CSC 4304 - Systems Programming
Fall 2010

Lecture - VIII
Process Control

Tevfik Koşar

Louisiana State University

September 16th, 2010

Environment Variables

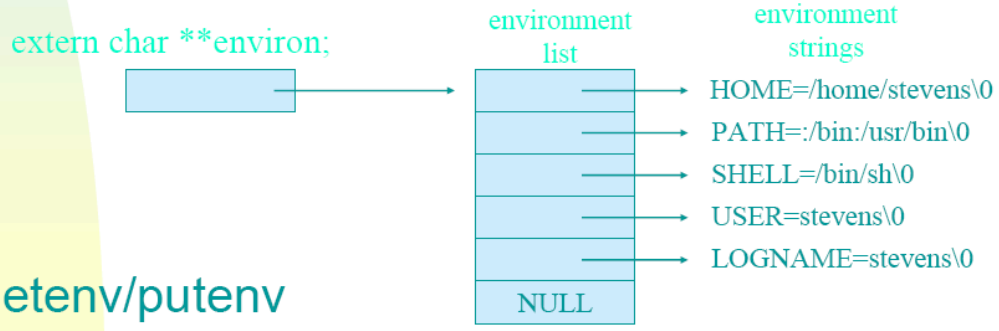
```
$ env
HOSTNAME=classes
TERM=xterm-color
USER=cs4304_kos
HOSTTYPE=x86_64
PATH=/usr/local/bin:/usr/bin:/opt/gnome/bin:/usr/lib/
mit/sbin:./
CPU=x86_64
PWD=/classes/cs4304/cs4304_kos
LANG=en_US.UTF-8
SHELL=/bin/bash
HOME=/classes/cs4304/cs4304_kos
MACHTYPE=x86_64-suse-linux
LOGNAME=cs4304_kos
...
```

Updating the Environment

```
$ course=csc4304
$ export course
$ env | grep course
course=csc4304

or

$export course="systems programming"
$ env | grep course
course=systems programming
```

How is Environment Implemented?

- Environment Variables
  - int main(int agrc, char **argv, char **envp);



extern char **environ;

