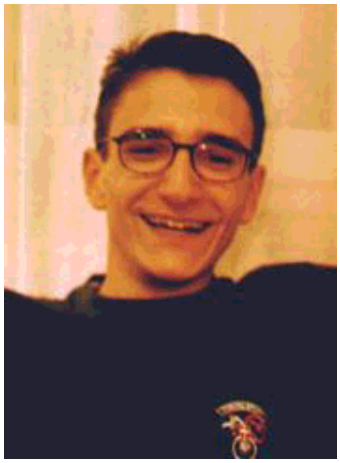
LinuxFocus article number 233
http://linuxfocus.org



par Reha K. Gerçeker
<gerceker/at/itu.edu.tr>

*L´auteur:*

Reha est étudiant en informatique à Istanbul, Turquie. Il aime la liberté que procure Linux pour le développement de logiciels. Il passe beaucoup de son temps à écrire des programmes sur son ordinateur. Il souhaite devenir un jour un programmeur de talent.

*Traduit en Français par:*
Paul Delannoy (homepage)

# Introduction à Ncurses


*Résumé*:

Ncurses est une bibliothèque qui fournit des fonctions de définition de touches du clavier, de couleurs d'écran et permet l'affichage de plusieurs fenêtres sans recouvrement sur un terminal texte.

## Qu'est ce que Ncurses?

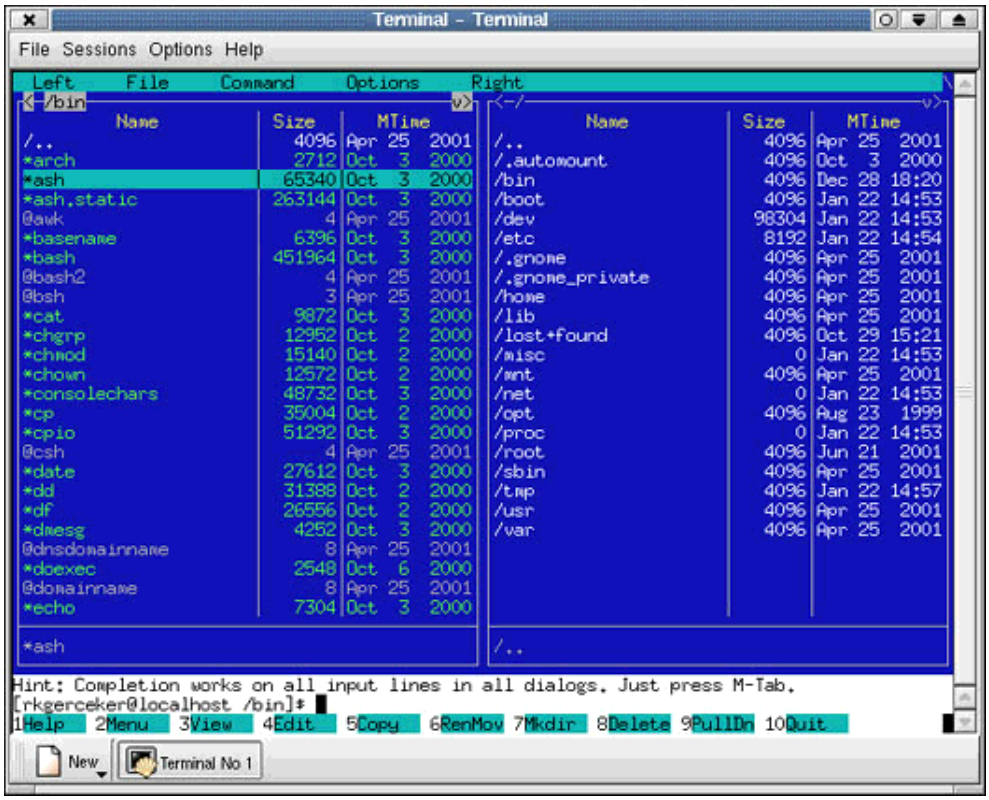
Voulez vous que vos programmes disposent d'une interface couleur sur un terminal texte ? Ncurses est une bibliothèque qui offre ce type de fonctionnalité pour les terminaux en mode texte. Ncurses peut :

