

CSI605

imake

What is imake?

- **imake is designed to deal with the insidious problem that make files are not portable**
- **make**
 - **We create a Makefile using our favorite editor**
 - **make then reads this Makefile and creates our target files according to the instructions contained within**
- **imake**
 - **We create an Imakefile with our editor**
 - **imake then reads the Imakefile along with system dependent configuration files and then creates a system appropriate Makefile**
 - **make then reads this system appropriate Makefile and outputs the desired target files**
- **When we move our software to a new machine we are not faced with editing the Makefile in order to hand craft the system specific code**
 - **Instead we merely rerun our imake procedure**
- **X11 is typically installed with it**

Advantages of imake

- **Easier to write an Imakefile than a Makefile**
- **Easier to collaborate with individuals**
- **Allows one to work painlessly in a heterogeneous environment**
- **Eases software distribution**
- **Obviously eases porting software to new platforms**

The imake Toolkit

- **Minimum requirements**
 - **imake and a set of configuration files**
 - **cpp and make**
 - **xmkmf**
 - **Bootstraps a Makefile from an Imakefile**
- **Other needed programs**
 - **makedepend**
 - **Generates header file dependencies from C source files**
 - **mkdirhier**
 - **Creates directories during file installation operations if your system does not support mkdir -p**
 - **bsdinst or install.sh**
 - **This installs files in the case where one does not have a BSD compatible install program**

Ascertaining the State of Your imake Installation

- **cpp and make are pretty standard on most UNIX/LINUX installations**
- **imake, xmkkmf are usually found in X11-related directories and perhaps some X11 configuration files**
- **Look here for configuration files**
 - **lib/X11/config under /usr or /usr/X11Rn**
 - **n indicates a release number**
 - **Might also try**
 - **/var/X11Rn, /opt/X11Rn /local/X11Rn, /usr/openwin/lib/config**
- **which imake**
- **Look for imake in**
 - **/usr/bin/X11**
 - **/usr/X11Rn/bin**
 - **/var/X11Rn/bin**
 - **/opt/X11Rn/bin**
 - **/local/X11Rn/bin**
 - **/usr/openwin/bin**
- **find / -name Imake.tmpl -print**
- **find / -name imake -print**
- **Setting path (under csh or tcsh)**
 - **set path = (dir1 dir2 dir 3 /usr/X11R6.1/bin)**

Obtaining the Necessary Software

- **Look for the software under**
 - **<http://www.primate.wisc.edu/software>**
- **Minimally one needs**
 - **One of the itools... files**
 - **Puruse the Readme files**

Simple Exercise 1

Write an Imakefile and Generate the Makefile

- **Step 1 - Create a vacuous Imakefile**
 - **cp /dev/null Imakefile**
- **Step 2 - The paradox**
 - **We wish to use a Makefile to run imake for us but we need to run imake to create the necessary Makefile**
 - **Approach 1 - run imake manually**
 - **Approach 2 -use bootstrap program**
 - **xmkmf**
 - **This produces the necessary Makefile to run imake**
- **Step 3 - use the makefile as usual**
 - **make clean**
 - **Removes any garbage files**
 - **make**
 - **Produces no output since there is nothing to build yet**
 - **make Makefile**
 - **Regenerates current Makefile after moving existing Makefile to Makefile.bak**

Simple Exercise Number 2

Building a Simple Program - I

- **Goal**
 - Use `imake` to build a simple hello world program.
- **The program**

```
#include <stdio.h>  
main(){  
    printf("hello, world \n");  
}
```
- **A hand configured Makefile**

```
hello: hello.o  
    cc -o hello hello.o
```
- **Compilation of the program using make**

```
make  
cc -c hello.c  
cc -o hello hello.o
```

Simple Exercise Number 2

Building a Simple Program - II

- Using imake for this compilation
 - Edit Imakefile to include
NormalProgramTarget
(hello,hello.o,NullParameter,NullParameter,NullParameter)
- NormalProgramTarget() is called the rule
 - Rules are defined in the configuration files
 - Rules take arguments
 - Ex.
 - NormalProgramTarget
 - First Arg = Name of program to build
 - Second Arg = Object file or files to use in building
 - Third , Fourth, and Fifth = Rules required for more complicated targets
- Step 2 - Rebuild the Makefile
 - make Makefile
- Step 3 - Build the program
 - make hello

Simple Exercise Number 2

Building a Simple Program - III

- **Step 4 - Run the program**
 - hello
- **Step 5 - Clean things up**
 - make clean

What Have We Done So Far?

