Méta-morphosis – Rapport 1

Owl's team

Pierre Bourdon <bourdo_p@epita.fr>
Florent Ditte <ditte_f@epita.fr>
Tiphaine Petit <petit_t@epita.fr>
David Stavaux <stavau_d@epita.fr>

Table des matières

1 Présentation du projet **4**
1.1 Méta-morphosis 4
1.2 Owl's Team 4
1.3 État actuel du projet 5

2 Fonctionnement de la PlayStation **7**
2.1 Différents composants 7
2.2 Communication inter-composants via registres 8
2.3 Communication inter-composants via DMA 9

3 Traducteur **10**
3.1 Principe de fonctionnement 10
3.2 LLVM 10
3.3 Possibilités d'amélioration 12

4 Bibliothèque de support **13**
4.1 Instructions du CPU 13
4.2 Accès à la mémoire 13
4.3 Position du code 14

5 Reprogrammation du BIOS **15**
5.1 Rôle du BIOS 15
5.2 Enjeu de la reprogrammation 16
5.3 État actuel 16

Introduction

Après plusieurs mois sans nouvelles de notre part, ce rapport de soutenance présente toutes les avancées qui ont été réalisées par rapport à notre projet durant ces deux derniers mois. Pendant ce laps de temps assez long, notre travail, pas forcèment continu ou régulier, nous a amené à un point où plusieurs composants sont terminés et où les fondations permettant de construire le reste sont bien en place. Ainsi, c'est sur l'idée d'un travail bien accompli que s'approche cette deadline marquée par la première soutenance.

Ce rapport est séparé en plusieurs parties qui parlent chacune d'un élément différent sur lequel nous avons travaillé. Nous commencerons par une présentation du projet, du groupe et de son état actuel, avant d'enchainer avec une description complète du fonctionnement de la PlayStation, la console sur laquelle notre sujet de projet porte. Ensuite, les trois composants principaux de notre projet : le traducteur, les bibliothèques de support et le BIOS, seront exposés et expliqués en détail.

Notez que ce rapport est téléchargeable en format numérique (PDF) sur le site web de notre projet : `http://meta-morphosis.ath.cx`. Bonne lecture !

1 Présentation du projet

1.1 Méta-morphosis

Notre projet, nommé *Méta-morphosis*, est un projet assez unique en son genre, dans le sens où il change la façon habituelle de résoudre le problème du lancement de jeu de consoles sur PC. En effet, contrairement à la méthode utilisée communèment qui constitue en l'émulation du système cible, notre but est de transformer l'exécutable d'une console, la PlayStation, en un exécutable directement compréhensible par le système hôte[1]. Ainsi, ce programme transformé (ou *métamorphosé*) sera directement exécutable par le processeur de l'hôte sans couche d'émulation et en appelant directement les API natives plutôt que les fonctions habituelles de la PlayStation.

Malgré tout, il n'est pas possible de toujours traduire mot pour mot chaque appel système de la PlayStation en son appel correspondant à une API habituelle comme OpenGL, ALSA ou SDL. Pour pallier à ce problème, l'exécutable traduit est lié à des bibliothèques dynamiques traduisant les accès au matériel de la PlayStation en appels aux APIs du système. Interchangeables, elles permettent une flexibilité et une évolutivité du programme au fil du temps, mais également une plus grande possibilité de configuration [2].

Ce projet, bien que difficile et nécessitant de plonger dans les entrailles du fonctionnement de la PlayStation, est également un exercice de programmation de haut niveau, nécessitant de décompiler un programme presque exactement et d'en déduire une traduction réalisant le même rôle. Ainsi, il mélange deux domaines opposés qui sont l'analyse de code et l'émulation de matériel, ce qui le rend intéressant à programmer.

1.2 Owl's Team

Notre groupe est parmi la majorité des groupes constitués de 4 membres, qui sont les suivants :

1. Pierre `bourdo_p` Bourdon, chef de projet
2. Florent `ditte_f` Ditte
3. Tiphaine `petit_t` Petit
4. David `stavau_d` Stavaux

[1] Le plus souvent, un exécutable ELF ou PE pour architecture x86.