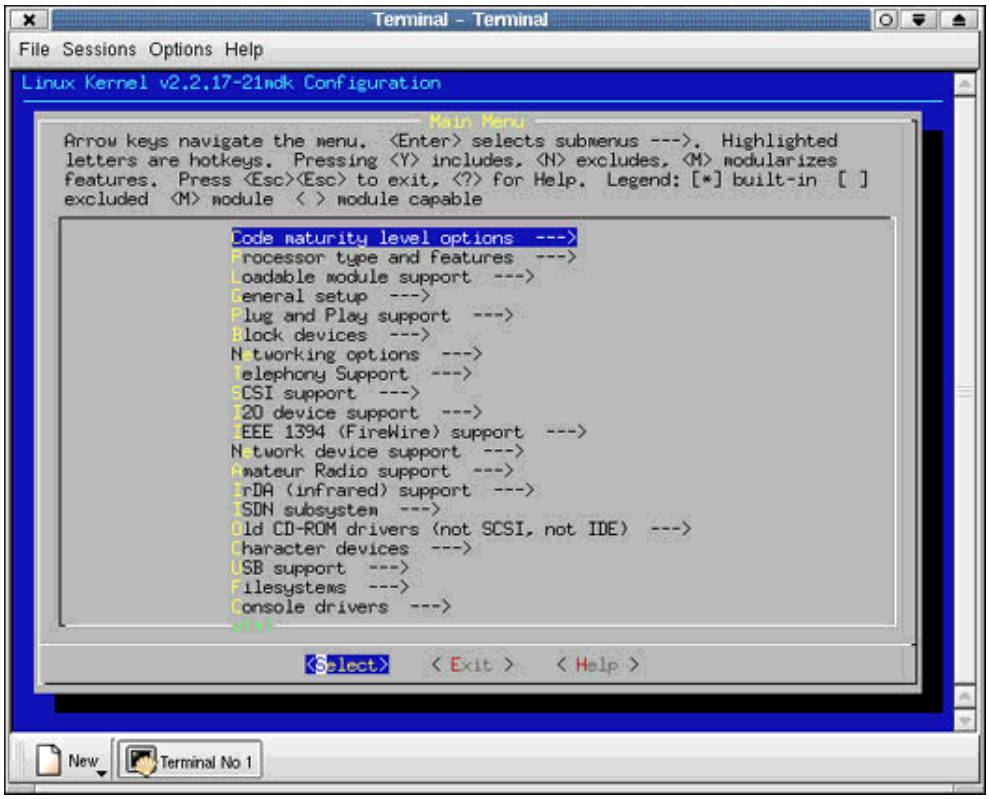
- Utiliser tout l'écran si vous le souhaitez.
- Créer et gérer des fenêtres.

- Utiliser 8 couleurs différentes.
- Permettre le contrôle de votre programme par la souris.
- Utiliser les touches de fonction du clavier.

Tout système UNIX conforme à la norme ANSI/POSIX supporte la bibliothèque Ncurses. De ce fait, elle est capable de détecter les propriétés du terminal à partir de la base de données du système, et donc de fournir une interface indépendante du terminal. Ainsi, ncurses est utilisable et fiable pour des travaux devant fonctionner sur différentes plates-formes et différents types de terminaux.

Un exemple type est Midnight Commander. De même l'interface de configuration du noyau à partir de la console est écrite avec ncurses. Voir ci-dessous des copies d'écran.






# Où le télécharger ?

Ncurses est développé sous GNU/Linux. Pour télécharger la dernière version, obtenir des informations détaillées, trouver des liens relatifs à ncurses, visitez www.gnu.org/software/ncurses/.

# Eléments de base

Pour pouvoir utiliser cette bibliothèque, vous devrez inclure *curses.h* dans votre code source et bien lier ce code à la bibliothèque curses. Cela s'obtient en donnant le paramètre -lcurses à gcc.

Pour travailler avec ncurses, il faut bien comprendre la structure de données fondamentale. Il s'agit de la structure WINDOW, et comme le nom l'indique, elle est utilisée pour représenter les fenêtres que vous allez créer. Pratiquement toutes les fonctions auront besoin d'un pointeur de type WINDOW comme paramètre.

Les éléments le plus souvent utilisés avec ncurses sont des fenêtres. Et même si vous ne créez pas vos propres fenêtres, l'écran est lui-même considéré comme une fenêtre. De même que le descripteur, de type FILE,*stdout* de la bibliothèque d'entrée/sortie standard représente l'écran (du moins en l'abscence de redirections), ncurses dispose du pointeur de type WINDOW *stdscr* qui a le même rôle. Outre *stdscr*, un autre pointeur de type WINDOW nommé *curscr* est défini dans la bibliothèque. De même que *stdscr* représente l'écran, *curscr* représente l'écran 'courant' pour la bibliothèque. Vous pouvez vous demander "Quelle est la différence ?". Continuez à lire.

Afin d'utiliser les fonctions et les variables dans vos programmes, un appel à la fonction *initscr*est obligatoire. Elle alloue la mémoire nécessaire aux variables telles que *stdscr*, *curscr* et initialise la bibliothèque. Autrement dit, toutes les fonctions de ncurses doivent être précédées de *initscr*. De façon analogue, vous devrez appeler la fonction *endwin* quand vous n'aurez plus besoin de ncurses. Cela libère la mémoire occupée par ncurses. Bien évidemment, après l'appel de *endwin* vous n'aurez plus accès aux fonctions ncurses, sinon en rappelant *initscr*.

