CSE 421/521 - Operating Systems
Fall 2012 Recitations

Recitation - II

Unix Processes

Prof. Tevfik Kosar

Presented by Sonali Batra

University at Buffalo
September 2012

In Today's Class

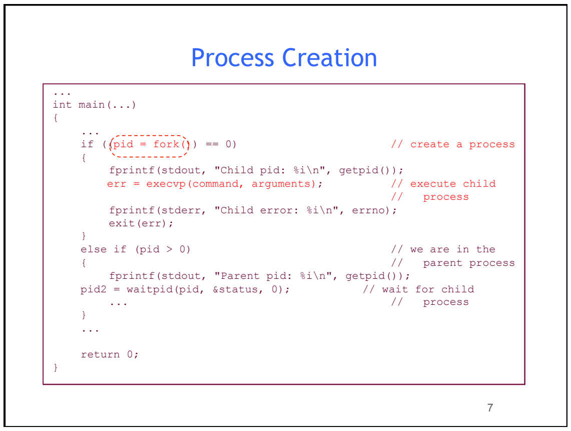
- Unix Process Environment
  - Creation & Termination of Processes
  - Exec() & Fork()
  - ps -- get process info
  - Shell & its implementation
  - Environment Variables
  - Process Control
  - Pipes

```
$ ps
PID   TTY      TIME     CMD
18684 pts/4    00:00:00 bash
18705 pts/4    00:00:00 ps
```

```
$ ps a
  PID TTY      STAT   TIME COMMAND
 6702 tty7     Ss+   15:10 /usr/X11R6/bin/X :0 -audit 0
 7024 tty1     Ss+    0:00 /sbin/mingetty --noclear tty1
 7025 tty2     Ss+    0:00 /sbin/mingetty tty2
 7026 tty3     Ss+    0:00 /sbin/mingetty tty3
 7027 tty4     Ss+    0:00 /sbin/mingetty tty4
 7028 tty5     Ss+    0:00 /sbin/mingetty tty5
 7029 tty6     Ss+    0:00 /sbin/mingetty tty6
17166 pts/6    Ss     0:00 -bash
17191 pts/6    S+     0:00 pico program3.cc
17484 pts/5    Ss+    0:00 -bash
17555 pts/7    Ss+    0:00 -bash
17646 pts/8    Ss     0:00 -bash
17809 pts/10   Ss     0:00 -bash
17962 pts/8    S+     0:00 pico prog2.java
17977 pts/1    Ss     0:00 -bash
18014 pts/9    Ss+    0:00 -bash
18259 pts/10   T      0:00 a.out
18443 pts/2    Ss     0:00 -bash
18511 pts/1    S+     0:00 pico program3.cc
18684 pts/4    Ss     0:00 -bash
```

