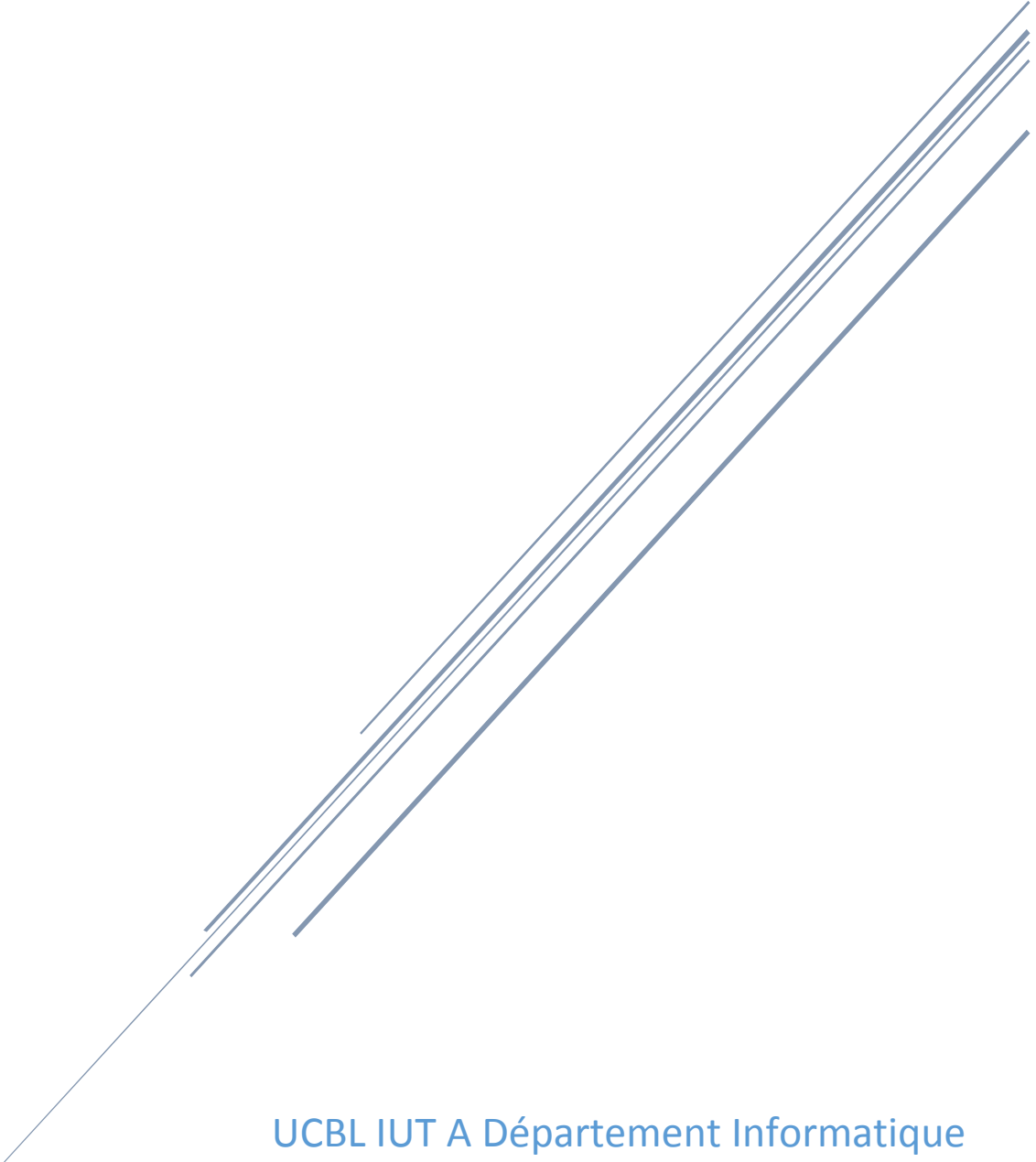


PTUT S2 COMPTE RENDU

Architecture des ordinateurs : Console de jeu rétro



UCBL IUT A Département Informatique
DREVET Yoann, MOURIK Hamza, VIVIER David, GOURGAUD
Alexandre

Table des matières

Fiche descriptive :	2
Identification du besoin et problématisation.....	3
Etude des besoins.....	4
Enquête	5
Solutions.....	6
Conserver Windows	6
Installer une distribution de Linux récente	7
Gentoo.....	7
Linux From Scratch	8
Solution Retenue	8
Étude préliminaire	9
Bootloader	9
Linux From Scratch	11
Méthodologie et planning.....	12
Introduction.....	12
Première partie	12
Deuxième partie	13
Troisième partie	14
Quatrième partie	15
Conclusion	16

Fiche descriptive :

Numéro du sujet : 2-244

Titre : Projet tuteuré architecture des ordinateurs, création d'une console de jeu rétro.