Entre l'appel de *initscr* et celui de *endwin*, assurez-vous de ne rien envoyer à l'écran par l'intermédiaire de la bibliothèque d'entrée/sortie standard. Sinon vous obtiendrez un affichage indésirable et habituellement corrompu. Dès que ncurses est actif, utilisez ses fonctions pour envoyer des données à l'écran. Avant l'appel de *initscr* ou après celui de *endwin*, faites ce qui vous plaît.

# Mise à jour de l'affichage : rafraîchissement

La structure de type WINDOW ne sert pas seulement à conserver les dimensions et la position d'une fenêtre, elle gère également son contenu. Pour écrire dans une fenêtre, vous modifiez son contenu, mais ce changement n'apparaît pas immédiatement à l'écran. Pour mettre à jour l'affichage, vous devez faire un appel à *refresh* ou à *wrefresh*.

Voici la différence entre *stdscr* et *curscr*. Alors que *curscr* conserve le contenu de l'écran courant, *stdscr* peut comporter des informations différentes à la suite d'appels à des fonctions d'affichage ncurses. Si vous voulez transférer les derniers changements de *stdscr* vers *curscr*, vous devez appeler *refresh*. Autrement dit, *refresh* est la seule fonction qui puisse dialoguer avec *curscr*. Il vaut donc mieux ne pas bidouiller avec *curscr* et laisser la fonction *refresh* s'occuper de sa mise à jour.

*refresh* dispose d'un mécanisme pour rafraîchir l'écran le plus vite possible. En effet, lors d'un appel, elle ne met à jour que les lignes de la fenêtre qui ont été modifiées. Cela économise du temps CPU et évite au programme de devoir répéter l'affichage d'informations inchangées. C'est la raison pour laquelle l'usage simultané de fonctions ncurses et de fonctions de la bibliothèque d'entrée/sortie standard produit des résultats erronnés; les fonctions ncurses positionnent un drapeau par ligne qui informe *refresh* que telle ligne a été modifiée, alors que rien de tel ne se produit lors de l'appel d'une fonction de la bibliothèque d'entrée/sortie standard.

*refresh* et *wrefresh* font en fait la même chose. *wrefresh* prend un pointeur de type WINDOW en paramétre et ne rafraîchit que le contenu de la fenêtre correspondante. L'appel à *refresh()* équivaut à celui de *wrefresh(stdscr)*. Nous verrons plus loin que, comme *wrefresh*, la plupart des fonctions de ncurses contiennent des macros qui permettent de les appliquer à *stdscr*.

# Créer des fenêtres

Parlons maintenant de *subwin* et de *newwin*, les fonctions qui créent de nouvelles fenêtres. Les deux réclament comme paramètres la hauteur, la largeur, et les coordonnées du coin supérieur gauche de la

fenêtre à créer. Elles renvoient un pointeur WINDOW représentant la nouvelle fenêtre. Vous pouvez utiliser ce pointeur avec *wrefresh* et d'autres fonctions dont nous parlerons plus loin.

"Si elles font la même chose, pourquoi en proposer deux ?" me demanderez-vous. C'est vrai qu'elles sont un peu différentes. *subwin* crée une fenêtre comme sous-fenêtre d'une fenêtre existante. La fenêtre ainsi créée hérite des propriétés de sa fenêtre mère. Mais ces propriétés peuvent être modifiées sans affecter la fenêtre mère.

Il existe toutefois quelque chose qui lie les fenêtres mère et fille. Le tableau de caractères qui gère le contenu d'une fenêtre est partagé entre les fenêtres mère et fille. Autrement dit, les caractères de leur frontière commune peuvent être changés par n'importe laquelle des deux. Si la mère y écrit, le contenu de la fille est altéré, et réciproquement.

Au contraire de *subwin*, *newwin* crée vraiment une nouvelle fenêtre. Cette fenêtre ne partage pas le tableau de caractères, sauf si elle possède ses propres sous-fenêtres. L'avantage procuré par *subwin* est que le partage d'un tableau de caractéres économise de la mémoire. Par contre, si les fenêtres doivent rester indépendantes, *newwin* est avantageux.

Le sous-fenêtrage n'est pas limité en profondeur. Chaque sous-fenêtre peut créer ses propres sous-fenêtres, mais faites attention au fait que plus de deux fenêtres utilisent alors le même tableau de caractères.

La fonction *delwin* libère la fenêtre dont vous n'avez plus besoin. Consultezez les pages de manuel pour découvrir la liste des paramètres possibles.

## Ecrire dans des fenêtres, lire depuis des fenêtres

Nous avons parlé de *stdscr*, *curscr*, du rafraîchissement d'écran et de la création de nouvelles fenêtres. Mais comment écrit-on dans une fenêtre ? Ou, comment lit-on les données d'une fenêtre ?

