Groupe 9
Compte rendu du projet

Phase 1

Étape 1

On a fait une structure pour manipuler les fichiers elf.

```
typedef struct ELF_STRUCT {

       FILE* elf_file;
       Elf32_Ehdr *elf_header;        // L'entête
       Elf32_Shdr *a_shdr;            // Les sections
       Elf32_Sym *a_sym;              // La table des symboles
       Elf32_Rel *a_rel;              // Réels de la table de réimplantation
       Elf32_Rela *a_rela;            // Relatifs de la table de réimplantation
       char **sections_content;       // Le contenu des sections
       int * tab_reimplant;           // Table de réimplantation
       int taille_rel;                // La taille des réels
       int taille_rela;               // La table des relatifs
       int str_table_idx;             // L'indice de la table de string
       int error_code;                // Identifier l'erreur dans la structure

} ELF_STRUCT;
```

Notre fonction display_header du fichier header_elf.c affiche l'entête d'un fichier de format elf. Cette fonction prend en paramètre un fichier de type Elf32_Ehdr. En premier lieu on s'assure que le fichier est du type elf 32 bits en comparant ses nombres magiques avec les constantes ELFMAG0, ELFMAG1, ELFMAG2, ELFMAG3 et sa classe avec ELFCLASS32. Si ces derniers sont corrects, on affiche la classe, la version de l'entête, le système d'exploitation, le type..etc. Et pour chacune des propriétés on a crée une fonction pour afficher correctement le nom par exemple pour la propriété e_version:

```
char* get_object_file_version(Elf32_Word e_version) {
    switch (e_version) {

        case EV_NONE:       return "0 (invalid version)";
        case EV_CURRENT:    return "1 (current version)";
        default:            return "Unknown";

 }
```

On a traité le cas où le fichier est en little endian, dans ce cas on le reverse en big endian.

Étape 2

Notre fonction header_section dans le fichier section_header.c parcourt toutes les sections et pour chacune la met dans le tableau elf->a_shdr (de dimension taille de l'entête de section * nombre de sections) - Le nombre des sections nous est donné dans la variable e_shnum. Et pareil que l'étape 1 on reverse en big endian si nécessaire et on traîne les différents cas de type (dans la fonction case_type) et les flags (dans la fonction case_flags).
//Section_header.c

