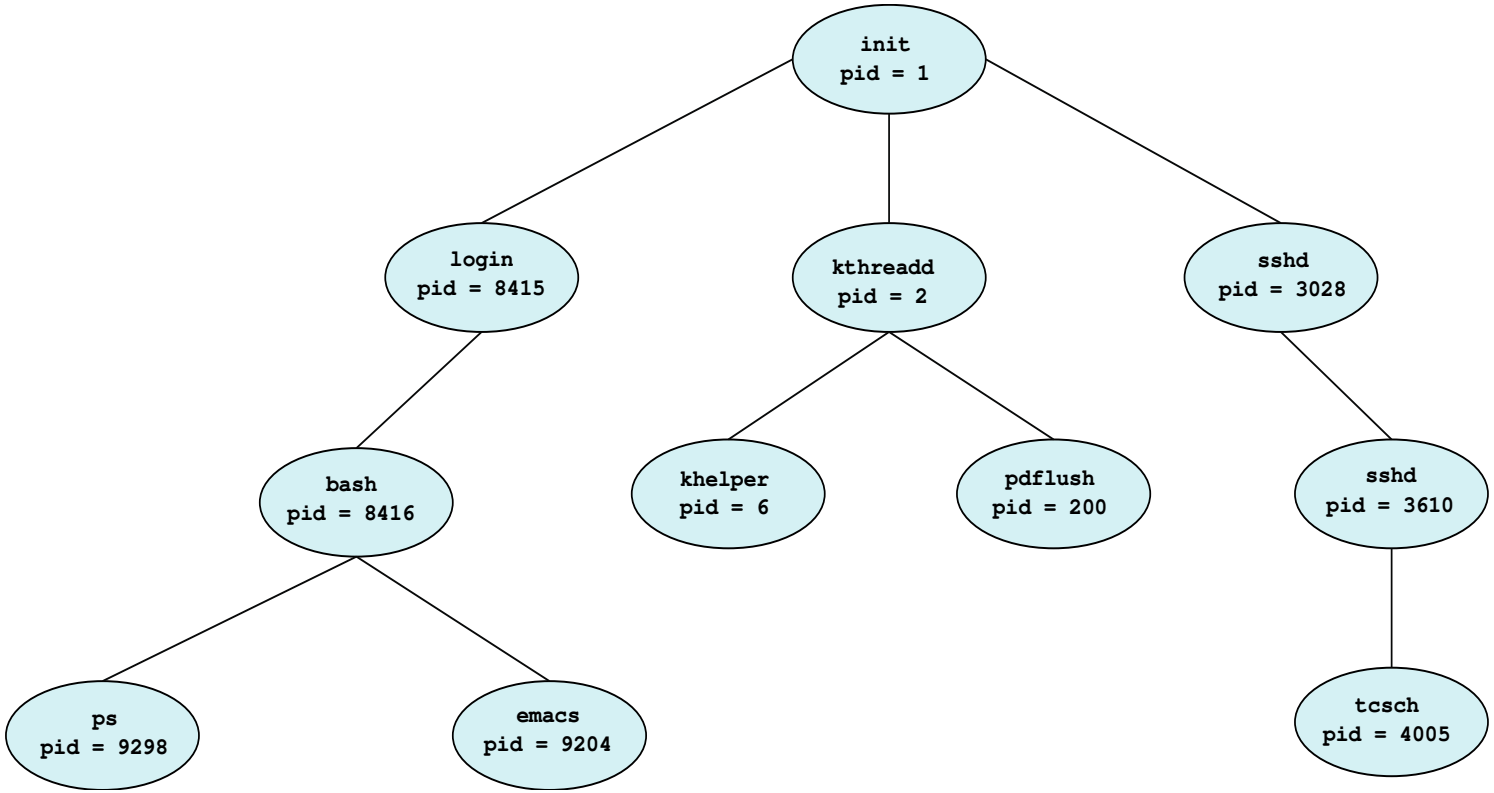
Processes (3)

