Compte Rendu - SAÉ RT1SA05

LUKAS CAKIC, WATALAKU JONATHAN

2025-11-30

Table des matières

1. Analyse et choix des structures de données 2
1.1 Structure principale : `Data` 2
1.2 Tableaux de constantes 2
2. Algorithmes 3
2.1 Lecture et parsing du fichier log : `func_opred()` 3
2.2 Extraction de la date : `extract_date()` 4
2.3 Affichage texte : `versiontexte()` 5
2.4 Affichage graphique : `versiongraphique()` 5
2.5 Affichage web (CGI) : `versionweb()` 7
2.6 Chargement du style CSS : `lecture_style()` 8
2.7 Point d'entrée : `main()` 8
3. Code source 9
3.1 Fichier : versiondynamique.c 9
3.2 Fichier : Makefile 15
3.3 Fichier : aecrire.html 16
4. Tests et résultats 18
4.1 Test du mode texte 18
4.2 Test du mode graphique 19
4.3 Test du mode web (CGI) 20
5. Conclusion 20

1. Analyse et choix des structures de données

1.1 Structure principale : `Data`

Nous avons choisi d'utiliser une structure pour centraliser toutes les données :

```c
typedef struct {
    int n_rq;              // Nombre total de requêtes
    int monthly[12];       // Distribution mensuelle (index 0 = Janvier)
    int first_day;         // Jour de la première connexion
    int first_month;       // Mois de la première connexion
    int first_year;        // Année de la première connexion
    int has_first_date;    // Flag : première date trouvée
} Data;
```

Justification des choix :

1. Le choix d'un tableau simple `int monthly[12]` s'explique par le fait que l'index du tableau (compris entre 0 et 11) représente naturellement les mois de l'année. Il n'est donc pas nécessaire de stocker le numéro du mois en plus du compteur, par exemple `monthly[3]` pour le mois d'avril.
2. Le passage des données par pointeur, répond à de bonnes pratiques. Cela évite les effets de bord liés aux globales et rend les fonctions réentrantes. Le fait de passer explicitement `&data` aux fonctions rend clair quelles parties du code modifient réellement les données, notamment avec une initialisation explicite dans `main()` via `Data data = {0};`.
3. L'utilisation du flag `has_first_date` permet d'identifier précisément la première ligne du fichier de log. Les informations de date sont donc enregistrées qu'une seule fois, ce qui évite de les écraser inutilement à chaque nouvelle ligne.

1.2 Tableaux de constantes

```c
const char *months_abbrev[] = {"Jan", "Feb", "Mar", ...};
const char *months_full[] = {"Janvier", "Fevrier", "Mars", ...};
```

Justification : Centralisation des noms de mois (évite les répétitions), `months_abbrev` : pour parser les logs (format anglais), `months_full` : pour l'affichage en français,

---

2. Algorithmes

2.1 Lecture et parsing du fichier log : `func_opred()`

Principe : Le programme ouvre le fichier `/var/log/apache2/access.log`, lit son contenu caractère par caractère avec un buffer dynamique, extrait la date de chaque ligne complète à l'aide de `extract_date()` pour incrémenter le compteur total si la date est valide, puis ferme le fichier et libère la mémoire.

Utilisation d'un buffer dynamique

L'utilisation d'un buffer dynamique est justifiée par le fait que les lignes de log peuvent avoir des tailles énormes. Le recours à `realloc()` permet d'agrandir le buffer uniquement lorsque cela est nécessaire, évitant ainsi une allocation excessive dès le départ. Il est essentiel d'utiliser un pointeur intermédiaire comme `new_buffer`, car si `realloc()` échoue et retourne `NULL`, le pointeur original `buffer` reste valide et peut être libéré correctement, ce qui évite toute fuite mémoire. À l'inverse, sans ce pointeur intermédiaire, la référence à la mémoire initialement allouée serait perdue.