Tuteur : Michael MARISSA

Membres du groupe :

DREVET Yoann

MOURIK Hamza

VIVIER David

GOURGAUD Alexandre

Résumé du projet :

Ce projet consiste en la mise en place sur une machine en ressources limitées, d'une solution permettant à n'importe qui de jouer à d'anciens jeux d'arcade comme il l'aurait fait sur la machine originale. Il doit pouvoir jouer sans connaître techniquement la machine ni réaliser de tâches complexes.

Technologies envisagées pour la réalisation du projet :

Bootloader personnalisé

Linux From Scratch

Emulateurs : MAME, puNES, MasterGear

Debian

Méthode de gestion de projet :

Méthode agile Scrum

Probabilité de poursuite :

Il y a de fortes chances que tout ou partie du groupe poursuive le projet aux semestres 3 et 4, cependant certains problèmes que nous préciserons pourront nécessiter quelques modifications dans la machine utilisée. Il s'agit notamment de l'espace disque disponible.

Difficultés rencontrées :

Tout d'abord nous avons dû prendre connaissance de la machine. Du fait de son ancienneté elle ne répond pas aux mêmes normes que celles d'aujourd'hui. De plus la moitié du groupe n'avait pas de connaissance approfondie des technologies LFS et Bootloader, c'est pourquoi nous avons fait une mise à niveau pour que chacun parte avec les mêmes bases.

Identification du besoin et problématisation

Considérons le scénario suivant :

Une personne retrouve dans son grenier la machine suivante :

IBM PC 300 XL :

Processeur : Pentium 2 - fréquence d'horloge 233 Mhz

Mémoire RAM : 64 MB

Disque dur ATA : 2560 MB

Carte Ethernet : 10 MB/s

Carte Son : Stéréo 16 bits 44000Mhz

Carte graphique : S3 Trio64V2 accélération graphique 2Mo

Bien que très limitée en ressources, il considère que l'on peut trouver une utilité à celle-ci. Il adore également les jeux vidéo et notamment les jeux rétro comme ceux de consoles types 8/16 bits ou bien de bornes d'arcades. Ne possédant pas les machines originales passées aujourd'hui dans le domaine public, il se demande comment faire pour utiliser son vieux PC en tant que machine de jeux.

Au-delà de la fiction ce projet a pour but de démontrer l'intérêt de la réutilisation, du recyclage de produits qui ne possèdent aujourd'hui plus aucune valeur ajoutée. Il va de soi que pour rentabiliser l'effort, aucun investissement n'est possible.

En plus de l'unité centrale, on dispose d'un large choix d'anciens composants pour compléter la machine. En voici quelques exemples :

Disque durs ATA : 10 Go, 20 Go, 40 Go, 200 Go

Carte d'acquisition vidéo

Carte FireWire

Claviers, Souris en Ps2

Moniteurs CRT de résolution 800x600

On cherche donc à répondre à la problématique suivante :

COMMENT, AU MOYEN DES COMPOSANTS LISTES CI- DESSUS PEUT-ON JOUER A DE VIEUX JEUX VIDEO EMULES DE LEUR MACHINE DE DEPART ?

Par-delà l'attrait ludique, il s'agit au final d'acquérir nombre de connaissances et de compétences dans les domaines du hardware (normalisations), du fonctionnement de Linux à très bas niveau mais aussi en programmation C et assembleur. Ainsi ce projet peut être vu comme un exercice applicatif aux cours d'architecture, de Linux et de programmation C. De plus des notions en UI (User Interface), et UX (User experience) seront également requises pour la partie graphique.

