16/12/2016

Projet système

ExiaSaver screen

CHAMPAULT QUENTIN, VAUBOURG MANDEL

Exia Cesi

Table des matières :

[Contexte du projet : 1](#_Toc469583973)

[Fichier pbm 2](#_Toc469583974)

[Lecture des fichiers 3](#_Toc469583975)

[Ouverture, lecture de répertoire 6](#_Toc469583976)

[Fonctionnement des fonds d’écrans 8](#_Toc469583977)

[Création tableau virtuel 9](#_Toc469583978)

[Moteur de rendu 10](#_Toc469583979)

[Affichage 11](#_Toc469583980)

[L’écran de veille statique 12](#_Toc469583981)

[L’écran de veille dynamique 15](#_Toc469583982)

[L’écran de veille interactif 18](#_Toc469583983)

[Le lanceur 22](#_Toc469583984)

[L’historique. 23](#_Toc469583985)

# Contexte du projet :

Le but de ce projet est de créer un écran de veille pour Linux.

Plus précisément, nous devons en créer trois.

- Un écran de veille statique, qui affichera une image PBM (contrainte) au milieu de l’écran.

- Un écran de veille dynamique, une horloge qui s’actualisera au bout de N secondes, centrée avec un message en dessous, centré aussi, avec un point qui s’ajoutera à chaque seconde d’attente.

- Et enfin un écran de veille interactif, ou l’utilisateur pourra diriger un avion avec les touches H, B, G et D (que l’on a simplifié par Z, Q, S, D) et les touches 8,2,4,6. L’avion doit pouvoir traverser les murs, et commencer à s’afficher de l’autre côté de l’écran. Il doit tourner sur le point central de son fichier (Pour un fichier 6\*6, le point 3\*3 conviendra). Pour quitter l’écran de veille on doit appuyer sur ‘x’.

Toutes les images à afficher à l’écran sont des fichiers PBM, qu’il a fallu lire et afficher. Les fichiers PBM sont composer d’un nombre magique, de la taille du fichier (hauteur colonne) et du schéma en 0/1 de l’image

Tous ces écrans doivent être lancé aléatoirement depuis un lanceur. Quand on exécute ce programme, il doit choisir un seul écran, et envoyer cette donnée, ainsi que les données concernant l’écran (Quelle image choisir pour le statique par exemple) a un fichier ‘historique’.

Ce fichier doit être accessible avec la commande « -stats ». Il affichera un menu avec les options voulue.

Dans ce rapport nous détaillerons que les parties importantes à expliquer afin de synthétiser, vous trouverez le code source sur notre dépôt github à cette adresse :

https://github.com/Akitoshi02/ExiaScreen

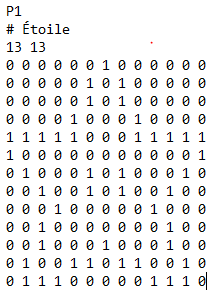
# Fichier pbm

Un fichier PBM est un fichier ***Portable BitMap file format.*** Il fait partie de la famille des fichiers *file format* avec les fichier PPM et PGM par exemple.

Il est composé d’un nombre magique du format qui indique quel type de fichier c’est (Entre P1 et P4 pour PBM). Il peut être composer de lignes de commentaire, commençant par un ‘#’.

Il est ensuite composé de la largeur de l’image, et de sa hauteur séparée par un espace. Cette hauteur représente le nombre de 0/1 par lignes et colonne.

Ensuite arrive les données de l’image *(Les 0 et les 1)*. Les 0 seront comptés comme des vides, et les 1 comme des pixels à afficher.



Exemple de PBM.

# Lecture des fichiers

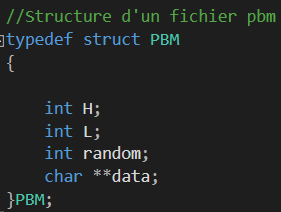
La lecture du fichier se fait via une fonction qui sera partagée par tous les exécutables TermStatic, TermDynamique, TermInteractif



Cette fonction prend en paramètre une structure qui représente notre fichier .pbm :

Il y a deux versions la première pour le statique et la second pour le dynamique et l’interactif,

Les deux versions changent peu, cependant la seconde version permet de remplir un tableau de structure de fichier pbm, afin de prendre en compte les pbm chiffre pour l’horloge et les 4 versions de l’avion



Les entiers **H** et **L** représente la **H**auteur et la **L**argeur de l’image.

Notre fonction de lecture est lancée dans un thread ainsi l’entier random permet de passer en paramètre le choix de l’images par le lanceur.

Enfin le char \*\*data représente un tableau dynamique à deux dimensions qui contiendra les successions de 0 et de 1 du fichier pbm décrivant l’images.

Pour remplir cette structure notre fonction doit pouvoir lire les fichier pbm pour cela nous utilisons les fonctions permettant d’ouvrir, lire et écrire dans un fichier se trouvant dans la bibliothèque **stdlib.h** (pour standard Library).

