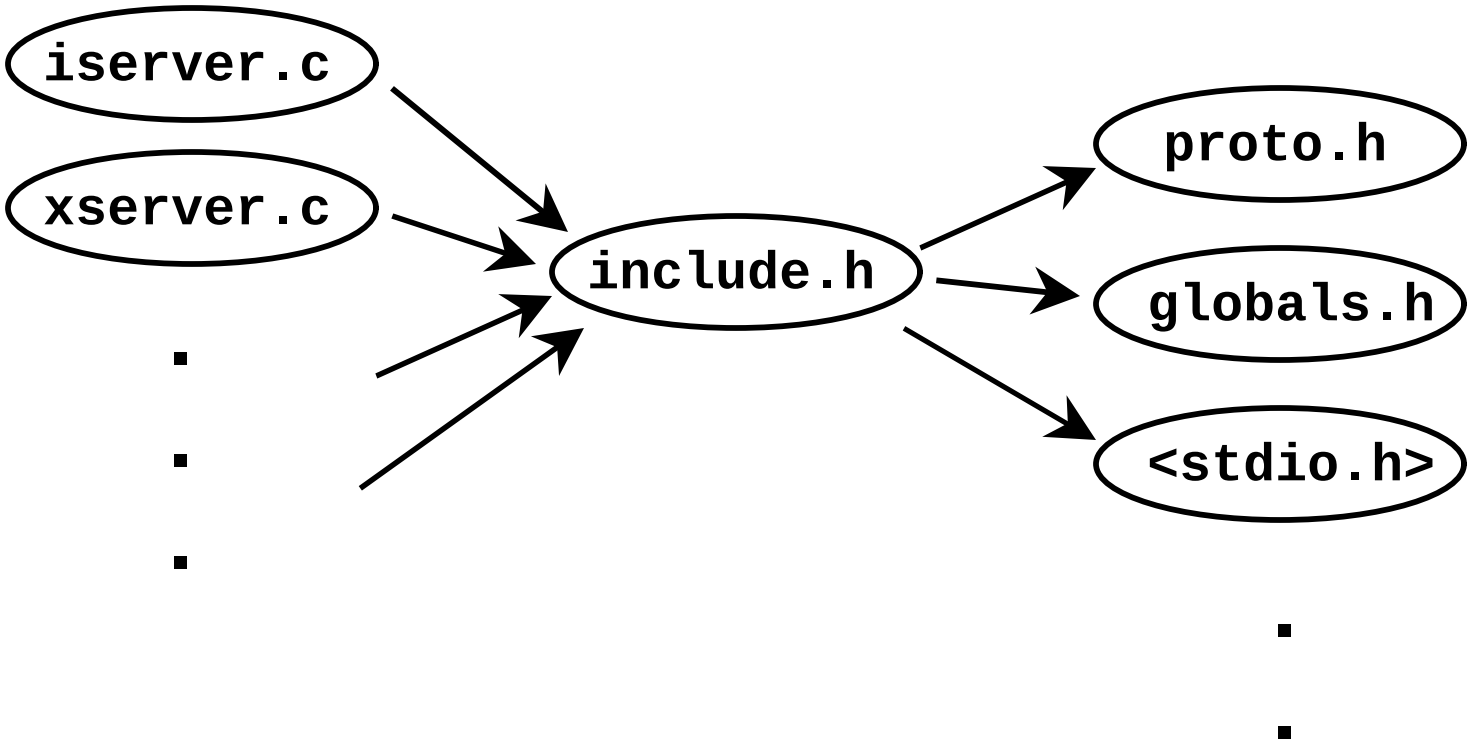
Project Management

Dependencies



Makefiles (p.430)

```
OBJS    = iserver.o xserver.o
CC      = gcc
CFLAGS  = -g
.c.o:
        $(CC) $(CFLAGS) -c $<

IServer: $(OBJS)
         $(CC) $(CFLAGS) $(OBJS) -o $@

iserver.o: include.h globals.h proto.h
xserver.o: include.h globals.h proto.h
clean:
        rm -f *.o IServer
```

