09/03/21-__/__/__ Melody BOY

GNL : Get Next Line

I. La Consigne :

Règles Communes :

- Normé ;
- Les fonctions ne doivent pas s'arrêter de manière inattendue ;
- Toute mémoire allouée sur la heap doit être libéré lors c'est nécessaire ;
- **Pas de Makefile** ;
- Le travail doit être rendue sur le git qui nous est assigné ;
- **Lifbft non-autorisé**.

Partie obligatoire :

Function Name	**get_next_line**
Prototype	Int get_next_line(int fd, char **line);
Paramètre	#1. Le file descriptor sur lequel lire #2. La valeur de ce qui a été lue
Valeur de Retour	1 : Une ligne a été lue 0 : La fin de fichier a été atteinte -1 : Une erreur est survenue
Fonctions Externes Autorisées	Read, malloc, free
Description	Ecrire une fonction qui retourne une ligne lue depuis un file description, sans le retour à la ligne
Compilation	Gcc -Wall -Werror -Wextra -D BUFFER_SIZE=xx get_next_line.c get_next_line_utils.c

Spécificité :

- Des **appels successifs** **à votre get_next_line doivent vous permettre de lire l'entièreté du texte disponibles sur le file description, une ligne à la fois, jusqu'au EOF** ;
- Puisque la libft n'est pas autorisée pour ce projet, il faudra ajouter le **fichier get_next_line_utils.c** qui **contiendra les fonctions nécessaires** au fonctionnement de notre get_next_line ;
- Le programme **doit compiler avec le flag -D BUFFER_SIXE=xx**, ce define dot être utilisé dans nos, appels de read du get_next_line, pour définir la taille du buffet (cette valeur sera modifiée lors de l'évolution et par la moulinette) ;
- Le **read doit utiliser le BUFFER_SIZE pour lire depuis un fichier ou depuis le stdin** ;
- Dans le **ficher header get_next_line.h, il doit y avoir au moins le prototype de la fonction**.
- Get_next_line a un **comportement indéterminé si, entre deux appels, le file description change de fichier alors qu'EOF n'a pas été atteint sur le premier fichier** ;
- **Sleek n'est pas une fonction autorisée** (la lecture du file description ne doit être faite qu'une seule fois ;
- Get_next_line à de nouveau un **comportement indéterminé s'il lis un fichier binaire**, cependant, l'on peut, si on le souhaite, rendre le comportement cohérent ;
- **Les variables globales sont interdites**.

09/03/21-__/__/__ Melody BOY

Conseils :

- Vérifier que la fonction fonctionne toujours si la valeur de **BUFFER_SIZE est de 9999 ? De 1 ? Ou 10 000 000 ?**
- Le programme **doit lire le moins possible à chaque fois** que get_next_line est appelé : **si le programme rencontre un retour à la ligne, il doit renvoyer la ligne et arrêter de la lire jusqu'au prochain appel.**
- **Eviter de tout lire d'abord puis de travailler les lignes une fois que tout est lue.**
- Bien **tester** le programme.
- **Test à faire** : lire depuis un fichier, depuis une redirection, depuis l'entrée standard, comment se comporte le programme lorsque vous lui envoyer un retour à la ligne ? Ou CTRL-D ?

II. Documentation :

1. Variable Globales et Locales :

Quand on dit qu'une variable est **global** ou **local**, on fait appelle à la **porté (c'est-à-dire où une variable va être utilisé)**. Attention, bien que la portée implique souvent une durée de vie, ici ce n'est pas le cas.

Def : Variable Globale
Variable déclarée à l'extérieur du corps du toute fonction ou classe, elles peuvent donc être utilisées n'importe où dans le programme (on parle également de variable de portée globale).

Def : Variable Locale
Variable qui ne peut être utilisée que dans la fonction ou le bloc où elle est définie. De fait, la variable locale s'oppose à la variable globale.

Dans de nombreux langages, les **variables globales sont toujours statiques**, mais dans certains elles **peuvent être dynamique**.

En revanche, **les variables locales sont généralement automatiques**, mais **peuvent être statiques**.