**Voici les fonctions que l’on utilise :**

Ouverture d’un fichier



Cette fonction prend en paramètre le nom ou chemin d’accès du fichier, ainsi qu’un mode d’ouverture

|  |  |
| --- | --- |
| MODE D’OUVERTURE | |
| **r** | Lecture seule le fichier doit exister |
| **r+** | Lecture et écriture le fichier doit exister |
| **w** | Ecriture seule si le fichier n’existe pas il sera créé |
| **w+** | Lecture et écriture le fichier sera vider de son contenu, s’il n’existe pas il sera créé |
| **a** | Mode ajout, on ajoute à la fin du fichier si le fichier n’existe pas il sera créé |
| **a+** | Lecture/écriture à partir de la fin du fichier s’il n’existe pas il sera créé |

Lecture dans un fichier

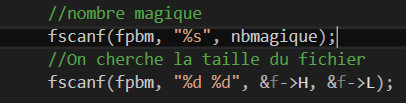


Cette fonction permet de lire une chaine de caractère à la fois

Elle prend en paramètre le fichier dans lequel elle doit lire, le format de lecture

C’est-à-dire quelle lira le format donner dans la chaine de caractère, elle prendra en dernier paramètre les variables dans lequel mettre le résultat de lecture

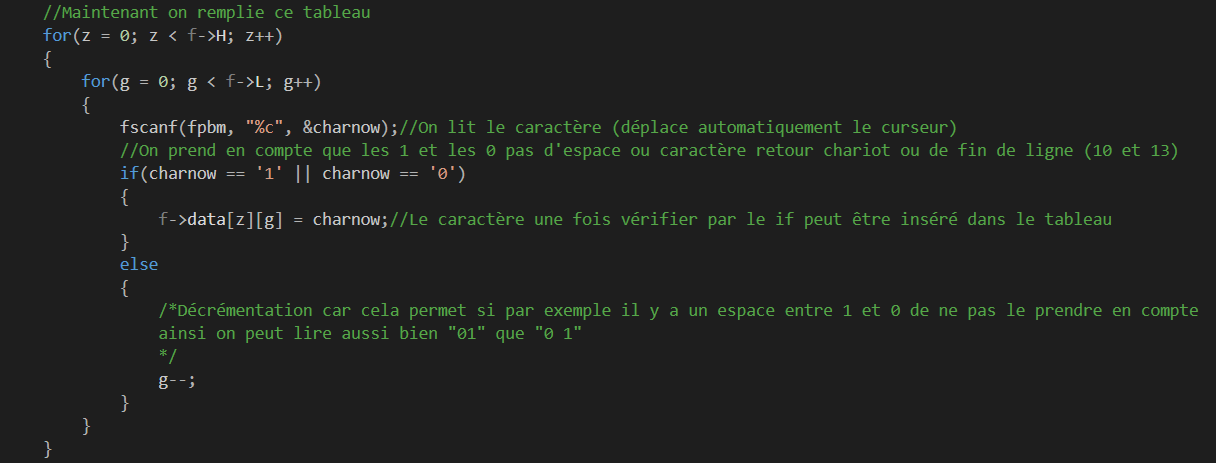
Exemple :



Le premier appel de fscanf() lis le premier format donné et récupère grâce au format donné **‘’%s’’** le nombre magique « P1 »

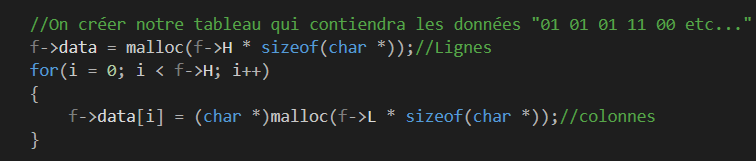
Le second appel permet de récupérer à la second ligne le format qui indique à fscanf() qu’on recherche deux entiers ‘**’%d %d’’** ces deux entiers représente la HAUTEUR et la LARGEUR de notre fichier pbm ces deux entier seront stockés dans la structure f->H et f->L

Enfin il faut lire les données du fichier pbm qui sont une succession de 0 et de 1



On lit chaque caractère ce code permet de prendre en compte si il y a des espaces entre les données ou non fscanf() récupère grâce au format qui lui est donné le caractère lu, le curseur dans le fichier avancera et passera au second caractère chaque lecture est enregistré dans charnow qui lui permettra de remplir le tableau data de notre structure.

Nous avons vu que data était en faite un double pointeur car la taille des fichiers n’est pas fixe et varie d’un à l’autre ainsi nous devons allouer dynamiquement notre tableau grâce aux taille Hauteur et Largeur récupérer dans le pbm précédemment :



|  |  |
| --- | --- |
| PILE | TAS |
| f->data | |  |  |  |  |  | | --- | --- | --- | --- | --- | | char \* | char \* | char \* | char \* | char \* | |
|  | |  |  |  |  |  | | --- | --- | --- | --- | --- | | char | char | char | char | char | |

Enfin on ferme le fichier grâce à fclose() elle prend en paramètre le pointeur de notre fichier.

