Programmation, TD 1: exercices pour se familiariser avec l'assembleur du Pentium, avec `gcc` et `gdb`

3 octobre 2002

Pour le moment, l'outil `ddd` n'est pas installé, à ma connaissance : ça viendra ! On va donc utiliser l'outil `gdb` à la place. Il fait autant de choses, mais est juste moins convivial...

On va commencer par écrire un petit programme. Pour ne pas faire trop compliqué, on va prendre celui dont je vous ai rebattu les oreilles au premier cours. Vous pouvez soit le télécharger depuis `http://www.lsv.ens-cachan.fr/~goubault/CoursProgrammation/cat.c`, soit le recopier, typiquement sous XEmacs :

```
#include <stdio.h>

main (int argc, char *argv[])
{
  int i, c;

  for (i=1; i<argc; i++)
    {
      FILE *f;

      f = fopen (argv[i], "r");
      while ((c = fgetc (f))!=EOF)
        fputc (c, stdout);
      fclose (f);
    }
  fflush (stdout);
  exit (0);
}
```

Je ne vous demande pas de comprendre ce que ça fait ! (pour l'instant en tout cas.)

1. Compilez ce programme. En supposant que le fichier source ci-dessus s'appelle `cat.c`, invoquez le compilateur C en tapant :

   ```
   gcc -ggdb -o mycat cat.c
   ```

   Ceci appelle le compilateur C `gcc`, lui demande de compiler (traduire en code machine) le fichier `cat.c` vers le fichier exécutable `mycat` (l'*option* `-o mycat`). L'option `-ggdb` demande à `gcc` d'inclure dans le fichier `mycat` suffisamment d'informations pour que le debugger (`gdb` ou `ddd`, voir plus loin) retrouve ses petits.

   Vérifiez que `gcc` a bien créé un fichier `mycat` en tapant `ls`.

   Pour les hackers : vous pouvez tout savoir sur `gcc` en tapant `man gcc`, ou mieux `info gcc`. Pour savoir ce que fait `man`, tapez `man man`.

2. On va maintenant regarder ce que fait ce programme. Tapez :

   ```
   ./mycat cat.c
   ./mycat cat.c cat.c
   ```

Ceci fait-il ce à quoi vous vous attendiez ?

3. Regardons maintenant sous le debugger `gdb`. Tapez :

   ```
   gdb mycat
   ```

   Sous `gdb`, tapez `b main` pour lui dire de poser un point d'arrêt (*breakpoint*) à l'entrée de la fonction `main`. `gdb` devrait répondre quelque chose comme :

   ```
   Breakpoint 1 at 0x8048536: file cat.c, line 7.
   ```

   Regardons où ça se trouve dans le source, en tapant :

   ```
   list
   ```

   Le point d'arrêt 1 que nous venons de poser est au début de la ligne 7, avant le `for (i=1; ....` On va regarder les valeurs des variables évoluer. Pour ceci, lançons `mycat` avec l'argument `cat.c` :

   ```
   run cat.c
   ```

   Le programme doit avoir démarré, et s'être arrêté au point d'arrêt numéro 1. Tapez :

   ```
   display argc
   display argv[0]
   display argv[1]
   display i
   display c
   display (char)c
   ```

   Tapez `n` (comme "next"). `gdb` vous montre qu'il a exécuté le code de la ligne 7, qui essentiellement met `i` à 1, et est passé au début de la ligne 11. Vérifiez que `i` vaut bien 1 maintenant, et qu'aucune autre variable n'a changé.

   Continuez à taper `n`, et observez le flux d'exécution du programme, en particulier l'évolution de la valeur de la variable `c`, qui récupère les caractères lus successivement depuis les fichiers pris en argument.

4. On va maintenant refaire tourner `mycat` en regarder ce qui se passe au niveau du code machine. Quitter `gdb` en tapant `q` et en répondant `y` à la question. Relancer par `gdb mycat`. Comme avant, poser un point d'arrêt par `b main`, puis lancer en tapant `run cat.c`. Maintenant, tapez :

   ```
   disassemble
   ```

   C'est le code machine Pentium de la fonction `main`. Arrivez-vous à mettre ce code en correspondance avec ce que fait le source ?

   Indications : `info registers` vous montre le contenu de tous les registres. `esp` est le registre de pointeur de pile. `x/4xw` suivi d'un nom de variable ou d'une adresse (`0xbffffe80` par ex.) vous montre les 4 mots 32 bits stockés à cette adresse, en hexadécimal. `x/s` vous montre le contenu en tant que chaîne de caractères (si c'en est bien une...). Le compilateur `gcc` a réservé l'espace suivant pour les variables :