Pseudo-code :

```
FONCTION func_opred(data : pointeur vers Data)
    OUVRIR fichier LOG en lecture
    SI échec ALORS
        Afficher erreur
        RETOURNER
    FIN SI

    Allouer buffer de taille 2048
    position ← 0

    TANT QUE caractère disponible FAIRE
        SI buffer plein ALORS
            Doubler la taille du buffer avec realloc
            SI échec ALORS
                Libérer mémoire
                RETOURNER
            FIN SI
        FIN SI

        Lire caractère → buffer[position]
        position ← position + 1

        SI caractère = '\n' ALORS
            buffer[position-1] ← '\0'
            SI extract_date(buffer, data) = succès ALORS
                data->n_rq ← data->n_rq + 1
            FIN SI
            position ← 0
```

```
        FIN SI
    FIN TANT QUE

    Libérer buffer
    Fermer fichier
FIN FONCTION
```

2.2 Extraction de la date : `extract_date()`

Principe :

Le traitement consiste d'abord à rechercher le caractère `[` afin d'identifier le début de la date. Une fois ce repère trouvé, la date est extraite et analysée au format `JJ/Mmm/AAAA` à l'aide de `sscanf()`. Le mois, exprimé sous forme abrégée (par exemple « Mar »), est ensuite converti en un index compris entre 0 et 11 afin de pouvoir être utilisé directement dans le tableau de comptage. La première date rencontrée est alors mémorisée pour servir de référence, puis, à chaque ligne analysée, le compteur correspondant au mois extrait est incrémenté.

Pseudo-code :

```
FONCTION extract_date(ligne, data : pointeur vers Data) : entier
    Chercher '[' dans ligne
    SI non trouvé ALORS RETOURNER échec

    Parser "JJ/Mmm/AAAA" avec sscanf
    SI échec ALORS RETOURNER échec

    Convertir jour et année en entiers

    mois ← -1
    POUR m de 0 à 11 FAIRE
        SI mois_texte = months_abbrev[m] ALORS
            mois ← m
            SORTIR de la boucle
        FIN SI
    FIN POUR

    SI mois = -1 ALORS RETOURNER échec

    SI data->has_first_date = faux ALORS
        data->first_day ← jour
        data->first_month ← mois
        data->first_year ← année
        data->has_first_date ← vrai
    FIN SI

    data->monthly[mois] ← data->monthly[mois] + 1
    RETOURNER succès
```

```
29 FIN FONCTION
```

2.3 Affichage texte : `versiontexte()`

Principe :

Le traitement commence par la vérification de la présence de données afin d'éviter tout affichage inutile ou d'erreur. Une fois cette condition remplie, un message est affiché, incluant la date de la première connexion. Enfin, le programme parcourt chaque mois, calcule le pourcentage correspondant par rapport au total, puis l'affiche.

Pseudo-code :

```
FONCTION versiontexte(data : pointeur vers Data)
    SI data->n_rq = 0 ALORS
        Afficher "Aucune donnée"
        RETOURNER
    FIN SI

    Afficher "Depuis le JJ/Mmm/AAAA on a enregistre N connexions"

    POUR m de 0 à 11 FAIRE
        count ← data->monthly[m]
        pourcentage ← (count / data->n_rq) × 100
        Afficher "Mois : XX.XX%"
    FIN POUR
FIN FONCTION
```

Format de sortie :

```
Depuis le 21/Feb/2016 on a enregistre 186181 connexions.
   Janvier : 19.50%
   Fevrier :  2.30%
   ...
```

2.4 Affichage graphique : `versiongraphique()`

Principe :

Le programme commence par vérifier qu'il y a des données à afficher. Il ouvre ensuite une fenêtre SDL de 430×350 pixels, qui servira de support à l'affichage graphique. Un message est affiché afin de contextualiser les données présentées. Les labels des mois sont placés en bas. Pour chaque mois, le pourcentage correspondant est calculé, puis converti en hauteur de barre en multipliant ce pourcentage par 10. La barre orange représente le mois, et le label n'est affiché que si la hauteur de la barre

atteint au moins 25 pixels afin de garantir une bonne lisibilité. Enfin, l'affichage est actualisé et le programme attend une interaction de l'utilisateur, sous la forme d'un clic, avant de se terminer.

