Spécification et conception d'un éditeur de texte page

R. Aubry

9 novembre 2010

Table des matières

Résumé

Il n'y a pas de notion de qualité sans contrôle, il n'y a pas de contrôle sans mesure.

Première partie

Introduction

Le document suivant présente la conception et la spécification d'un éditeur de texte page glissante. L'éditeur s'adressant à un utilisateur débutant en informatique, très peu de fonctions sont proposées à l'utilisateur afin de limiter le temps d'apprentissage nécessaire. Le texte affiché est brute, *i.e.* sans traitement et sans formatage.

Nous nous concentrerons en priorité sur l'ergonomie, puis sur la portabilité, et enfin sur la maintenabilité et l'efficacité.

Deuxième partie
Spécification

1 Caractéristiques

1.1 Périphériques requis

Périphériques :
- Un souris deux bouton.
- Un clavier standart.
- Un terminal pouvant afficher au moins 80 car/ligne, 50 lignes/page + les quatres lignes appartenant à l'interface de l'éditeur.

L'éditeur de texte ne gère qu'un seul encodage : l'**ASCII étendu**.

1.2 Zone tampon

Cette zone mémoire contient la dernière sélection effectuée.

2 Comportement de l'interface

Curseur : Le curseur est un rectangle semi-transparent,de la hauteur d'une ligne, de la largeur d'un caractère, et pouvant être déplacé via un clic de la souris. Il ne peut être placé que sur un caractère existant ou en début de ligne.

Ligne d'entête : Elle affiche le nom du fichier, son état, le numéro de la page en édition, le numéro de la ligne et de la colonne où se trouve le curseur, et eventuellement le mode de lecture (lecture seule ou lecture/écriture).

Un fichier ne peut être que dans 2 états :
- Enregistré : La version affichée à l'écran et la version sur le disque sont les même.
- Modifié : La version affichée à l'écran et la version sur le disque sont différentes.

Zone de saisie : Cette zone, reservée à la saisie du texte, fait :
- 80 caractères de large
- 50 lignes de haut

C'est la dimension d'une page *affichée*.

Si l'utilisateur dépasse :
- 80 caractères : Le nouveau caractère est transféré sur la ligne suivante. Aucun saut de ligne n'est cependant ajouté, seul l'affichage est modifié.
- 50 lignes : L'éditeur insère un caractère de fin de page et place le curseur au début de la page suivante.

Dans cette zone, l'utilisateur peut insérer du texte en tapant les caractères qu'il veut insérer au clavier. Les caractères saisis seront placés avant le curseur. Le caractère de fin de ligne est inséré par la touche ≪ Entrée ≫.

L'utilisateur peut sélectionner du texte en appuyant sur le bouton de la souris, en déplacant le curseur jusqu'au dernier caractère de la sélection, puis en relachant le bouton. Le texte sélectionné est automatiquement mis en zone tampon.

L'utilisateur peut aussi coller au niveau du curseur le texte présent dans la zone tampon (*i.e* le dernier texte sélectionné) en appuyant sur le second bouton de sa souris.

L'utilisateur peut supprimer le caractère courant en appuyant sur la touche standart « Suppr ». Il peut aussi supprimer le texte sélectionné au curseur en sélectionnant le texte, puis en appuyant sur la touche standart « Suppr ».

Page glissante : L'utilisateur a aussi la possibilité de naviguer entre les pages en appuyant sur les flèches directionnelles du clavier :

- « Flêche haut » : Change la page courante avec la page précédente si elle existe.
- « Flêche bas » : Change la page courante avec la page suivante si elle existe.

Dans les deux cas, le curseur est placé en première ligne.

Menu contextuel Constitué de 3 lignes, une *ligne de saisie* et deux lignes d'aide contextuel, il repertorie toutes les commandes accessibles dans le contexte courant sous la forme : TODO INSËRER SNITPPET

Les différents éléments de l'aide :

- **Charger** : « CTRL + o »
- **Enregistrer** : « CTRL + s »
- **Fin de page** : « CTRL + p »
- **Quitterr** : « CTRL + q »

Dans le cas où les commandes de gestion de fichiers sont appelée, l'aide contextuelle affiche (dans le cas d'un système en Français) :

Veuillez saisir le chemin absolu du fichier	
Enter : Valider	ESC : Annuler

Toutes les saisies relatives aux noms de fichiers seront effectuée dans la *ligne de saisie*.