environment list

environment strings

HOME=/home/stevens\0

PATH=:/bin:/usr/bin\0

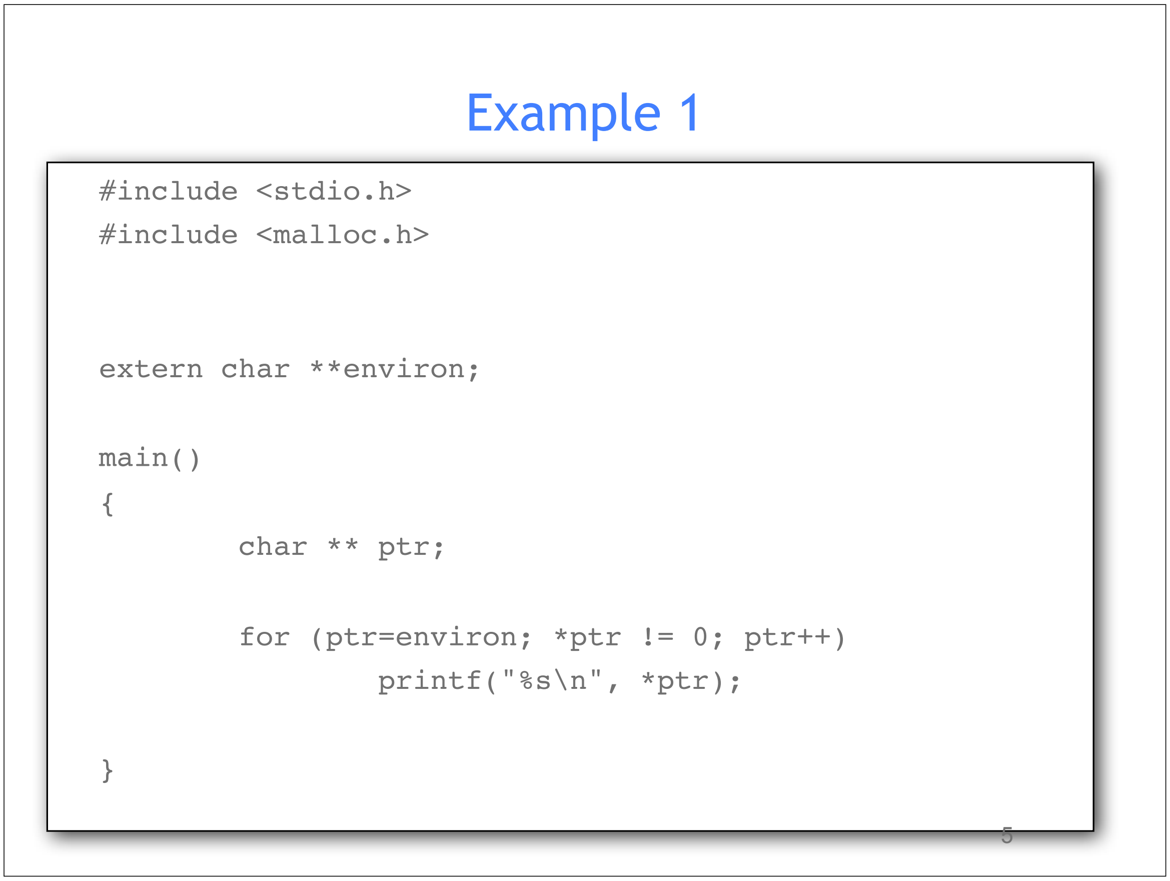
SHELL=/bin/sh\0

USER=stevens\0

LOGNAME=stevens\0

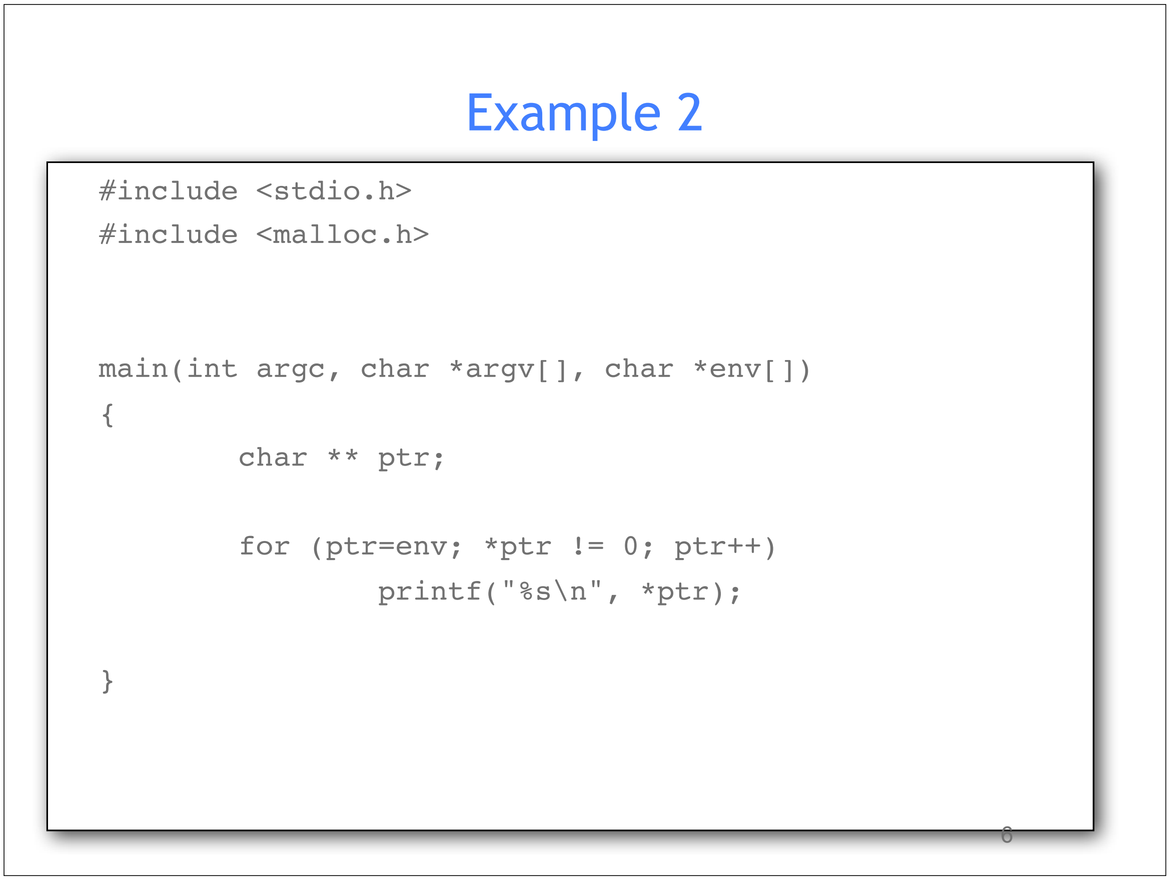
NULL

- getenv/putenv



Example 1

```
#include <stdio.h>
#include <malloc.h>


extern char **environ;

main()
{
        char ** ptr;

        for (ptr=environ; *ptr != 0; ptr++)