[2] Par exemple, si le système ne supporte pas l'accélération OpenGL, il pourrait toujours utiliser une bibliothèque graphique comme SDL ou XVideo.

Tous aussi motivés pour travailler, nous nous répartissons équitablement les tâches à réaliser en priorité afin d'arriver aux objectifs de chaque deadline à temps.

1.3 État actuel du projet

Après deux mois de travail plus ou moins régulier, notre projet a atteint un état où il est capable de faire fonctionner différents composants de la PlayStation et plusieurs programmes de démonstration. Ces programmes de démonstration sont issus du SDK le plus utilisé pour le développement de homebrews [3] sur PlayStation. Ces programmes ont l'avantage d'être disponibles sous forme source, ce qui facilite le debug des problèmes de notre projet. De plus, ils illustrent pas à pas toutes les possibilités de la PlayStation et l'utilisation de tous ses composants.

Méta-morphosis est actuellement capable de faire fonctionner la majorité de ces démos utilisant uniquement le GPU et les manettes (la gestion du son n'étant pas encore au programme), y compris lors de tâches comme l'affichage de texte ou la gestion de polygones transparents. Il gère également un joueur utilisant soit une manette de PlayStation en USB, soit un clavier. Ces fonctionnalités le rendent potentiellement utilisable pour la traduction d'un jeu 2D sans son pour la PlayStation, ce qui n'est, malheureusement (ou heureusement), pas quelque chose de courant.

[3] Programmes développés de manière amateur par des passionnés n'ayant pas de quoi payer une licence officielle de Sony pour développer des jeux.

FIG. 1 – Le quatrième programme de démo du SDK – l'oiseau est controlable en utilisant la manette ou le clavier.

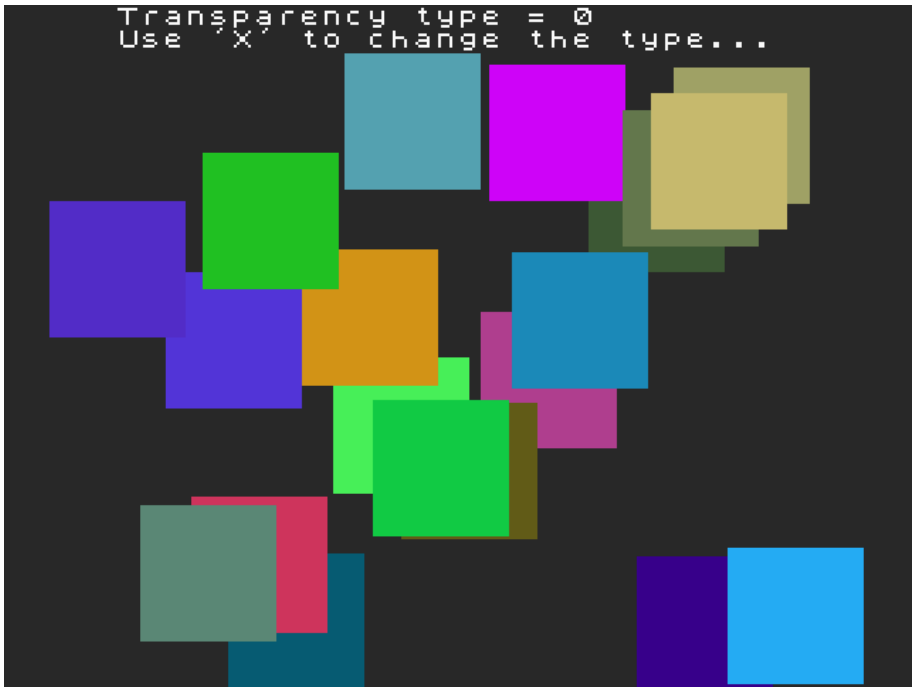

FIG. 2 – Le huitième programme de démo du SDK – la transparence peut être modifiée via un appui sur le bouton X.