Pseudo-code :

```
FONCTION versiongraphique(data : pointeur vers Data)
    SI data->n_rq = 0 ALORS
        Afficher "Aucune donnée"
        RETOURNER
    FIN SI

    Ouvrir fenêtre 430×350
    Remplir fond en blanc

    Créer message "Depuis le JJ/Mmm/AAAA..."
    Afficher message en position (30, 50)

    Afficher "Jan Fev Mar ... Dec" en position (30, 300)

    POUR m de 0 à 11 FAIRE
        count ← data->monthly[m]
        pourcentage ← (count / data->n_rq) × 100
        hauteur ← pourcentage × 10
        x ← 30 + (m × 31)
        y ← 300 − hauteur

        Dessiner rectangle orange à (x, y) de taille 30×hauteur

        SI hauteur ≥ 25 ALORS
            Créer label avec pourcentage arrondi
            Afficher label en blanc à (x+8, y+8)
        FIN SI
    FIN POUR

    Actualiser affichage
    Attendre clic
    Fermer fenêtre
FIN FONCTION
```

Calculs importants :

- Hauteur = pourcentage × 10 (ex : 13.7% → 137px)
- Espacement horizontal = 31 pixels entre barres
- Position Y inversée : y = 300 - hauteur (SDL place (0,0) en haut à gauche)
- Label affiché si hauteur ≥ 25px (sinon trop petit)

2.5 Affichage web (CGI) : `versionweb()`

Principe :

Le traitement débute par l'envoi du header HTTP `Content-type: text/html`, indispensable pour indiquer au navigateur que la réponse est au format HTML. Le code HTML est ensuite généré dynamiquement à l'aide de `printf()`, ce qui permet d'intégrer directement les données calculées. Le style CSS est chargé depuis le fichier `aecrire.html` à l'aide de la fonction `lecture_style()` afin de séparer la présentation du contenu et de simplifier la maintenance. Le message d'en-tête est alors affiché pour contextualiser les résultats. La fonction vérifie ensuite la présence de données avant de générer les barres de l'histogramme en HTML/CSS, en fonction des valeurs calculées pour chaque mois.

CGI :

Le programme génère du HTML via `printf()`, Apache l'envoie au navigateur, le header `Content-type` précise le format, et les barres `<li>` sont positionnées en CSS.

Pseudo-code :

```
FONCTION versionweb(data : pointeur vers Data)
    Afficher "Content-type: text/html\n\n"
    Afficher "<!doctype html>..."

    Charger et afficher aecrire.html (via lecture_style())

    Afficher "<p>Depuis le JJ/Mmm/AAAA...</p>"

    SI data->n_rq = 0 ALORS
        Afficher "Aucune donnée"
        RETOURNER
    FIN SI

    Afficher "<div id='vertgraph'><ul>"

    POUR m de 0 à 11 FAIRE
        count ← data->monthly[m]
        pourcentage ← (count / data->n_rq) × 100
        hauteur ← pourcentage × 10
        x ← 10 + (m × 31)

        SI hauteur ≥ 25 ALORS
            Afficher "<li style='left:Xpx;height:Hpx'>P</li>"
        SINON
            Afficher "<li style='left:Xpx;height:Hpx'></li>"
        FIN SI
    FIN POUR

    Afficher "</ul></div></body></html>"
```

```
30 FIN FONCTION
```

2.6 Chargement du style CSS : `lecture_style()`

Principe :

Cette fonction utilitaire est appelée par `versionweb()` pour charger et afficher le contenu du fichier CSS. Elle ouvre le fichier `aecrire.html` en mode lecture, lit chaque ligne à l'aide de `fgets()` avec un buffer de 1024 caractères, et affiche directement chaque ligne sur la sortie standard via `printf()`. En cas d'erreur d'ouverture du fichier, un message d'erreur est affiché sur la sortie d'erreur standard. Enfin, le fichier est fermé proprement.

Pseudo-code :

```
FONCTION lecture_style()
    Ouvrir "aecrire.html" en lecture
    SI échec ALORS
        Afficher erreur sur stderr
        RETOURNER
    FIN SI

    TANT QUE ligne disponible FAIRE
        Lire ligne (max 1024 caractères)
        Afficher ligne sur stdout
    FIN TANT QUE

    Fermer fichier
FIN FONCTION
```

2.7 Point d'entrée : `main()`

Principe :

La fonction `main()` sert de point d'entrée du programme et gère le choix du mode d'affichage en fonction des arguments de ligne de commande. Elle initialise d'abord la structure de données à zéro, puis appelle `func_opred()` pour charger et analyser le fichier de logs. Ensuite, elle examine les arguments : si `-txt` est passé, elle active le mode texte ; si `-gr` est passé, elle active le mode graphique ; sinon, elle active par défaut le mode web (CGI). Le programme retourne `EXIT_SUCCESS` à la fin de son exécution.