                printf("%s\n", *ptr);

}
```



Example 2

```
#include <stdio.h>
#include <malloc.h>


main(int argc, char *argv[], char *env[])
{
        char ** ptr;

        for (ptr=env; *ptr != 0; ptr++)
                printf("%s\n", *ptr);

}
```

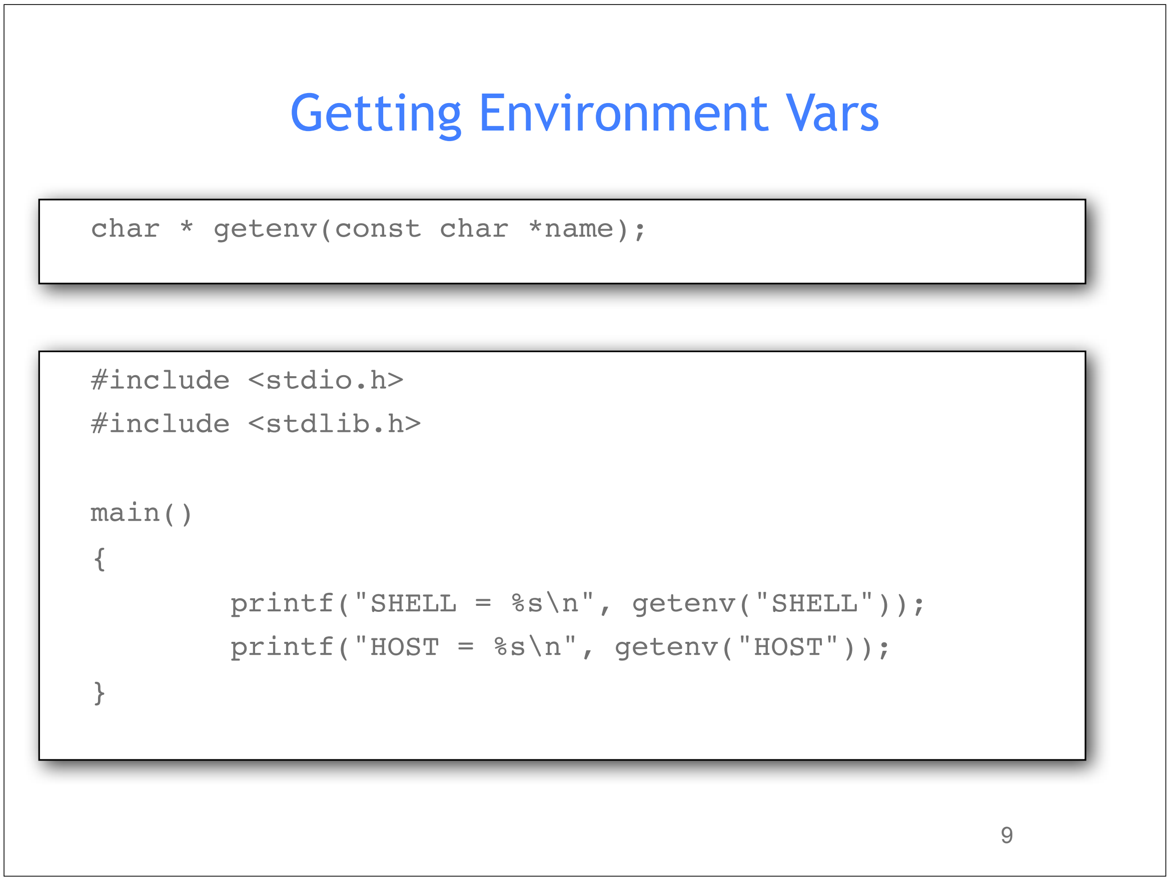
system function

```
int system(const char *command);
```