2 Fonctionnement de la PlayStation

Comme le système sur lequel nous nous attardons pour notre projet est la très célèbre console de jeu de Sony, la PlayStation, il serait bon de faire un résumé rapide de son fonctionnement et de son architecture interne pour rendre la suite du rapport plus compréhensible et plus abordable. Également, cette présentation permet d'expliquer plusieurs de nos choix techniques qui découlent directement d'un choix fait par les ingénieurs de Sony dans l'architecture de la PlayStation.

Nous verrons tout d'abord les différents composants constituant la PlayStation avant de nous attarder sur leurs intéractions et leur communication qui se fait par deux canaux différents : via des registres mappés en mémoire et via un canal DMA.

2.1 Différents composants

Une PlayStation n'est pas uniquement constituée d'un processeur et d'une carte graphique : c'est l'ensemble des composants gravitant autour de cet ensemble qui ont permi à cette console de rivaliser avec les autres de sa génération, et qui par conséquent ont amélioré la qualité des jeux vidéos sur cette console. Les composants sont les suivants, triés par ordre d'importance dans le fonctionnement de la console :

- Le CPU : de type MIPS R3000A, c'est la pièce principale du puzzle qu'est l'architecture interne de la PlayStation. Cadencé à 33MHz, il exécute toutes les instructions régissant la logique d'un programme, et permet également au jeu de communiquer avec le reste des composants de la console. Cependant, sa faible puissance ne lui permet pas de réaliser des opérations lourdes telles que de la décompression d'images, de sons, ou de vidéos ;
- La RAM : de capacité assez faible (2 Mio), c'est l'emplacement où toutes les données qui ont vocation à être utilisées immédiatement sont stockées. Ces données incluent d'ailleurs les instructions que le CPU exécute en allant les chercher dans la RAM à une certaine adresse qui varie en fonction de la logique du programme. Sa faible capacité ne lui permet pas de stocker énormément de données, d'où la nécessité d'un lecteur CD-ROM rapide permettant d'accéder aux données plus lourdes rapidement ;
- Le GPU : son but étant de travailler sur des pixels, c'est une unité de calcul peu puissante mais fortement parallélisée : plusieurs traitements auront lieu en même temps, ce qui le rend très adapté au traitement d'images. Il communique avec le CPU via deux registres mappés en mémoire et deux canaux DMA (DMA2 et DMA6) ;

- La ROM et le BIOS : ces deux éléments sont rassemblés ici étant donné que la ROM, de capacité 512 Kio, est le conteneur du BIOS, qui est un ensemble de bibliothèques de fonctions visant à simplifier la programmation sur PlayStation. Ces fonctions font abstraction au-dessus des registres mappés en mémoire et permettent ainsi une grande aisance pour la plupart des opérations ;
- Le SPU : responsable du son et de la musique dans les jeux de PlayStation, c'est une puce dopée pour les traitements de signaux nécessaires à son travail, Il communique avec le reste du système via plusieurs registres mappés et un canal DMA ;
- Le lecteur de CD-ROM : une des grandes nouveautés de la PlayStation à son époque et son plus gros atout pour les jeux vidéos, capables de contenir des univers immenses et des cinématiques de qualité superbe dans un jeu grâce à sa forte capacité pour l'époque (350x la taille de la RAM !). Il communique avec les programmes via un canal DMA permettant de stocker les secteurs lus depuis le CD ;
- Le GTE : son acronyme signifiant *Graphics Transformation Engine*, il s'agit d'une puce spécialisée dans les calculs vectoriels et matriciels dont abondent les moteurs 3D. Capable de calculer des choses allant du produit de matrice à l'éclairage d'une surface via son vecteur normal, cette puce sert de soutien au CPU qui ne pourrait pas soutenir tous ces calculs à cause de sa généricité et de sa faible cadence ;
- Le DDE : responsable des cinématiques que vous voyez parfois dans les jeux de PlayStation, le *Data Decompression Engine* décompresse les données d'une vidéo MPEG reçue dans un canal DMA vers un autre canal DMA. Il réalise ainsi une tâche irréalisable par le CPU du fait de sa faible vitesse, et peut décoder un flux vidéo et un flux audio simultanément.