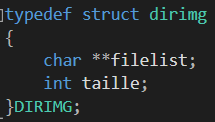
# Ouverture, lecture de répertoire

La lecture d’un répertoire nous permet de lister les images afin d’en choisir une aléatoirement pour le statique ou remplir un tableau pour le dynamique et l’interactif

Nous avons pour cela créer une fonction qui prend en paramètre le répertoire à lire :



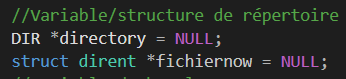
Elle retourne une structure représentant le répertoire, sa taille (le nombre de fichier) ainsi que les noms des fichiers



Filelist est un tableau de chaine de caractère qui contiendra les noms des fichiers, utile afin de former les chemins d’accès aux images

Pour l’ouverture/fermeture et lecture d’un répertoire nous avons besoin des bibliothèques **dirent.h**

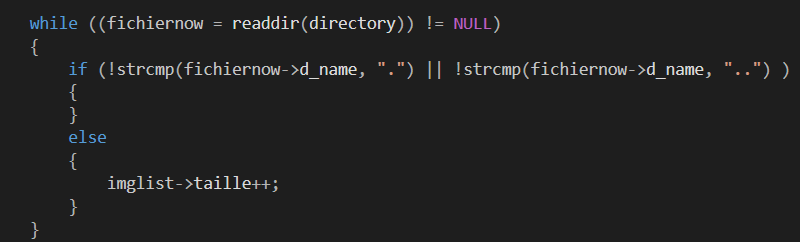
Premièrement notre répertoire sera représenté par un pointeur « DIR \* » puis les fichiers lu seront représentés par un pointeur de structure dirent \*



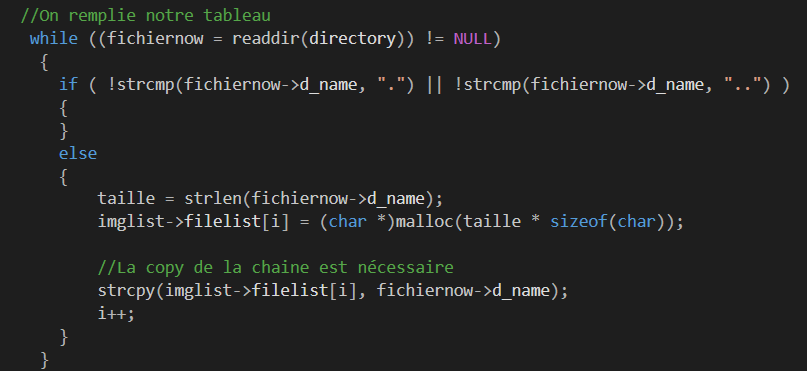
L’ouverture d’un répertoire se fait via la fonction opendir() elle prend en paramètre le chemin du répertoire qu’on souhaite ouvrir.

Puis nous devons parcourir le répertoire grâce à readdir()

Elle prend en paramètre le répertoire précédemment ouvert via opendir()



Dans cette partie de la fonction nous comptons le nombre de fichier que contient le répertoire, on supprime bien entendu « . » et « .. » qui représente le répertoire actuel et le répertoire contenant le répertoire dans lequel on lit.



Puis nous réutilisons le même code que précédemment, cependant celui-ci va nous permettre de remplir notre tableau qui contient les noms des fichiers pbm contenu dans le répertoire

Nous avons rencontré un problème avec cette partie. En effet la structure fichiernow qui représente le fichier en cours de lecture est une variable LOCALE à la fonction donc existante dans la pile, ainsi quand nous sortons de la fonction cette variable est détruite et son information avec…

Nous avons dû ainsi copier, grâce à strcpy(), le nom du fichier dans notre tableau de notre structure, ainsi nous ne perdons pas les données !

# Fonctionnement des fonds d’écrans

Nous avons voulu un système modulaire, qui puissent s’adapter en temps réel aux dimensions du shell, ainsi les contraintes de taille n’en sont plus !

Pour cela nous créons un tableau virtuel qui contiendra l’image, le but de ce tableau virtuel est de redimensionner le l’image.

Nous avons créé pour cela trois fonctions :

* Une qui créer le tableau virtuel
* Une qui vas inclure le pbm dans le tableau et redimensionner l’image
* Une qui affiche le tableau virtuel au centre du shell



Qu’est qu’on appelle tableau virtuel ? En fait interagir nous offre que ses dimensions et nous des emplacements comme un tableau (coordonnées). Ainsi, bien que le C n’ait pas cette approche, nous avons pensé notre système sous forme d’objet se matérialisant via des structures. L’avantage de cette approche et que nos fonctions sont modulaires, nous verrons la puissance de ce système dans l’écran dynamique.

Ainsi notre fichier est inclus dans notre Tableau virtuel (virtuel car l’utilisateur ne le voit pas) ce tableau nous offre donc des dimensions mais aussi des positions X et Y contrairement au shell qui nous offre que les dimensions…

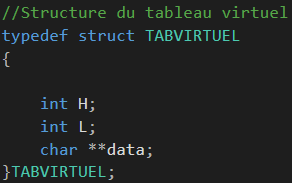
L’image pbm sera redimensionner dans ce tableau on divise pour cela la hauteur par la hauteur du pbm et pareil pour la largeur ce rapport nous donnera la hauteur et largeur d’un pixel.