Gestion des fichiers : L'utilisateur peut charger un fichier existant en appuyant simultanément sur les touches « CTRL + O ». Dans ce cas, le programme demande à l'utilisateur de saisir le chemin absolu du fichier à ouvrir dans la ligne de saisie.

Il peut aussi sauvegarder le texte en cours d'édition. Il est demandé à l'utilisateur de saisir le chemin absolu du nouveau fichier. Si le nom de fichier est le même que celui qui est en cours d'édition, les modifications sont apporées sur le fichier déjà sauvegardé.

Gestion des langues : La langue de l'éditeur, *i.e.* les message d'erreur et les messages du menu contextuel, prend automatiquement celle du système.

2.1 Lancement

Deux modes de lancement sont disponibles :

1. Sans paramètre : Un fichier vide appelé « Untitled » est ouvert, mais pas enregistré.
2. Avec un paramètre : Le fichier passé en paramètre est chargé et affiché s'il existe.

3 Liste des erreurs

Seules les opérations sur les fichiers peuvent générer des erreurs. Les messages dépendent de la langue du système. Ici pour un système un Français :

Le nom de fichier à charger n'existe pas : Le menu contextuel affiche « Fichier inexistant »

Le fichier où aura lieux la sauvegarde est protégé en écriture : Le menu contextuel affiche « Fichier protégé »

4 Bilan

Tout en respectant un cahier des charges réaliste, notre application a le mérite de proposer toutes les fonctions de bases pour manipuler un document source. Il permet, dans un contexte *multilangue*, avec des *requis matériels minimalistes*, quelque soit le *système d'exploitation*, ou la *machine cible* :

- D'effectuer toutes les opérations de fichiers du type : Ouvrir, charger, sauvegarder
- D'insérer, modifier, supprimer du texte
- De naviguer dans un fichier texte, au travers de ses différentes pages

Troisième partie
Conception

5 Rappel du contexte

La suite de ce document présente la démarche de conception entreprise dans le cadre de ce projet. Nous souhaitons proposer un éditeur de textes page capable de manipuler efficacement un fichier texte séquentiel de longueur variable.

L'architecture retenue à l'issue de cette phase de conception doit permettre d'offrir un outil ergonomique (aspect essentiellement traité dans le cadre de la spécification), portable sur différents environements Unix, maintenable et efficace. Nous devons garder en mémoire que cet outil doit pouvoir être déployé dans un environnement ou les ressources sont limitées.

Nous analyserons et critiquerons trois solutions de gestion des données en mémoire centrale et mémoire secondaire selon les critères de portabilité, maintenabilité et efficacité. Nous retiendrons celle jugée la plus pertinente et nous la décrirons en détail.

6 Choix d'une solution

6.1 Présentation des alternatives

6.1.1 Points commun des trois solutions

À l'ouverture d'un fichier, son contenu est copié dans un fichier temporaire — éventuellement en le structurant de manière adaptée à l'éditeur. Le fichier est indexé de manière à pouvoir accéder aux lignes directement.

Un tampon de taille limitée est créé en mémoire centrale. Lorsque l'utilisateur demande à accéder à des lignes qui n'y sont pas encore chargées, les lignes les plus anciennement accédées sont remplacées par celles qui viennent d'être demandées.

Le tampon est modifié directement au fur et à mesure de la frappe de l'utilisateur, et la ligne modifiée est transférée dans le fichier temporaire dès que l'utilisateur passe à une nouvelle ligne ou demande une sauvegarde du fichier. Le fichier source, quant à lui, est mis à jour sur demande de l'utilisateur (sauvegarde du fichier).

6.1.2 Solution 1 : structure consécutive

Toute la mémoire est allouée dans des tableaux, d'un seul bloc.

Le fichier temporaire est lui aussi placé dans une structure consécutive dans un zone contigüe de la mémoire centrale.

La solution est donc homogène : le même mécanisme est mis en place entre la mémoire centrale et la mémoire secondaire. La compréhension du système est alors simplifiée.

L'accès est direct : chaque caractère du texte peut être accédé en tant constant ($O(n)$). De fait, si l'on a besoin uniquement de remplacer des caractères par d'autres caractères, cette solution est efficace.

De la même manière, si l'on a besoin de rajouter des caractères à la fin du texte, la solution est là encore très efficace, dans les limites de la mémoire disponible sur le système.

De plus, cette solution présente l'avantage de ne pas devoir réserver de la mémoire pour le chaînage, celui-ci n'étant pas présent dans cette solution. Par contre, nous devons impérativement réserver un *buffer* de taille importante dès le début, pour éviter la réallocation intempestive. Nous pourrions envisager d'utiliser la technique bien connue de la réallocation par pas de n^2, c'est à dire réserver la mémoire selon un schéma quadratique, et donc réserver de plus en plus de mémoire à chaque réallocation.

