Section: Manuel du programmeur Linux (3) Mise à jour de la version anglaise : 26 novembre 2007 Index Menu\_principal

# **NOM**

printf, fprintf, sprintf, sprintf, vprintf, vfprintf, vsprintf, vsnprintf - Formatage des sorties

int printf(const char \*format, ...);

**SYNOPSIS** #include <stdio.h>

int fprintf(FILE \*stream, const char \*format, ...); int sprintf(char \*str, const char \*format, ...); int snprintf(char \*str, size\_t size, const char \*format, ...);

#include < stdarq.h>

int vprintf(const char \*format, va\_list ap); int vfprintf(FILE \*stream, const char \*format, va\_list ap); int vsprintf(char \*str, const char \*format, va\_list ap); int vsnprintf(char \*str, size\_t size, const char \*format, va\_list ap);

snprintf(), vsnprintf(): BSD SOURCE || XOPEN SOURCE >= 500 || ISOC99 SOURCE; ou cc -std=c99

Exigences de macros de test de fonctionnalités pour la glibc (voir **feature\_test\_macros**(7)) :

**DESCRIPTION** Les fonctions de la famille **printf()** produisent des sorties en accord avec le format décrit plus bas. Les fonctions **printf()** et **vprintf()** écrivent leur sortie

#### sur stdout, le flux de sortie standard. fprintf() et vfprintf() écrivent sur le flux stream indiqué. sprintf(), snprintf(), vsprintf() et vsnprintf() écrivent leurs sorties dans la chaîne de caractères str.

Les fonctions **snprintf**() et **vsnprintf**() écrivent au plus *size* octets (octet nul ( $\ll \0$ ) inclus) dans *str*. Les fonctions vprintf(), vfprintf(), vsprintf(), vsnprintf() sont équivalentes aux fonctions printf(), fprintf(), sprintf(), sprintf(), respectivement, mais elles emploient un tableau va list à la place d'un nombre variable d'arguments. Ces fonctions n'appellent pas la macro va end. Aussi, la valeur de

va\_arg est-elle indéfinie après l'appel. Voir **stdarg**(3).

Le caractère % peut être éventuellement suivi par un ou plusieurs attributs suivants :

donnera des chiffres arabes pour la localisation perse (« fa IR »).

Ces huit fonctions créent leurs sorties sous le contrôle d'une chaîne de format qui indique les conversions à apporter aux arguments suivants (ou accessibles à travers les arguments de taille variable de **stdarg**(3)). VALEUR RENVOYÉE

Si elles réussissent, ces fonctions renvoient le nombre de caractères imprimés, sans compter l'octet nul « \0 » final dans les chaînes. Les fonctions snprintf() et vsnprintf() n'écrivent pas plus de size octets (y compris le « \0 » final). Si la sortie a été tronquée à cause de la limite, la valeur de retour est le nombre de caractères (sans le « \0 » final) qui auraient été écrits dans la chaîne s'il y avait eu suffisamment de place. Ainsi, une valeur de retour size ou plus signifie que la sortie a été tronquée. (Voir aussi la section NOTES plus bas). Si une erreur de sortie s'est produite, une valeur négative est

Chaîne de format Le format de conversion est indiqué par une chaîne de caractères, commençant et se terminant dans son état de décalage initial. La chaîne de format est composée d'indicateurs : les caractères ordinaires (différents de %), qui sont copiés sans modification sur la sortie, et les spécifications de conversion, qui sont mises en correspondance avec les arguments suivants. Les spécifications de conversion sont introduites par le caractère %, et se terminent par un indicateur de conversion. Entre eux peuvent se trouver (dans l'ordre), zéro ou plusieurs attributs, une valeur optionnelle de largeur minimal de champ, une

# valeur optionnelle de *précision*, et un éventuel *modificateur de longueur*.

renvoyée.