Etude des besoins

Pour l'utilisateur :

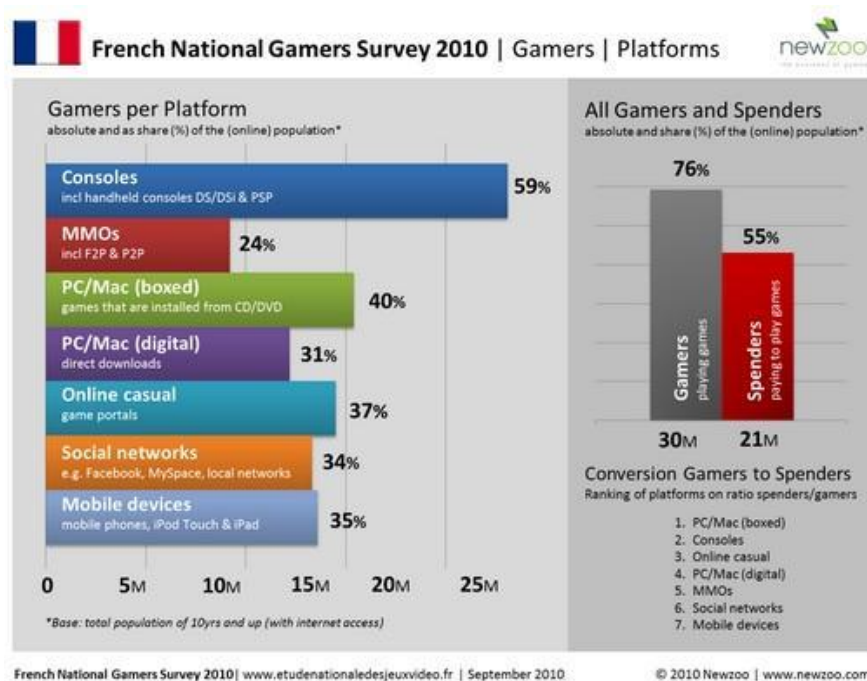
Il devra pouvoir allumer la machine et accéder sans aucune action au menu de sélection des jeux.

Il ne devra jamais être confronté directement au système d'exploitation de la machine.

Dans l'absolu même si aucun travail sur les émulateurs eux-mêmes ne sera accompli, le choix entre ceux-ci devra lui se faire dans une interface la plus minimale possible.

Pour appuyer notre argumentation nous rappelons que la majorité des joueurs, d'une manière générale ont comme principal support de jeu la console et donc, une interface similaire s'impose pour notre machine.

Fig. 1 - Répartition des joueurs



Pour le système :

Sachant qu'aucun investissement n'est possible comme précisé plus haut, le système final devra se contenter des composants à disposition.

Il faudra dans la recherche de solutions garder à l'esprit l'existence des éléments limitants de la machine qui ne peuvent être modifiés tel que le processeur, la mémoire RAM, et la carte graphique qui dans le cas présent est soudée sur la carte mère.

Enquête

Conscients de la difficulté du projet, il nous a semblé judicieux de faire appel à un, voire à plusieurs intervenants extérieurs pour nous conseiller sur les solutions envisageables.

Dans un premier temps nous avons pris contact avec M. Mickael Fortunato, professeur en technologies Linux (et plus généralement en système) à l'école d'informatique Supinfo de Lyon.

Lui exposant comme ici le problème, nous lui avons tout d'abord demandé un avis sur la faisabilité générale du projet. Il valida celui-ci tout en précisant que le maximum d'optimisation système était à prévoir.

Il nous a donc aiguillés vers l'une des solutions détaillées ci-après.

De plus, ce contact a été sollicité à nouveau comme nous le verrons dans la partie concernant le cahier des charges.

Forts de cette bonne expérience nous avons ensuite contacté M. Corentin Beal, étudiant à Supinfo et ami du groupe qui a eu l'occasion de mettre en place une borne d'arcade sur machine virtuelle VMware. Bien que n'étant pas expert sur la question système il nous a permis de démystifier l'épineuse question dans le choix des émulateurs à notre disposition pour chacune de nos solutions envisagées.

De plus nous avons également établi bon nombre de contacts sur les forums officiels de LFS qui ont pu largement nous aider à obtenir les informations que nous livrons aujourd'hui.