Pseudo-code :

```
FONCTION main(argc : entier, argv : tableau de chaînes) : entier
    Initialiser data ← {0}
```

```
3
4       Appeler func_opred(&data)
5
6       SI argc > 1 ALORS
7           SI argv[1] = "-txt" ALORS
8               Appeler versiontexte(&data)
9           FIN SI
10
11          SI argv[1] = "-gr" ALORS
12              Appeler versiongraphique(&data)
13          FIN SI
14      SINON
15          Appeler versionweb(&data)
16      FIN SI
17
18      RETOURNER EXIT_SUCCESS
19  FIN FONCTION
```

Modes d'exécution :

- Sans argument : mode web (CGI)
- Avec `-txt` : mode texte (terminal)
- Avec `-gr` : mode graphique (SDL)

---

3. Code source

3.1 Fichier : versiondynamique.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <libgraphique.h>

#define LOG "/var/log/apache2/access.log"

typedef struct {
    int n_rq;
    int monthly[12];
    int first_day;
    int first_month;
    int first_year;
    int has_first_date;
} Data;

const char *months_abbrev[] = {"Jan", "Feb", "Mar", "Apr", "May", "Jun"
    ,
```

```c
                                    "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"};
const char *months_full[] = {"Janvier", "Fevrier", "Mars", "Avril", "Mai", "Juin",
                             "Juillet", "Aout", "Septembre", "Octobre", "Novembre", "Decembre"};

int extract_date(const char *s, Data *data) {
    char *start;
    char day_str[3];
    char month_str[4];
    char year_str[5];
    int day, year, month, m;

    start = strchr(s, '[');
    if (!start) return 0;
    start++;

    if (sscanf(start, "%2s/%3s/%4s", day_str, month_str, year_str) != 3)
        return 0;

    day = atoi(day_str);
    year = atoi(year_str);
    month = -1;

    for (m = 0; m < 12; m++) {
        if (strncmp(month_str, months_abbrev[m], 3) == 0) {
            month = m;
            break;
        }
    }

    if (month == -1) return 0;

    if (!data->has_first_date) {
        data->first_day = day;
        data->first_month = month;
        data->first_year = year;
        data->has_first_date = 1;
    }

    data->monthly[month]++;
    return 1;
}

void func_opred(Data *data) {
    FILE *fp;
    char *buffer;
    char *new_buffer;
    size_t buffer_size;
```

```
65      size_t position;
66      size_t new_size;
67      int c;
68
69      fp = fopen(LOG, "r");
70      if (!fp) {
71          fprintf(stderr, "Erreur : Impossible d'ouvrir %s\n", LOG);
72          return;
73      }
74
75      buffer_size = 2048;
76      buffer = malloc(buffer_size);
77      if (!buffer) {
78          fprintf(stderr, "Erreur : Allocation mémoire\n");
79          fclose(fp);
80          return;
81      }
82
83      position = 0;
84
85      while ((c = fgetc(fp)) != EOF) {
86          if (position >= buffer_size - 1) {
87              new_size = buffer_size * 2;
88              new_buffer = realloc(buffer, new_size);
89
90              if (!new_buffer) {
91                  fprintf(stderr, "Erreur : Réallocation mémoire\n");
92                  free(buffer);
93                  fclose(fp);
94                  return;
95              }
96
97              buffer = new_buffer;
98              buffer_size = new_size;
99          }
100
101         buffer[position] = c;
102         position++;
103
104         if (c == '\n') {
105             buffer[position - 1] = '\0';
106             if (extract_date(buffer, data))
107                 data->n_rq++;
108             position = 0;
109         }
110     }
111
112     free(buffer);
113     fclose(fp);
114 }
115
```

```c
void lecture_style(void) {
    FILE *fd;
    char str[1024];

    fd = fopen("aecrire.html", "r");
    if (!fd) {
        fprintf(stderr, "ERREUR DE CHARGEMENT DE FICHIER\n");
        return;
    }

    while (fgets(str, 1024, fd)) {
        printf("%s", str);
    }

    fclose(fd);
}

void versiontexte(Data *data) {
    int m;
    int count;
    float percent;

    if (data->n_rq == 0) {
        printf("Aucune donnée à afficher.\n");
        return;
    }

    printf("Depuis le %02d/%s/%d on a enregistre %d connexions.\n",
           data->first_day,
           months_abbrev[data->first_month],
           data->first_year,
           data->n_rq);

    for (m = 0; m < 12; m++) {
        count = data->monthly[m];
        percent = (float)count * 100.0f / data->n_rq;
        printf("%10s : %4.2f%%\n", months_full[m], percent);
    }
}

void versiongraphique(Data *data) {
    int height, x_pos, y_pos;
    int m, count;
    char msg[256];
    char label[4];
    float percent;
    Point p;

    if (data->n_rq == 0) {
        printf("Aucune donnée à afficher.\n");
        return;
```