Enfin la fonction d’affichage aura juste à printf() le tableau virtuel qui contient l’image pbm redimensionner en prenant soin de le placer au centre du shell

## Création tableau virtuel

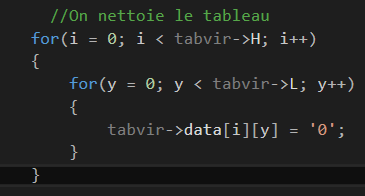
La fonction qui créer le tableau virtuel createtabvir()

Elle prend en paramètre une structure qui représente notre tableau ainsi que ses dimensions, la déclaration de la structure tabvir se fait dans le main()

La structure TABVIRTUEL :

Cette structure contient la Hauteur et la largeur du tableau virtuel ainsi que les données dans un tableau en deux dimensions, ce tableau contiendra l’image du pbm redimensionner par rapport à la taille de ce tableau

Enfin cette fonction met toutes les cases à zéro



## Moteur de rendu

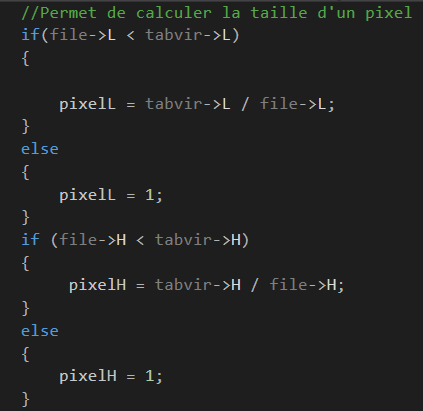
Notre moteur de rendu est une fonction qui va permettre deux choses

1. Redimensionner l’image pbm aux dimensions du tableau virtuel (zoom ou réduction)
2. Et centrer l’image au centre de ce tableau !



Elle prend en paramètre le tableau virtuel précédemment créé et le fichier pbm à inclure dans le tableau

Donc en premier lieu on calcul la taille d’un pixel

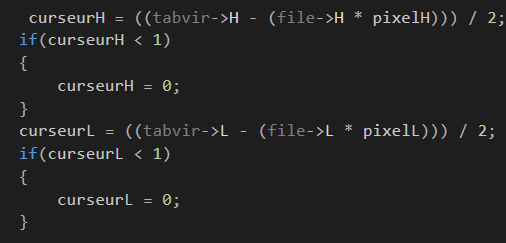


On sécurise : un pixel ne peux pas faire 0 en largeur ou en hauteur

Puis on divise la Hauteur du tableau virtuel par celle du fichier et pareil pour la largeur **pour** **avoir la largeur et la hauteur d’un pixel**

Puis on va se déplacer dans le pbm pour lire les données et dans le tableau pour les imprimer n fois la taille d’un pixel.

On utilise des curseurs pour centrer l’image en hauteur et largeur et aussi éviter que chaque pixel écrase le précédent en faisant avancer les coordonnées en X et Y ainsi chaque tour de boucle sera X + taille\_du\_pixel et Y +taille\_du\_pixel





## Affichage

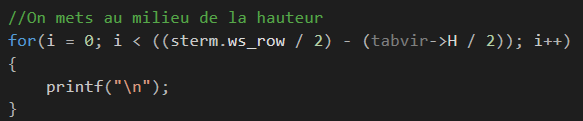
L’affichage du fichier se fait via une fonction

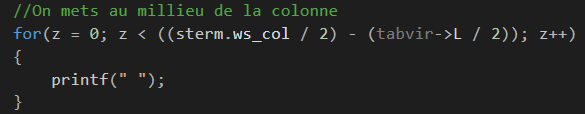


Cette fonction prend en paramètre l’adresse du tableau virtuel déclaré dans le main()

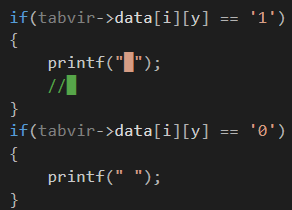
Dans un premier temps elle va centrer le tableau virtuel, comme nous sommes en interaction avec le Shell ce centrage se fera via des printf() pour connaitre la position du milieu nous utilisons un calcule simple :

On divise par deux la hauteur et la largeur du Shell moins la hauteur et la largeur du tableau dynamique divisé par deux :





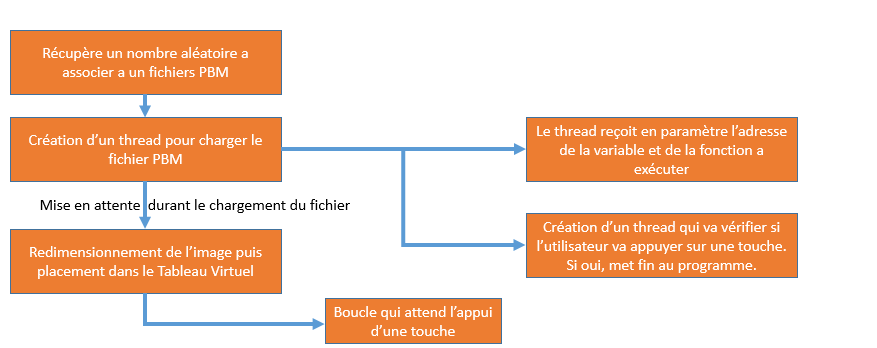
La structure sterm est une structure fournis par la bibliothèque **termios.h** elle permet de récupérer via la fonction  la hauteur et la largeur du shell via le flag TIOCGWINSZ