Les fonctions correspondantes ressemblent à leur pendant de la bibliothèque d'entrée/sortie standard. Parmi elles on trouve *printw* à la place de printf, *scanw* au lieu de scanf, *addch* au lieu de putc ou putchar, *getch* au lieu de getc ou getchar. Leur usage est le même, seuls les noms diffèrent. De même, *addstr* sera utilisée pour écrire une chaîne dans une fenêtre et *getstr* pour lire une chaîne depuis une fenêtre. Si vous ajoutez une lettre 'w' devant ces noms, et que vous passiez un pointeur de type WINDOW comme premier paramètre, ces fonctions travailleront sur une fenêtre différente de *stdscr*. Ainsi par exemple, *printw(...)* et *wprintw(stdscr, ...)* sont équivalents,comme le sont *refresh()* et *wrefresh(stdscr)*.

Entrer dans le détail de ces fonctions serait une longue histoire. Les pages de manuel sont la meilleure source pour apprendre leurs descriptions, prototypes, valeurs de retour et autres notes. Je vous encourage à lire les pages de manuel pour chaque fonction que vous voulez utiliser. Leur information est préci(eu)se. L'exemple développé dans la dernière partie de cet article constitue également un tutoriel sur l'utilisation des dites fonctions..

# Curseur physique et curseurs logiques

Après avoir parlé des entrées/sorties liées à une fenêtre, il est nécessaire de bien distinguer le curseur physique des curseurs logiques. Le curseur physique est le caractère clignotant visible dans l'écran, et il est unique. Au contraire, un curseur logique est lié à chaque fenêtre ncurses. Il peut donc y en avoir plusieurs.

Ce curseur logique se situe à la position où doit commencer la prochaine lecture ou écriture. Comme vous pouvez le dépacer à votre guise, vous pouvez écrire à n'importe quel endroit d'une fenêtre quand vous le voulez. C'est l'avantage de ncurses par rapport à la bibliothèque d'entrée/sortie standard.

Pour déplacer un curseur logique, appelez *move* ou, comme vous pouvez le deviner, *wmove*. *move* est une macro de *wmove* écrite pour *stdscr*.

Il est intéressant de pouvoir coordonner les curseurs physique et logiques. Après une écriture, la position du curseur logique est déterminée par le drapeau _leave présent dans la structure de type WINDOW. Si le drapeau *_leave*, est défini, la position du curseur logique devient celle du curseur physique (position du dernier caractère écrit). Si le drapeau *_leave* n'est pas défini, c'est le curseur physique qui revient à la position du curseur logique (où le premier caractère est écrit). Le drapeau *_leave* est contrôlé par la fonction *leaveok* .

Pour déplacer le curseur physique, la fonction *mvcur* est utilisée. Contrairement à d'autres, *mvcur* agit sans attendre le prochain *refresh.* Pour rendre le curseur physique invisible, utilisez *curs_set*. Vous aurez plus de détails en tapant "man ncurses".

Des macros existent combinant écriture et déplacement du curseur. Elles sont décrites dans les pages de manuel aux chapitres concernant *addch, addstr, printw, getch, getstr, scanw* etc.

# Nettoyer les fenêtres

Nous avons écrit dans des fenêtres. Mais comment nettoyons-nous les fenêtres, les lignes ou les caractères ?

Dans ncurses, nettoyer signifie remplir la zone, la ligne ou le contenu par des espaces. Les fonctions décrites ci-dessous remplissent la zone appropriée par des espaces et par conséquent nettoient l'écran.

Commençons par un caractère ou une ligne. Les fonctions *delch* et *wdelch* suppriment le caractère sous le curseur logique de la fenêtre et décalent les suivants vers la gauche. *deleteln* et *wdeleteln* effacent la ligne où se trouve le curseur logique et remontent les suivantes d'une ligne.

Les fonctions *clrtoeol* et *wclrtoeol* effacent les caractères à droite du curseur logique jusqu'à la fin de la ligne. *clrtobot* et *wclrtobot* appellent d'abord *wclrtoeol* pour effacer la fin de la ligne à partir du curseur puis détruisent les lignes suivantes.

Enfin, vous pouvez effacer l'écran entier ou une fenêtre. Deux méthodes pour effacer l'écran entier : la première consiste à remplir la zone par des espaces puis d'appeler *refresh* et la seconde consiste à utiliser le code de contrôle propre au terminal. La seconde est bien sûr beaucoup plus rapide puisque la première nécessite une réécriture de tous les caractères de l'écran.

*erase* et *werase* remplissent le tableau de caractères d'une fenêtre par des espaces. La fenêtre sera vide au prochain rafraîchissement. Il n'est pas très malin de s'en servir si la fenêtre à nettoyer remplit tout l'écran. Elles utilisent la première des deux méthodes décrites ci-dessus. Pour une fenêtre occupant tout l'écran, il vaut mieux utiliser les fonctions ci-dessous.