Étape 3

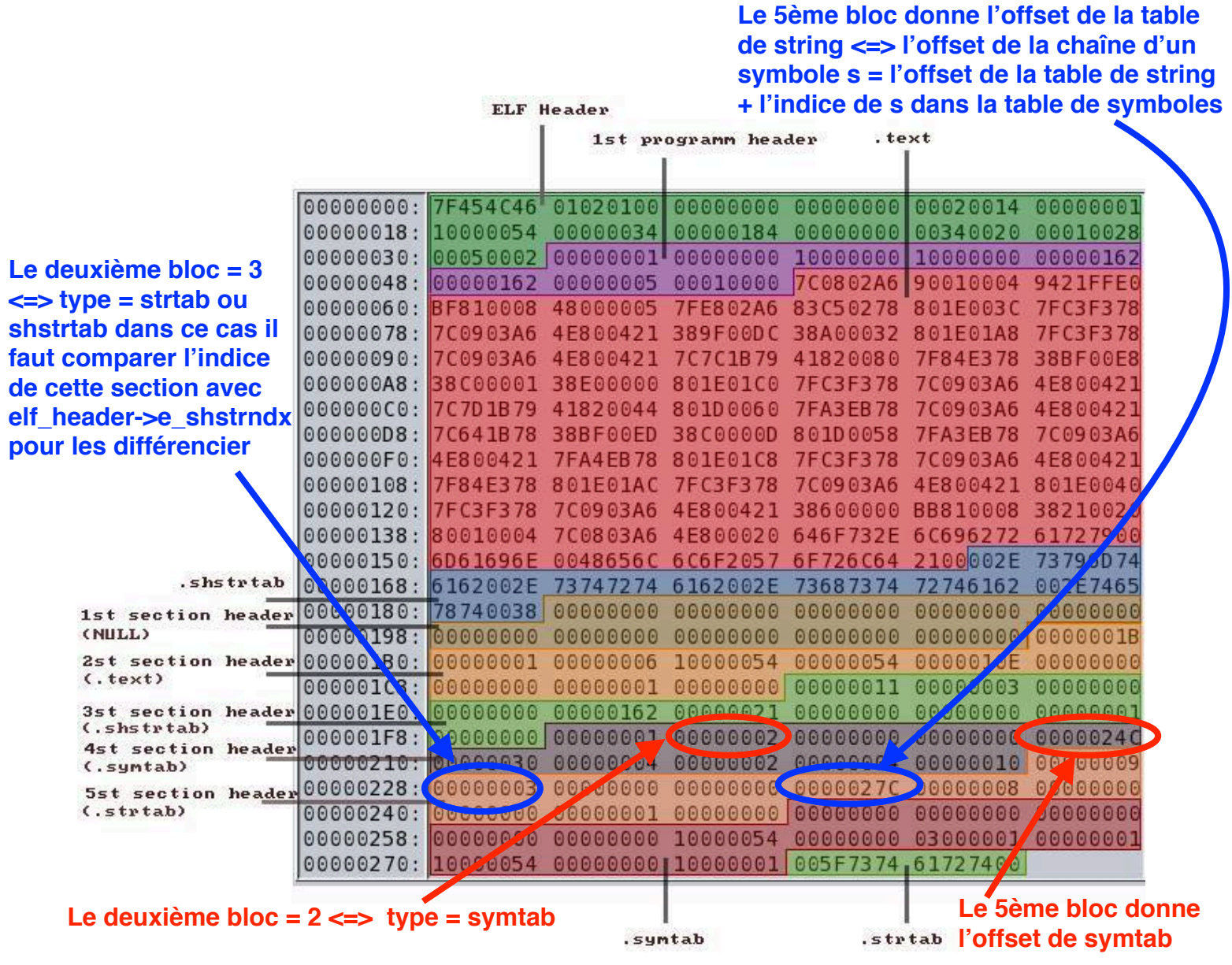
Dans cette étape on s'intéresse à la structure Elf32_Shdr. Dans le fichier section_elf.c, la fonction choix_section demande à l'utilisateur le numéro de la section et puis appelle la fonction display_section si le numéro de section existe et est valide. Display_section, qui prend en paramètre un pointeur vers un fichier elf et l'indice de la section, affiche la section choisie. La procédure read_section prend en paramètre un pointeur vers un fichier de type ELF_STRUCT, l'indice de la section pour afficher et un pointeur vers un tableau où on la section sera stockée par

la procédure fread. Dans cette fonction on utilise les procédures fseek (pour déplacer le curseur sur l'offset de section.

Étape 4

Ici, our accéder à la table de symboles on s'intéresse à la structure Elf32_Sym. Dans le fichier table_symbole.c les fonctions afficher_type, afficher_ndx, afficher_vis et afficher_bind traitent les différents cas de chaque propriété. Par exemple, afficher_bind prend en paramètre st_info et le décale de 4 bits à gauche puis selon ce décalage on affiche le bind qui correspond. La fonction creer_table_symbole prend en paramètre un pointeur vers un fichier de type ELF_STRUCT. Dans cette fonction on trouve le numéro de l'entête de section de la table de symbole en parcourant toutes les sections et en incrémentant un compteur jusqu'à ce que l'on trouve l'indice de celle dont le type = 2. Ensuite on récupère l'offset et la taille de la table de symbole dans sh_offset et sh_size de la section dont l'indice est le compteur. On utilise fseek pour déplacer le curseur à l'offset de la table de symbole pour ensuite lire les symboles un par un. On reverse en big endian si il le faut.

Quant à l'affichage de la table de symbole, on cherche la section dont le type est 3. Mais si l'indice est égal à elf_header->e_shstrndx (qui correspond à l'indice de shstrtab) on le saute. Si on a trouvé la bonne section, on récupère le décalage/offset pour se rendre à la table des strings puis on cherche la section symtab et on récupère la taille de la table des symboles (Le nombre de symboles = la taille de la table des symboles / la taille de la structures d'un seul symbole). On parcours la structure de chaque symbole si st_name n'est pas nul on calcule le décalage à effectuer jusqu'au premier caractère du symbole = décalage jusqu'à la table des strings + indice du symbole au sein de la table des strings. On met le curseur sur cet offset pour lire la chaîne de caractères du symbole (on s'arrête quand on arrive à \0).