Tous ces composants forment l'ensemble que l'on appelle communément une PlayStation. Même si tous ne sont pas nécessaires pour l'exécution de programmes de démonstration comme notre programme en est capable actuellement, ils sont néanmoins utilisés dans leur intégralité par la plupart des jeux commerciaux tournant sur la console de Sony.

2.2 Communication inter-composants via registres

Une des possibilités pour communiquer entre composants de la PlayStation que j'ai évoqué plus haut est celle des registres mappés en mémoire. Ce qu'on appelle un registre mappé est une petite mémoire d'un composant, de taille allant de 8 à 32 bits, qui est accessible en lisant ou en écrivant à une adresse mémoire spécifique. C'est une méthode assez pratique pour rendre accessible des zones mémoire de différents périphériques aux autres : en effet, pour un programme PlayStation, la lecture ou l'écriture à une adresse est l'affaire d'une ou deux instructions.

Ces registres mappés sont notamment utilisés pour indiquer l'état d'un péripérhique : par exemple, le GPU les utilise pour communiquer si le rendu d'une frame est en cours, et donc si le programme peut lui envoyer de nouvelles données. Ils sont également utilisés pour signaler le type et le début d'un accès DMA : trois registres mappés en mémoire sont alloués par DMA (CHCR, MPR et BCR) qui indiquent le type, l'adresse et la taille du transfert DMA.

2.3 Communication inter-composants via DMA

DMA est un acronyme qui signifie *Direct Memory Access*, une méthode de communication entre composants qui passe par des transferts directs de zones de mémoires de plusieurs kilo-octets (voire méga-octets) plutôt qu'un transfert des données une par une via un registre mappé.

Il existe 6 canaux DMA utilisés par la PlayStation, qui sont alloués pour divers composants tels que le GPU, le SPU ou encore le lecteur de CD-ROM. En effet, tous ces composants ont besoin de transfert rapide de blocs de données pour des entités comme des textures, des effets sonores ou des secteurs venant d'être lus depuis un disque. Chacun de ces canaux DMA peut être utilisé dans différents modes, que ce soit de la RAM à un composant, d'un composant vers la RAM ou également pour qu'un composant accède directement à une adresse dans la RAM sans copie de données.

3 Traducteur

Le traducteur de binaires est la partie la plus importante et la plus algorithmiquement complexe de notre projet. Cependant, c'est également la partie que nous avons (relativement à sa complexité) le moins eu la chance de travailler : en effet, tester des algorithmes complexes sans pouvoir tester la qualité du résultat est une pratique qui nous semble plus que douteuse. Nous nous sommes donc concentrés sur le minimum pour faire fonctionner des programmes d'exemple.

De ce fait, l'algorithme que nous avons utilisé pour cette soutenance est plus que basique : on pourrait même le qualifier de naif. Cependant, c'est un défaut qui ne saurait rester dans les futures versions de notre projet.

3.1 Principe de fonctionnement

La description de l'algorithme utilisée est assez simple étant donné sa naiveté par rapport à la complexité du problème : en effet, le principe est que la seule phase réalisée par le traducteur est le décodage des instructions et des headers du fichier exécutable PlayStation (qui contient, par exemple, les adresses des sections et leur taille).

Le traducteur itère sur le code du programme et traduit chaque entier représentant une instruction par un couple contenant la fonction exécutant le code de l'instruction et ses paramètres. Par exemple, l'instruction suivante :

```
addiu v0, r0, 42
```

Sera traduite par le traducteur en ce couple :

```
(psx_instr_addiu, (V0, R0, 42, 0))
```

Un tableau est ensuite généré contenant l'ensemble des fonctions et des paramètres des instructions. Un fichier objet contenant ces tableaux est généré en utilisant la bibliothèque LLVM, et est linké avec la bibliothèque de support qui exécutera le code contenu dans les tableaux. Ainsi, tout ce qui est généré par le traducteur est un ensemble de données statiques et pas encore du code.

3.2 LLVM

LLVM est la bibliothèque que nous utilisons pour générer du code exécutable à partir d'une API objet et haut niveau. À la base de nombreux projets

de compilateurs, soutenue par des sociétés comme AMD ou Apple qui l'utilisent pour générer du code pour les cartes graphiques via l'API OpenCL, c'est une bibliothèque éprouvée et qui suffit largement pour les usages que nous comptons en faire.