Mais avant d'aborder de nouvelles fonctions, parlons du drapeau *_clear* . Il est défini dans la structure de type WINDOW et s'il est positionné, il force refresh à envoyer le code de contrôle au terminal lors de son appel. Lorsqu'il est appelé, *refresh* détermine (grâce à la valeur du drapeau _FULLWIN) si la fenêtre occupe tout l'écran, et si c'est le cas, il vide l'écran en se servant de la méthode propre au terminal. Ainsi l'effacement est bien plus rapide. La raison pour laquelle la méthode interne du terminal ne s'applique qu'aux fenêtres plein écran vient de ce que le code de contrôle du terminal nettoie l'écran entier et non la fenêtre proprement dite. Le drapeau *_clear* est géré par la fonction*clearok*.

Les fonctions *clear* et *wclear* sont utilisées pour vider les fenêtres plein écran. Elles sont équivalentes à un appel de *werase* puis de *clearok*. Le tableau de caractères de la fenêtre est d'abord rempli par des espaces. Ensuite le drapeau *_clear* étant positionné, elles nettoient l'écran par la méthode interne du terminal si la fenêtre est plein écran, ou elles rafraichissent tous les caractères de la fenêtre en les remplissant par des espaces.

Donc, si vous savez que la fenêtre occupe tout l'écran, utilisez *clear* ou *wclear*. Cela va plus vite. Sinon, il n'y a pas de différence entre les deux types de fonctions.

# Utiliser la couleur

Vous devez penser aux couleurs de l'affichage comme à des paires. En effet chaque caractère comporte une couleur de fond et une couleur d'écriture. Ainsi, écrire en couleurs avec ncurses implique la définition de vos paires de couleur et de leur utilisation dans l'écriture vers une fenêtre.

De même qu'un appel de *initscr* est indispensable pour lancer ncurses, *start_color* doit être appelé pour l'initialisation des couleurs. Pour définir vos paires de couleur, vous vous servez d'*init_pair*. Lorsque vous créez une paire de couleurs par init_pair, celle-ci est associée au numéro donnée à la fonction en tant que premier paramètre. Ensuite, lorsque vous souhaitez utiliser cette paire, vous y faites référence en appelant COLOR_PAIR et son numéro associé.

En dehors de la création des paires de couleurs, il vous faut les fonctions capables d'écrire avec les différentes paires. Ce sont les fonctions *attron* et *wattron*. Tant que vous n'appelez pas *attroff* ou *wattroff*, tout sera écrit dans la fenêtre correspondante avec la paire de couleurs sélectionnée.

Les fonctions *bkgd* et *wbkgd* sont capables de modifier la paire de couleurs associée à une fenêtre. Lorsqu'elles sont appelées, elles modifient les couleurs de fond et d'écriture de chaque emplacement de caractère de la fenêtre. Au prochain appel de *refresh*, chaque emplacement de caractère de la fenêtre sera

réécrit avec la nouvelle paire de couleurs.

Les pages de manuel vous préciseront les couleurs disponibles et vous donneront tous les détails sur ces fonctions.

## Des fenêtres avec un cadre

Vous pouvez améliorer l'aspect de vos programmes en encadrant vos fenêtres. La bibliothèque vous propose pour cela la macro *box*. Il n'existe pas de *wbox* contrairement aux autres fonctions; *box* réclame un pointeur de type WINDOW comme argument.

Tout autre détail sur *box* figure dans les pages de manuel. Juste une précision à ce sujet : mettre votre fenêtre dans un cadre signifie que vous écrivez les caractères nécessaires correspondant aux limites de la fenêtre dans son tableau de caractères. Si vous écrivez ultérieurement dans ces zones de caractères, le cadre sera altéré. Vous éviterez ce désagrément en créant une sous-fenêtre dans la fenêtre mère avec subwin, en mettant la fenêtre originale dans un cadre et en n'écrivant que dans la fenêtre fille.