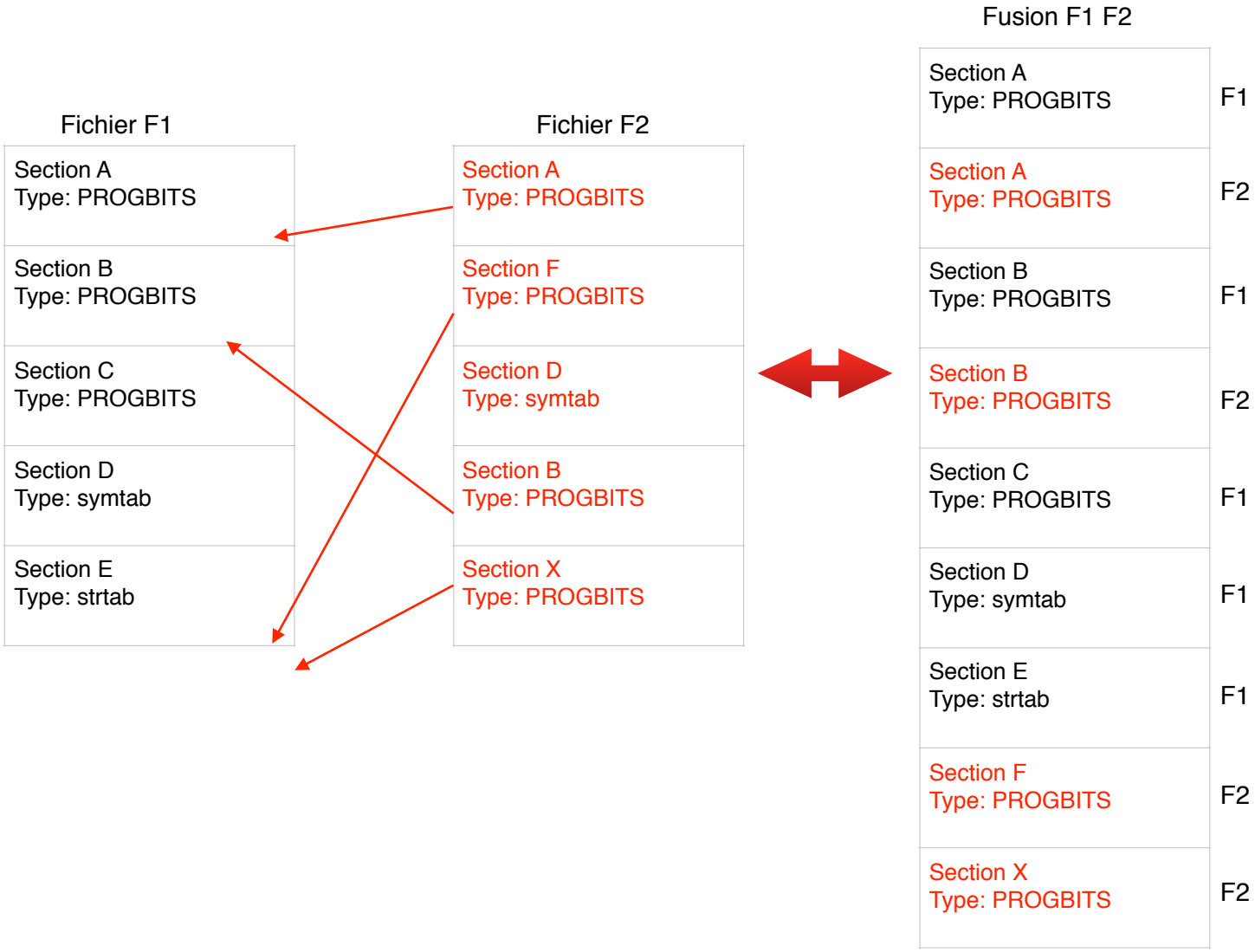
Étape 5

Dans le fichier réimplentation.c on commence par remplir un tableau de char par les types possibles de Rel. On fait une première boucle parcourant les sections pour calculer le nombre total d'éléments de Rel de toutes les sections (pour chaque section on divise la taille de la section par la taille de Rel: Elf32_Rel) puis une deuxième boucle qui parcourt de nouveau toutes les séquences de type Rel pour remplir le tableau a_rel dont la taille est fournie par la première boucle.