Puis la fonction va lire les données du tableau virtuel en le parcourant et les imprimer via printf() à l’écran

Si la donnée lu est un 1 on imprime un pixel «**█ »** et si la donnée et 0 on imprime un espace

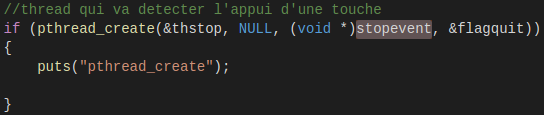
# L’écran de veille statique



L’écran de veille statique doit afficher une image aléatoire centrée dans la console.

C’est le lanceur qui générera le nombre aléatoire et lui enverra, l’écran statique quant à lui choisira le fichier à lire via les fonctions [d’ouverture du répertoire](#_Ouverture,_lecture_de) et de [lecture du fichier](#_Lecture_des_fichiers)

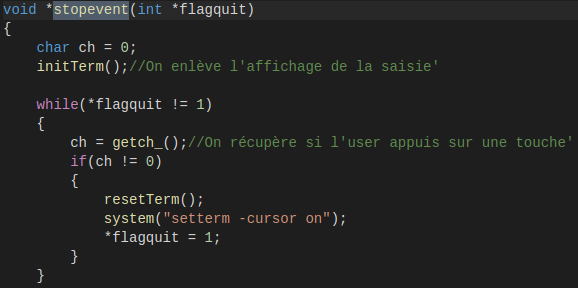
On a donc un dossier ou sont stocker les images PBM pour les récupérer.

Pour les charger, on va utiliser un thread, un thread se créer via une variable de type pthread\_t et la fonction pthread\_create qui prend l’adresse de la variable du thread ainsi que la fonction à exécuter dans ce thread et en dernier paramètre les paramètre à envoyer à la fonction chargée dans le thread :

Enfin le main () doit attendre la fin du chargement du fichier ; il doit donc attendre la fin du thread de lecture : pthread\_join() prend en paramètre le pointeur du thread et la valeur de retour de la fonction chargée dans le thread, ici à NULL car la fonction ne retourne rien…



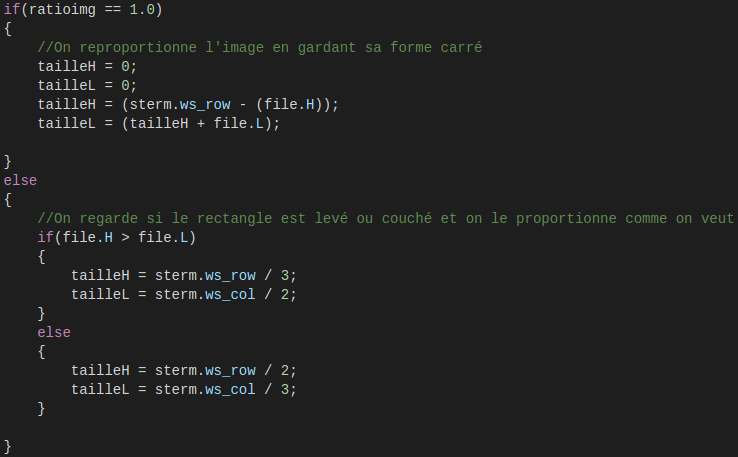
En parallèle à ce thread, il y en a un autre qui va vérifier lorsqu’on appuie sur une touche et qui va arrêter le programme si on détecte une touche :



Ensuite, petit plus, on va redimensionner l’image.

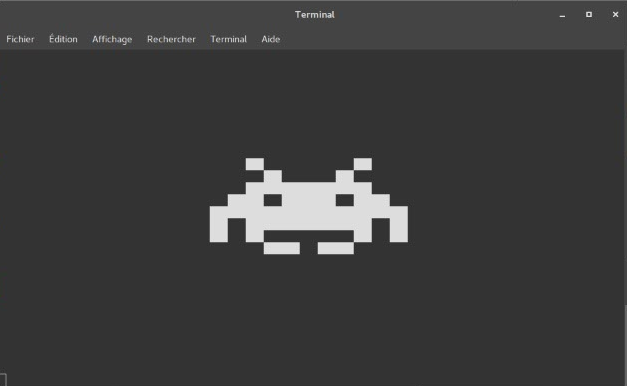
On va d’abord vérifier si elle est carrée. Si elle l’est, on va juste l’adapter à la console pour qu’elle ne perde pas son ratio et être déformer pareil s’il s’agit d’un rectangle

Si non, on va regarder dans quel sens est le rectangle, et adapter… pour que cela Rende bien

