Process Management

Advanced Embedded Linux Development

with **Dan Walkes**

University of Colorado **Boulder**

Learning objectives:

What is a process

PATH and process execution

System calls: execl(), fork(), wait(), system()

Process Management

- What is a process?
  - A running program
  - Binary image in memory
  - Virtualized memory instance
  - Files
  - Associated user
  - One or more threads of execution

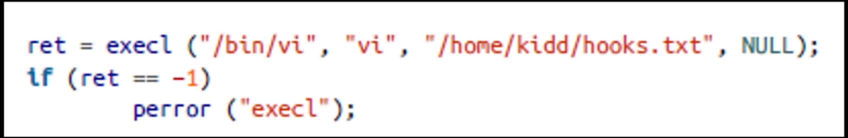
Process ID and Defined Processes

- pid Uniquely identifies a process at any single point in time.
  - allocated/reused by the kernel
- Special process IDs:
  - idle process - pid 0 - One per processor, handles power control
  - init process - pid 1 - Initializes system, starts services, launches a login program

Linux System Programming Chapter 5

https://unix.stackexchange.com/questions/361245/what-does-an-idle-cpu-process-do

Executing a Program



```
ret = execl ("/bin/vi", "vi", "/home/kidd/hooks.txt", NULL);
if (ret == -1)
        perror ("execl");
```

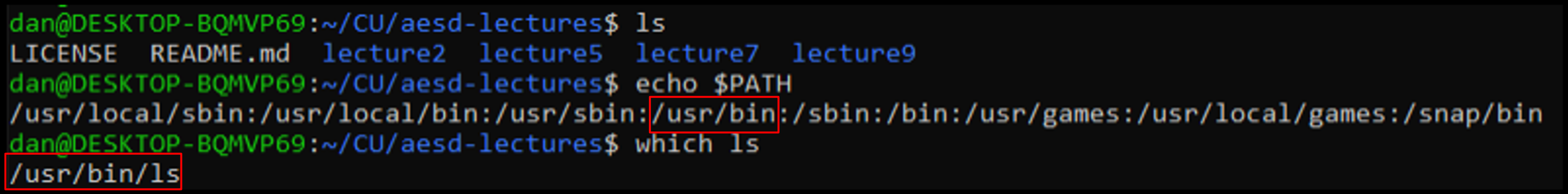
- execl()
  - replaces the current process new content
  - does not return on success
  - changes pretty much everything in the current process image, with the exception of:
    - pid, priority, owning user and group

Executing a Program

- Other family members execlp() execle() execv() execvp() execve() differ on
  - How arguments are specified
  - Whether environment is specified
  - Whether PATH is included

Linux PATH variable

- Colon separated list of places to search for executables when specified by name rather than path



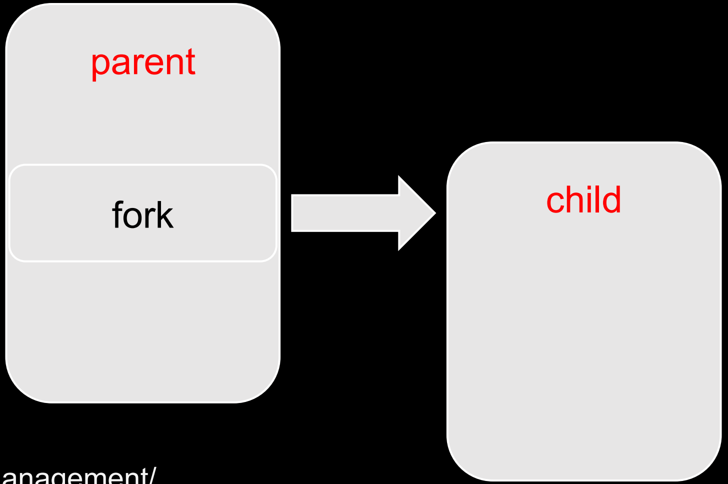
```
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures$ ls
LICENSE  README.md  lecture2  lecture5  lecture7  lecture9
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
dan@DESKTOP-BQMVP69:~/CU/aesd-lectures$ which ls
/usr/bin/ls
```

PATH Injection

- Modifications to your PATH can potentially influence which executable is used
  - /home/hacker/ls instead of /usr/bin/ls
- What’s the solution?
  - Avoid relying on PATH, use absolute path to executables
  - Especially when we are in full control of the embedded system image!

fork()

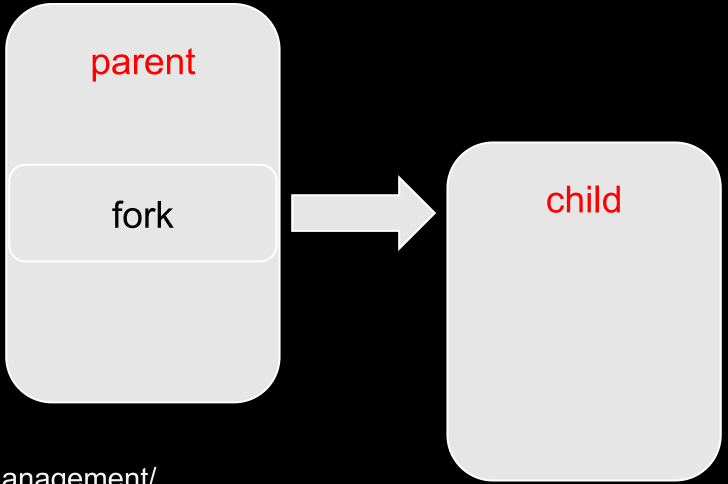
- Creates a new process running the same image as the current process
- Creates a new PID for child.
- Often used to start Daemons
  - process running in the background