- used to execute command strings
- e.g. system(“date > file”);
- implemented using fork(), exec(), and waitpid()

Example 3

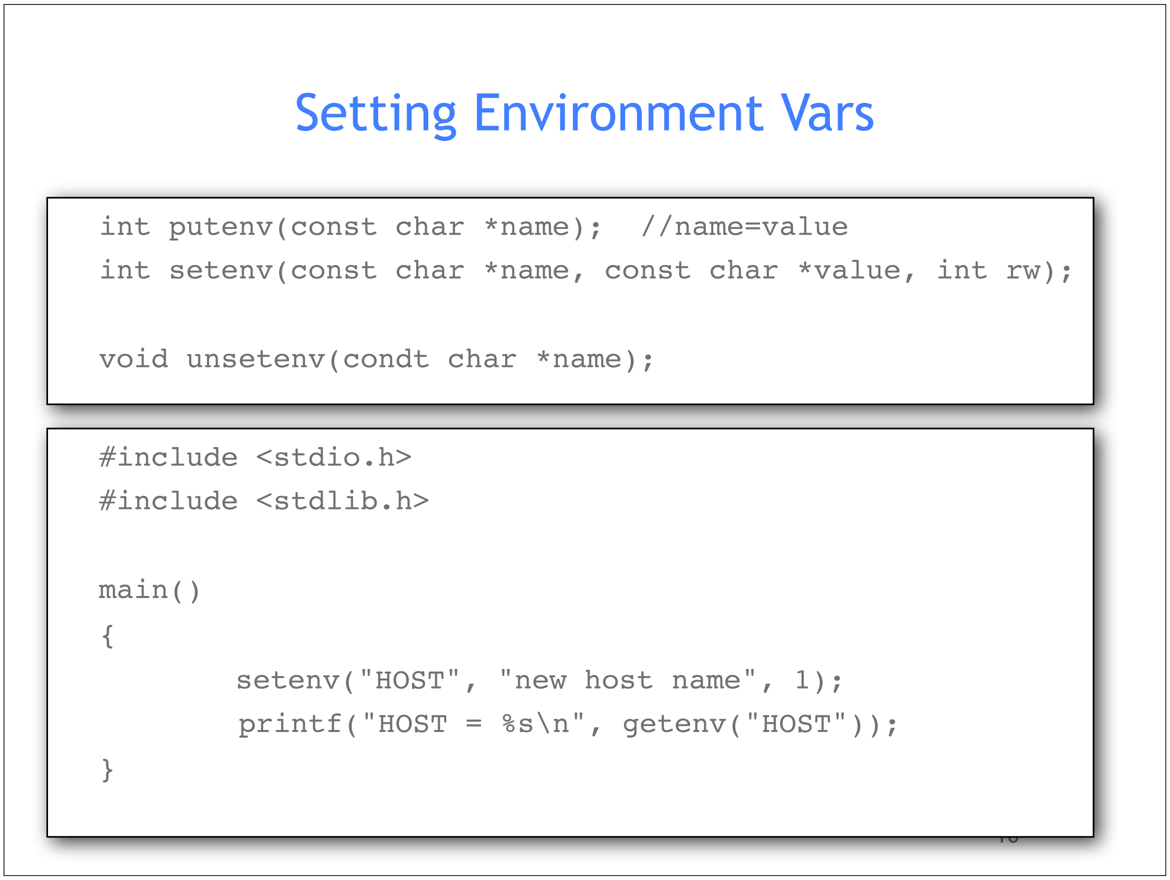
```
#include <stdio.h>
#include <unistd.h>
extern char **environ;

main()
{
        char    *newenv[5];
        printf("The current environment is..\n");
        system("env");

        printf("***** Now Replacing Environment...\n"); getchar();
        newenv[0] = "HOME=/on/the/range";
        newenv[1] = "LOGNAME=nobody";
        newenv[2] = "PATH=.:/bin:/usr/bin";
        newenv[3] = "DAY=Wednesday";
        newenv[4] = 0 ;
        environ = newenv;
        execlp("env", "env", NULL);
}
```