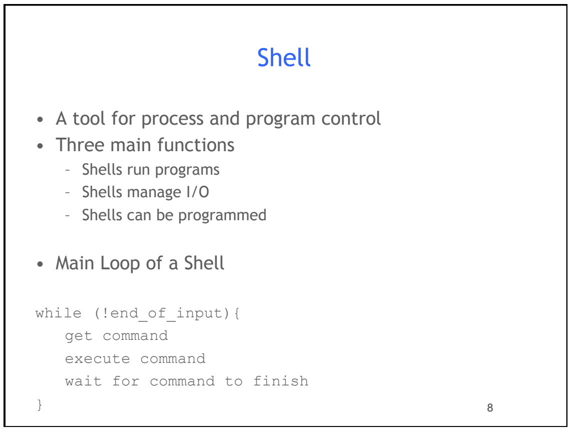
```
$ ps la
 F   UID   PID  PPID PRI  NI    VSZ   RSS WCHAN  STAT TTY        TIME COMMAND
4     0  6702  6701  15   0  25416  7204 -      Ss+  tty7      15:10 /usr/X11R6/bin/X :0 -
audit 0 -auth /var/lib/g
4     0  7024     1  17   0   3008     4 -      Ss+  tty1       0:00 /sbin/mingetty --noclear
tty1
4     0  7025     1  16   0   3008     4 -      Ss+  tty2       0:00 /sbin/mingetty tty2
4     0  7026     1  16   0   3012     4 -      Ss+  tty3       0:00 /sbin/mingetty tty3
4     0  7027     1  17   0   3008     4 -      Ss+  tty4       0:00 /sbin/mingetty tty4
4     0  7028     1  17   0   3008     4 -      Ss+  tty5       0:00 /sbin/mingetty tty5
4     0  7029     1  17   0   3008     4 -      Ss+  tty6       0:00 /sbin/mingetty tty6
0  2317 17166 17165  15   0   9916  2300 wait   Ss   pts/6      0:00 -bash
0  2317 17191 17166  16   0   8688  1264 -      S+   pts/6      0:00 pico program3.cc
0  2238 17484 17483  16   0   9916  2300 -      Ss+  pts/5      0:00 -bash
0  2611 17555 17554  15   0   9912  2292 -      Ss+  pts/7      0:00 -bash
0  2631 17646 17644  16   0   9912  2300 wait   Ss   pts/8      0:00 -bash
0  2211 17809 17808  15   0   9916  2324 wait   Ss   pts/10     0:00 -bash
0  2631 17962 17646  16   0   8688  1340 -      S+   pts/8      0:00 pico prog2.java
0  2320 17977 17976  16   0   9912  2304 wait   Ss   pts/1      0:00 -bash
```

```
$ ps -ax
 PID TTY      STAT   TIME COMMAND
   1 ?        S      0:02 init [5]
   2 ?        S      0:00 [migration/0]
   3 ?        SN     0:00 [ksoftirqd/0]
   4 ?        S      0:00 [migration/1]
   5 ?        SN     0:01 [ksoftirqd/1]
   6 ?        S      0:00 [migration/2]
   7 ?        SN     0:16 [ksoftirqd/2]
   8 ?        S      0:00 [migration/3]
   9 ?        SN     0:16 [ksoftirqd/3]
  10 ?        S<     0:00 [events/0]
  11 ?        S<     0:00 [events/1]
  12 ?        S<     0:00 [events/2]
  13 ?        S<     0:00 [events/3]
  14 ?        S<     0:00 [khelper]
  15 ?        S<     0:00 [kthread]
 653 ?        S<     0:00 [kacpid]
 994 ?        S<     0:00 [kblockd/0]
 995 ?        S<     0:00 [kblockd/1]
 996 ?        S<     0:01 [kblockd/2]
 997 ?        S<     0:00 [kblockd/3]
1062 ?        S      0:24 [kswapd0]
```



Process Creation

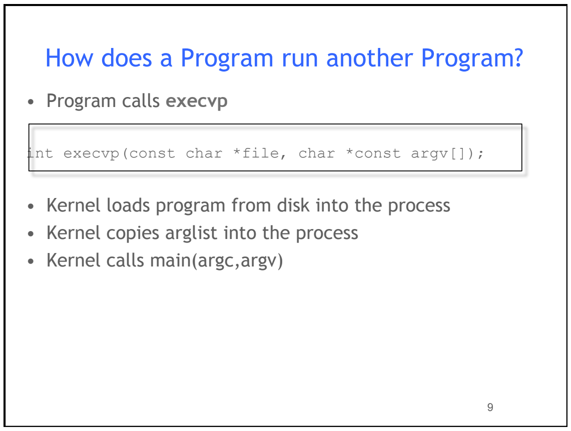
```
...
int main(...)
{
    ...
    if ((pid = fork()) == 0)                    // create a process
    {
        fprintf(stdout, "Child pid: %i\n", getpid());
        err = execvp(command, arguments);       // execute child
                                                //   process
        fprintf(stderr, "Child error: %i\n", errno);
        exit(err);
    }
    else if (pid > 0)                           // we are in the
    {                                           //   parent process
        fprintf(stdout, "Parent pid: %i\n", getpid());
    pid2 = waitpid(pid, &status, 0);        // wait for child
        ...                                     //   process
    }
    ...

    return 0;
}
```



