

System programming: process management and IPC



CS 2204

Class meeting 11



UNIX system programming

- Programming that uses special features of the UNIX system
- Programs make *system calls*
- Types of system calls
 - File I/O
 - Process management
 - Inter-process communication (IPC)
 - Signal handling



Processes in UNIX

- Process: basic unit of execution
 - Executing instance of a program
 - Has a process ID (PID)
 - Is in a hierarchy of processes (parents, children)
 - Has its own state/context/memory
- Shell commands dealing with processes: `ps`, `top`, `kill`, `nice`, ...



Process management

- System calls dealing with:
 - Process creation
 - Setting the program a process executes
 - Waiting for a process to terminate
 - Process termination
 - Sending signals to a process



Process creation

- `pid = fork()`
- Creates a new child process that is an exact copy of the current process
 - Same program running at same location
 - Same variable values
 - Same open files
- Only difference: child has new PID
- Returns 0 in the child process
- Returns child's PID in the parent process



Setting the program a process executes

- `exec` family of functions
- e.g. `execlp(executable_name, arg0, arg1, ...)` ;
- Replaces the current process with a new process image running the executable specified with the arguments listed
- New process retains old PID and any open files
- Other functions: `execl`, `execle`, `execv`, `execvp`, `execve`



Waiting for a process to terminate

- `pid = wait(&status)`
 - Suspends execution of the calling process until any child process terminates
 - Returns PID of terminating child
 - Puts exit status of child in status
- `pid = waitpid(pid, &status, options)`
 - Suspends execution of the calling process until a specific child terminates



Using fork/exec/wait together

```
pid = fork();  
if (pid == 0)  
    execl("./program", "program",  
          arg1, NULL);  
else  
    pid = wait(&status);  
continue execution
```



Process termination

- `exit(status)`
- Terminates the calling process
- Closes all open file descriptors
- Returns status to the parent process



Sending signals to a process

- `retval = kill(pid, signal)`
- Some common signals a user program might send:
 - SIGINT: interrupt (CTRL-c)
 - SIGKILL: kill
 - SIGUSR1, SIGUSR2: user-defined



Inter-process communication (IPC)

- Information passing between processes
- Two basic paradigms
 - Message passing: processes send information back and forth in messages/packets
 - Shared Memory: processes share a chunk of physical memory and read/write data there to share that information



IPC with pipes

- Example of message passing
- `int fds[2]; retval = pipe(fds);`
- Creates two file descriptors (a pipe is a file), the first for reading, and the second for writing
- How does another process connect to this pipe?



Basic pipe example

```
int fds[2]; char
    s[100];
retval = pipe(fds);
pid = fork();
if(pid == 0){
    read(fds[0], s, 100);
    printf("Read %s\n",
        s);
}
```

```
else
    write(fds[1],
        "hello", 6);
```

NOTES:

- data is written/read in order (FIFO)
- reads block until there's something to read
- writes block if the pipe is full
- closing the writing fd causes EOF to be read on the other end