Getting Environment Vars

```
char * getenv(const char *name);
```

```
#include <stdio.h>
#include <stdlib.h>

main()
{
        printf("SHELL = %s\n", getenv("SHELL"));
        printf("HOST = %s\n", getenv("HOST"));
}
```



Setting Environment Vars

```
int putenv(const char *name);   //name=value
int setenv(const char *name, const char *value, int rw);

void unsetenv(condt char *name);
```

```
#include <stdio.h>
#include <stdlib.h>

main()
{
        setenv("HOST", "new host name", 1);
        printf("HOST = %s\n", getenv("HOST"));
}
```

vfork function

```
pid_t vfork(void);
```

- Similar to fork, but:
  - child shares all memory with parent
  - parent is suspended until the child makes an **exit** or **exec** call

fork example

```
main()
{
        int     ret, glob=10;


        printf("glob before fork: %d\n", glob);
        ret = fork();

        if (ret == 0) {
                glob++;
                printf("child: glob after fork: %d\n", glob) ;
                exit(0);
        }

        if (ret > 0) {

                if (waitpid(ret, NULL, 0) != ret) printf("Wait error!\n");
                printf("parent: glob after fork: %d\n", glob) ;
        }
```

vfork example

```
main()
{
        int     ret, glob=10;


        printf("glob before fork: %d\n", glob);
        ret = vfork();

        if (ret == 0) {
                glob++;
                printf("child: glob after fork: %d\n", glob) ;
                exit(0);
        }

        if (ret > 0) {

                //if (waitpid(ret, NULL, 0) != ret) printf("Wait error!\n");
                printf("parent: glob after fork: %d\n", glob) ;
        }
```