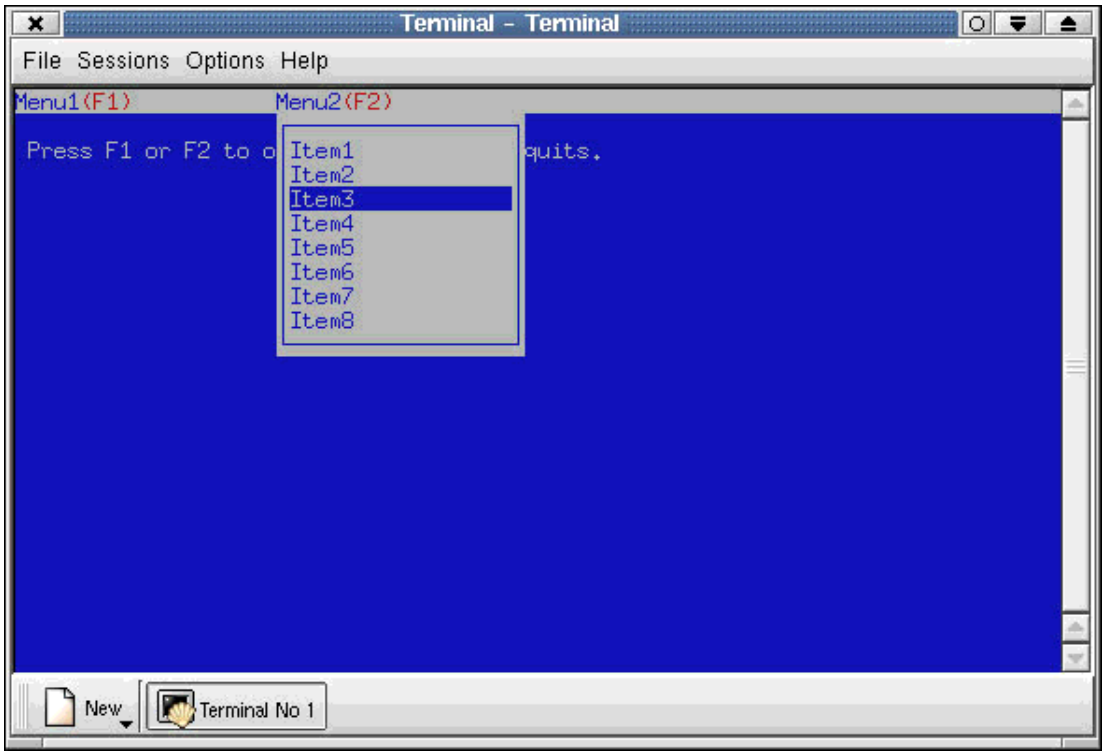
## Touches de fonction

Utiliser les touches de fonction du clavier implique de placer le drapeau *_use_keypad* dans la fenêtre concernée. La fonction *keypad* définit la valeur du drapeau *_use_keypad*. S'il est positionné, les touches de fonction et de direction peuvent être utilisées pour la saisie de données.

Dans ce cas, si vous utilisez *getch* par exemple, stockez la donnée reçue plutôt dans une variable de type *int* que dans une variable de type *char*. En effet les valeurs numériques des touches de fonction sont trop élevées pour contenir dans une variable *char*. Il n'est pas nécessaire de connaître ces valeurs, mais plutôt les noms sous lesquels la bibliothèque ncurses les désigne. Leur liste figure dans la page de manuel de la fonction *getch*.

## Un exemple

Analysons maintenant un programme simple et sympathique. On y crée des menus avec ncurses et on y montre comment sélectionner une option de menu. Un aspect intéressant de ce programme vient du fait qu'il utilise des fenêtres ncurses pour générer un menu. Voici une copie d'écran :




Le programme débute comme toujours par les entêtes *include* nécessaires. Les constantes sont les valeurs ASCII des touches *entrée* et *escape*.

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

#define ENTER 10
#define ESCAPE 27
```

La fonction ci-dessous est la première appelée à l'exécution du programme. Elle appelle *initscr* puis *start_color* pour bénéficier des couleurs. Les paires de couleurs utilisées dans le programme seront définies ultérieurement. L'appel à *curs_set(0*) rend le curseuur physique invisible. *noecho* arrête l'affichage des caractères tapés au clavier. La fonction noecho permet aussi de contrôler ce qui est tapé et de ne faire apparaître que ce que vous souhaitez. La fonction *echo* doit être appelée pour annuler l'effet de noecho. Enfin un appel de *keypad* active l'usage des touches de fonction pour la fenêtre *stdscr*. Nous utiliserons en effet F1, F2 et les touches de gestion du curseur dans le cours du programme.

```
void init_curses()
{
        initscr();
        start_color();
        init_pair(1,COLOR_WHITE,COLOR_BLUE);
        init_pair(2,COLOR_BLUE,COLOR_WHITE);
        init=_pair(3,COLOR_RED,COLOR_WHITE);
        curs_set(0);
        noecho();
        keypad(stdscr,TRUE);
}
```

La fonction suivante crée la barre de menu en haut de l'écran. En étudiant la fonction, vous pouvez

constater que la barre de menu, qui apparaît comme une simple ligne en haut de l'écran, est définie comme une sous-fenêtre de *stdscr* en une seule ligne. Cette fonction prend le pointeur vers cette fenêtre comme paramètre, change sa couleur de fond puis écrit les titres des menus. Nous avons utilisé *waddstr* pour faire cette écriture, mais nous aurions pu en utiliser d'autres. Soyez attentifs aux appels de *wattron* qui permettent de modifier la paire de couleurs (number 3) en remplacement de la paire par défaut (number 2). Cette paire a été définie par défaut dès la première ligne par l'appel de *wbkgd*. L'appel de *wattroff* permet de revenir à la paire par défaut.