```
167     }
168
169     ouvrir_fenetre(430, 350);
170     dessiner_rectangle((Point){0, 0}, 430, 350, blanc);
171
172     sprintf(msg, "Depuis le %02d/%s/%d on a enregistre %d connexions.",
173             data->first_day,
174             months_abbrev[data->first_month],
175             data->first_year,
176             data->n_rq);
177
178     p.x = 30;
179     p.y = 50;
180     afficher_texte(msg, 12, p, noir);
181
182     p.y = 300;
183     afficher_texte(" Jan  Fev  Mar  Avr  Mai  Juin  Juil  Aou  Sep  Oct
          Nov  Dec", 14, p, vert);
184
185     for (m = 0; m < 12; m++) {
186         count = data->monthly[m];
187         percent = (float)count * 100.0f / data->n_rq;
188         height = (int)(percent * 10.0f);
189         x_pos = 30 + (m * 31);
190         y_pos = 300 - height;
191
192         dessiner_rectangle((Point){x_pos, y_pos}, 30, height, orange);
193
194         if (height >= 25) {
195             sprintf(label, "%d", (int)percent);
196             afficher_texte(label, 14, (Point){x_pos + 8, y_pos + 8},
                 blanc);
197         }
198     }
199
200     actualiser();
201     attendre_clic();
202     fermer_fenetre();
203 }
204
205 void versionweb(Data *data) {
206     int height, left_pos;
207     int m, count;
208     float percent;
209
210     printf("Content-type: text/html\n\n");
211     printf("<!doctype html>\n");
212     printf("<html lang=\"fr\">\n");
213     printf("<body>\n");
214
215     lecture_style();
```

```
216
217     printf("<p>Depuis le %02d/%s/%d on a enregistr&eacute; %d
            connexions.</p>\n",
218             data->first_day,
219             months_abbrev[data->first_month],
220             data->first_year,
221             data->n_rq);
222
223     if (data->n_rq == 0) {
224         printf("Aucune donnée à afficher.\n");
225         return;
226     }
227
228     printf("<div id=\"vertgraph\">\n");
229     printf("<ul>\n");
230
231     for (m = 0; m < 12; m++) {
232         count = data->monthly[m];
233         percent = (float)count * 100.0f / data->n_rq;
234         height = (int)(percent * 10.0f);
235         left_pos = 10 + (m * 31);
236
237         if (height >= 25) {
238             printf("<li style=\"left:%dpx;height:%dpx\" title=\"%s\">%d
                    </li>\n",
239                     left_pos, height, months_abbrev[m], (int)percent);
240         } else {
241             printf("<li style=\"left:%dpx;height:%dpx\" title=\"%s\"></
                    li>\n",
242                     left_pos, height, months_abbrev[m]);
243         }
244     }
245
246     printf("</ul>\n");
247     printf("</div>\n");
248     printf("</body>\n");
249     printf("</html>\n");
250 }
251
252 int main(int argc, char* argv[]) {
253     Data data = {0};
254
255     func_opred(&data);
256
257     if (argc > 1) {
258         if (strcmp(argv[1], "-txt") == 0) {
259             versiontexte(&data);
260         }
261         if (strcmp(argv[1], "-gr") == 0) {
262             versiongraphique(&data);
263         }
```

```
264     } else {
265         versionweb(&data);
266     }
267 
268     return EXIT_SUCCESS;
269 }
```

3.2 Fichier : Makefile