Multiply-defined globals

iserver.c:

```
#include "include.h"

void main( void )
{
   X_ServerPid++;
   PrintPid();
}
```

xserver.c:

```
#include "include.h"
void PrintPid()
{
   printf( "X_ServerPid:%d\n",
           X_ServerPid );
}
```

include.h:

```
#include <stdio.h>
#include "proto.h"
#include "globals.h"
```

proto.h:

```
void PrintPid();
```

globals.h:

```
int X_ServerPid = 14;
```

Two Solutions

for initialized globals:

globals.h:

```
#ifdef _MAIN
   int X_ServerPid = 14;
#else
   extern X_ServerPid;
#endif
```

iserver.c:

```
#define _MAIN
#include "include.h"
```

for uninitialized globals:

globals.h:

```
#ifdef _MAIN
   #define EXTERN
#else
   #define EXTERN extern
#endif

EXTERN X_ServerPid;
/* set in Init()*/
```

Miscellanea

gzip, **compress (p.111)**

- Usage: **gzip [filename]**: compress specified filename
  gunzip [filename]: uncompress specified filename

- Examples:

gzip file1	*creates file1.gz*
gunzip <file2.gz | more	*leaves file2.gz intact*
cat file3 | gzip > newFile.gz	*leaves file3 intact*

- **compress** behaves like **gzip**, using a different (less efficient) compression algorithm is used (resulting files have **.Z** extension).

- Similarly, **uncompress** behaves like **gunzip**

tar (5.11)

- Traditionally, tar (short for Tape ARchive) was used for backups to tape drives
- It's also useful to create archive files on disk.

- Example: creating an archive of a directory structure:

```
tar fcvp dir1.tar dir1
```

- Example: uncompressing and extracting a tar file:

```
gunzip < dir2.tar.gz | tar fxvp -
```

- Example: copying a directory structure:

```
tar fcvp - dir1 | ( cd newloc; tar fxvp - )
```

- Advantage over "**`cp -rp`**": preserves symbolic links

nice, **nohup**

- **nice** (csh built-in) sets the priority level of a command. The higher the priority number, the slower it will run.
- Usage: **nice [+ n | - n] command**
- Example:

  nice +20 emacs &

  nice -20 importantJob *only root can give negative value*

- **nohup** (csh built-in) makes a process immune to hangup conditions
- Usage: **nohup command**
- Example:

  nohup bigJob &

- in **~/.logout**: **/usr/bin/kill -HUP -1 >& /dev/null**

Named pipes: **mknod( )**

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int main() {
   unlink( “namedPipe” );
   mknod(  “namedPipe”, S_IFIFO, 0 );
   chmod(  “namedPipe”, 0600 );
   if( fork() == 0 ) {
      int fd = open( “namedPipe”, O_WRONLY );
      dup2( fd, fileno(stdout) ); close( fd );
      execlp( "ruptime", "ruptime", (char *) 0 );
   } else {
      int fd = open( “namedPipe”, O_RDONLY );
      dup2( fd, fileno(stdin) ); close( fd );
      execlp( "sort", "sort", "-r", (char *) 0 );
   }
}
```

vfork()

- The typical **fork()**/**exec()** sequence is inefficient because **fork()** creates a copy of the data, heap, and stack area of the original process, which is then immediately discarded when **exec()** is called.
- **vfork()** is intended to create a new process when the purpose of the new process is to **exec()** a new program. **vfork()** has the same calling sequence and the same return values as **fork()**.
- **vfork()** creates the new process, just like **fork()**, without fully copying the address space of the parent into the child, since the child won't reference that address space -- the child just calls **exec()** right after the **vfork()**.
- Another difference between **vfork()** and **fork()** is that **vfork()** guarantees that the child runs first, until the child calls **exec()** or **exit()**.

`system()` (11.7)

- It is sometimes convenient to execute a command string from within a program.
- For example, to put a time and date stamp into a certain file, one could:
  - use **`time()`**, and **`ctime()`** to get and format the time, then open a file for writing and write the resulting string.
  - use **`system( "date > file" );`** (*much simpler*)
- **`system()`** is typically implemented by calling **`fork()`**, **`exec()`**, and **`waitpid()`**

lint

- **lint** is a useful utility that checks programs more thoroughly that **gcc** or other compilers
- Usage:

```
lint file1 [file2] ...
```

```
% cat main.c

#include <stdio.h>
void main()
{
   int i;
   printf("Hello\n");
}
```

```
% lint main.c

variable unused in function:
     (5) i in main

function returns value
which is always ignored:
     printf
```