Solutions

Nous avons donc établi 4 solutions supposées jusque-là pouvant répondre à tout ou partie du problème.

Bien que certaines d'entre elles puissent de prime abord sembler inenvisageables, nous avons tout de même décidé de les conserver afin de mieux cerner l'intérêt que nous portons à notre choix final.

Conserver Windows

A l'origine, le système qu'intègre le PC est Windows 98. Il serait donc très facile de mettre en place un émulateur par simple installation de celui-ci. Les avantages et inconvénients ci-dessous sont le résultat d'essais pratiques durant notre étude préliminaire de la machine.

Avantages

Bien que l'on ne puisse pas réellement personnaliser les tâches, l'interface est connue de tous.

Déjà installé, il tient sur le disque de 2,5 Gb et reste dans l'absolu un scénario efficace et rapide à mettre en place.

Inconvénients

Le système ne permet pas de personnaliser le boot loader.

On ne pourra pas non plus utiliser de bootsplash personnalisé, cette fonction n'existe plus depuis la version ME.

Le support technique de Windows 98 (seule version de Windows pouvant s'exécuter) est arrêté depuis longtemps. Ainsi, des failles resteront non corrigées, notamment avec les protocoles réseau. Le système peut ainsi devenir vulnérable aux attaques.

De plus, 64 MB de RAM reste peu si l'émulateur dépasse les 10 MB de RAM consommés.

Installer une distribution de Linux récente

Avec un système IA32 x86, le choix des distributions est large : n'importe quelle version dont le kernel peut s'exécuter sur processeur possédant le jeu d'instruction MMX est utilisable. On choisit ici d'en prendre une relativement légère : Debian.

Avantages

Cette distribution est assez simple à installer.

Elle a également l'avantage d'être très stable, sans failles et constamment à jour.

Tous les mécanismes tels que l'authentification sont automatisables.

On peut utiliser un bootloader personnalisé ainsi qu'un bootsplash.

Inconvénients

Debian est beaucoup trop gourmand en ressources une fois l'interface graphique installée. Il nécessite dans tous les cas 128 MB de RAM à lui seul.

Gentoo

Après discussion avec Monsieur Fortunato, nous avons décidé d'inclure Gentoo, cette distribution peu connue part avec l'idée d'optimiser, depuis le kernel, l'intégralité du système. Nous avons pu essayer celui-ci dans une machine virtuelle créée par Corentin Beal.

Avantages

Gentoo possède les mêmes avantages que ceux cités précédemment avec Debian en étant beaucoup plus léger.

On compile quasiment tout le système et on peut également optimiser le kernel.

Inconvénients

Basé sur un bloc fixe de logiciel dans son paquet de base appelé stage 3, Gentoo est lourd sur le disque et même si on approche l'optimisation parfaite en utilisant un kernel totalement personnel, le nombre de services est tel que trop peu de RAM reste utilisable pour les processus utilisateurs et donc pour les émulateurs.

Linux From Scratch

Littéralement cette dernière option semble radicale, on décide de ne partir de rien.

On aura ainsi comme on le détaillera plus tard la possibilité de choisir absolument tout, de tout compiler et d'optimiser le système en chaque point.

Avantages

Il correspond à la solution parfaite pour répondre à tous nos besoins, en reprenant les avantages de Gentoo.

Il est créé de A à Z ce qui nous permet de mettre l'accent sur le fait que la machine ne saura faire et sera optimisée que pour la seule tâche de jeu.

Au final il permet d'obtenir un niveau maximum d'abstraction entre le joueur et la machine.

Toutes les sources proposées par la communauté sont sans cesse vérifiées et mises à jour.

Par-delà le côté minimaliste, la communauté de développeur LFS, très étendue, sera un précieux allié pour la réussite du projet.

Inconvénients

Il va de soi que le fait de tout faire soi-même est long.

On explicitera ce point de vue ultérieurement.

Enfin LFS reste le plus difficile d'un point de vue technique.

Solution Retenue

Après avoir pesé le pour et le contre de chaque système, nous avons décidé de mettre en place notre machine avec Linux From Scratch.