Par contre, si l'on a besoin d'insérer des caractères au milieu du texte, la solution est très couteuse. En effet chaque insertion ou suppression de caractères provoque des opérations en $O(n)$

(où n est le nombre de caractères entre le point de fin d'insertion ou de suppression et la fin de la mémoire allouée (le *buffer*). Comme il s'agit d'une opération assez classique et fréquente, de nombreuses opération sont à prévoir. Ces opération sont du type `memmove()`, qui permet de déplacer des données dans un *buffer*, et qui a une complexité de n.

6.1.3 Solution 2 : structure mixte

La seconde solution proposée est une variation de la première. La structure de données en mémoire secondaire reste identique : nous conservons le fichier sous la forme d'une structure de caractères contigüs. Cependant, le bloc texte est fragmenté en sous-blocs de taille donnée en mémoire centrale et stockée dans des éléments d'une liste doublement chaînée. Les éléments sont chaînés dans l'ordre des données du fichier.

La structure d'un élément de la liste contiendra les données suivantes :

@précédant Adresse de l'élément précédant dans la liste,

@suivant Adresse de l'élément suivant dans la liste,

lgUtil Longueur du bloc utilisée données,

lgMax Longueur totale du bloc de données,

info Bloc de données (contient le fragment de texte stocké dans l'élément).

Le bloc de données peut éventuellement être plus large que la chaîne de caractères qu'il contient. En effet, lors de l'édition du document, de nouveaux blocs peuvent être alloués mais pas intégralement remplis par la saisie de l'utilisateur. Il est donc nécessaire de retenir deux informations sur ce bloc : $lgUtil$, la longueur effectivement utilisée, qui correspond à la taille de la chaine stockée (marqueur de chaîne inclu), et $lgMax$, la taille du bloc $info$.

Cette seconde solution est moins efficace que la première lors du chargement et de la sauvegarde du fichier. Les structures manipulées en mémoire centrale et mémoire secondaire ne seront plus homogènes, il sera donc nécessaire de proposer des procédures permettant de convertir le bloc de texte d'un format de structure à l'autre. Les opérations de chargement et sauvegarde du fichier seront plus coûteuses du fait de ces opérations de traduction, mais aussi des opérations d'allocation de mémoire proportionnelles à la taille du fichier.

Par ailleurs, cette solution est plus coûteuse en espace pour la mémoire centrale, puisqu'il est nécessaire de prévoir quelques octets correspondant à la taille des pointeurs manipulés.

En contrepartie, cette solution offre de biens meilleurs temps d'accès en lecture et écriture, puisqu'il est possible de naviguer d'un élément de la chaîne à un autre sans lire l'ensemble du bloc. La complexité algorithmique d'une telle opération sera en $O(n/m)$, où n est la longueur maximale d'un bloc et m le nombre d'éléments de la chaîne. L'insertion de texte nécessite de se placer à la fin du bloc de données et d'ajouter le caractère saisi. Si la longueur de la chaîne atteint la longueur maximale, il sera nécessaire d'allouer un nouvel élément et de l'insérer dans la chaîne à la suite de l'élément courant.

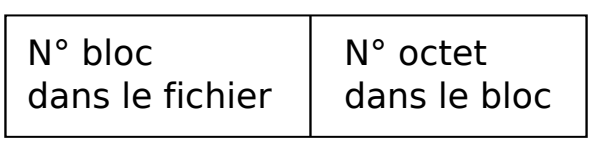
Une double liste chaînée est une structure de données classique qui n'est pas difficile à implémenter et à manipuler. L'impact d'une telle solution sur la maintenabilité est très faible.

6.1.4 Solution 3 : structure chaînée

Toutes les structures de données sont chaînées.

