1. Introduction à make

1.1 Commencez par un petit « man make » juste pour voir ...

1.2 Testez les commandes suivantes :

- Avec un fichier Makefile dans le répertoire courant (utilisé par défaut) :
  make
  make -n
  make -i
  make autre
- Avec un fichier « makefile2 » masquant les commandes exécutées (via « @ ») :
  make -f makefile2
  make -nf makefile2
  make -if makefile2
  make -f makefile2 autre
- Avec un fichier « makefile3 » ignorant certaines erreurs (via « - ») :
  make -f makefile3
  make -nf makefile3
  make -if makefile3
  make -f makefile3 autre
- Variables passées par commande :
  ATTENTION : $(TOTO) et ${TOTO} sont valides mais pas $TOTO ni $[TOTO]
  make toto
  make toto TOTO=object.c
- Variables passées par commande avec valeur par défaut :
  make titidef
  make titidef TITI=valeur
- Manipulation des variables (au passage regardez comment fonctionne « AFFICHE », il est possible de simplifier en mettant « affiche AFFICHE: » dans cet exemple) :
  make affiche
  make AFFICHE
- Variables automatiques (exécutez les commandes suivantes dans l'ordre !) :
  rm printc
  make printc
  touch printc
  make printc
  touch *.c
  make printc
- Makefile récursif :
  Déjà vu (ou sera vu) lors de la compilation du module/driver SCULL (en TP SRS/Kernel)

```
SUBDIRS = misc-modules skull scull pci simple
all: subdirs
subdirs:
	for n in $(SUBDIRS); do $(MAKE) -C $$n || exit 1; done
clean:
	for n in $(SUBDIRS); do $(MAKE) -C $$n clean; done
```

- Exemple de règles implicites (accessibles via default*.mk sur certains O.S.) :

```
OUTPUT_OPTION=
AR=ar
ARFLAGS=cr
RM=rm -f
CC=gcc
CFLAGS=
CPPFLAGS=
COMPILE.c=$(CC) $(CFLAGS) $(EXTRA_CFLAGS) $(CPPFLAGS) -c
LINK.c=$(CC) $(CFLAGS) $(EXTRA_CFLAGS) $(CPPFLAGS) $(LDFLAGS)
.c:
	$(LINK.c) -o $@ $< $(LDLIBS)
.c.o:
	$(COMPILE.c) $(OUTPUT_OPTION) $<
.c.a:
	$(COMPILE.c) -o $% $<
	$(AR) $(ARFLAGS) $@ $%
	$(RM) $%
```

2. Problème basé sur l'utilisation de make

2.1 Problème posé

Cet exercice est basé sur un cas réel où il s'agit de construire une « librairie » ou plutôt une archive au format « .a » destinée à contenir un ensemble de fichiers objet « .o » en vue de mettre en place des travaux pratiques pour des étudiant(e)s en L2. Dans le cadre de ce problème, l'archive a été fortement simplifiée, mais le problème posé reste le même...

L'enseignant dispose de toutes les sources des procédures et fonctions destinées à simplifier la vie des étudiant(e)s en masquant au maximum les aspects liés au système sous-jacent. Il les compile sous forme d'un ou plusieurs fichiers objet et les rassemble dans une archive au moyen des commandes suivantes :

```
ar cr libddtools.a monfichier1.o monfichier2.o monfichier3.o ...
ranlib libddtools.a
```

Les fichiers sources (.h et .c) les fichiers objets (.o) et l'archive (.a) sont dans un même répertoire « TP1Exercice1/TP1EX1SourcesCompletes/src ». Le fichier Makefile destiné à créer cette archive est malheureusement incomplet. Il est lui aussi situé dans ce répertoire, vous devrez le compléter en vous assurant que les commandes lancées ne soient pas affichées pour faciliter la lecture de l'affichage généré par la commande make.