Phase 2

Étape 6

Pour fusionner deux fichiers de type ELF_STRUCT on ajoute les sections du type PROGBITS du deuxième fichier aux sections du premier. Pour savoir où ajouter les sections on regarde le nom de la section. Si le nom de le section existe dans le premier fichier on l'ajoute à la suite de cette dernière sinon on l'ajoute à la fin des sections.



Dans un premier temps, on parcours toutes les sections du premier fichier, et si une est du type PROGBITS, on parcours toutes les sections du deuxième:
Dans le cas où on trouve une section du deuxième fichier de type PROGBITS et de même nom, on alloue une place dans la mémoire de la somme des tailles des deux sections et on appelle la procédure seccat pour concatener la section du deuxième fichier à celle de la première dans

l'espace alloué. On décale ensuite les offsets des sections dont l'indice est supérieure à celui de la section qui a été le sujet de la concaténation par la procédure maj_offset. Cette dernière parcours toutes les sections du fichier donné en paramètre en comparant leur indice avec l'indice donné en paramètre et si il est supérieur décale son offset du size donné en paramètre. Ce size est évidemment le size de la section du deuxième fichier qu'on vient d'ajouter.
Ensuite on parcours toutes les séquences du deuxième fichier, si elle est de type PROGBITS mais dont le nom n'est pas identique à aucun des noms des sections du premier fichier on appelle la procédure ajout_section qui:
- Incrémente le nombre de sections dans le header du premier fichier
- Fait une réallocation de l'espace alloué aux sections headers
- Copie la nouvelle section header
- Modifie le nom et l'offset de la nouvelle section header
- Modification du size de shstrtab
- Ajoute le nom dans shstrtab (avec la procédure ajout_nom_section, défini dans util.c)
- Ajoute le contenu de la section à la fin (l'indice de la dernière section est donné par la fonction max_offset_section.

Étape 7

Étape 8

La première partie de l'étape 8 est proche de l'étape 6. Ici, on veut fusionner les sections dont le type est SH_REL de deux fichiers. On a crée un tableau de booléens missing qui met à TRUE l'indice des sections à traiter et qui est initialisé comme ceci:
Les indices des sections dont le type est SH_REL sont à TRUE et les autres sont à FALSE.
On parcours ensuite toutes les sections du deuxième fichier de type ELF_STRUCT donné en paramètre si ils sont à TRUE et il existe une section du premier fichier de type ELF_STRUCT donné en paramètre qui a le même nom on la concatène à la suite de cette dernière et on met à jour les offsets, on les met ensuite à FALSE dans missing. Si la section du fichier 2 est à TRUE mais il n'existe aucune section du fichier 1 ayant le même nom on l'ajoute à la fin et on modifie l'offset de la fonction ajoutée et on la met à FALSE dans missing. A chaque fois qu'on ajoute une section, on modifie le nombre de section dans l'offset de la première entête de section, on modifie le size de sh_strtab (+1 pour le \0) et on ajoute le nom de la section à la fin de .shstrtab. Pour récupérer le nom d'une section donnée on utilise la fonction get_name définie dans util.c:

```
char* get_name(ELF_STRUCT * elf,int numero){
	int offset = elf->a_shdr[numero].sh_name;
	char * shstr = elf->sections_content[elf->elf_header->e_shstrndx];
	char * str;
	int taille_mot = 1;
	int j = 0;
	int i = offset;
	while (shstr[i] != '\0') {
		taille_mot++;
		i++;
	}
	taille_mot++;
	str = malloc(sizeof(char)*taille_mot);
	for (i = offset; i < taille_mot+offset-1; i++) {
		str[j] = shstr[i];
		j++;
	}
	str[taille_mot] = '\0';
	return str;
}
```