2. Variables Statiques et Dynamiques :

Les **variables statiques existent aussi longtemps que le programme**, et sont **initialisées (éventuellement) en même temps que lui.**

Les **variables dynamiques sont créées (et initialisées éventuellement) à un certain moment dans le programme, puis détruites ultérieurement**.

Les trois types de variables statiques

	Variables Globales	**Variables Statiques**	**Variables Statiques déclarées dans le bloc principal de la fonction main**
Création	Toutes les variables statiques sont créées dans le segment de données de compilation	//	//
Déclaration	En dehors de toute fonction (souvent au début du fichier)	Explicitement déclarées avec le mot réservé **static** dans une fonction	
Visibilité et Utilisation	Visibles et utilisables dans toutes les fonctions qui les suivant dans le fichier	Ces variables ne sont visibles et utilisables que dans le bloc où elle est déclarée.	Comme la durée de parcours de main et du programme entier, ces variables existent tout au long du programme mais ne sont que visibles dans la fonction main
Initialisation	Automatique initialisées à zéro si l'utilisateur ne les initialises pas	//	Ne sont jamais initialisées automatiquement

Les deux types de variables dynamiques

(Créées par le programmeur explicitement dans le tas à l'aide de malloc ou de new, et qui détruit quand bon lui semble (ne sont pas prise en compte par le compilateur))

	Variables Dynamiques de Programmeur	**Variables Dynamique Automatiques**
Création/Destruction	Crée explicitement dans le tas à l'aide de malloc ou de new, et peuvent être détruire quand bon lui semble	Elles sont automatiquement créées et détruites par le programme de façon transparente pour le programmeur (elle sont placées dans la pile, et il en existe différents types)

Autre types de variable (dites automatiques)

	Variables Dynamique Automatiques	**Variables Locales**	**Variables Cachées**
Création/ Déclaration	Automatiquement créées et détruites de façon transparente pour le programmeur	Déclarées à l'intérieur d'un bloc, et sont détruites à la fin de ce bloc	Variable temporaires créées par le programme lorsqu'on initialise une référence sur une constante, dans ce cas, une variable provisoire et créée et initialisée, dont la durée de vie est celle de référence afférente.
Petit Plus	Dans la pile	Aucune de ces variables sont initiées automatiquement, donc il est préférable de toujours préciser une valeur initiale	Ne sont jamais nommées

3. Que signifie **EOF** ? **End-of-file.**

4. Directives **#ifdef** et **#ifndef :**

Les directives **#ifdef et #ifndef ont le même effet que la directive #if lorsqu'elle est utilisée avec l'opérateur défini**.

Syntaxe :
- Identificateur de #if (directive : identificateur définir par l'#if);
- Identificateur de #ifndef (directive : identificateur #if ! défini).

Vous pouvez utiliser les directives #ifdef et #ifndef n'importe où #if peut être utilisé.

L' #ifdef instruction d' identificateur est **équivalente à #if 1 lorsque identifier a été défini**. Elle est **équivalente à #if 0 lorsque identifier n'a pas été défini ou a été non défini par la #undef directive.** Ces directives vérifient uniquement la présence ou l'absence d'identificateurs définis

avec #define, et non d'identificateurs déclarés dans le code source C ou C++. Ces directives sont fournies uniquement pour des raisons de **compatibilité avec les versions antérieures du langage. L'expression constante définie (identificateur) utilisée avec la #if directive est préférée.**

La directive **#ifndef vérifie l'inverse de la condition vérifiée par #ifdef :**
- **Si l'identificateur n'a pas été défini ou si sa définition a été supprimée avec #undef** , la **condition est vraie (différente de zéro**) ;
- Sinon, **la condition n'est pas vérifiée (0).**

III. Fonction Read :

1. Nom : Read - **Lire depuis un descripteur de fichier**.

2. Synopsis : **#include <unistd.h>**
   ssize_t read (int fd, void *buf, size_t count)

3. Description et Valeur de Retour :

Read() lit jusqu'à Count octets depuis le descripteur de fichier fd dans le tampon pointé par buf.