Cette bibliothèque, écrite à la base en C++, a été portée en OCaml par les développeurs du projet, et est donc utilisable directement depuis notre traducteur. Même si elle n'est actuellement utilisée que pour générer des valeurs constantes à lier ensuite avec des bibliothèques dynamiques, elle permet de générer tout type de code, qui peut ensuite être traduit en bytecode portable, en code natif pour l'architecture actuelle ou même être exécuté directement via un Just In Time compiler (JIT).

Nous avons porté notre choix sur cette bibliothèque plutôt qu'une autre car aucune ne propose une telle couverture au niveau des architectures cibles et autant de possibilités au niveau de la sortie. En plus de cela, LLVM est capable d'optimiser énormément le code en sortie, et s'approche petit à petit du niveau de compilateurs matures tels que GCC ou ICC.

FIG. 3 – Le logo de LLVM, donné gracieusement par Apple à la communauté.

3.3 Possibilités d'amélioration

L'algorithme actuellement implémenté étant loin d'être intelligent, cela nous laisse le champ clair pour de nombreuses possibilités d'améliorations pour le fonctionnement du traducteur. En particulier, un des domaines actuellement envisagés est l'analyse du graphe représentant le flot d'exécution afin de détecter les zones de code jamais appelées. Ce graphe permetterait également de pointer vers les adresses de branchement afin de plus tard permettre de traduire ce code efficacement (un label par adresse de branchement, par exemple).

Une première implémentation de cet algorithme est déjà réalisée et permet de générer des graphes au format Graphviz.

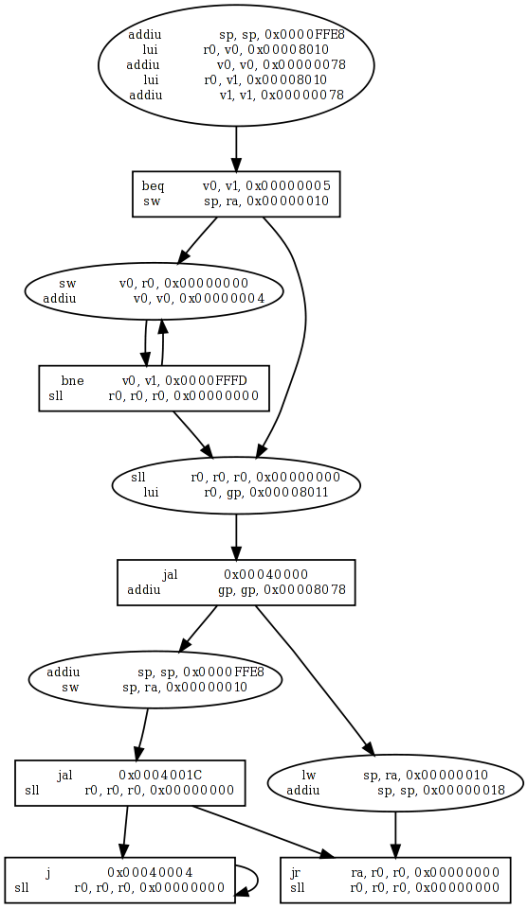

FIG. 4 – Le graphe du code d'un programme simple – les programmes plus complexes génèrent généralement des graphes trop gros pour l'affichage.

4 Bibliothèque de support

L'architecture d'une PlayStation et d'une machine comme celles que nous utilisons tous les jours sont suffisamment différentes pour qu'une traduction mot à mot, ou instruction par instruction, ne soit pas toujours adaptée ou réalisable. Pour résoudre ce problème, une bibliothèque de support est utilisée par les programmes traduits par notre projet afin de traduire, par exemple, les accès à la RAM de la PlayStation en accès à la RAM du PC.

Cette bibliothèque de support est nommée `libpsx.so` et contient entres autres les fonctions implémentant les instructions de micro-processeur de la PlayStation ainsi que la recherche d'une instruction à partir de son adresse mémoire.