- **Written an Imakefile**
- **Bootstrapped the initial Makefile with xmkmf**
- **Used the Makefile to regenerate itself**
- **Used the Makefile to build a program**
- **Used the Makefile to remove debris**

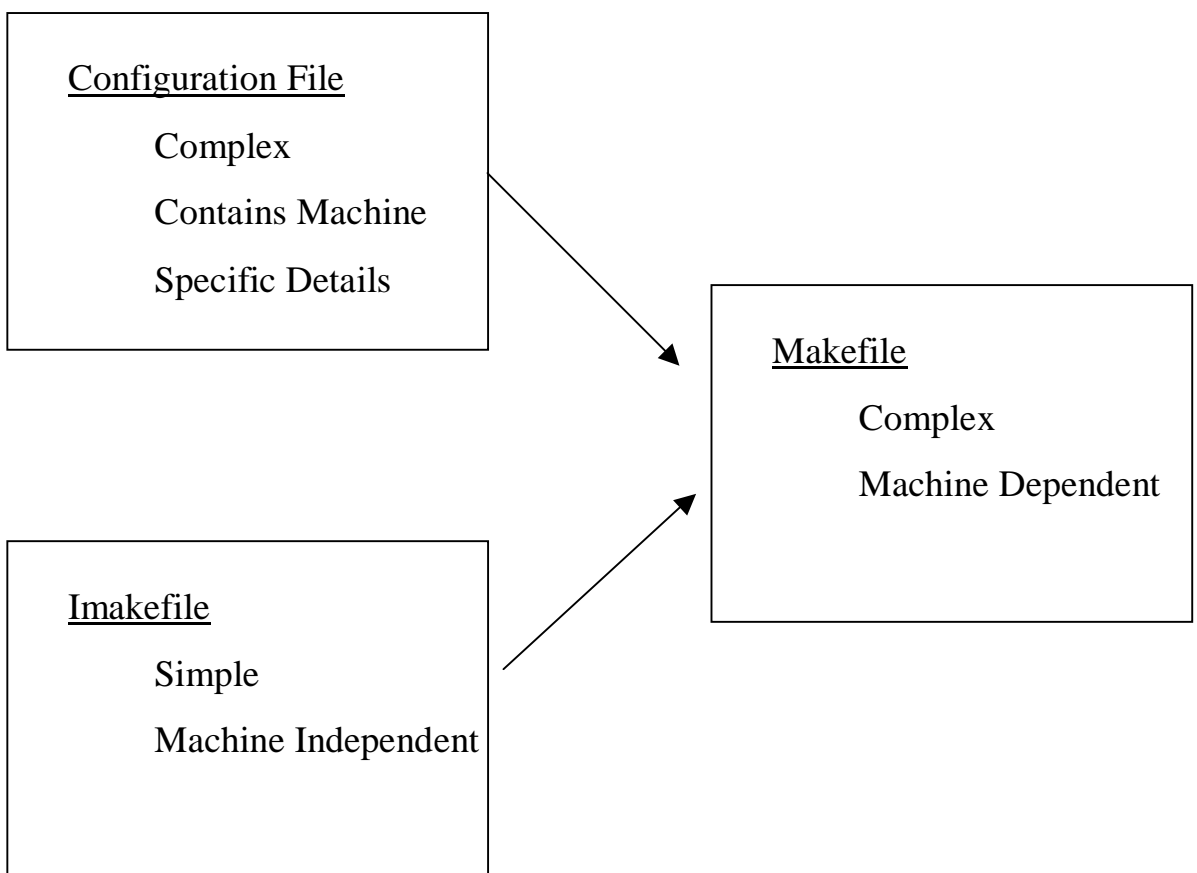
Exercise 3 - Examining the Makefile

- **What is in our imake created Makefile?**
 - **Along with a larger amount of other stuff there should be the 5 lines**

```
hello: hello.o
$(RM) $@
$(CCLINK) - o $@ $(LDOPTIONS) hello.o
$(LDLIBS) $(EXTRA_LOAD_FLAGS)

clean::
$(RM) hello
```
- **What about all of the other complexity?**
 - **Provides flexibility**
 - **Enters via the configuration files**
 - **Most users don't need to write configuration files and hence do not need to understand the full complexity of the matter**
 - **One often does not even need to look at the imake created Makefiles**
 - **How often does one modify postscript files by hand?**

How It All Fits Together?



