable PDC_RESTORE_SCREEN is set, this function may also store the existing screen image for later restoration by PDC_scr_close().

pdcsetsc.c:

int PDC_curs_set(int visibility);

Called from curs_set(). Changes the appearance of the cursor – 0 turns it off, 1 is normal (the terminal's default, if applicable, as determined by SP->orig_cursor), and 2 is high visibility. The exact appearance of these modes is not specified.

pdcutil.c:

void PDC_beep(void);

Emits a short audible beep. If this is not possible on your platform, you must set SP->audible to FALSE during initialization (i.e., from PDC_scr_open() – not here); otherwise, set it to TRUE. This function is called from beep().

void PDC_napms(int ms);

This is the core delay routine, called by napms(). It pauses for about (the X/Open spec says "at least") ms milliseconds, then returns. High degrees of accuracy and precision are not expected (though desirable, if you can achieve them). More important is that this function gives back the process' time slice to the OS, so that PDCurses idles at low CPU usage.

const char *PDC_sysname(void);

Returns a short string describing the platform, such as "DOS" or "X11". This is used by longname(). It must be no more than 100 characters; it should be much, much shorter (existing platforms use no more than 5).

More functions

The following functions are implemented in the platform directories, but are accessed directly by apps. Refer to the user documentation for their descriptions:

pdcclip.c:

int PDC_clearclipboard(void);

int PDC_freeclipboard(char *contents);

int PDC_getclipboard(char **contents, long *length);

int PDC_setclipboard(const char *contents, long length);

pdcsetsc.c:

int PDC_set_blink(bool blinkon);

int PDC_set_bold(bool boldon);

void PDC_set_title(const char *title);