et

#

aq aq

aq

Les arguments doivent correspondre correctement (après les promotions de types) avec les indicateurs de conversion. Par défaut les arguments sont pris dans l'ordre indiqué, où chaque « \* » et chaque indicateur de conversion réclament un nouvel argument (et où l'insuffisance en arguments est une erreur). On peut aussi préciser explicitement quel argument prendre, en écrivant, à chaque conversion, « %m\$ » au lieu de « % », et « \*m\$ » au lieu de « \* ». L'entier décimal m indique la position dans la liste d'arguments, l'indexation commençant à 1. Ainsi, printf("%\*d", width, num);

quelque part dans la chaîne de format. Pour certaines conversions numériques, un caractère de séparation décimale (le point par défaut) est utilisé, ainsi qu'un caractère de regroupement par milliers. Les véritables caractères dépendent de la localisation LC\_NUMERIC. La localisation POSIX utilise « . » comme séparateur décimal, et n'a pas de caractère de regroupement. Ainsi,

indique que la valeur doit être convertie en une autre forme. Pour la conversion o le premier caractère de la chaîne de sortie vaudra zéro (en ajoutant un préfixe 0 si ce n'est pas déjà un zéro). Pour les conversions x et X une valeur non nulle reçoit le préfixe « 0x » (ou « 0X » pour

indique le remplissage avec des zéros. Pour les conversions d, i, o, u, x, X, a, A, e, E, f, F, g, et G, la valeur est complétée à gauche avec des zéros

plutôt qu'avec des espaces. Si les attributs **0** et **-** apparaissent ensemble, l'attribut **0** est ignoré. Si une précision est fournie avec une conversion

sont équivalents. La seconde notation permet de répéter plusieurs fois le même argument. Le standard C99 n'autorise pas le style utilisant « \$ », qui provient des Spécifications Single Unix. Si le style avec « \$ » est utilisé, il faut l'employer pour toutes conversions prenant un argument, et pour tous les arguments de largeur et de précision, mais on peut le mélanger avec des formats « %% » qui ne consomment pas d'arguments. Il ne doit pas y avoir de sauts dans les numéros des arguments spécifiés avec « \$ ». Par exemple, si les arguments 1 et 3 sont spécifiés, l'argument 2 doit aussi être mentionné

s'affichera comme « 1234567.89 » dans la localisation POSIX, « 1 234 567,89 » en localisation fr\_FR, et « 1.234.567,89 » en localisation da\_DK. **Caractère d'attribut** 

printf("%aq.2f", 1234567.89);

printf("%2\$\*1\$d", width, num);

l'indicateur X). Pour les conversions a, A, e, E, f, F, g, et G le résultat contiendra toujours un point décimal même si aucun chiffre ne le suit (normalement, un point décimal n'est présent avec ces conversions que si des décimales le suivent). Pour les conversions g et G les zéros en tête ne sont pas éliminés, contrairement au comportement habituel. Pour les autres conversions, cet attribut n'a pas d'effet.

numérique (d, i, o, u, x, et X), l'attribut 0 est ignoré. Pour les autres conversions, le comportement est indéfini.

n'inclue pas %agF.

indique que la valeur doit être justifiée sur la limite gauche du champ (par défaut elle l'est à droite). Sauf pour la conversion **n**, les valeurs sont complétées à droite par des espaces, plutôt qu'à gauche par des zéros ou des blancs. Un attribut - surcharge un attribut 0 si les deux sont fournis.

(une espace) indique qu'une espace doit être laissée avant un nombre positif (ou une chaîne vide) produit par une conversion signée Un signe (+ ou -) doit toujours être imprimé avant un nombre produit par une conversion signée. Par défaut, un signe n'est utilisé que pour des valeurs négatives. Un attribut + surcharge un attribut « espace » si les deux sont fournis. Les cinq caractères d'attributs ci-dessus sont définis dans le standard C, les spécifications SUSv2 en ajoute un :

Pour les conversions décimales (i, d, u, f, F, q, G) indique que les chiffres d'un argument numérique doivent être groupés par milliers en fonction de la localisation. Remarquez que de nombreuses versions de gcc(1) n'acceptent pas cet attribut et déclencheront un avertissement (warning). SUSv2

