CSE 120

UCSD

17 January 2007
CVS Basics

CVS = "Concurrent Versions System"

Designed to:

- Keep a history of changes made to your source code
  - make it easy to go back if you break something
- Allow multiple people to collaboratively work on the same code
  - each person gets a private copy
  - automates merging changes made by other people, so changes are not lost
Repository: Stores entire history of all files, in compressed form (one per project)

Working Directory: Where you make develop and make changes (one per person)
export CVSROOT=/path/to/repository
export CVSROOT=/path/to/repository

- cvs checkout nachos
CVS Commands

```bash
export CVSROOT=/path/to/repository

▶ cvs checkout nachos

▶ cvs add, cvs remove

▶ cvs commit

▶ cvs checkout nachos
```

![Repository Diagram]
CVS Commands

```bash
export CVSROOT=/path/to/repository
```

- `cvs checkout nachos`
- `edit files`
- `cvs add, cvs remove`

```
cvs checkout nachos
```

```
cvs commit
```

```
cvs checkout nachos
```

Repository

```
Working Directory
```

```
Working Directory
```
CVS Commands

- export CVSROOT=/path/to/repository
- cvs checkout nachos
- edit files
- cvs add, cvs remove
- cvs commit

Repository

Working Directory

Working Directory
export CVSROOT=/path/to/repository

- cvs checkout nachos
- edit files
- cvs add, cvs remove
- cvs commit

- cvs commit
- cvs update
Conflicts in CVS

Two people edit the same file...

$ cvs update
RCS file: /home/linux/ieng6/cs120w/cs120w1/cvstest/nachos/code/threads/synch.h
retrieving revision 1.1.1.1
retrieving revision 1.2
Merging differences between 1.1.1.1 and 1.2 into synch.h
rcsmerge: warning: conflicts during merge
cvs update: conflicts found in code/threads/synch.h
C code/threads/synch.h
Conflicts in CVS

CVS tells you what changes were made, but up to you to sort it out:

```
synch.h
...
<<<<<<<<< synch.h
// Your code
========
// Their code
>>>>>>1.2
...
```

Fix the changes (and remove conflict markers!), then commit.
Final Words on CVS

Other useful commands:

- `cvs import`
- `cvs diff`
- `cvs log`
- `cvs tag`

Lots of documentation available

We’ll send out a few more details on using CVS for the projects.
For project 1, you’ll be asked to implement and work with thread synchronization

- No userspace yet, only kernel threads
- Everything runs in same address space
A Brief Tour of NACHOS

Modules in code/threads:

- **list**: linked list data structure with support for priorities
- **main**: NACHOS “boot” code and command-line processing
- **scheduler**: maintains queue of threads in “ready” state
- **switch**: context-switch code, if you’re curious
- **synch**: implements semaphores for synchronization
- **synchlist**: synchronized version of list
- **system**: sets up interrupts, timer, etc.
- **thread**: defines NACHOS kernel threads
- **threadtest**: code for testing your threading code
- **utility**: random assortment (such as ASSERT)
A Brief Tour of NACHOS

Modules in code/threads:

- **list**: linked list data structure with support for priorities
- **main**: NACHOS “boot” code and command-line processing
- **scheduler**: maintains queue of threads in “ready” state
- **switch**: context-switch code, if you’re curious
- **synch**: implements semaphores for synchronization
- **synchlist**: synchronized version of list
- **system**: sets up interrupts, timer, etc.
- **thread**: defines NACHOS kernel threads
- **threadtest**: code for testing your threading code
- **utility**: random assortment (such as ASSERT)
A Brief Tour of NACHOS

Modules in code/threads:

- **list**: linked list data structure with support for priorities
- **main**: NACHOS “boot” code and command-line processing
- **scheduler**: maintains queue of threads in “ready” state
- **switch**: context-switch code, if you’re curious
- **synch**: implements semaphores for synchronization
- **synchlist**: synchronized version of list
- **system**: sets up interrupts, timer, etc.
- **thread**: defines NACHOS kernel threads
- **threadtest**: code for testing your threading code
- **utility**: random assortment (such as ASSERT)

Also see: code/machine/interrupt, for how interrupts are enabled/disabled
Hints for reading the code:

- Read through the `.h` files to see what the interfaces are
- Look at the `.cc` files if you want to see the implementation
- The code is well-commented, not too large, so don’t be intimidated
- Don’t need to read through everything