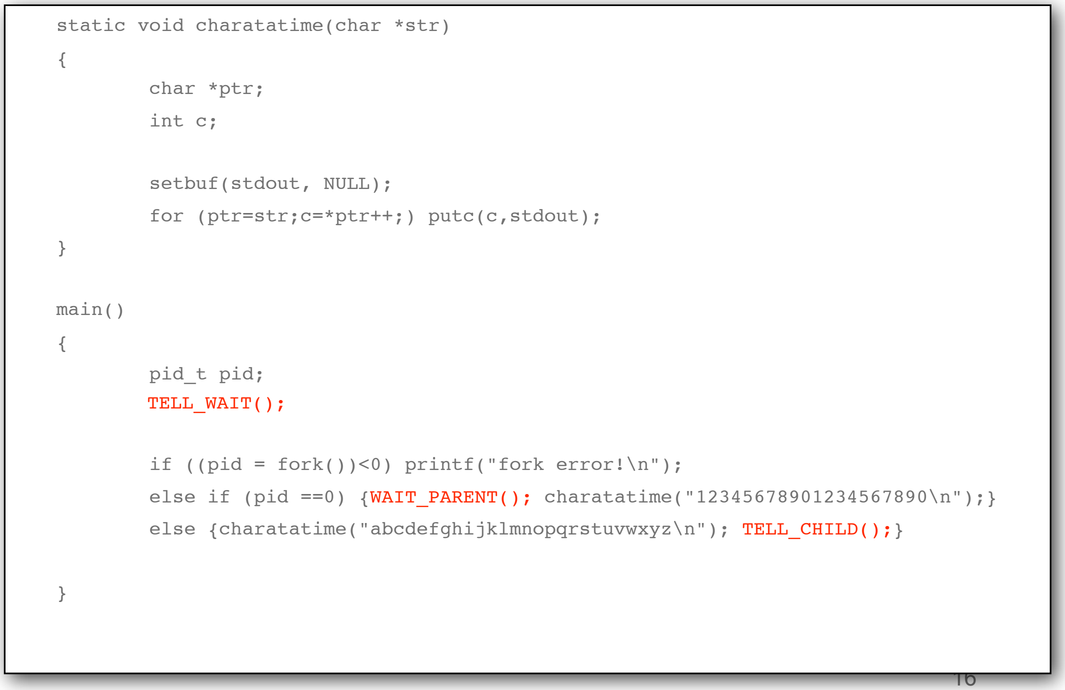
Race Conditions

```
static void charatatime(char *str)
{
        char *ptr;
        int c;

        setbuf(stdout, NULL);
        for (ptr=str;c=*ptr++;) putc(c,stdout);
}

main()
{
        pid_t pid;

        if ((pid = fork())<0) printf("fork error!\n");
        else if (pid ==0) charatatime("12345678901234567890\n");
        else charatatime("abcdefghijklmnopqrstuvwxyz\n");


}
```

Output

```
$ fork3
12345678901234567890
abcdefghijklmnopqrstuvwxyz

$ fork3
12a3bc4d5e6f78901g23hi4567jk890
lmnopqrstuvwxyz
```

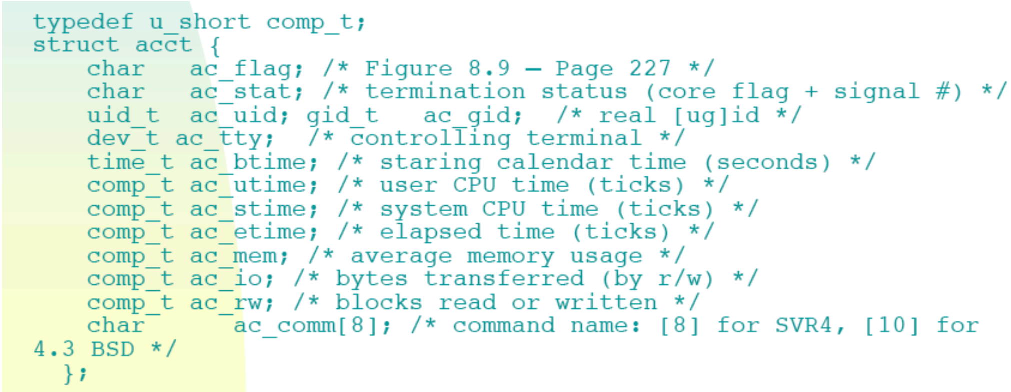
Avoid Race Conditions



```
static void charatatime(char *str)
{
        char *ptr;
        int c;

        setbuf(stdout, NULL);
        for (ptr=str;c=*ptr++;) putc(c,stdout);
}

main()
{
        pid_t pid;
        TELL_WAIT();

        if ((pid = fork())<0) printf("fork error!\n");
        else if (pid ==0) {WAIT_PARENT(); charatatime("12345678901234567890\n");}
        else {charatatime("abcdefghijklmnopqrstuvwxyz\n"); TELL_CHILD();}

}
```