Pour faciliter la vie de l'enseignant, ce dernier doit juste invoquer la commande « make » depuis le répertoire « TP1Exercice1/TP1EX1SourcesCompletes » pour créer l'archive « libddtools.a » dans le répertoire « TP1Exercice1/TP1EX1SourcesCompletes/src ». De même « make clean » fait le ménage (sans détruire l'archive créée !).

Une fois créée, l'enseignant doit juste copier l'archive « libddtools.a » dans le répertoire « TP1Exercice1/TP1EX1PourEtudiants/src » (attention pour l'instant l'archive est remplacée par un fichier vide !). Et les étudiant(e)s ont juste besoin d'appeler la commande « make » depuis le répertoire « TP1Exercice1/TP1EX1PourEtudiants » pour compiler tous les exemples (fork, pingpong, lecteurs-rédacteurs, skieurs-téléski). Notez qu'il n'y a pas de fichier Makefile dans le répertoire « TP1Exercice1/TP1EX1PourEtudiants/src ». C'est tout à fait normal car l'enseignant a compilé une bonne fois pour toutes l'archive à la place des étudiant(e)s ! Par contre, dans chaque sous répertoire (fork, pingpong, lecteursredacteurs, skieursteleski) figure un fichier Makefile à compléter !

Pour compiler ces exemples à la main, l'enseignant utilisait auparavant une commande du genre :

```
gcc -O2 -Wall -I../src -L../src -o exemple1 exemple1.o -lddtools -lm
```

Vous devrez vous en inspirer en utilisant notamment les « variables » définies dans le fichier Makefile.

2.2 Votre mission en quelques mots...

Complétez les fichiers « Makefile » suivants (pas forcément dans cet ordre bien sûr !) :

```
TP1Exercice1/TP1EX1PourEtudiants/fork/Makefile
TP1Exercice1/TP1EX1PourEtudiants/lecteursredacteurs/Makefile
TP1Exercice1/TP1EX1PourEtudiants/pingpong/Makefile
TP1Exercice1/TP1EX1PourEtudiants/skieursteleski/Makefile
TP1Exercice1/TP1EX1SourcesCompletes/src/Makefile
```

3. Introduction à cmake & cie

3.1 Commencez par installer les paquets utiles pour la suite :

Installez les paquets suivants à l'aide de la commande « aptitude install » ou ...

Il n'est pas nécessaire d'installer tous les paquets, regardez AVANT ce que contient chaque paquet et choisissez éventuellement de ne pas installer certains paquets !

```
cmake swig swig-doc swig-examples libboost libboost-doc
glade glade-gnome glademm eclipse eclipse-cdt
gdb xxgdb kdbg insight ddd ddd-doc
ccmalloc sysprof valgrind
alleyoop nemiver kcachegrind kcachegrind-converters
python python-mode python-application python-doc
diveintopython eric boa-constructor
pida pychecker cython pydb pymacs pylint
libcorelinux-dev libcorelinux-doc libcorelinux-examples
```

3.2 Détour par un petit « man cmake » juste pour voir ...

Allez voir également le contenu du répertoire /usr/share/doc/cmake.

3.3 Depuis le sous-répertoire cmake testez les commandes suivantes :

Pour créer automatiquement le fichier Makefile : cmake . (n'oubliez pas le « . »)

Pour compiler le programme : make

En cas de problème avec cmake : supprimez le fichier CmakeCache.txt et relancez cmake

Un certain nombre de cibles sont créées automatiquement, testez via : make help

Pour faire le ménage : make clean

3.4 Commencez par installer les paquets utiles pour la suite :

Installez les paquets suivants à l'aide de la commande « aptitude install » ou ...

Il n'est pas nécessaire d'installer tous les paquets, regardez AVANT ce que contient chaque paquet et choisissez éventuellement de ne pas installer certains paquets !

```
g++ htop apt-file
git git-core git-gui git-doc
libboost-dev libboost-test-dev libboost-date-time-dev
libxml++2.6-2 libxml++2.6-dev libsqlxx-dev
libglademm-2.4 libglademm-2.4-dev
```