En effet c'est le seul à véritablement répondre à toutes les contraintes que nous posons au départ. Bien que sa mise en place soit longue, nous pensons qu'en parallélisant celle-ci avec la création du bootloader et en partageant le groupe en deux les délais seront respectés.

La mise en place des services système et des programmes qui intervient après LFS répond parfois au nom de BLFS pour Beyond Linux From Scratch. On ne parlera pas de BLFS à proprement parler car on ne traitera que certains aspects, les autres n'étant pas requis par le système final.

D'ores et déjà un changement du disque dur pour une capacité plus grande est nécessaire. On choisit donc un disque de 20 Gb dont le temps d'utilisation mesuré est inférieur à 2000 heures, donnée acceptable compte tenu de l'âge du disque.

Étude préliminaire

Bootloader

Lors de la mise en place d'un système d'exploitation, il est nécessaire de mettre en place le Master Boot Record (MBR), sur le secteur 0 du disque dur. Le MBR contient la table des partitions (sur 64 octets), le *magic word* de vérification (2 octets) et la routine d'amorçage ou bootloader (sur 446 octets).

Nous avons besoin de nous affranchir des bootloaders classiques comme Grub, qui nécessitent une partition supplémentaire d'espace non négligeable (256 MB) du disque dur et donc réduiraient l'espace disponible pour les émulateurs.

Ainsi, dans l'optique de la construction d'un système à partir de rien, et avec l'enjeu d'optimisation de l'espace, nous allons nous-mêmes construire un bootloader personnalisé. Cela permettra d'une part d'utiliser uniquement le premier secteur pour l'amorçage, sans recours à un espace disque supplémentaire, et d'autre part de personnaliser en profondeur la machine dès son démarrage.

Que peut-on personnaliser dans un bootloader ? Le texte affiché au démarrage, voire même l'affichage d'une image.

Cependant, l'affichage d'une image nécessiterait de la stocker au préalable. Or nous désirons optimiser l'espace disque : le stockage d'une image de qualité décente n'est donc pas quelque chose d'envisageable sur l'espace disponible pour le bootloader.

En revanche, l'affichage d'une image reste possible : en utilisant une image qui soit non pas stockée mais générée par un algorithme.

Nous avons alors pensé à l'ensemble de Mandelbrot. Il s'agit d'une figure fractale, historiquement la première à avoir été étudiée et générée avec un ordinateur. L'algorithme est relativement simple, et de plus il est bien maîtrisé par l'un des membres du groupe.

En outre, la machine étant quelque peu vétuste, le calcul et l'affichage de la fractale permettront à chaque utilisation de vérifier automatiquement le fonctionnement correct de la machine dès le démarrage. En effet, le calcul de l'ensemble de Mandelbrot manipule des réels et fait appel à des routines d'affichage. Un mauvais affichage de la fractale permettra de détecter un mauvais fonctionnement de la machine.

Ainsi, le bootloader devra effectuer les tâches suivantes :

- Afficher le nom de la machine

- Afficher l'ensemble de Mandelbrot

- Paramétrer le noyau puis lui passer la main

En disposant pour cela de 446 octets.

Il était par conséquent indispensable d'étudier la faisabilité de l'intégration de cet algorithme dans le bootloader. Nous avons donc effectué une estimation de la place minimale qu'occupera l'algorithme.

Nous avons d'abord écrit l'algorithme en C. Cet algorithme utilise des nombres complexes, ce qui pourrait être lourd à la compilation, on a donc effectué des calculs afin de se ramener à des réels. On commencera par afficher la fractale en utilisant des caractères plutôt que directement des pixels (nous verrons l'utilisation des modes vidéo plus loin dans ce document).

Ce fichier .c a ensuite été compilé en fichier objet (code machine) pour une architecture 32 bits, sans utiliser aucune fonction de la librairie standard. En effet la librairie standard du C, requise pour bénéficier de la fonction printf(), est trop lourde et surtout elle n'est pas utilisable avant le démarrage du noyau. Par conséquent on crée une fonction assembleur qui utilise une routine du BIOS pour afficher un caractère. Le fichier objet obtenu a une taille de 988 octets.

Nous avons récupéré le code source en assembleur d'un secteur de boot depuis un tutoriel de développement de systèmes d'exploitation (le code est dans le domaine public). Après étude de ce secteur de boot, nous l'avons réduit en ne conservant que les parties qui nous étaient utiles (affichage d'un message, chargement d'un noyau...). Après assemblage, le code exécutable occupe 206 octets.