```
void draw_menubar(WINDOW *menubar)
{
    wbkgd(menubar,COLOR_PAIR(2));
    waddstr(menubar,"Menu1");
    wattron(menubar,COLOR_PAIR(3));
    waddstr(menubar,"(F1)");
    wattroff(menubar,COLOR_PAIR(3));
    wmove(menubar,0,20);
    waddstr(menubar,"Menu2");
    wattron(menubar,COLOR_PAIR(3));
    waddstr(menubar,"(F2)");
    wattroff(menubar,COLOR_PAIR(3));
}
```

La fonction suivante affiche le menu lorsque F1 (ou F2) est pressée. Pour générer l'effet menu, une nouvelle fenêtre est créée au-dessus de la fenêtre bleu servant de fond et elle possède la même couleur blanche que la barre des menus. Cette nouvelle fenêtre ne doit pas écraser les caractères déjà écrits dans la fenêtre de fond. Ils doivent toujours être présents à la fermeture du menu. C'est pourquoi la fenêtre de menu ne peut être créée comme sous-fenêtre de *stdscr*. La fenêtre items[0] est d'abord créée avec *newwin* et les 8 autres fenêtres sont créées comme sous-fenêtres de items[0]. items[0] permet de tracer un cadre autour du menu et les autres fenêtres servent à montrer la rubrique sélectionnée ainsi qu'à empêcher l'écrasement des caractères du cadre entourant le menu. Pour qu'une rubrique paraisse sélectionnée, il suffit de lui donner un fond différent de celui des autres rubriques. C'est ce qui est réalisé dans la troisième ligne en partant du bas; la couleur de fond de la première rubrique est différente des autres, ainsi, lorsque le menu apparaît, c'est la première rubrique qui est sélectionnée.

```
WINDOW **draw_menu(int start_col)
{
    int i;
    WINDOW **items;
    items=(WINDOW **)malloc(9*sizeof(WINDOW *));

    items[0]=newwin(10,19,1,start_col);
    wbkgd(items[0],COLOR_PAIR(2));
    box(items[0],ACS_VLINE,ACS_HLINE);
    items[1]=subwin(items[0],1,17,2,start_col+1);
    items[2]=subwin(items[0],1,17,3,start_col+1);
    items[3]=subwin(items[0],1,17,4,start_col+1);
    items[4]=subwin(items[0],1,17,5,start_col+1);
    items[5]=subwin(items[0],1,17,6,start_col+1);
    items[6]=subwin(items[0],1,17,7,start_col+1);
    items[7]=subwin(items[0],1,17,8,start_col+1);
    items[8]=subwin(items[0],1,17,9,start_col+1);
    for (i=1;i<9;i++)
        wprintw(items[i],"Item%d",i);
    wbkgd(items[1],COLOR_PAIR(1));
    wrefresh(items[0]);
    return items;
```

```
}
```

Cette fonction efface le menu créé par la précédente. Elle efface d'abord les fenêtres rubrique à l'aide de *delwin* puis elle libère la mémoire occupée par leurs pointeurs :

```
void delete_menu(WINDOW **items,int count)
{
    int i;
    for (i=0;i<count;i++)
        delwin(items[i]);
    free(items);
}
```

La fonction *scroll_menu* permet de se déplacer dans et entre les menus. Elle scrute les frappes du clavier avec *getch*. Les flèches HAUT et BAS sélectionnent les rubriques au-dessus ou au-dessous. Ce qui, rappelez vous, correspond à la différence de couleur de fond entre la rubrique sélectionnée et les autres. Les flèches GAUCHE et DROITE provoquent le changement du menu actuellement ouvert. Presser la touche Entrée retourne la valeur de la rubrique sélectionnée. La touche ESC ferme les menus sans sélection de rubrique. Toute autre touche est ignorée. Ici *getch* lit les touches de direction frappées au clavier. Rappelez-vous que ceci est rendu possible par l'appel, dans *init_curses*, de *keypad(stdscr,TRUE)* et par le stockage de la valeur retournée dans une variable de type *int* plutôt que de type *char* car ces touches renvoient des valeurs trop élevées.