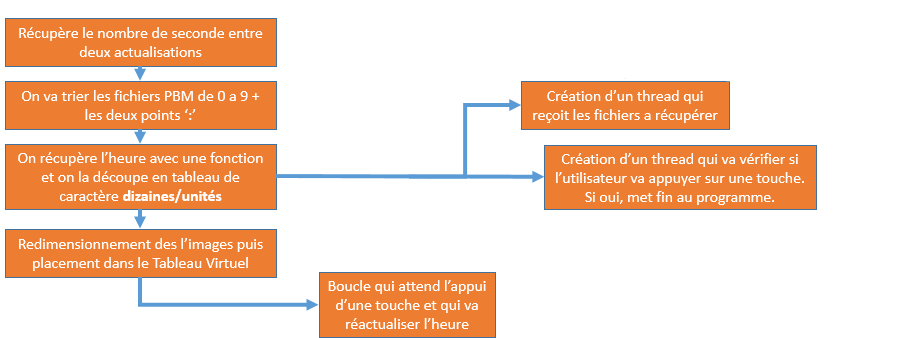


Ensuite, on a plus qu’à le placer. Pour cela on va créer un tableau virtuel ou on mettra l’image dedans. (Comme vu dans les sections sur le [fonctionnement des écrans](#_Fonctionnement_des_fonds)) puis l’afficher

Le programme rentre dans une boucle, qui, tant que l’utilisateur n’appuie pas sur une touche l’écran s’affiche. L’avantage d’une boucle est que les fonctions de création du tableau virtuel, du moteur de rendu et d’affichage sont ré appeler ainsi le programme va redimensionner l’image à la taille du Shell en temps réel (à une seconde d’intervalle) !



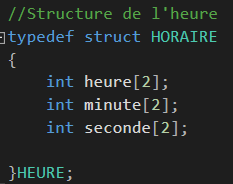
# L’écran de veille dynamique



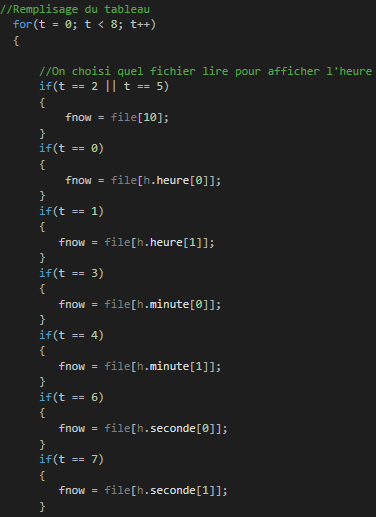
Cet écran de veille doit afficher l’heure centrée au milieu qui s’actualisera toute les N secondes. Il doit aussi afficher un message en dessous ou s’ajoutera un point toute les secondes sans actualisation.

Les fichiers sont lus par la fonction de lecture de fichier modifiée afin de remplir un tableau de structure (tableau qui contient tous les nombres + les deux point « : ») ce tableau est envoyé à la fonction du moteur de rendu modifié elle aussi elle prend en paramètre le tableau virtuel, le tableau contenant les fichier pbm ainsi que l’heure actuel :



Nous avons représenté l’heure dans une structure :

Les heures, minutes, seconde son un tableau de deux entiers pour représenter les dizaines et les unités cela nous permettra de choisir quel chiffre à afficher.



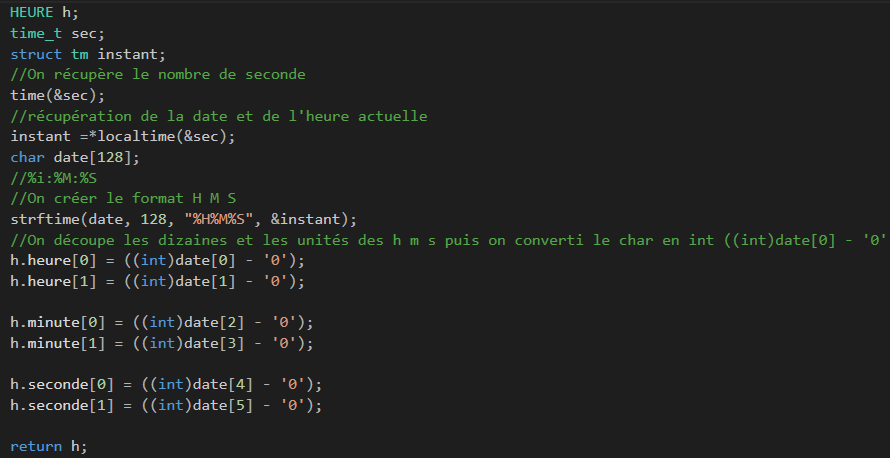
Dans un premier temps le moteur de rendu va trier les fichiers pbm de 0 à 9 et : en indice de 0 à 10, ces indices permettrons de choisir par rapport à l’heure qu’il est le bon chiffre :

Il y a 8 digit pour afficher l’heure : deux pour l’heure : deux pour les minutes, deux pour les secondes et deux pour les points « : », t est un curseur qui va imprimer les 8 digits dans le tableau virtuel en se déplaçant de la taille d’un digit (pour ne pas écraser le digit précédant) enfin sellons l’emplacement on met dans le tableau virtuel soit des dizaines soit les unités des heure, minutes et secondes…