On arrive donc à un total de $988 + 206 = 1194$ octets. Ceci est évidemment trop gros pour tenir sur les 446 octets de secteur de boot.

Nous avons donc décidé d'étendre l'espace attribué à notre bootloader, en créant une partition (appelée stage 2 du bootloader) contenant une seconde partie de code exécutable. Il s'agit de la même méthode qu'utilise GRUB. Cependant, la partition que nous allons mettre en place sera bien moins volumineuse : 2 Mo (qui est le minimum possible), contre 256 Mo pour Grub, ce qui permettra donc malgré tout d'optimiser l'espace disque.

Linux From Scratch

Conscients de la donnée élémentaire qu'est la variation du temps de compilation en fonction de la puissance d'une machine, il était alors nécessaire de savoir quel temps allait mettre le système à se compiler sur un Pentium II.

Pour répondre à cette question, nous nous baserons sur la documentation officielle de LFS.

Sachant que beaucoup de personnes se posaient la question qui était de savoir quel est le temps de compilation de chaque composante de LFS, et surtout que LFS est compilé sur grande variété de processeurs aux performances variables, donner un temps de compilation est impossible.

C'est ainsi qu'apparaît la SBU pour Standard Build Unit. Cette unité représente l'étalon temps de compilation du système. En effet il est donné que le premier package que l'on compilera à savoir Binutils aura pour un SBU égal à 1. Connaissant cette durée et connaissant pour chaque programme son temps SBU, on pourra donc estimer avec une précision relative les temps de compilation total.

Cependant cet indice reste d'une fiabilité incertaine. Il dépend de plusieurs facteurs tels que la version du compilateur gcc utilisée.

De plus le fait que certains processeurs soit multicoeurs rend la tâche plus complexe, certaines tâches étant parallélisables on ne peut prédire précisément le speed-up pour chaque processeur car cela reviendrait à obtenir la répartition entre le temps d'exécution séquentiel et le temps d'exécution total sur chaque processeur.

La documentation officielle ne traite pas directement des ressources système minimales requises. C'est pourquoi cette partie de l'étude peut sembler si informelle.

Pour répondre à la question nous avons fait appel à monsieur Fortunato. Il confirme la faisabilité du système ayant eu personnellement l'expérience de mettre en place un routeur sur la même configuration avec seulement 32 MB de RAM.

Enfin, la communauté très active sur les forums de LFS a pu nous confirmer que de telles tentatives avaient déjà connu le succès précédemment.

Méthodologie et planning

Introduction

Cette partie va décrire le déroulement de chaque étape du projet, en commençant par la finalisation du présent document jusqu'aux premières parties de jeu et de ce fait à la fin du semestre 4.

Nous avons décidé de scinder notre projet en 4 grandes parties, qui comportent chacune une succession d'étapes que nous allons décrire.

La première partie aura pour but de traiter toutes les tâches sans lesquelles la création du bootloader et de LFS ne pourraient commencer.

Les deuxième et troisième parties s'effectueront en parallèle et constitueront la création du système LFS d'une part et la création du bootloader d'autre part.

La dernière partie intégrera la mise en commun des parties 2 et 3 mais également la finalisation du projet.

On pourra suivre l'avancement du projet en se basant sur le document Microsoft Project attaché à ce document.

La plage de travail hebdomadaire du groupe sur le projet est le mercredi de 14h à 18h.

Bien que selon la tâche représentée sur le planning l'un des membres puisse ne pas être présent lors d'une séance, les ressources affectés ne sont que minimales : l'un ou l'autre des membres pourra, si besoin est, aider un autre membre dans sa tâche.

Première partie

Nous avons décidé pour le début du projet de réserver un temps afin que chacun puisse reprendre connaissance du présent document ainsi que du planning. En effet, plus de deux mois se seront écoulés et se lancer pour ainsi dire tête baissée nous semble difficile.

Ensuite nous devons mettre à jour la machine sur le plan matériel. On commencera par un nettoyage complet : restée très longtemps à l'abandon, il est dangereux d'utiliser la machine en l'état, et ce sachant qu'elle va fonctionner de longues heures durant.