```
int scroll_menu(WINDOW **items,int count,int menu_start_col)
{
    int key;
    int selected=0;
    while (1) {
        key=getch();
        if (key==KEY_DOWN || key==KEY_UP) {
            wbkgd(items[selected+1],COLOR_PAIR(2));
            wnoutrefresh(items[selected+1]);
            if (key==KEY_DOWN) {
                selected=(selected+1) % count;
            } else {
                selected=(selected+count-1) % count;
            }
            wbkgd(items[selected+1],COLOR_PAIR(1));
            wnoutrefresh(items[selected+1]);
            doupdate();
        } else if (key==KEY_LEFT || key==KEY_RIGHT) {
            delete_menu(items,count+1);
            touchwin(stdscr);
            refresh();
            items=draw_menu(20-menu_start_col);
            return scroll_menu(items,8,20-menu_start_col);
        } else if (key==ESCAPE) {
            return -1;
        } else if (key==ENTER) {
            return selected;
        }
    }
}
```

Maintenant, voici le programme principal *main()*. Il utilise toutes les fonctions décrites ci-dessus. Il récupère la valeur des touches pressées grâce à getch et si l'utilisateur tape F1 ou F2, il dessine la fenêtre de menu correspondante à l'aide de *draw_menu*. Puis il appelle *scroll_menu* et attend la sélection d'une

rubrique par l'utilisateur . Enfin lorsque *scroll_menu* se termine, il efface le menu courant et affiche la rubrique sélectionnée dans la barre des menus.

Il faudrait aborder la fonction *touchwin*. Si *refresh* était appelé directement après la fermeture des menus, sans *touchwin*, le dernier menu affiché resterait à l'écran. En effet la fonction menu ne modifie pas *stdscr* et lors de l'appel à *refresh* elle ne réécrit aucun caractère de *stdscr* puisqu'elle considère que la fenêtre n'a pas changé d'aspect. *touchwin* place tous les drapeaux dans la structure WINDOW qui indique à refresh que toutes les lignes de la fenêtre ont été modifiées, ainsi, au prochain rafraîchissement, la fenêtre entière est régénérée même si son contenu n'a pas changé. L'information présente dans *stdscr* reste en place après la fermeture des menus puisque ces derniers n'écrasent pas stdscr, étant créés comme de nouvelles fenêtres.

```
int main()
{
    int key;
    WINDOW *menubar,*messagebar;

    init_curses();

    bkgd(COLOR_PAIR(1));
    menubar=subwin(stdscr,1,80,0,0);
    messagebar=subwin(stdscr,1,79,23,1);
    draw_menubar(menubar);
    move(2,1);
    printw("Tapez F1 ou F2 pour ouvrir les menus. ");
    printw("ESC quitte.");
    refresh();

    do {
        int selected_item;
        WINDOW **menu_items;
        key=getch();
        werase(messagebar);
        wrefresh(messagebar);
        if (key==KEY_F(1)) {
            menu_items=draw_menu(0);
            selected_item=scroll_menu(menu_items,8,0);
            delete_menu(menu_items,9);
            if (selected_item<0)
                wprintw(messagebar,"Vous n'avez pas sélectionné de rubrique.");
            else
                wprintw(messagebar,
                  "Vous avez sélectionné la rubrique %d.",selected_item+1);
            touchwin(stdscr);
            refresh();
        } else if (key==KEY_F(2)) {
            menu_items=draw_menu(20);
            selected_item=scroll_menu(menu_items,8,20);
            delete_menu(menu_items,9);
            if (selected_item<0)
                wprintw(messagebar,"Vous n'avez pas sélectionné de rubrique.");
            else
                wprintw(messagebar,
                  "Vous avez sélectionné la rubrique %d.",selected_item+1);
            touchwin(stdscr);
            refresh();
        }
    } while (key!=ESCAPE);
```

```
    delwin(menubar);
    delwin(messagebar);
    endwin();
    return 0;
}
```

Si vous copiez ce code dans un fichier *example.c* et que vous en retiriez mes explications, vous pourrez le compiler par la commande :

```
gcc -Wall example.c -o example -lcurses
```

et tester ensuite le programme obtenu. Vous pouvez aussi télécharger le code à partir des "Références" ci-dessous.

# Conclusion

J'ai parlé des fonctions de base de ncurses, suffisantes pour interfacer au mieux vos programmes. Mais les capacités de cette bibliothèque ne s'arrêtent pas là. Les pages de manuel sont à votre disposition pour en découvrir bien d'autres, et si vous suivez ma recommandation de les lire, vous comprendrez que cet article n'est vraiment qu'une introduction.

# Références

- Le programme d'exemple: exemple.c
- Le site de ncurses:www.gnu.org/software/ncurses/

---


| | |
|---|---|
| Site Web maintenu par l´équipe d´édition LinuxFocus | Translation information: |
|  | tr --> -- : Reha K. Gerçeker <gerceker/at/itu.edu.tr><br>tr --> en: Reha K. Gerçeker <gerceker/at/itu.edu.tr><br>en --> fr: Paul Delannoy (homepage) |
| http://www.LinuxFocus.org | |

2005-01-14, generated by lfparser_pdf version 2.51