Shell

- A tool for process and program control
- Three main functions
  - Shells run programs
  - Shells manage I/O
  - Shells can be programmed
- Main Loop of a Shell

```
while (!end_of_input){
   get command
   execute command
   wait for command to finish
}
```

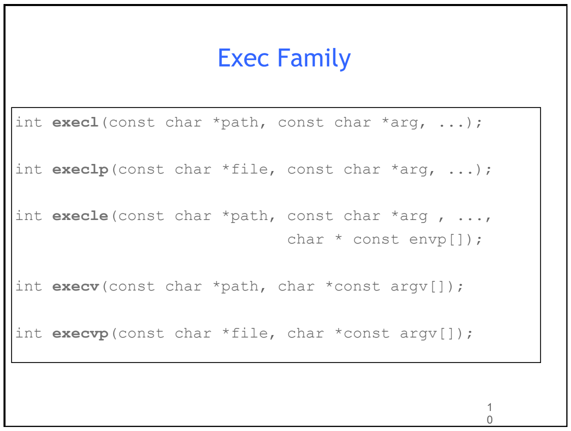


How does a Program run another Program?

- Program calls **execvp**

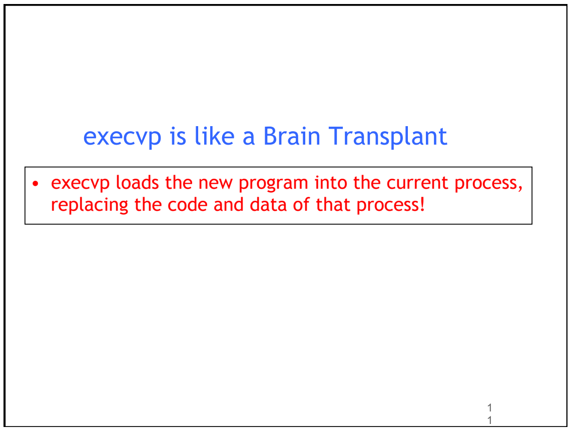
```
int execvp(const char *file, char *const argv[]);
```

- Kernel loads program from disk into the process
- Kernel copies arglist into the process
- Kernel calls main(argc,argv)



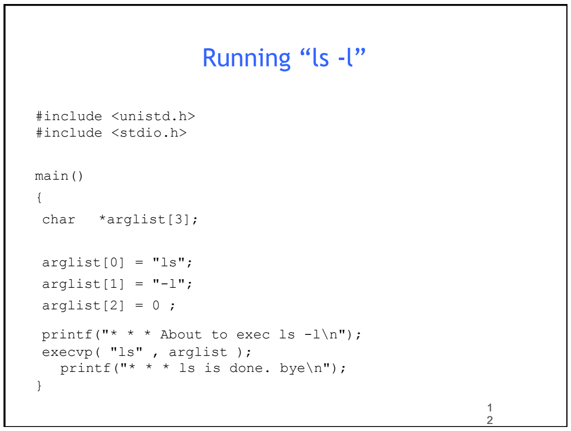
Exec Family

```
int execl(const char *path, const char *arg, ...);

int execlp(const char *file, const char *arg, ...);

int execle(const char *path, const char *arg , ...,
                             char * const envp[]);

int execv(const char *path, char *const argv[]);

int execvp(const char *file, char *const argv[]);
```



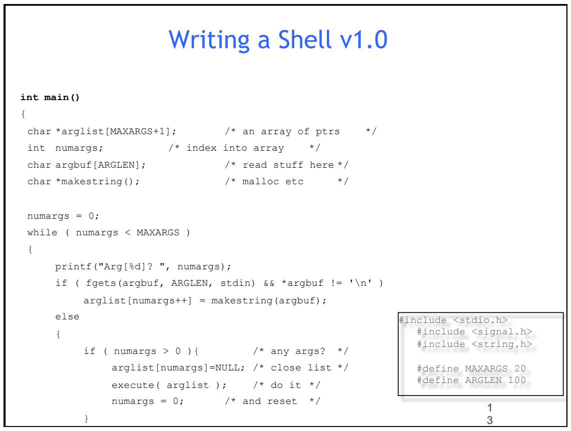
execvp is like a Brain Transplant

- execvp loads the new program into the current process, replacing the code and data of that process!