```
1  CC = gcc
2  CFLAGS = -Wall -Wextra -std=c11
3  LIB_DIR = ../lib
4  CFLAGS += -I$(LIB_DIR)
5  LDFLAGS = -lSDL -lSDL_ttf -lm
6  TARGET = versiondynamique
7  SOURCES = versiondynamique.c $(LIB_DIR)/libgraphique.c
8  
9  CGIDIR = /usr/lib/cgi-bin/
10 LOGDIR = /var/log/apache2/
11 WWWDIR = /var/www/
12 HTMLDIR = $(WWWDIR)/html/
13 
14 # Règle par défaut : nettoyer les builds locaux, compiler, puis
       installer
15 all: clean $(TARGET) install
16 
17 # Compiler le programme
18 $(TARGET): $(SOURCES)
19     $(CC) $(CFLAGS) -o $(TARGET) $(SOURCES) $(LDFLAGS)
20 
21 # Installer
22 install: install-html install-cgi
23     @sudo chmod a+rx $(LOGDIR)
24     @sudo chmod a+r $(LOGDIR)/access.log
25 
26 install-cgi:
27     sudo cp aecrire.html versiondynamique ../www/codescompiles/
28     @cd ../www/codescompiles/ ; \
29     sudo cp aecrire.html versiondynamique \
30          $(CGIDIR) ; \
31     cd ../../src/
32     @echo "Installation du programme :"
33     sudo cp $(TARGET) $(CGIDIR)
34 
35 install-html:
36     @echo "Copie des fichiers aux bons emplacements"
37     @sudo tar xf ../www/www.tar -C /
38     @echo "Copie d'un fichier de log de taille conséquente"
39     @sudo cp ../www/access.log.big $(LOGDIR)/access.log
```

```
40
41 # Nettoyer les fichiers de compilation
42 clean:
43     rm -f $(TARGET) *.o
44
45 # Nettoyer l'installation
46 clean-install:
47     @sudo rm -rf /var/www/*
48     @sudo mkdir $(HTMLDIR)
49     @sudo cp ../www/index.html $(HTMLDIR)
50     @sudo rm -rf $(CGIDIR)/*
51
52 .PHONY: all install install-cgi install-html clean clean-install
```

Explication du Makefile :

- `CFLAGS` : Options de compilation + inclusion du dossier `lib`
- `LDFLAGS` : Liens vers les bibliothèques SDL
- `all` : Règle par défaut, nettoie, compile et installe
- `install` : Copie l'exécutable dans `/usr/lib/cgi-bin/` avec les permissions
- `install-cgi` : Installation du CGI et des fichiers associés
- `install-html` : Installation des fichiers web et du fichier de log
- `clean` : Supprime les fichiers générés
- `clean-install` : Réinitialise l'installation web

3.3 Fichier : aecrire.html

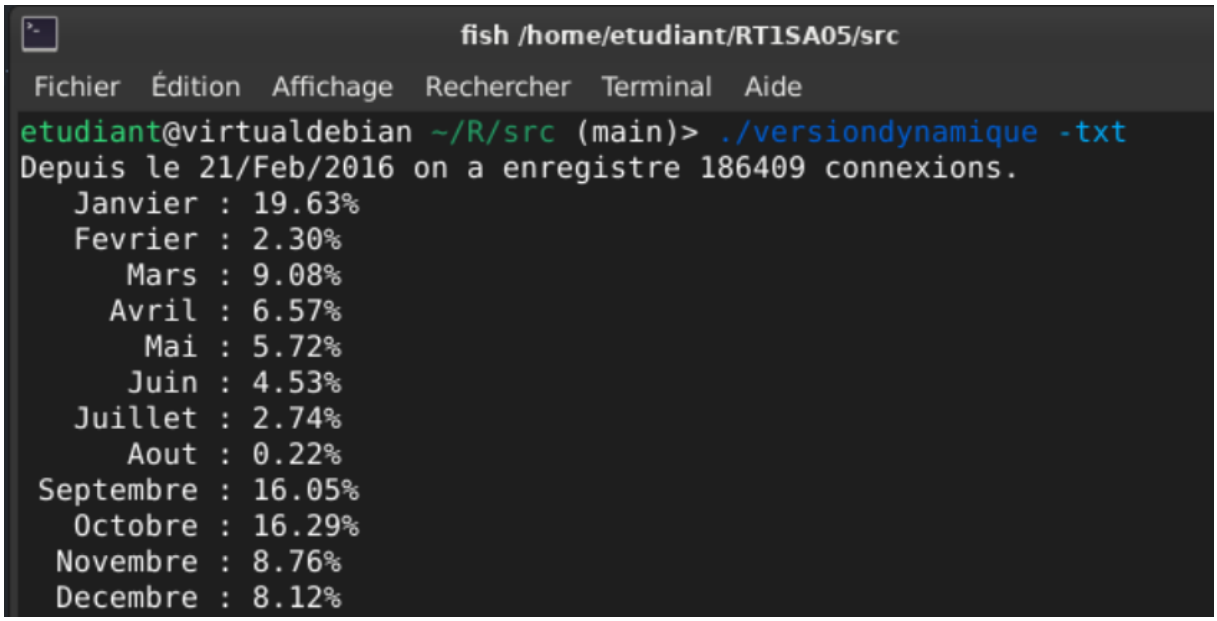
```
1  <style>
2      #vertgraph {
3          width: 378px;
4          height: 230px;
5          position: relative;
6          text-align: left;
7          font-weight: bold;
8          color: #5F8000;
9          line-height: 2.5em;
10         background: url("../img/mois.gif") no-repeat;
11         background-position:9px 210px;
12     }
13     #vertgraph .graph-area {
14         width: 100%;
15         top: 11px;
16         height: 200px;
17         position: relative;
18         overflow: hidden;
19     }
20     #vertgraph ul {
```