L’heure quant à elle est récupérer via la fonction getheure()



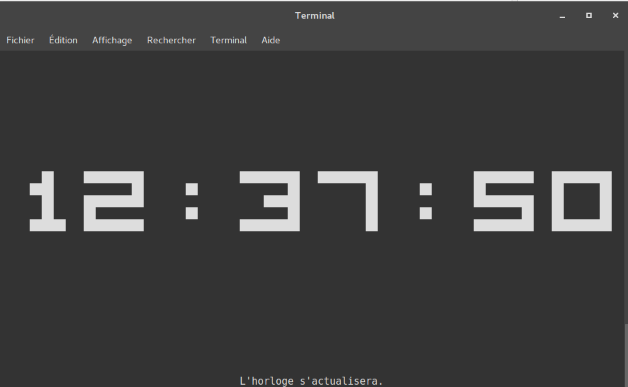
Elle retourne la structure représentant l’heure. Pour récupérer le temps nous avons inclus la bibliothèque **time.h**



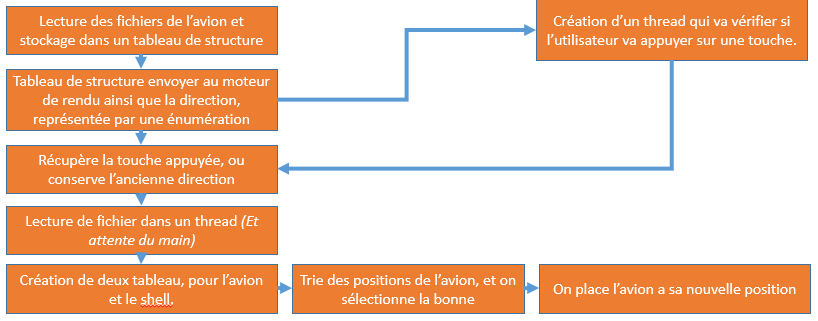
Nous utilisons la fonction time() qui prend en paramètre l’adresse d’une variable de type time\_t elle contiendra le nombre de secondes écoulées depuis le premier janvier 1970, cela correspond à la date maximum que peut remonter les calendriers des ordinateurs…

Puis le localtime() vas permettre remplir notre structure tm et ainsi découper ces secondes en date et heure. Enfin strftime permet de formaliser notre temps via le format qui lui est donné ici « %H%M%S » qui correspond aux heures minutes secondes

Enfin on découpe la chaine « date » et on remplir notre structure heure pour séparer les dizaines et les unités. Puis nous affichons l’écran via la fonction d’affichage



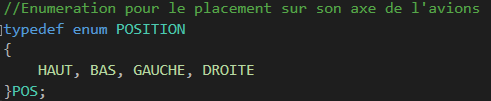
# L’écran de veille interactif



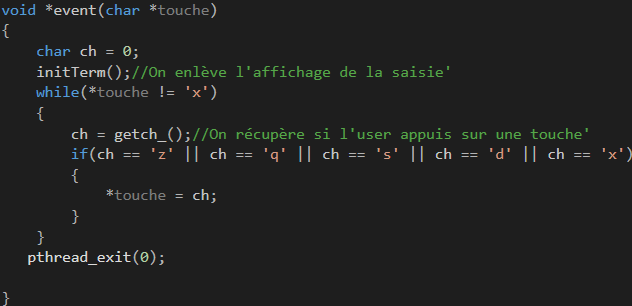
Cet écran de veille affiche un avion. Pour que cet écran ait un intérêt, il faut que l’utilisateur appuie sur les touches ZQSD pour bouger l’avion. Une fois que l’avion à commencer à bouger, il va continuer dans la direction jusqu’à ce que l’utilisateur ré appuie sur une touche de direction, ou sur la touche de sortie ‘x’. L’avion pourra évidement traverser les bords et réapparaitre de l’autre côté de l’écran.

Les fichiers sont lus par la fonction de lecture de fichier modifiée afin de remplir un tableau de structure (tableau qui contient toutes les positions de l’avion) ce tableau est envoyé à la fonction du moteur de rendu modifié elle aussi elle prend en paramètre le tableau virtuel, le tableau contenant les fichier pbm ainsi que la direction de l’avion HAUT BAS, DROITE, GAUCHE

Cette direction est représentée par une énumération :



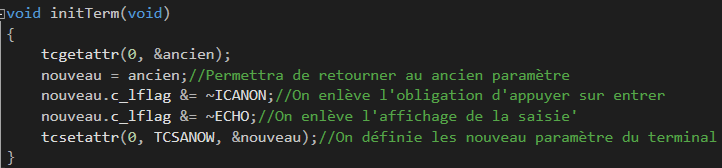
La récupération de la saisie de l’utilisateur se fait dans un thread :



Il prend en paramètre l’adresse de la variable qui contiendra la touche appuyée par l’utilisateur

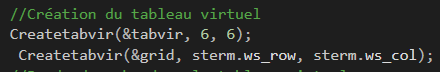
Nous avons désactivé l’affichage de la saisie et le fait de devoir appuyer sur ENTRER, pour cela nous avons inclus la bibliothèque **termios.h** et **unistd.h** qui nous permettrons de changer les entrées/sorties standard