La glibc 2.2 ajoute un caractère d'attribut supplémentaire. Ι Pour les conversions décimales (i, d, u) la sortie emploie les chiffres alternatifs de la localisation s'il y en a. Par exemple, depuis la glibc 2.2.3, cela

Largeur de champ Un nombre optionnel ne commençant pas par un zéro, peut indiquer une largeur minimale de champ. Si la valeur convertie occupe moins de caractères

l'attribut « - » vu plus haut suivi d'une largeur positive. En aucun cas une largeur trop petite ne provoque la troncature du champ. Si le résultat de la conversion est plus grand que la largeur indiquée, le champ est élargi pour contenir le résultat. **Précision** 

Une précision éventuelle, sous la forme d'un point (« . ») suivi par un nombre. À la place de la chaîne représentant le nombre décimal, on peut écrire « \* » ou « \*m\$ » (m étant entier) pour indiquer que la précision est fournie dans l'argument suivant, ou dans le m-ième argument, respectivement. L'argument fournissant la précision doit être de type int. Si la précision ne contient que le caractère « . », ou une valeur négative, elle est considérée comme nulle.

Cette précision indique un nombre minimum de chiffres à faire apparaître lors des conversions d, i, o, u, x, et X, le nombre de décimales à faire apparaître pour les conversions a, A, e, E, f et F, le nombre maximum de chiffres significatifs pour q et G, et le nombre maximum de caractères à imprimer depuis

suivant, ou dans le m-ième argument, respectivement. L'argument fournissant la largeur doit être de type int. Une largeur négative est considéré comme

que cette largeur, elle sera complétée par des espaces à gauche (ou à droite si l'attribut d'alignement à gauche a été fourni). À la place de la chaîne représentant le nombre décimal, on peut écrire « \* » ou « \*m\$ » (*m* étant entier) pour indiquer que la largeur du champ est fournie dans l'argument

# une chaîne pour les conversions s et S.

Modificateur de longueur Ici, une conversion entière correspond à d, i, o, u, x ou X. hh La conversion entière suivante correspond à un signed char ou unsigned char, ou la conversion **n** suivante correspond à un argument pointeur sur un

La conversion entière suivante correspond à un short int ou unsigned short int, ou la conversion **n** suivante correspond à un argument pointeur sur

(elle) La conversion entière suivante correspond à un long int ou unsigned long int, ou la conversion n suivante correspond à un pointeur sur un long

int, ou la conversion c suivante correspond à un argument wint\_t, ou encore la conversion s suivante correspond à un pointeur sur un wchar\_t.

#### Ш (elle-elle) La conversion entière suivante correspond à un long long int, ou unsigned long long int, ou la conversion **n** suivante correspond à un pointeur sur un long long int.

utiliser).

Indicateur de conversion

signed char.

un *short int*.

h

L

Z

t

o, u, x, X

e, E

n

m

%

q (« quad » BSD 4.4 et Linux sous libc5 seulement, ne pas utiliser) Il s'agit d'un synonyme pour II. j La conversion entière suivante correspond à un argument *intmax\_t* ou *uintmax\_t*.

La conversion entière suivante correspond à un argument size\_t ou ssize\_t. (La bibliothèque libc5 de Linux proposait l'argument **Z** pour cela, ne pas

La conversion a, A, e, E, f, F, g, ou G suivante correspond à un argument long double. (C99 autorise %LF mais pas SUSv2).

La conversion entière suivante correspond à un argument ptrdiff t. Les spécifications SUSv2 ne mentionnent que les modificateurs de longueur h (dans hd, hi, ho, hx, hX, hn), I (dans Id, Ii, Io, Ix, IX, In, Ic, Is) et L (dans Le, LE, Lf, Lg, LG).

Un caractère indique le type de conversion à apporter. Les indicateurs de conversion, et leurs significations sont : d, i L'argument int est converti en un chiffre décimal signé. La précision, si elle est mentionnée, correspond au nombre minimal de chiffres qui doivent apparaître. Si la conversion fournit moins de chiffres, le résultat est rempli à gauche avec des zéros. Par défaut la précision vaut 1. Lorsque 0 est

avec des zéros. La précision par défaut vaut 1. Lorsque 0 est converti avec une précision valant 0, la sortie est vide.