Running “ls -l”

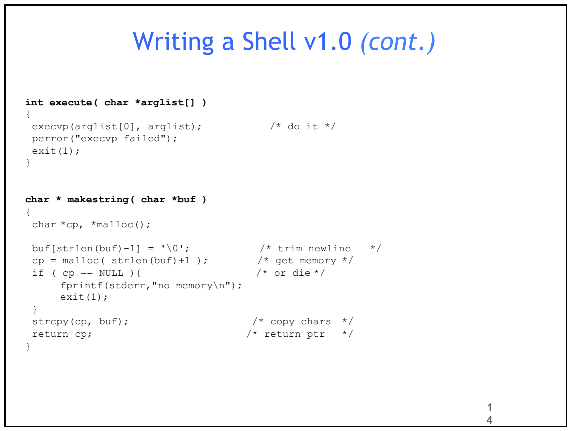
```
#include <unistd.h>
#include <stdio.h>

main()
{
 char   *arglist[3];

 arglist[0] = "ls";
 arglist[1] = "-l";
 arglist[2] = 0 ;

 printf("* * * About to exec ls -l\n");
 execvp( "ls" , arglist );
   printf("* * * ls is done. bye\n");
}
```



Writing a Shell v1.0

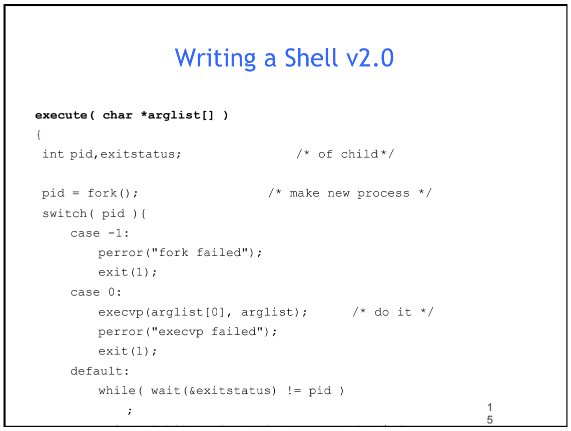
```
int main()
{
 char *arglist[MAXARGS+1];        /* an array of ptrs    */
 int  numargs;            /* index into array    */
 char argbuf[ARGLEN];             /* read stuff here */
 char *makestring();              /* malloc etc      */

 numargs = 0;
 while ( numargs < MAXARGS )
 {
     printf("Arg[%d]? ", numargs);
     if ( fgets(argbuf, ARGLEN, stdin) && *argbuf != '\n' )
         arglist[numargs++] = makestring(argbuf);
     else
     {
         if ( numargs > 0 ){         /* any args?  */
             arglist[numargs]=NULL; /* close list */
             execute( arglist );    /* do it */
             numargs = 0;       /* and reset  */
         }
```

```
#include <stdio.h>
   #include <signal.h>
   #include <string.h>

   #define MAXARGS 20
   #define ARGLEN 100
```



