In Today’s Class

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

- An operating system executes a variety of programs:
  - Batch system - jobs
  - Time-shared systems - user programs or tasks
- Process - a program in execution; process execution must progress in sequential fashion
- A process includes:
  - program counter
  - stack: temporary data

Process in Memory

Process Control Block (PCB)

Information associated with each process
- Process state
  - (running, waiting..)
- Program counter
- CPU registers
- CPU scheduling information
  - (i.e. process priority)
- Memory-management information
  - (i.e. page & segment tables)
- Accounting information
- I/O status information
Process State

- As a process executes, it changes state
  - new: The process is being created
  - ready: The process is waiting to be assigned to a process
  - running: Instructions are being executed
  - waiting: The process is waiting for some event to occur
  - terminated: The process has finished execution

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

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

### $ ps la

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

<table>
<thead>
<tr>
<th>UID</th>
<th>PID</th>
<th>PPID</th>
<th>C</th>
<th>STIME</th>
<th>TTY</th>
<th>TIME</th>
<th>CMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>cs1253as</td>
<td>17191</td>
<td>17166</td>
<td>0</td>
<td>10:13</td>
<td>pts/6</td>
<td>00:00:00</td>
<td>pico program3.cc</td>
</tr>
<tr>
<td>cs135032</td>
<td>17962</td>
<td>17646</td>
<td>0</td>
<td>10:57</td>
<td>pts/8</td>
<td>00:00:00</td>
<td>pico prog2.java</td>
</tr>
<tr>
<td>cs1253av</td>
<td>18511</td>
<td>17977</td>
<td>0</td>
<td>11:20</td>
<td>pts/1</td>
<td>00:00:00</td>
<td>pico program3.cc</td>
</tr>
<tr>
<td>15059</td>
<td>19109</td>
<td>18684</td>
<td>0</td>
<td>11:44</td>
<td>pts/4</td>
<td>00:00:00</td>
<td>ps -fa</td>
</tr>
</tbody>
</table>

### $ ps -ax

<table>
<thead>
<tr>
<th>PID</th>
<th>TTY</th>
<th>STAT</th>
<th>TIME</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>?</td>
<td>S</td>
<td>0:02</td>
<td>init [5]</td>
</tr>
<tr>
<td>2</td>
<td>?</td>
<td>S</td>
<td>0:00</td>
<td>[migration/0]</td>
</tr>
<tr>
<td>3</td>
<td>?</td>
<td>SN</td>
<td>0:00</td>
<td>[ksoftirqd/0]</td>
</tr>
<tr>
<td>4</td>
<td>?</td>
<td>S</td>
<td>0:00</td>
<td>[migration/1]</td>
</tr>
<tr>
<td>5</td>
<td>?</td>
<td>SN</td>
<td>0:01</td>
<td>[ksoftirqd/1]</td>
</tr>
<tr>
<td>6</td>
<td>?</td>
<td>S</td>
<td>0:00</td>
<td>[migration/2]</td>
</tr>
<tr>
<td>7</td>
<td>?</td>
<td>SN</td>
<td>0:16</td>
<td>[ksoftirqd/2]</td>
</tr>
<tr>
<td>8</td>
<td>?</td>
<td>S</td>
<td>0:00</td>
<td>[migration/3]</td>
</tr>
<tr>
<td>9</td>
<td>?</td>
<td>SN</td>
<td>0:16</td>
<td>[ksoftirqd/3]</td>
</tr>
<tr>
<td>10</td>
<td>?</td>
<td>S&lt;</td>
<td>0:00</td>
<td>[events/0]</td>
</tr>
<tr>
<td>11</td>
<td>?</td>
<td>S&lt;</td>
<td>0:00</td>
<td>[events/1]</td>
</tr>
<tr>
<td>12</td>
<td>?</td>
<td>S&lt;</td>
<td>0:00</td>
<td>[events/2]</td>
</tr>
<tr>
<td>13</td>
<td>?</td>
<td>S&lt;</td>
<td>0:00</td>
<td>[events/3]</td>
</tr>
<tr>
<td>14</td>
<td>?</td>
<td>S&lt;</td>
<td>0:00</td>
<td>[khelper]</td>
</tr>
<tr>
<td>15</td>
<td>?</td>
<td>S&lt;</td>
<td>0:00</td>
<td>[kthread]</td>
</tr>
<tr>
<td>653</td>
<td>?</td>
<td>S&lt;</td>
<td>0:00</td>
<td>[kacpid]</td>
</tr>
<tr>
<td>994</td>
<td>?</td>
<td>S&lt;</td>
<td>0:00</td>
<td>[kblockd/0]</td>
</tr>
<tr>
<td>995</td>
<td>?</td>
<td>S&lt;</td>
<td>0:00</td>
<td>[kblockd/1]</td>
</tr>
<tr>
<td>996</td>
<td>?</td>
<td>S&lt;</td>
<td>0:01</td>
<td>[kblockd/2]</td>
</tr>
<tr>
<td>997</td>
<td>?</td>
<td>S&lt;</td>
<td>0:00</td>
<td>[kblockd/3]</td>
</tr>
</tbody>
</table>
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