Exercise 4 : Generating Header File Dependencies

- Sometimes targets are dependent on header files.
 - Example
 - `hello.c` is dependent on `stdio.h`
- Location of `stdio.h` is machine dependent
 - Hence header file dependencies are machine dependent
 - We will use `makedepend` to handle this
 - This will be incorporated into our `Imakefile`
- Modify `Imakefile` to resemble
 - `SRCS = hello.c`
 - `NormalProgramTarget(hello, hello.o, NullParameter,`
`NullParameter, NullParameter)`
 - `DependTarget()`
- Rebuild the Makefile
 - `make Makefile`
- Generate the dependencies
 - `make depend`
- Rebuild the program
 - `make hello`

Exercise 5 - Multiple Source Files and Dependencies

- Suppose that we had two programs

- `good.c`

```
main()  
{  
  bye();  
}
```

- `bye.c`

```
#include <stdio.h>  
bye()  
{  
  printf("goodbye, cruel world\n");  
}
```

- Here is the modified `Imakefile`

```
SRCS = hello.c good.c bye.c
```

```
NormalProgramTarget(hello, hello.o, NullParameter,  
  NullParameter, NullParameter)
```

```
NormalProgramTarget(goodby, good.o, bye.o,  
  NullParameter, NullParameter, NullParameter)
```

```
DependTarget()
```

- Rebuild Everything

```
make makefile
```

```
make depend
```

Why We Need the All Target

- We would like to be able to execute make all
 - Will not work with current version of the Makefile
 - There is no all target
- What if we execute just make
 - Only makes the first target in the Makefile which is hello
 - This is only true for certain distributions

Exercise 6 - The All Target

- We modify our Imakefile as follows
 - SRCS = hello.c goodbye.c**
 - AllTarget(hello)**
 - NormalProgramTarget(hello, hello.o, NullParameter, NullParameter, NullParameter)**
 - AllTarget(goodbye)**
 - NormalProgramTarget(goodbye, good.o, bye.o, NullParameter, NullParameter, NullParameter)**
 - DependTarget()**
- We have changed the Imakefile so we must rebuild the Makefile
 - **make Makefile**
- Rebuild the dependencies
 - **make depend**

Observations about the All Target

- It is the first target in the file so just make works as we desire
- make all also has the desired effect

Exercise 7: Checking Source Files

- There is a UNIX source code checker known as lint. It is used to find problems in your source code.
- Our modified Imakefile looks like
 - SRCS = hello.c goodbye.c
 - AllTarget(hello)
 - NormalProgramTarget(hello, hello.o, NullParameter, NullParameter, NullParameter)
 - AllTarget(goodbye)
 - NormalProgramTarget(goodby, good.o,bye.o, NullParameter, NullParameter, NullParameter)
 - DependTarget()
 - LintTarget()
- Rebuild the Makefile and regenerate the dependencies
- To run our program through lint we merely execute
 - make lint

Exercise 8 : Installing the Program

- Currently our Imakefile does not install our program anywhere. It does not generate any install entries.
- `InstallProgram(program, directory)`
- Our modified Imakefile
 - `SRCS = hello.c goodbye.c`
 - `AllTarget(hello)`
 - `NormalProgramTarget(hello, hello.o, NullParameter, NullParameter, NullParameter)`
 - `InstallProgram(hello, $(BINDIR))`
 - `AllTarget(goodbye)`
 - `NormalProgramTarget(goodbye, good.o, bye.o, NullParameter, NullParameter, NullParameter)`
 - `InstallProgram(goodbye, $(BINDIR))`
 - `DependTarget()`
 - `LintTarget()`
- `BINDIR`
 - `/usr/X11R6.1/bin`
 - `/usr/bin/X11`
- Rebuild Makefile and dependencies
- To invoke the Makefile to show what it would do without actually running it we execute
 - `make -n install`

Recommended References

- **“Practical Software Engineering Software Portability with imake,” Paul Dubois**