4.1 Instructions du CPU

Comme la version actuelle de notre traducteur ne traduit pas directement les instructions MIPS en instructions x86 ou code natif mais passe plutôt par des fonctions émulant le fonctionnement des instructions, l'implémentation de ces fonctions instructions est nécessaire.

Ces fonctions, généralement d'une ou deux lignes et manipulent les registres généraux du processeur [4] pour renvoyer leur résultat. Cependant, malgré leur petite taille, elles sont en très grand nombre : environ 60 instructions différentes déjà implémentées, et une vingtaine encore non implémentées.

Toute cette partie de la bibliothèque de support devrait cependant petit à petit disparaitre au fur et à mesure de l'évolution des algorithmes du traducteur, qui permettront par la suite d'éviter des appels inutiles de fonction quand une simple addition est nécessaire.

4.2 Accès à la mémoire

Le modèle mémoire de la PlayStation est fondamentalement différent de la vue de la mémoire que nous propose la majorité des systèmes d'exploitation. En effet, l'ensemble de l'espace mémoire adressable, soit 2^32 octets, est réparti entre différentes zones : la RAM, la ROM, les zones DMA, les registres mappés, etc.

Pour permettre un accès aisé à la mémoire depuis le BIOS et les instructions implémentées dans la bibliothèque de support, un module gère les accès en lecture et écriture vers la mémoire pour les rediriger si besoin est vers

[4] Ou GPR, *General Purpose Registers*.

une autre zone (RAM, ROM, registres, etc.). Sans avoir la simplicité d'un simple pointeur, ces fonctions d'accès permettent d'éviter de recoder la roue à chaque accès mémoire.

4.3 Position du code

Un des derniers modules importants de cette bibliothèque de support est ce qui est appelé dans notre code la « codegate ». Ce nom n'est pas choisi au hasard : ce module gère la récupération d'une instruction à une adresse donnée, et ce que cette adresse soit dans la RAM ou dans le BIOS.

En effet, un appel à une fonction du BIOS consiste simplement à un saut vers une adresse spéciale (0xA0, 0xB0 ou 0xC0) avec un registre positionné à une valeur représentant le numéro d'appel système. Il est donc nécessaire de gérer ce cas en renvoyant l'exécution vers une instruction spéciale dans le cas d'un appel au BIOS. Dans les autres cas, la codegate renvoie simplement l'instruction à exécuter.

5 Reprogrammation du BIOS

Le BIOS, acronyme signifiant *Basic Input/Output System*, est un programme stocké dans la ROM de la PlayStation qui permet de gèrer facilement l'accès aux différents périphériques tels que les manettes de jeu ou les cartes mémoire. Étant un programme comme un autre, il est en théorie lui aussi possible de le traduire directement via notre projet, mais il est cependant plus avantageux de le réecrire sous forme d'une bibliothèque dynamique, et ce pour plusieurs raisons :

- C'est un programme de taille conséquente : 512Kio constitués uniquement de code qui implémente une des 300 fonctions exportées par le BIOS de la PlayStation. C'est également un programme complexe qui tire atout de toutes les possibilités de la console, ce qui le rend non trivial à traduire ;
- Tous les chemins mènent à lui, ce qui incite à le programmer pour obtenir les meilleurs performances possibles. En effet, des opérations comme la lecture de l'état de la manette sont faites 60 fois par seconde par un jeu de PlayStation ;
- Enfin, c'est un programme qui appartient à Sony. Par conséquent, le redistribuer sous une forme certes modifiée serait une violation de copyright, ce qui est bien entendu un problème majeur pour un projet tel que le notre.

Pour ces diverses raisons, nous avons entamé un recodage des fonctions du BIOS dont nous avons besoin pour nos programmes de démonstration, qui sont disponibles dans la bibliothèque dynamique `libbios.so`. Allant des fonctions de manipulation de chaines de caractères à un allocateur de mémoire, on y trouve de tout et c'est un travail de grande envergure.