Process Accounting

- Kernel writes an accounting record each time a process terminates
- acct struct defined in <sys/acct.h>



```
typedef u_short comp_t;
struct acct {
    char   ac_flag; /* Figure 8.9 — Page 227 */
    char   ac_stat; /* termination status (core flag + signal #) */
    uid_t  ac_uid; gid_t   ac_gid;  /* real [ug]id */
    dev_t ac_tty;   /* controlling terminal */
    time_t ac_btime; /* staring calendar time (seconds) */
    comp_t ac_utime; /* user CPU time (ticks) */
    comp_t ac_stime; /* system CPU time (ticks) */
    comp_t ac_etime; /* elapsed time (ticks) */
    comp_t ac_mem; /* average memory usage */
    comp_t ac_io; /* bytes transferred (by r/w) */
    comp_t ac_rw; /* blocks read or written */
    char        ac_comm[8]; /* command name: [8] for SVR4, [10] for
4.3 BSD */
  };
```

Process Accounting

- Data required for accounting record is kept in the process table
- Initialized when a new process is created
  - (e.g. after fork)
- Written into the accounting file (binary) when the process terminates
  - in the order of termination
- No records for
  - crashed processes
  - abnormal terminated processes

Pipes

- one-way data channel in the kernel
- has a reading end and a writing end

- e.g. **who | sort** or **ps | grep ssh**

Process Communication via Pipes

```
int pipe(int filedes[2]);
```

- pipe creates a pair of file descriptors, pointing to a pipe inode, and places them in the array pointed to by filedes. filedes[0] is for reading filedes[1] is for writing

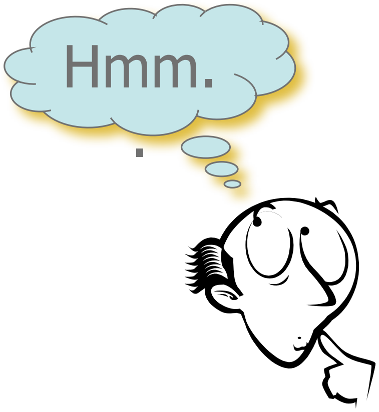
```
main(int ac, char *av[])
{
        int     thepipe[2], newfd, pid;*/
        if ( ac != 3 ){fprintf(stderr, "usage: pipe cmd1 cmd2\n");exit(1);}

        if (pipe(thepipe) == -1){perror( "cannot create pipe"); exit(1); }

        if ((pid = fork()) == -1){fprintf(stderr,"cannot fork\n"); exit(1);}

         /*
         *      parent will read from reading end of pipe
         */

        if ( pid > 0 ){                 /* the child will be av[2]      */
                close(thepipe[1]);      /* close writing end            */
                close(0);               /* will read from pipe          */
                newfd=dup(thepipe[0]);  /* so duplicate the reading end */
                if ( newfd != 0 ){      /* if not the new stdin..       */
                        fprintf(stderr,"Dupe failed on reading end\n");
                        exit(1);
                }
                close(thepipe[0]);      /* stdin is duped, close pipe   */
                execlp( av[2], av[2], NULL);
                exit(1);                /* oops                         */
        }
```

```
        /*
         *      child will write into writing end of pipe
         */
        close(thepipe[0]);      /* close reading end            */
        close(1);               /* will write into pipe         */
        newfd=dup(thepipe[1]);  /* so duplicate writing end     */
        if ( newfd != 1 ){      /* if not the new stdout..      */
                fprintf(stderr,"Dupe failed on writing end\n");
                exit(1);
        }
        close(thepipe[1]);      /* stdout is duped, close pipe  */
        execlp( av[1], av[1], NULL);
        exit(1);                /* oops                         */
}
```

Summary

- Process Control
  - Environment
  - Process Accounting
  - Pipes

- Next Class: Signals



Acknowledgments

- Advanced Programming in the Unix Environment by R. Stevens
- The C Programming Language by B. Kernighan and D. Ritchie
- Understanding Unix/Linux Programming by B. Molay
- Lecture notes from B. Molay (Harvard), T. Kuo (UT-Austin), G. Pierre (Vrije), M. Matthews (SC), and B. Knicki (WPI).