Dr. Jun Zheng

CSE325 Principles of Operating Systems

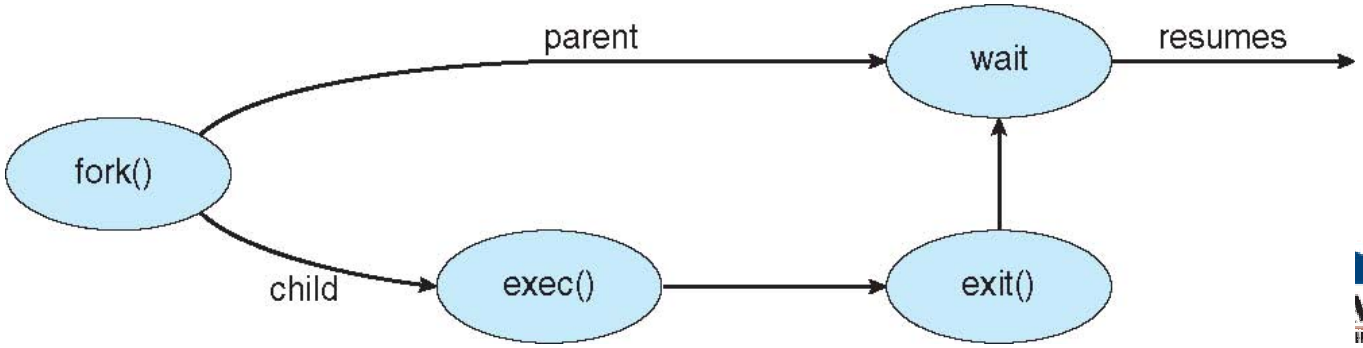
9/6/2019

A Tree of Processes in Linux



Process Creation (Cont.)

- Address space
  - Child duplicate of parent
  - Child has a new 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



Process Creation: Example

- When you log in to a machine running Unix, you create a shell process.
- Every command you type into the shell is a child of your shell process and is an implicit *fork* and *exec* pair.
- For example, you type emacs, the OS *"forks"* a new process and then *"exec"* (executes) emacs.
- If you type an & after the command, Unix will run the process in parallel with your shell, otherwise, your next shell command must wait until the first one completes.

C Program Forking Separate Process

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
       fprintf(stderr, "Fork Failed");
       return 1;
    }
    else if (pid == 0) { /* child process */
       execlp("/bin/ls","ls",NULL);
    }
    else { /* parent process */
       /* parent will wait for the child to complete */
       wait(NULL);
       printf("Child Complete");
    }

    return 0;
}
```

- In the parent process, fork returns the process id of the child
- In the child process, the return value is 0

NEW MEXICO TECH
SCIENCE · ENGINEERING · RESEARCH · UNIVERSITY

Process Termination

- Process executes last statement and then asks the operating system to delete it using the `exit()` system call.
  - Returns status data from child to parent (via `wait()`)
  - Process' resources are deallocated by operating system
- Parent may terminate the execution of children processes using the `abort()` system call. Some reasons for doing so:
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - The parent is exiting and the operating systems does not allow a child to continue if its parent terminates

Process Termination

- Some operating systems do not allow child to exists if its parent has terminated. If a process terminates, then all its children must also be terminated.
  - **cascading termination.** All children, grandchildren, etc. are terminated.
  - The termination is initiated by the operating system.
- The parent process may wait for termination of a child process by using the `wait()` system call. The call returns status information and the pid of the terminated process

```
pid = wait(&status);
```

- If child process terminates but no parent waiting (did not invoke `wait()`), the child process is a **zombie**
- If parent terminated but child process remains running itself, the child process is an **orphan**

In-Class Work 1

How many processes are created by the following program including the parent process? What will be printed by the program? Assume children always print before their parent.

```
int main(){
    int i;
    int a = 3;
    if(fork() == 0){
        a = a*4;
        if(fork() == 0)
            a = a - 2;
    }
    printf("%d\n", a);
}
```

Answer

3 processes

10

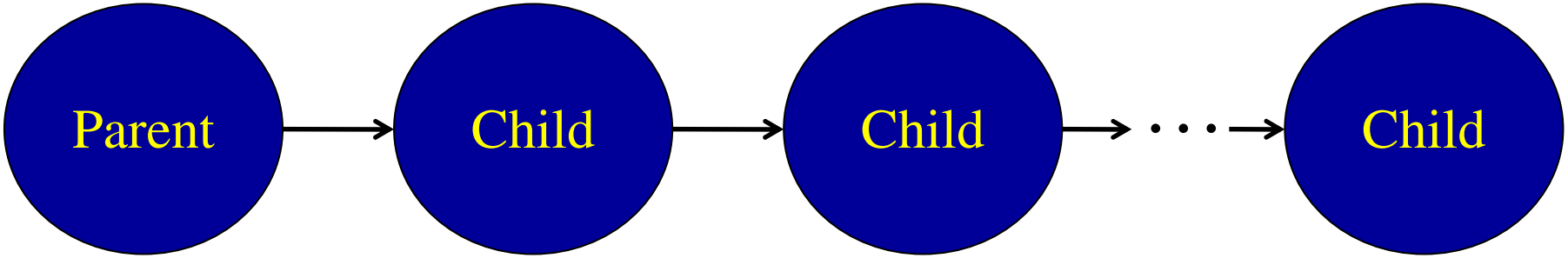
12

3

Process Chain