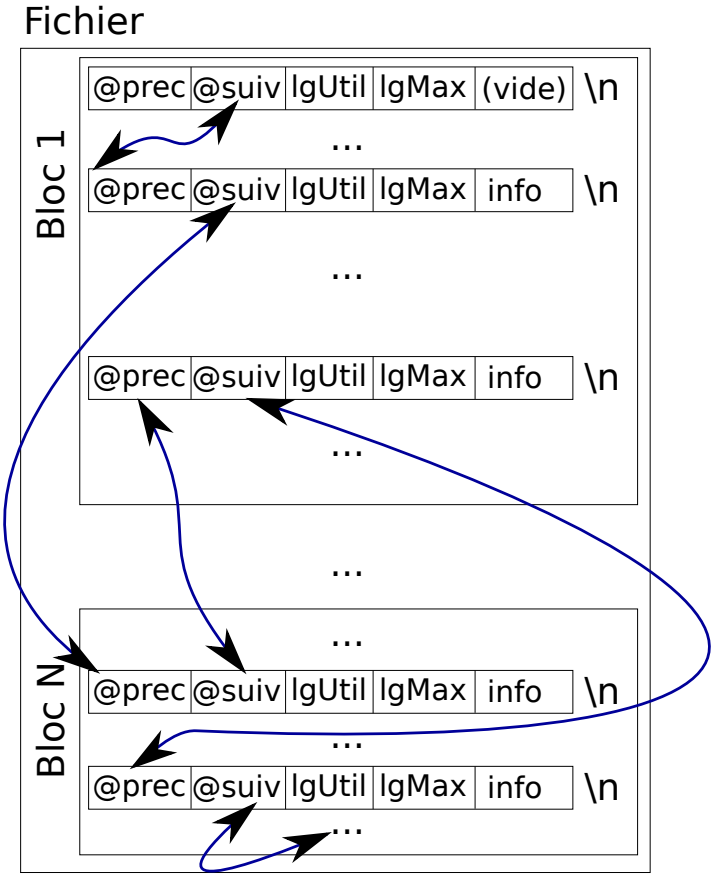
Le ficher temporaire est chaîné grâce à des adresses virtuelles, comme présentées ci-dessous.

FIGURE 1 – Structure d'une adresse virtuelle



Ainsi chaque ligne (y compris la « ligne 0 » virtuelle) fait référence à son successeur et son prédécesseur. On stocke également la longueur de l'information (« longueur utile ») et le nombre d'octets disponibles dans l'enregistrement (« longueur max », supérieure ou égale à la longueur utile).

FIGURE 2 – Structure du fichier temporaire de la solution 3



L'accès direct (lorsque l'utilisateur demande d'accéder à la ligne l, par exemple) est possible toutes les L lignes (L à définir) grâce à la table de *mapping*, qui donne pour chaque bloc la dernière ligne stockée dans celui-ci. Cette table est bien sûr complétée et mise à jour au fur et à mesure de l'édition du fichier (suppressions, ajouts, ...). Pour accéder à une ligne non indexée, il suffit d'accéder à une ligne proche et indexée, et de suivre le chaînage (complexité bornée).

Cette table contient bien sûr d'autres informations, comme le montre la représentation suivante.

FIGURE 3 – Structure de la table de *mapping* de la solution 3

N° ligne	N° bloc	Octet	Chargé	Date dernier accès	(Autres infos)	Adresse MC
10	1	156	Non	1289135888657534079	...	0x000000
...						
119	N	451	Oui	1289135888658945079	...	0x4006d4

6.2 Comparaison des alternatives

Les critères d'ergonomie et de portabilité ne sont pas pertinents pour la comparaison des alternatives. En effet, une bonne ergonomie, au niveau de la conception, sera traduite par une bonne efficacité, afin d'éviter à l'utilisateur des attentes inutiles. De plus, nous n'utiliserons pas

d'assembleur dans les différentes solutions, se reposant sur le compilateur C++ pour assurer cette portabilité. Le système sera donc portable tant que la plateforme dispose d'un compilateur C++.

Nous avons choisis de pondérer ces critères, pour prendre en compte leur importance respectives.

Critère	Pondération	Solution 1	Solution 2	Solution 3
Maintenabilité	20	++ (34)	++ (35)	+++ (56)
Décomposable en couches	8	+	++	+++
Homogénéité	5	+++	+	+++
Couplage entre couches	5	+	++	+++
Facilité d'implémentation	2	+++	++	+
Efficacité	15	++ (27)	++ (27)	+++ (41)
Complexité en insertion	3	+	++	+++
Complexité en suppression	3	+	++	+++
Complexité au changement de ligne	3	+	+	+++
Complexité de changement de page	3	+++	++	+++
Économie mémoire	1	+++	++	+
Complexité de chargement du fichier temporaire	1	+++	++	+++
Complexité de la sauvegarde	1	+++	++	+
Total	35	61	60	98

TABLE 1 – Table comparative des avantages et inconvénients de chaque solution

7 Détail de la solution choisie

7.1 ...

7.2 ...

7.3 ...

7.4 ...

7.5 Proposition d'architecture

8 Bilan