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

How does a Program run another Program?

- Program calls `execvp`

```c
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

```c
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[]);
```

Running “ls -l”

```c
#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");
}```
execvp is like a Brain Transplant

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

```c
#include <stdio.h>   #include <signal.h>   #include <string.h>
#define MAXARGS 20   #define ARGLEN 100

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 */
      }
    }
  }
  return 0;
}
```
Writing a Shell v1.0 (cont.)

```c
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 */
}
```

How to Create a New Process?

- Parent process create children processes, which, in turn create other processes, forming a tree of processes
- Resource sharing
  - Parent and children share all resources
  - Children share subset of parent’s resources
  - Parent and child share no resources
- Execution
  - Parent and children execute concurrently
  - Parent waits until children terminate
Process Creation (Cont.)

- Address space
  - Child duplicate of parent
  - Child has a program loaded into it
- UNIX examples
  - `fork` system call creates new process
  - `exec` system call used after a `fork` to replace the process’ memory space with a new program

How fork works?

```c
pid_t fork(void);
```

- Allocates a new chunk of memory and data structures
- Copies the original process into the new process
- Adds the new process to the set of running processes
- Returns control back to both processes
Fork Implementation

```c
int main()
{
    Pid_t pid;
    /* fork another process */
    pid = fork();
    if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork Failed");
        exit(-1);
    }
    else if (pid == 0) { /* child process */
        execlp("/bin/ls", "ls", NULL);
    }
    else { /* parent process */
        /* parent will wait for the child to complete */
    }
}
```

Fork Example 1

```c
#include <stdio.h>

main()
{
    int ret_from_fork, mypid;
    mypid = getpid();    /* who am i? */
    printf("Before: my pid is %d\n", mypid); /* tell pid */
    ret_from_fork = fork();
    sleep(1);
    printf("After: my fork returns pid: %d, said %d\n", 
        ret_from_fork, getpid());
}
```
Fork Example 2

```c
#include <stdio.h>

main()
{
    fork();
    fork();
    fork();
    printf("my pid is %d\n", getpid());
}
```

How many lines of output will this produce?

Writing a Shell v2.0

```c
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 )
            {
                printf("child exited with status %d,%d\n", exitstatus>>8, exitstatus&0377);
            }
    }
}
```
Exercise

Improve the Shell v2.0 by:

- Allow the user to type all the arguments on one line
- Allow the user to quit by typing exit

Process Termination

- Five ways to terminate:
  - Normal Termination
    - Return from main()
    - Call exit()
    - Call _exit()
  - Abnormal termination
    - Call abort()
    - Be terminated by a signal
Process Start and Termination

Summary

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

- Next Class: Process Control
- Try “fork” and “shell” examples
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).