L'argument *unsigned int* est converti en un chiffre octal non signé (o), un chiffre décimal non signé (u), un chiffre hexadécimal non signé (x et X). Les lettres **abcdef** sont utilisées pour les conversions avec **x**, les lettres **ABCDEF** sont utilisées pour les conversions avec **X**. La précision, si elle est indiquée, donne un nombre minimal de chiffres à faire apparaître. Si la valeur convertie nécessite moins de chiffres, elle est complétée à gauche

L'argument réel, de type double, est arrondi et présenté avec la notation scientifique [-]c.ccce\\*(Pmcc dans lequel se trouve un chiffre avant le

L'argument réel, de type double, est arrondi, et présenté avec la notation classique [-]ccc.ccc, où le nombre de décimales est égal à la précision

SUSv2 ne mentionne pas **F** et dit qu'il existe une chaîne de caractères représentant l'infini ou NaN. Le standard C99 précise « [-]inf » ou « [-

réclamée. Si la précision n'est pas indiquée, l'affichage se fera avec 6 décimales. Si la précision vaut zéro, aucun point n'est affiché. Lorsque le point

point, puis un nombre de décimales égal à la précision demandée. Si la précision n'est pas indiquée, l'affichage contiendra 6 décimales. Si la précision vaut zéro, il n'y a pas de point décimal. Une conversion **E** utilise la lettre **E** (plutôt que **e**) pour introduire l'exposant. Celui-ci contient toujours au moins deux chiffres. Si la valeur affichée est nulle, son exposant est 00. f, F

converti avec une précision valant 0, la sortie est vide.

est affiché, il y a toujours au moins un chiffre devant.

## linfinity » pour les infinis, et une chaîne commençant par « nan » pour NaN dans le cas d'une conversion f, et les chaînes « [-]INF » « [-]INFINITY » « NAN\* » pour une conversion **F**. g, G

S'il n'y a pas de modificateur I, l'argument de type const char \* est supposé être un pointeur sur un tableau de caractères (pointeur sur une chaîne). Les caractères du tableau sont écrits jusqu'à l'octet nul « \0 » final, non compris. Si une précision est indiquée, seul ce nombre de caractères sont écrits. Si une précision est fournie, il n'y a pas besoin d'octet nul. Si la précision n'est pas donnée, ou si elle est supérieure à la longueur de la chaîne, l'octet nul final est nécessaire.

Si un modificateur I est présent, l'argument de type const wchar\_t \* est supposé être un pointeur sur un tableau de caractères larges. Les caractères larges du tableau sont convertis en une séquence de caractères multi-octets (chacun par un appel de wcrtomb(3), avec un état de conversion dans l'état initial avant le premier caractère large), ceci jusqu'au caractère large nul final compris. Les caractères multi-octets résultants sont écris jusqu'à l'octet nul final (non compris). Si une précision est fournie, il n'y a pas plus d'octets écrits que la précision indiquée, mais aucun caractère multi-octet n'est écrit partiellement. Remarquez que la précision concerne le nombre d'octets écrits, et non pas le nombre de caractères larges ou de positions d'écrans. La chaîne doit contenir un caractère large nul final, sauf si une précision est indiquée, suffisamment petite pour que le nombre d'octets

Un caractère « % » est écrit. Il n'y a pas de conversion. L'indicateur complet est « %% ». **CONFORMITÉ** 

### **BOGUES** Comme **sprintf**() et **vsprintf**() ne font pas de suppositions sur la longueur des chaînes, le programme appelant doit s'assurer de ne pas déborder l'espace d'adressage. C'est souvent difficile. Notez que la longueur des chaînes peut varier avec la localisation et être difficilement prévisible. Il faut alors utiliser snprintf() ou vsnprintf() à la place (ou encore asprintf(3) et vasprintf(3)).

glibc 2.1. Jusqu'à la glibc 2.0.6, elles renvoyaient -1 si la sortie avait été tronquée.