```
        width: 100%;
        height: 100%;
        position: relative;
        bottom:40;
        margin: 0;
        padding: 0;
        list-style-type: none;
    }
    #vertgraph ul li {
        list-style-type: none;
        position: absolute;
        width: 30px;
        height: 100px;
        bottom: 20px;
        padding: 0;
        margin: 0;
        background: #008080;
        text-align: center;
        font-weight: bold;
        color: white;
        line-height: 2.5em;
    }
</style>
```

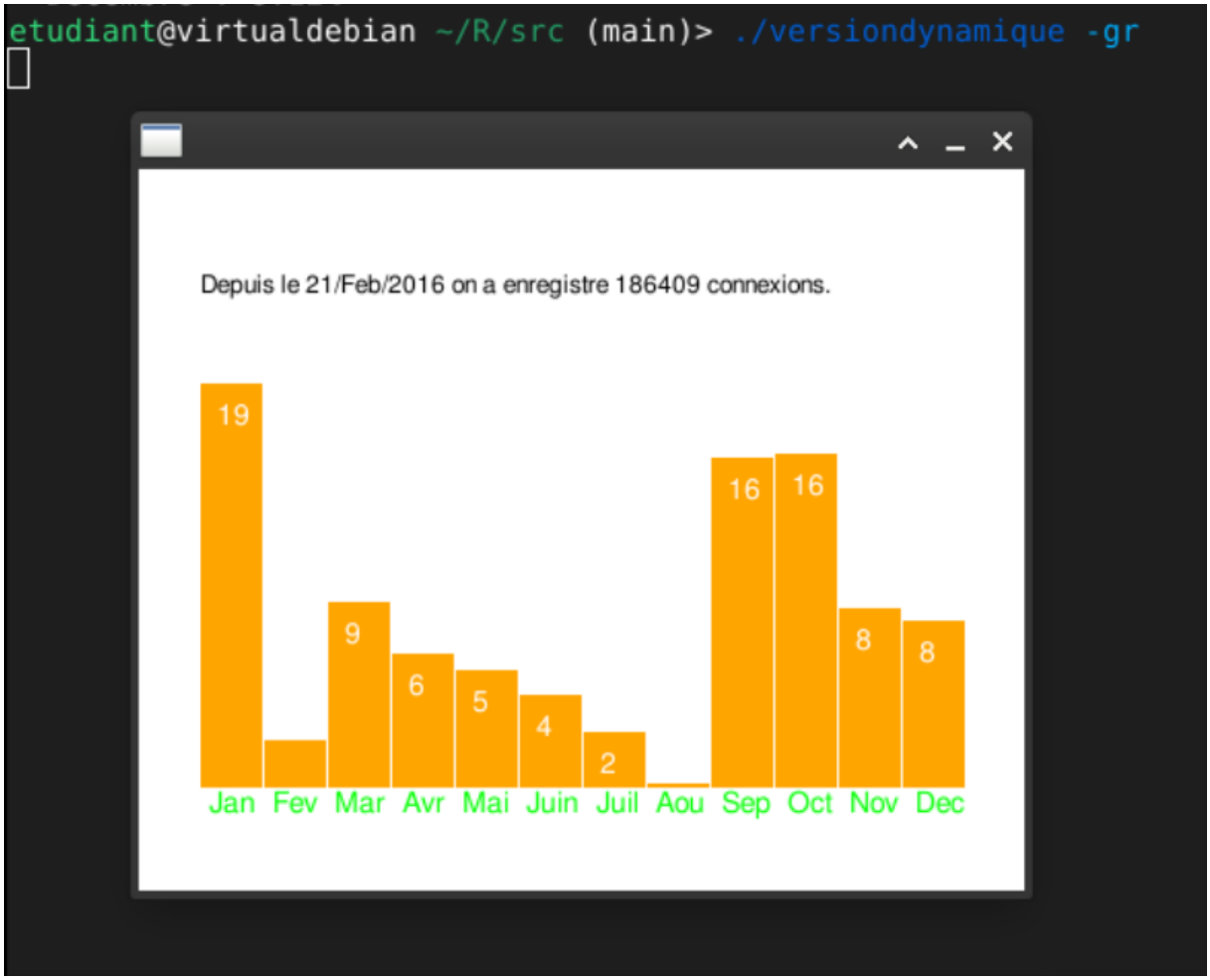