```
pid_t childpid = 0;

for(i = 0; i < n; i++)
    if(childpid = fork())
        break;
```

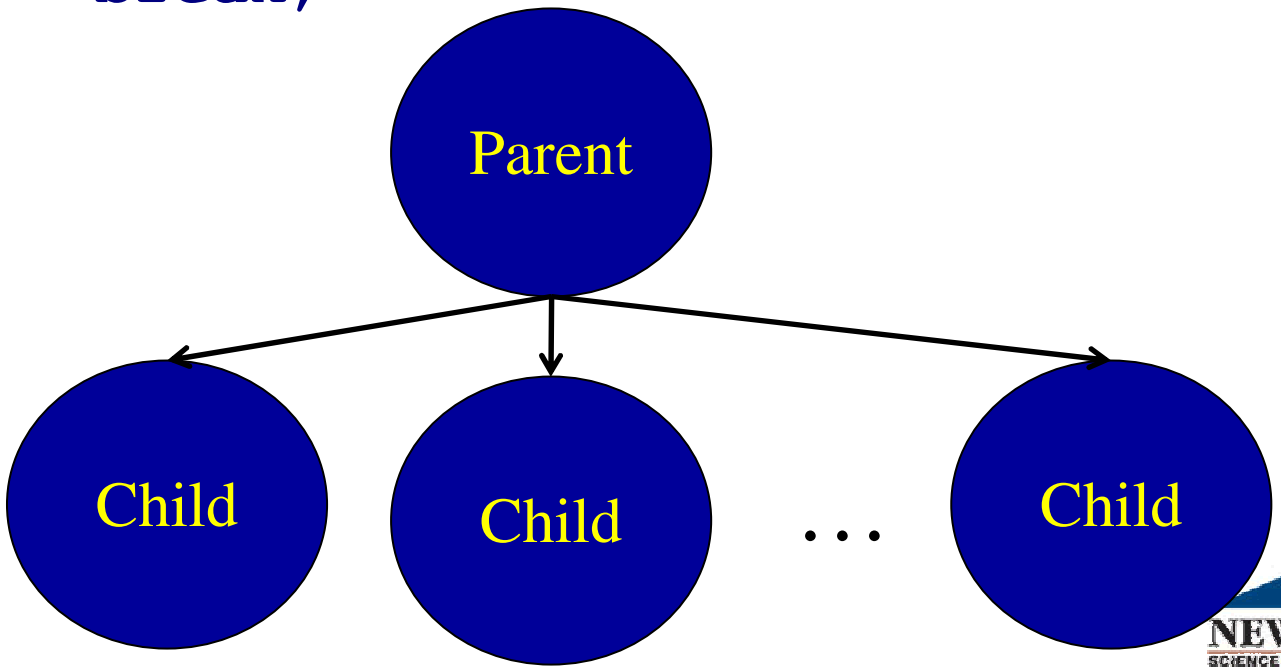


NEW MEXICO TECH
SCIENCE · ENGINEERING · RESEARCH · UNIVERSITY

Process Fan

```
pid_t childpid = 0;

for(i = 0; i < n; i++)
    if((childpid = fork()) <= 0)
        break;
```



NEW MEXICO TECH
SCIENCE · ENGINEERING · RESEARCH · UNIVERSITY

xv6 Process Management

In `sysproc.c, proc.c`

- `fork()` - create a new process,
- `exit()` - terminate current process
- `wait()` - wait for the child to exit
- `kill()` - set proc-killed to 1

In `proc.c`

- `sleep(chan)` - sleeps on a "channel", an address to name the condition we are sleeping on
- `wakeup(chan)` - wakeup all all threads sleeping on chan (may be more than one)

In `sysfile.c, exec.c`

- `exec()` - replace memory of a current process with a memory image (of a program) loaded from a file