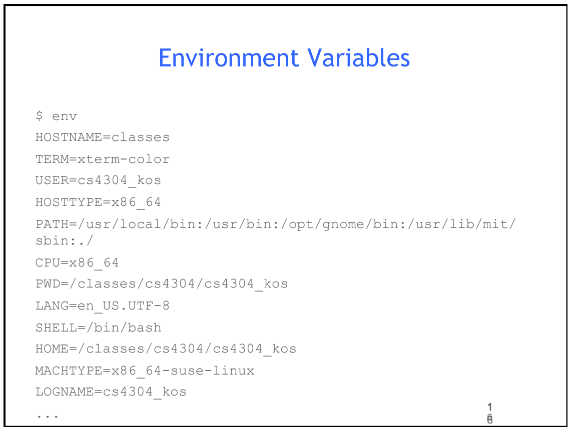
Writing a Shell v1.0 *(cont.)*

```
int execute( char *arglist[] )
{
 execvp(arglist[0], arglist);           /* do it */
 perror("execvp failed");
 exit(1);
}


char * makestring( char *buf )
{
 char *cp, *malloc();

 buf[strlen(buf)-1] = '\0';            /* trim newline   */
 cp = malloc( strlen(buf)+1 );         /* get memory */
 if ( cp == NULL ){                    /* or die */
      fprintf(stderr,"no memory\n");
      exit(1);
 }
 strcpy(cp, buf);                      /* copy chars  */
 return cp;                           /* return ptr   */
}
```



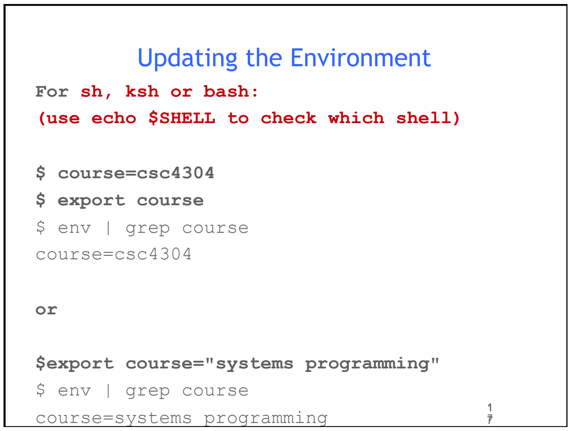
Writing a Shell v2.0

```
execute( char *arglist[] )
{
 int pid,exitstatus;                /* of child*/

 pid = fork();                   /* make new process */
 switch( pid ){
     case -1:
         perror("fork failed");
         exit(1);
     case 0:
         execvp(arglist[0], arglist);      /* do it */
         perror("execvp failed");
         exit(1);
     default:
         while( wait(&exitstatus) != pid )
             ;
```



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
...
```



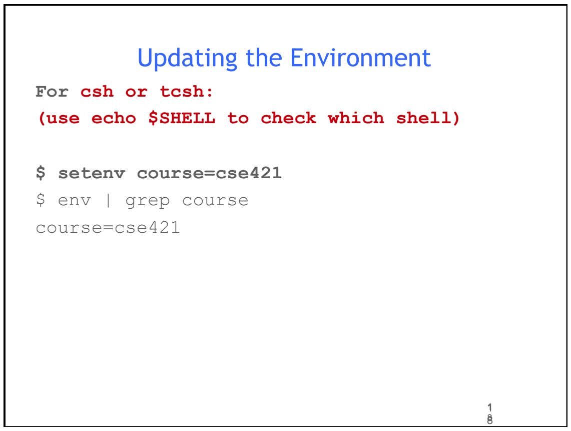
Updating the Environment

For sh, ksh or bash:
(use echo $SHELL to check which shell)

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

or

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



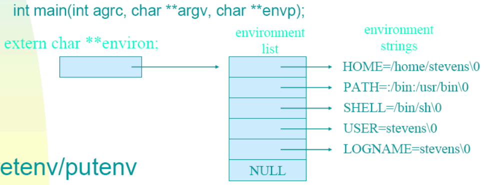
Updating the Environment

For csh or tcsh:
(use echo $SHELL to check which shell)

```
$ setenv course=cse421
$ env | grep course
course=cse421
```

How is Environment Implemented?

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



- 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);
}
```

Updating the Environment

For sh, ksh or bash:
(use echo $SHELL to check which shell)

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


or


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

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"));}
        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                         */
}
```

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).