---

4. Tests et résultats

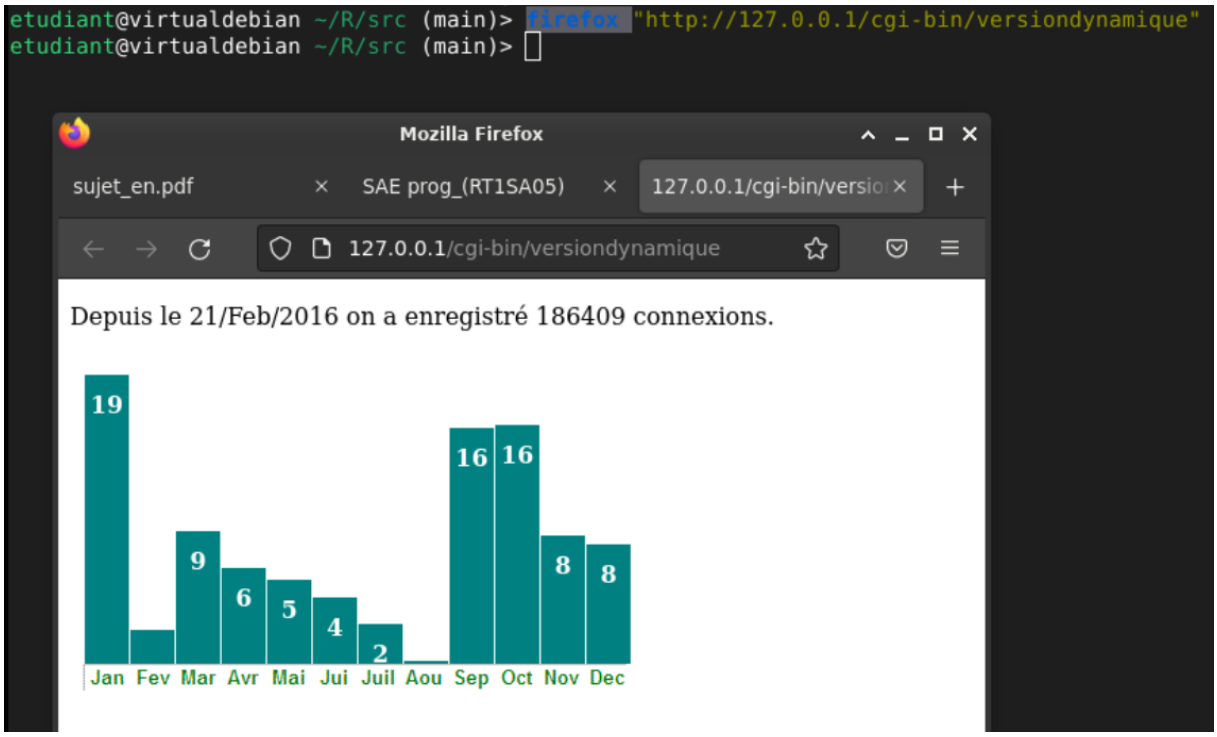
4.1 Test du mode texte



4.2 Test du mode graphique



4.3 Test du mode web (CGI)



---

5. Conclusion

Analyse du fichier log : Parsing complet du format CLF **Calcul des statistiques** : Total et distribution mensuelle **Trois interfaces** : Texte, graphique (SDL), web (CGI) **Makefile fonctionnel** : Compilation et installation automatiques **Code robuste** : Gestion des erreurs et buffer dynamique **Passage par paramètre** : Aucune variable globale (hormis constantes)