Avant de débiter leurs parties respectives, les deux binômes prendront connaissance de la partie qui leur incombe.

Point final avant la mise en parallèle des tâches, nous installerons un système linux de type TazLinux qui sera utilisé au début de la création de Lfs.

Deuxième partie

Ayant réglé dans la première partie tous les pré-requis liés à l'installation, on débute ici avec une prise de connaissance générale de LFS. On doit commencer par maîtriser les règles typographiques propres à LFS, qui ne sont pas directement des standards POSIX mais aussi s'informer sur les errata publiés au sujet de chaque paquet. Ceci dans le but ne pas, plus tard, télécharger de versions instable des sources.

Bien que LFS ne soit pas un système global en soit, il est mentionné un système de version dont les notes de mise à jour devront avoir fait l'objet d'une attention particulière.

Par la suite nous allons devoir créer la partition LFS. C'est un point clé stratégique car elle devra être assez grande pour contenir les sources ainsi que les fichiers compilés. On prévoira aussi un large espace de type swap en raison de la très faible quantité de RAM et de l'usage intensif de celle-ci par le compilateur. De plus nous conserverons 8Mb d'espace libre (en raw, pour ne pas s'encombrer d'un système de fichiers) entre le début de la première partition et la MBR pour insérer le stage 2 de notre bootloader. Pour la partition LFS, on choisira ext4 comme système de fichier en raison de sa robustesse et de sa fonction de journalisation très performante. Cette partition sera ensuite montée au sein du système hôte, et, une variable d'environnement LFS sera également définie pour rendre les accès au point de montage plus rapide.

Nous devons ensuite télécharger l'intégralité des paquets et des patches utilisés au cours de l'installation.

Avant de passer à la construction d'un système temporaire, nous allons créer l'utilisateur LFS, ce qui rendra nos commandes plus simples et qui aura pour deuxième but de conserver l'intégrité du système en hiérarchisant les tâches.

Nous ne détaillerons pas la partie suivante pour chaque paquet car cela serait en toute logique redondant à expliquer.

Ainsi pour chaque paquet on créera un nouveau répertoire, on copie les sources, on lance un `./configure` avec les paramètres propres au paquet puis on lance `make` et `make install`.

Une fois tous les paquets compilés on rend l'utilisateur root propriétaire de la partition LFS montée.

Maintenant que le système temporaire est créé, la création du système en lui-même commence et avec celle-ci on déplace la racine du système courant à la racine de notre disque Lfs.

Le kernel vient en premier et sera optimisé pour l'occasion, on choisira de conserver seulement les fonctionnalités utilisées sur un Pentium II.

Ensuite tout comme pour le système temporaire, on compile un par un les paquets.

Avant de rendre le système bootable on implémente les scripts que l'on utilisera pour passer d'un noyau seul au processus init puis ensuite au système complet.

Bien que le bootloader final soit différent on installe pour faire des tests, le bootloader Grub.

Troisième partie

Nous allons procéder pas à pas en implémentant plusieurs versions du bootloader, d'un simple secteur de boot affichant un message jusqu'au bootloader final capable de générer un boot splash et charger le système final.

Avant de procéder à l'implémentation, nous devons tout d'abord nous renseigner sur les moyens d'afficher du texte à l'écran. Une recherche rapide nous a permis de trouver une liste des routines proposées par le BIOS de l'ordinateur, facilitant ainsi le développement d'applications bas niveau. Le BIOS nous fournit donc une routine permettant d'afficher des caractères à l'écran via le vecteur d'interruption 0x10.

A ce stade, nous pouvons implémenter un premier secteur de boot affichant un message. Le fichier final devra faire 512 octets (taille d'un secteur) et sera écrit sur le secteur 0 d'une disquette ou d'une image de disque. Le test pourra donc se faire au choix sur une machine physique ou virtuelle.

Une fois l'affichage géré, nous pouvons nous occuper du chargement et de l'exécution d'un programme. Pour nos tests, nous commencerons tout d'abord par charger un programme simple qui, dans la version finale, sera remplacé par le noyau. Tout comme pour l'affichage, le BIOS nous fournit une routine permettant d'effectuer des lectures ou écritures sur un disque, via le vecteur d'interruption 0x13.