Si **count vaut zéro**, read() **renvoie zéro et n'a pas d'autres effet**.
Si **count est supérieur à SSIZE_MAX**, **le résultat est indéfini.**

4. Valeur de Retour :

Read() renvoie -1 s'il échoue, auquel cas errno contient le code d'erreur, et la position de la tête de lecture est indéfinie.

Sinon, **read() renvoie le nombre d'octets lus (0 en fin de fichier)**, **et avance la tête de lecture de ce nombre.** Attention, **le fait que le nombre renvoyé soit plus petit que le nombre demandé n'est pas une erreur**, ceci se produit **à la fin du fichie**r, ou **si on lit depuis un tube ou un terminal**, ou encore **si read() a été interrompu par signal**.

5. Erreurs Possibles :

Erreur	Raison 1	Raison 2	Raison 3
EAGAIN	On utilise une lecture non bloquante (attribut O_NONBLOCK du descripteur de fichier) et aucune donnée n'était possible		
EBADF	Fd n'est pas un descripteur de fichier valide	Ou fd n'est pas ouvert en lecture	
EFAULT	Buf pointe en dehors de l'espace d'adressage accessible		
EINVAL	Fd correspond à un objet ne permettant pas la lecture	Ou le fichier a été ouvert avec l'attribut O_DIRECT	Ou Soit l'adresse spécifiée dans le buf, soit la valeur spécifiée dans le count, soit la tête de lecture du fichier ne sont pas correctement alignés
EINVAL	Fd a été crée par appel à timerfd_creat	Et une mauvaise taille de tampon a été donné a read()	
EIO	Errreur d'entrée-sorite : ceci arrive si un processus est dans un groupe en arrière-plan et tente de lire depuis le terminal (il reçoit un signal SIGTTIN mais il l'ignore ou le bloque)	Ou si une erreur d'entrée-sortie bas-niveau s'est produite pendant la lecture d'un disque ou d'une bande	
EISDIR	Fd est un répertoire		

IV. Getline (Manuel) :

1. Nom : **Getline** ou encore **Getdelim**, sont des fonction de **saisie de chaîne délimité**.

2. Synopsis : **#define _GNU_SOURCE**
   #include <stdio.h>

   ssize_t getline(char **lineptr, size_t *n, FILE *stream)
   ssize_t getdelime(char **lineptr, size_t *n, int delim, FILE *stream)

3. Description :

Getline, comme get_next_line, **lit une ligne entière dans stream et stocke l'adresse du tampon contenant le texte dans *lineptr**. Le tampon **se termine par un caractère nul.**

Si *lineptr vaut NULL, getline() **alloue un tampon pour recevoir la ligne**. **Ce tampon devra être libéré** par le programme utilisateur (la valeur dans *n est ignorée).

Alternativement, avant d'appeler getline(), ***lineptr peut contenir un pointeur vers un tampon alloué par malloc et de taille *n octets**. **Si le tampon n'est pas suffisant** pour recevoir la ligne de saisie, **getline() le redimensionne avec realloc**, **mettant à jour *lineptr et *n comme il soit**.

Quoi qu'il en soit, **en cas de succès, *lineptr et *n seront adaptés afin de rendre compte respectivement de l'adresse et de la taille tampon**.

Getdelime() fonctionne comme getline(), si ce n'est qu'**un séparateur différent de saut-de-ligne peut être spécifié en tant qu'argument délimiter**. Tout comme getline(), aucun séparateur n'est ajouté s'il n'y a en avait pas dans l'entrée avant que la fin du fichier ne soit atteinte.

4. Valeur de Retour :

En cas de succès, getline() et getdelim() **renvoient le nombre de caractères lus, séparateur inclus, mais sans comptez le zéro terminal.** Cette valeur peut-être utilisée afin de traiter les octets nuls insérés dans la ligne lue.

Les deux fonctions **renvoient -1 en cas d'échec de lecture de la ligne** (condition de fin de fichier incluse).

5. Erreurs : **Paramètres erronés (n ou l'inepte valent NULL, ou bien stream n'est pas valide).**

6. Conformité : Getline() tout comme getdelim() sont des **extensions GNU** (disponibles depuis **libc 4.6.27**)