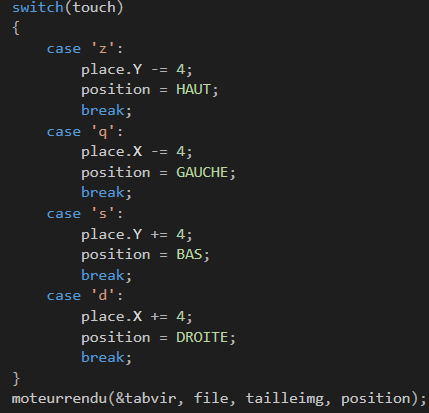
Bien sûr on créer deux structure termios ‘ancien’ pour avoir les anciennes configurations et revenir à la configuration avant utilisation et nouveau qui permettra d’utiliser les configurations changées comme indiqué ci-dessus

La lecture des fichiers se fait toujours dans un thread. Cependant nous créons deux tableaux virtuels un qui contiendra l’avion et un second qui fera la taille du shell et représentera la grille sur laquelle se déplacera l’avions

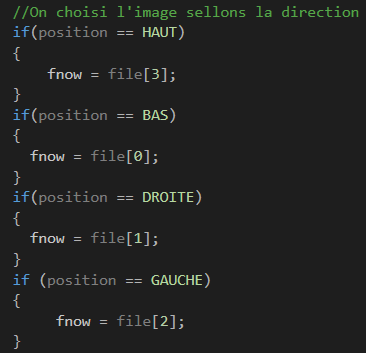


La première ligne correspond au tableau qui contiendra le pbm de l’avions, la second la grille où se placera le tableau virtuel contenant l’avion

Puis nous avons un switch qui sellons sur quelle touche l’utilisateur a appuyé nous enverrons au moteur de rendu quel fichier pbm à afficher :



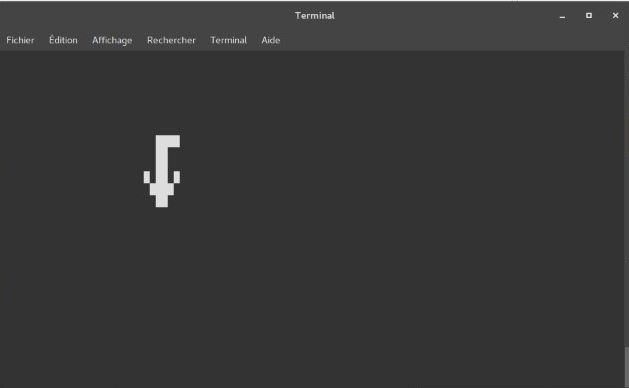
Comme pour l’affichage dynamique (avec l’horloge) on tri les pbm de l’avions de 0 à 3 dans le moteur de rendu. Puis sellons la direction choisie par l’utilisateur on chargera tel ou tel pbm :



Enfin pour permettre le déplacement à l’écran du tableau virtuel contenant le pbm de l’avion nous avons une fonction placement()



Elle prend en paramètre la grille qui représente le Shell (elle permet de le quadriller) le tableau virtuel qui vas être placé dans cette grille et enfin les coordonnées x et y où placer le tableau virtuel. Enfin cette fonction permet de faire passer d’un côté et de l’autre l’avion quand il sort de la fenêtre, on affiche enfin la grille avec la fonction d’affichage :



# Le lanceur

Le lanceur doit exécuter un des écrans de veille aléatoirement, tout en envoyant son choix à l’historique. Il doit aussi choisir quelle image afficher dans le statique, et la position initiale de l’avion dans l’interactif. L’utilisateur peut aussi choisir lui-même quel écran afficher via un paramètre

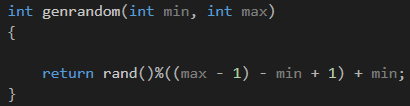
Le lanceur doit pouvoir générer un nombre aléatoire pour le choix de l’exécutable, mais aussi des images pour l’écran statique et les coordonnées de l’avoir pour l’interactif pour cela nous initialisons le système de génération aléatoire



Time retourne ne nombre de seconde depuis 1970

Srand génère un seed qui permet de générer un nombre aléatoire via la fonction rand()

Notre fonction de génération aléatoire prend un nombre minimum et maximum :



Puis l’utilisateur pourra aussi choisir un des trois fonds d’écran via les paramètre

-static pour le statique + le numéro de l’image à afficher

-dyn pour le dynamique + le nombre de seconde d’actualisation

-inter pour l’interactif + les coordonnées de l’avion

-stat pour lancer un menu qui proposera d’afficher soit les logs soit de les supprimer.

## L’historique.

Le but de l’historique est d’enregistrer les choix d’écran de veille, ainsi que la date et l’heure du début de l’affichage.

On a plusieurs informations à stocker dans l’historique :

* La date et l’heure de l’exécution du Lanceur
* Le type d’écran de veille lancer
* Quelle image a été choisie pour le statique
* La période d’actualisation pour le dynamique
* La position initiale de l’avion pour l’interactif