5.1 Rôle du BIOS

Comme précisé en introduction, le BIOS a un rôle plus que majeur dans la programmation sur PlayStation, étant donné qu'il exporte la plupart des fonctions de la bibliothèque standard du C (`strlen`, `memcpy`, `printf`, etc.) en plus des fonctions caractéristiques de la PlayStation.

En plus de ce rôle de bibliothèque de fonctions, le BIOS a également la charge de gèrer les interruptions liées au manettes, qui régissent le rythme des programmes sur PlayStation : en effet, pour attendre l'image suivante parmi les 60 à faire en une seconde, un jeu de PlayStation va attendre l'arrivée de données venant des manettes. C'est le BIOS qui se charge de copier ces données là où il le faut et quand il le faut, ce qui le rend donc chef d'orchestre qui tente de conserver la synchronisation.

5.2 Enjeu de la reprogrammation

La reprogrammation du BIOS en tant que bibliothèque dynamique a un enjeu important pour notre projet : ce n'est pas pour rien que de nombreux projets d'émulateurs incluent leurs propre BIOS reprogrammés en C avec leur émulateur [5].

Cet enjeu se mesure premièrement par la réduction de la taille de l'exécutable généré : si pour chaque jeu le BIOS devait être inclu et traduit, cela ajouterait au moins un méga-octet de code pour chaque jeu traduit par Métamorphosis.

En plus de cette consommation de place inutile, le BIOS étant un programme peu trivial, il est difficile d'optimiser chaque fonction traduite du BIOS : ainsi on a une perte de vitesse de l'ordre d'un facteur 20 pour chaque appel de fonction (qui sont, comme déjà dit précédemment, appellées plusieurs centaines de fois par seconde).

Enfin, le problème de la licence du BIOS et de sa redistribution est réglé par un recodage complet du BIOS de la PlayStation : n'étant pas considéré comme une oeuvre dérivée, il est possible de redistribuer une reprogrammation du BIOS sous n'importe quelle licence.

5.3 État actuel

Notre réimplémentation du BIOS est actuellement dans un état assez peu avancé, ne réimplémentant qu'une petite poignée de l'intégralité des fonctions exportées par cette immense bibliothèque. En effet, nous avons choisi de petit à petit déterminer les fonctions à programmer en observant les appels systèmes utilisés par les jeux que nous traduisons.

La liste des appels que nous implémentons actuellement est la suivante :

- `printf` – implémentée pour aider au debug ;
- `pad_init, pad_stop` – implémentés pour gérer les manettes ;
- `strlen` – implémentée en tant qu'exemple d'implémentation

L'appel des fonctions du BIOS se fait communément via un saut aux adresses mémoire spéciales `0xA0, 0xB0` et `0xC0`. Ces appels sont interceptés par le dispatcheur de code, qui redirige le flux d'exécution vers une pseudo-instruction `psx_instr_bios_call`, qui se charge de regarder l'appel système demandé (passé via un registre du processeur) et d'appeller la fonction adéquate avant de revenir au flux d'exécution normal.

[5] Par exemple, PCSX.

En interne, les appels du BIOS sont stockés dans une table, où les appels non implémentés sont représentés par un pointeur nul et les autres via un pointeur vers leur code. L'appel d'une fonction du BIOS déclenche une recherche dans cette table, et un appel d'une fonction non réimplémentée stoppe le programme avec un message d'erreur explicite nous indiquant la fonction manquante.

Ceci, par exemple, est l'implémentation de la fonction `strlen` pour notre BIOS :

```
void psx_bios_strlen(uint32_t* GPR)
{
    uint32_t n = 0;
    uint32_t string_addr = GPR[A0];

    while (psx_mem_read_8(string_addr) != '\0')
    {
        n += 1;
        string_addr += 1;
    }

    GPR[V0] = n;
}
```

Conclusion

C'est ainsi sur ces lignes que s'achève ce rapport de première soutenance assez chargé en contenu et qui présente assez sommairement le travail qui a été réalisé pendant les deux derniers mois.

Cette première période fut surtout l'occasion de poser les bases qui nous permetteront de mieux expérimenter au niveau des algorithmes de traduction de code par la suite, et de peut-être revenir avec un algorithme performant, efficace et original pour notre deuxième soutenance.