« %n », ce qui autorise **printf**() à écrire dans la mémoire, et crée une faille de sécurité.

#include <stdio.h> fprintf(stdout, "%s, %s %d, %.2d:%.2d\n", jour\_semaine, mois, jour, heure, minute);

char \*p, \*np; va list ap;

return NULL;

va end(ap);

while (1) {

return p; /\* Sinon réessayer avec plus de place \*/ if (n > -1) /\* glibc 2.1 \*/ size = n+1; /\* ce qu'il fallait \*/ /\* glibc 2.0 \*/

/\* Essayons avec l'espace alloué. \*/

/\* Si ça marche, renvoyer la chaîne. \*/

n = vsnprintf(p, size, fmt, ap);

printf(1), asprintf(3), dprintf(3), scanf(3), setlocale(3), wcrtomb(3), wprintf(3), locale(5) **TRADUCTION** 

**VOIR AUSSI** 

de manuel.

Index

Chaîne de format Caractère d'attribut Largeur de champ Précision

> Modificateur de longueur Indicateur de conversion

VALEUR RENVOYÉE

NQM.

**SYNOPSIS** 

**TRADUCTION** 

**CONFORMITÉ** NOTES **BOGUES EXEMPLE VOIR AUSSI** 

Pour allouer une chaîne de taille suffisante et écrire dedans (code correct aussi bien pour glibc 2.0 que glibc 2.1) :

#include <stdio.h> #include <stdlib.h> #include <stdarg.h>

"%1\$s, %3\$d. %2\$s, %4\$d:%5\$.2d\n" on peut obtenir « Dimanche, 3 juillet, 23:15 ».

où le format dépend de la localisation et peut permuter les arguments. Avec la valeur :

L'argument réel, de type double, est converti en style f ou e (ou F ou E pour la conversion G) La précision indique le nombre de décimales significatives. Si la précision est absente, une valeur par défaut de 6 est utilisée. Si la précision vaut 0, elle est considérée comme valant 1. La notation scientifique e est utilisée si l'exposant est inférieur à -4 ou supérieur ou égal à la précision demandée. Les zéros en fin de partie décimale sont supprimés. Un point décimal n'est affiché que s'il est suivi d'au moins un chiffre. a, A (C99 mais pas SUSv2). Pour la conversion a, l'argument de type double est transformé en notation hexadécimale (avec les lettres abcdef) dans le style [-]0xh.hhhhp\\*(Pmd; Pour la conversion A, le préfixe 0X, les lettres ABCDEF et le séparateur d'exposant P sont utilisés. Il y a un chiffre hexadécimal avant la virgule, et le nombre de chiffres ensuite est égal à la précision. La précision par défaut suffit pour une représentation exacte de la valeur, si une représentation exacte est possible en base 2. Sinon, elle est suffisamment grande pour distinguer les valeurs de type double. Le chiffre avant le point décimal n'est pas spécifié pour les nombres non normalisés, et il est non nul pour les nombres normalisés. C S'il n'y a pas de modificateur I, l'argument entier, de type int, est converti en un unsigned char, et le caractère correspondant est affiché. Si un modificateur I est présent, l'argument de type wint\_t (caractère large) est converti en séquence multi-octet par un appel à wcrtomb(3), avec un état de conversion débutant dans l'état initial. La chaîne multi-octet résultante est écrite. S

écrits la remplisse avant la fin de la chaîne. C (dans SUSv2 mais pas dans C99) Synonyme de Ic. Ne pas utiliser. S (dans SUSv2 mais pas dans C99) Synonyme de Is. Ne pas utiliser. p L'argument pointeur, du type void \* est affiché en hexadécimal, comme avec %#x ou %#lx.

Le nombre de caractères déjà écrits est stocké dans l'entier indiqué par l'argument pointeur de type int \*. Aucun argument n'est converti.

(Extension glibc.) Afficher la sortie de strerror(errno). Aucun argument n'est nécessaire.

toujours) le nombre de caractères qui auraient été écrits si la chaîne de sortie avait été assez grande.