Une nouvelle version peut donc être implémentée, capable à la fois de générer un boot splash en mode texte puis de charger un programme depuis un autre secteur du disque.

L'étape suivante consiste à remplacer le boot splash textuel par une version graphique. Pour cela, nous devons nous renseigner sur les différents modes d'affichage graphique que peuvent gérer l'ordinateur. Heureusement pour nous, la plupart des BIOS prennent en charge les VESA BIOS Extensions, des routines permettant de changer le mode vidéo de l'ordinateur. Il nous reste donc à nous renseigner sur l'utilisation de ces modes vidéo.

On s'affaira ensuite à mettre en place le lancement du stage 2.

Puis dans ce stage 2 on compilera l'algorithme de Mandelbrot pour un affichage graphique.

Finalement, on ajoute dans le stage 2, après Mandelbrot, le code qui permettra enfin de configurer le noyau avant de lui donner le relais.

Nous pouvons donc nous servir des étapes précédentes afin d'implémenter la version finale du bootloader, capable, dans un premier temps de générer et d'afficher la fractale de Mandelbrot, puis, dans un second temps, de charger le noyau Linux.

Quatrième partie

Nous débuterons cette partie par l'implémentation de la MBR contenant le bootloader personnalisé. On va ainsi recopier à la place de la MBR existante contenant le stage 1 de Grub, la nouvelle MBR contenant d'une part le bootloader personnalisé et d'autre part la copie de la table des partitions présente dans la MBR initiale. Ensuite nous copierons notre stage 2 à la place de celui de grub dans l'espace en raw crée dans la Deuxième partie..

Comme nous le disions plus tôt l'ensemble des actions suivantes peuvent être regroupées sous l'acronyme BLFS. Cependant nous n'emploierons pas ce terme car notre travail est ici plus modeste tant les fonctionnalités requises par le système sont réduites.

On commencera par ajouter les utilisateurs, puis, on configurera le système de login.

Toujours dans l'approche sécuritaire on devra vérifier les failles sur les principaux composants réseaux ainsi que sur le paquet Sudo. Pour plus tard se connecter à internet on devra également installer un pare-feu.

Ensuite, comme dans la [Deuxième partie](#), on installera les bibliothèques et les bibliothèques graphiques requises par les émulateurs.

Avant de se connecter au réseau, d'autres utilitaires seront ajoutés pour la carte son notamment.

On procédera ensuite à la connexion au réseau en configurant un client DHCP pour que chaque connexion soit ensuite automatisée.

Avant de pouvoir utiliser l'interface graphique pour nos émulateurs, on commencera par installer XOrg, le configurer et lui ajouter un gestionnaire de fenêtres minimaliste. On choisit ici openbox pour sa très grande légèreté.

On peut maintenant installer les émulateurs, Mame, en charge des bornes d'arcade, puNES pour Nintendo et MasterGear pour la Sega Master System.

Pour terminer on automatisera l'authentification de l'utilisateur en configurant le shell qui aura aussi pour tâche d'afficher le directement l'interface graphique mettant à disposition les émulateurs.

Enfin on ajoutera bootstrap qui cachera le code affiché pendant le temps de boot de la machine.

Si tout s'est bien déroulé il ne reste plus qu'à allumer la machine et à jouer.

Conclusion

A partir des analyses et planifications dont nous rendons compte dans le présent rapport, nous pouvons à présent affirmer que notre projet est faisable sur la durée des semestres 3 et 4.

Les études préliminaires réalisées vont permettre à chaque membre du groupe de bien cerner et maîtriser la partie qui lui est assignée. La répartition des tâches et leur planification dans le temps nous permettront à la fois de garder une vue d'ensemble du projet, et de structurer le travail de chacun en s'organisant correctement.

Finalement, la rigueur dans le travail fourni ainsi que le respect du planning assureront l'aboutissement du projet.

Lorsque le projet aura abouti, il sera encore possible d'en améliorer l'ergonomie, afin que l'utilisateur n'ait même pas à passer par le bureau. Par exemple en programmant une interface Swing avec des icônes qui, de manière très intuitive, permettront à l'utilisateur de sélectionner l'émulateur, de choisir son jeu et de commencer à jouer.