News

• Exam 1
  – Average: 70
  – Hand back at the end of class

• tshlab will be released tomorrow
  – Due next Thursday
  – Lots of tricky details, so start early!
News

• Exam 1
  – Average: 70
  – Hand back at the end of class

• tshlab will be released tomorrow
  – Due next Thursday
  – Lots of tricky details, so start early!

• The most interesting and important part of this class starts now
Today

- Processes
- Signals
- tshlab
Processes

• What is a program?
  – No state, just instructions
Processes

• What is a process?
  – An instance of a program in execution
  – A Great Idea in Computer Science
    • Ubiquitous on multitasking systems
  – Fundamental abstraction provided by the OS
    • Single thread of execution (control flow)
    • Full, private memory space and registers
    • Various other state (files, etc.)
Processes

• Four basic process control functions
  – fork
  – exec
  – exit
  – wait

• Standard on all Unix systems
fork

- Creates a process
- More like mitosis than birth
- Parent and child are exactly alike
  - Except for return value
  - Equal, but private
    - Threads of execution
    - Registers (except %eax)
    - Memory
    - File descriptors
      - Files are shared
fork

**Code**

```c
int rc = fork();
printf("my rc is %d\n", rc)
```

**Output**

```
my rc is 0
my rc is 4233
/* or */
my rc is 4233
my rc is 0
```
exec

- **Replaces** process
  - No processes created
  - New process (mostly) unaware of old state

- **How programs are run**
  - Replace memory image with new program
  - Set up stack with arguments
  - Start execution at the entry point (main)

- **Actually a family of functions**
  - `man 3 exec`
int rc = fork();
if (!rc) {
    printf("child\n");
    execl("/bin/echo",
          "/bin/echo",
          "/bin/echo",
          "sup!", NULL);
    exit(EXIT_FAILURE);
}
printf("parent\n");
Processes

• exit
  – Terminates a process
  – OS frees resources used by the process
  – Tiny leftover data
    • Zombie state
    • Exit status for parent
    • Must be freed
Processes

• wait
  – Blocks until child process changes state
  – Reaps child if it terminated
    • Frees all remaining resources and gets exit status
  – Can be used for synchronization
  – Lots of details
    • man 2 wait
wait

Code

```c
int rc = fork();
if (!rc) {
    printf("child\n");
    execl("/bin/echo",
        "/bin/echo",
        "sup!", NULL);
    exit(EXIT_FAILURE);
}
printf("parent\n");
```

Output

```
child
sup!
parent
/* how can we ensure the same output every time? */
```
```c
int rc = fork();
if (!rc) {
    printf(“child\n”);
    execl(“/bin/echo”,
          “/bin/echo”,
          “sup!”, NULL);
    exit(EXIT_FAILURE);
}
int status;
waitpid(rc, &status, 0);
printf(“child status %d\n”,
       WEXITSTATUS(status));
```
Processes

• **States**
  – **Running**
    • Executing instructions on the CPU
    • Number bounded by the number of CPU cores
  – **Runnable**
    • Waiting to be running
  – **Blocked**
    • Waiting for an event
    • Not runnable
  – **Zombie**
    • Terminated but not yet reaped
Today

• Processes
• **Signals**
• tshlab
Signals

- Primitive form of interprocess communication
- Notify process of an event
- Asynchronous with normal execution
- Several types
  - man 7 signal
- Sent in various ways
  - ^C, ^Z, ^\n  - kill command
  - kill system call
Signals

• What to do when receiving a signal
  – Ignore
  – Catch and run signal handler
  – Terminate
  – `man sigaction`

• Blocking
  – `man sigprocmask`

• Can’t modify behavior of SIGKILL and SIGSTOP

• Signals don’t queue
Signals

- Signal handlers
  - Can be installed to run when a signal is received
  - Type is "void (*sa_handler)(int)"
  - Separate control flow in the same process
  - Resumes control flow on return
  - Signal handlers can be called anytime
Signals

• Interesting signals
  – SIGKILL – force kill a process
  – SIGINT – kill a process
  – SIGSTOP and SIGCONT – suspend and resume a process
  – SIGCHLD – child changed state
  – SIGSEGV – everyone knows this
Today

- Processes
- Signals
- tshlab
tshlab

• Write a not-so-basic shell
  – Fork and execute a new program
  – Wait for foreground jobs
  – Support background jobs
  – React to changes in child state
  – Input and output redirection
    • `ls > ls_out` and `wc < filename`

• Many different designs
  – Some much better than others

• Read the spec carefully and start early!
Thanks!