Projet Unix

Nathan GUYOT Vincent LEFOULON

18 décembre 2015

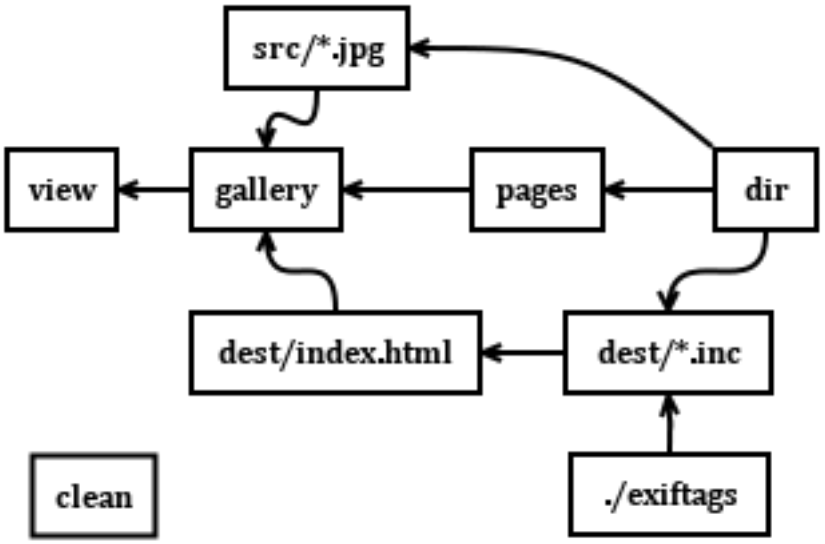
1 Le code

Dans les grandes lignes, le code s'organise autour d'une boucle : celle parcourant les images du dossier source. Cette boucle se base sur les indices et non directement sur les éléments (stockés dans un `array`) afin de pouvoir plus facilement récupérer les images adjacentes.

A chaque tour de boucle, on crée la miniature, puis génère la page d'aperçu et enfin ajoute l'image au fichier HTML.

2 Le Makefile

Le graphe des tâches est le suivant :



Chaque règle se contente d'appeler un script shell de la forme `make-X.sh`, lequel analyse ses paramètres et exécute un des scripts implémentés dans la première version de la galerie.

3 Mesure du temps d'exécution

Avec six images dans le dossier source, la commande `time make gallery` affiche :

```
real  0m1.724s
user  0m1.010s
sys 0m0.157s
```

La parallélisation donne :

```
$ make clean; time make -j 15 gallery
real  0m0.548s
user  0m1.257s
sys 0m0.137s

$ make clean; time make -j 12 gallery
real  0m0.521s
user  0m1.140s
sys 0m0.160s

$ make clean; time make -j 9 gallery
real  0m0.510s
user  0m1.183s
sys 0m0.127s

$ make clean; time make -j 6 gallery
real  0m0.526s
user  0m1.207s
sys 0m0.150s

$ make clean; time make -j 2 gallery
real  0m0.530s
user  0m0.937s
sys 0m0.077s
```

On constate que la parallélisation diminue fortement le temps d'exécution. Les performances commencent par augmenter avec le nombre de processus parallèles, puis on atteint un seuil à partir duquel on a trop de processus par rapport aux opérations réalisables en parallèle, ce qui coûte des ressources sans nous en faire économiser.

4 Les données EXIF

Les suffixes dans les Makefiles sont obsolètes, notamment parce qu'ils ne permettent pas ne définir de dépendances. Dans le cas présent, les fichiers

`.c` n'ont donc pas pour dépendance le `.h` correspondant. On pourrait plutôt faire :

```
%.c: %.h

%.o: %.c
$(CC) $(CFLAGS) -o $@ -c $<
```