fork(2): create a new process

exit(3): exit a process

Waiting:

```
pid_t wait(int *status);
pid_t waitpid(pid_t wpid, int *status, int options);
#include <sys/resource.h>
pid_t wait3(int *status, int options, struct rusage *rusage);
pid_t wait4(pid_t wpid, int *status, int options, struct rusage *rusage);
```

zombie process -- exited & not waited on

(example: zombie.c)

How to "kill" the zombies?

Wait on them!

Killing Zombies (e.g. waiting!)

pid_t waitpid(pid_t wpid, int *status, int options);

wpid

- -1   waits for any child process.
- 0   waits for any child process in the process group of the caller.
- >0   waits for the process with process id wpid.
- <-1   waits for any process whose process group id equals the absolute value of wpid.

status

- exit value, other information about exit status.
- see man page

options

- WNOHANG -- This option is used to indicate that the call should not block if there are no stopped or exited children. wait*() call returns 0 if no stopped or exited children.

Exit Status ... macros

- WIFEXITED(status)
  - true if process called _exit(2) or exit(3)
  - WEXITSTATUS(status)
    - The low-order 8 bits of the argument passed to _exit(2)
- WIFSIGNALED(status)
  - True if the process terminated due to receipt of a signal.
  - WTERMSIG(status)
    - The number of the signal that caused the termination.
  - WCOREDUMP(status)
    - True if a core file was created.
- WIFSTOPPED(status)
  - True if the process has not terminated, but has stopped and can be restarted.
  - WSTOPSIG(status)
    - Number of signal that stopped process.

Race Conditions

- Multiple processes working together
- Results depend on the order of the processes running.

  (Example race.c)

- Example -- character at a time output
- Accessing a file, "lock file"

More on the exec functions ....

System Call:

- int execve (char *path, char * argv[], char * envp[]);

Library Calls:

- int execl (char *path, char *arg, ...);
- int execlp (char *file, char *arg, ...);
- int execle (char *path, char *arg, ..., char *envp[]);
- int execv(char *path, char *argv[]);
- int execvp(char *file, char *argv[]);

Use:

- *p -- search the path for the file named
- others require full file path.

Example program exec.c

Exec functions (page 2)

Things that remain the same across an exec

- process ID (pid), parent process ID (ppid)
- real user ID (uid), real group ID (gid)
- supplementary groups IDs
- process group ID
- session ID
- controlling terminal
- time left until alarm clock
- current working directory
- file mode creation mask (umask)
- file locks
- process signal mask
- pending signals
- resource limits
- time accounting values

Exec functions (page 3)

Open files:

- file will remain open unless set to close.
- r = fcntl (fd, F_SETFD, FD_CLOEXEC);

Changing User IDs

Kinds of UIDs

- real user ID
- effective user ID (set by exec if setuid bit is set)
- saved set-user-ID (set by exec if setuid bit is set)

int setuid(uid_t newuid);

- POSIX (and Linux)
  - effective uid == 0 => sets all three
  - real uid == newuid => sets effective uid
  - saved uid == newuid => sets effective uid
  - otherwise error
- BSD (NetBSD)
  - effective uid == 0 => sets all three
  - real uid == newuid => sets all three
  - otherwise error

Changing User IDs (page 2)

int seteuid(uid_t euid);

- POSIX -- no such call
- BSD (NetBSD)
  - set effective to either real or saved UID

int setgid(gid_t gid);

int setegid(gid_t egid);

- Same rules for these.

Use?

Interpretor files

Script files can be run directly from command line

```
#!name [opt parameter]
contents
```

Uses:

- shells
- perl
- python