Linux System Programming Chapter 5
http://www.it.uu.se/education/course/homepage/os/vt18/module-2/process-management/

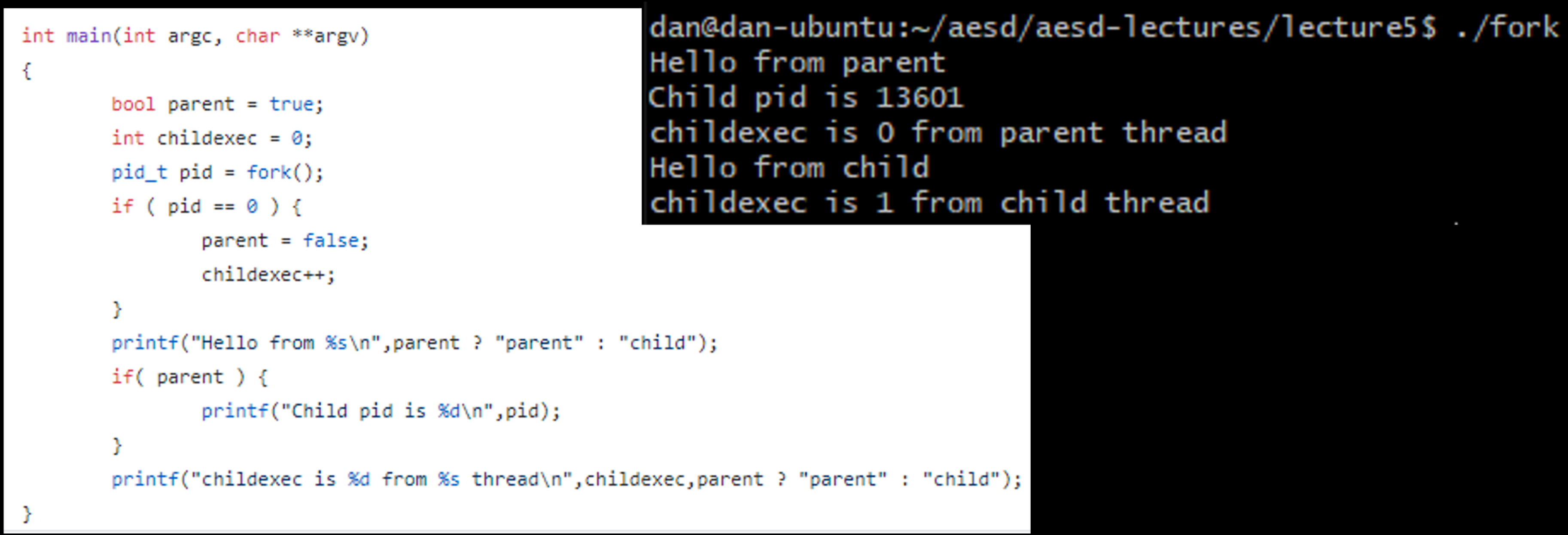
fork()

- Copy on Write for any changes to process memory space (memory is shared until modified)
  - Local variables
  - Kernel resources
    - Open file handles



Linux System Programming Chapter 5
http://www.it.uu.se/education/course/homepage/os/vt18/module-2/process-management/

fork()



```c
int main(int argc, char **argv)
{
        bool parent = true;
        int childexec = 0;
        pid_t pid = fork();
        if ( pid == 0 ) {
                parent = false;
                childexec++;
        }
        printf("Hello from %s\n",parent ? "parent" : "child");
        if( parent ) {
                printf("Child pid is %d\n",pid);
        }
        printf("childexec is %d from %s thread\n",childexec,parent ? "parent" : "child");
}
```

```
dan@dan-ubuntu:~/aesd/aesd-lectures/lecture5$ ./fork
Hello from parent
Child pid is 13601
childexec is 0 from parent thread
Hello from child
childexec is 1 from child thread
```

https://github.com/cu-ecen-5013/aesd-lectures/blob/master/lecture5/fork.c

Zombie Process

- Happen when a child process dies before the parent
- Process remains in zombie state until the parent is informed about child terminating.

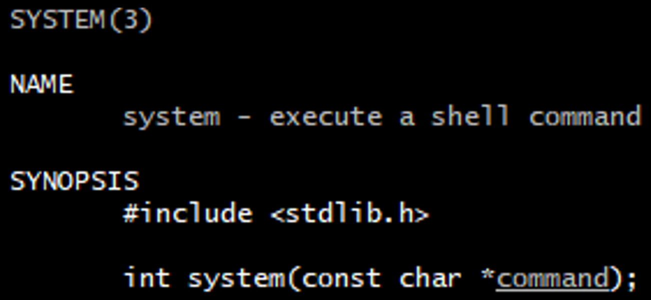
Waiting for Processes

- Parent uses wait() to obtain information about terminated children.
  - Reason for terminated (signal, etc)
  - Return code
- waitpid() waitid() - additional option
  - especially if the process starts multiple children.

Launching a new process

- So far we’ve discussed:
    - Replacing a process with exec()
    - Launching a child process with fork()
    - Waiting for completion with wait()
- How do we launch a completely new process and wait for its completion?
    - Combine the three - fork() + exec() + wait()

Launching a new process



```
SYSTEM(3)

NAME
       system - execute a shell command

SYNOPSIS
       #include <stdlib.h>

       int system(const char *command);
```

- Is there a less complicated option than fork()+exec()+wait()?
  - system() - which does all these for us.
- What’s the drawback of system()?
  - Uses the PATH - can have PATH injection concerns
  - Expands shell input like $HOME