La bibliothèque glibc 2.1 ajouta les modificateurs de longueur hh, t, z, et les caractères de conversion a, A.

La bibliothèque glibc 2.2. ajouta le caractère de conversion F avec la sémantique C99, et le caractère d'attribut I.

Les fonctions **fprintf()**, **printf()**, **sprintf()**, **vprintf()**, **vfprintf()**, et **vsprintf()** sont conformes à C89 et C99. Les fonctions **snprintf()** et **vsnprintf()** sont conformes à C99. En ce qui concerne la valeur de retour de **snprintf**(), SUSv2 et C99 sont en contradiction : lorsque **snprintf**() est appelée avec un argument size=0 SUSv2 précise une valeur de retour indéterminée, inférieure à 1, alors que C99 autorise str à être NULL dans ce cas, et réclame en valeur de retour (comme

cdeEfFgGinopsuxX, où F était synonyme de f. De plus, elle acceptait D, O, U comme synonymes de ld, lo et lu. (Ce qui causa de sérieux boques par la suite

lorsque le support de %D disparut). Il n'y avait pas de séparateur décimal dépendant de la localisation, pas de séparateur des milliers, pas de NaN ou

La bibliothèque libc5 de Linux connaissait les 5 attributs standard C, l'attribut « ag », la localisation, %m\$ et \*m\$. Elle connaissait les modificateurs de

L'implémentation des fonctions **snprintf**() et **vsnprintf**() de la glibc se conforme au standard C99, et se comporte comme décrit plus haut depuis la

Un code tel que **printf**(foo); indique souvent un bogue, car foo peut contenir un caractère « % ». Si foo vient d'une saisie non sécurisée, il peut contenir

longueur h, I, L, Z, q, mais acceptait L et q pour les long double et les long long int (ce qui est un boque). Elle ne reconnaissait plus FDOU, mais ajoutait le

La bibliothèque libc4 de Linux connaissait les 5 attributs standard du C. Elle connaissait les modificateurs de longueur h, l, L et les conversions

La libc4.[45] de Linux n'avait pas snprintf(), mais proposait une bibliothèque libbsd qui contenait un snprintf() équivalent à sprintf(), c'est-à-dire qui ignorait l'argument size. Ainsi, l'utilisation de snprintf() avec les anciennes libc4 pouvait conduire à de sérieux problèmes de sécurité.

Pour afficher avec cinq décimales :

fprintf (stdout, "pi =  $%.5f\n$ ", 4 \* atan (1.0));

jour semaine, mois, day, hour, min);

**EXEMPLE** 

#include <math.h> #include <stdio.h>

#include <stdio.h> fprintf(stdout, format,

d'infinis, et pas de %m\$ ni \*m\$.

**NOTES** 

caractère de conversion **m**, qui affiche *strerror(errno)*.

La bibliothèque glibc 2.0 ajouta les caractères de conversion C et S.

De nombreux pays utilisent un format de date différent, comme jour-mois-année. Une version internationale doit donc être capable d'afficher les arguments dans l'ordre indiqué par le format :

Pour afficher une date et une heure sous la forme « Sunday, July 3, 23:15 », ou jour\_semaine et mois sont des pointeurs sur des chaînes :

make message(const char \*fmt, ...) /\* Supposons que 100 octets suffisent. \*/ int n, size = 100;

if ((p = malloc(size)) == NULL)

if (n > -1 && n < size)

return NULL;

p = np;

} else {

va start(ap, fmt);

size \*= 2; /\* deux fois plus \*/ if ((np = realloc(p, size)) == NULL) { free(p);

Ce document est une traduction réalisée par Christophe Blaess < http://www.blaess.fr/christophe/> le 10 novembre 1996 et révisée le 17 juillet 2008. L'équipe de traduction a fait le maximum pour réaliser une adaptation française de qualité. La version anglaise la plus à jour de ce document est toujours consultable via la commande : « LANG=C man 3 printf ». N'hésitez pas à signaler à l'auteur ou au traducteur, selon le cas, toute erreur dans cette page

**DESCRIPTION** 

Dernière mise à jour : 17 juillet 2008