MP4: The UNIX Shell

- exec*()
- fork()
- wait()
- system()
- pipe
- Background command
- Special command

exec*( ) family

- Basic
  - execl(pathname, …): take a list of args
  - execv(pathname, argv): take an array of args
  - E.g
```
int ret = execl("/bin/ls", "ls", "-l", (char*)0);
```
- Custom environment
  - execle(pathname, …, envp)
  - execve(pathname, argv, envp)
  - E
```
char *env[] = { "HOME=/usr/home", "LOGNAME=home", (char*)0 };
int ret = execle("/bin/ls", "ls", "-l", (char*)0, env);
```
- PATHü
  - execlp(filename, …)
  - execvp(filename, argv)
  - E.g
```
char *cmd[] = { "ls", "-l", (char*)0 };
int ret = execvp("ls", cmd);
```

exec*( ) family

- exec*() is a brain transplant function
  - Clears out the machine code of the calling program
  - Loads the code of the called program
  - Does not return on success

```
#include<stdio.h>

int main ()
{
        char *  arglist [] = {"doesnotmatter","-l", "-a"};
        printf ("<<<<<<<<< About to execute ls -l>>>>>>>>>\n");
        execvp ("ls", arglist);
        printf ("<<<<<<<<< ls is done >>>>>>>>>\n");
        return 0;
}
```

- Can we directly call exec() to handle a user's command in our shell program?
  - NO! We'll lose our shell
  - So what can we do?

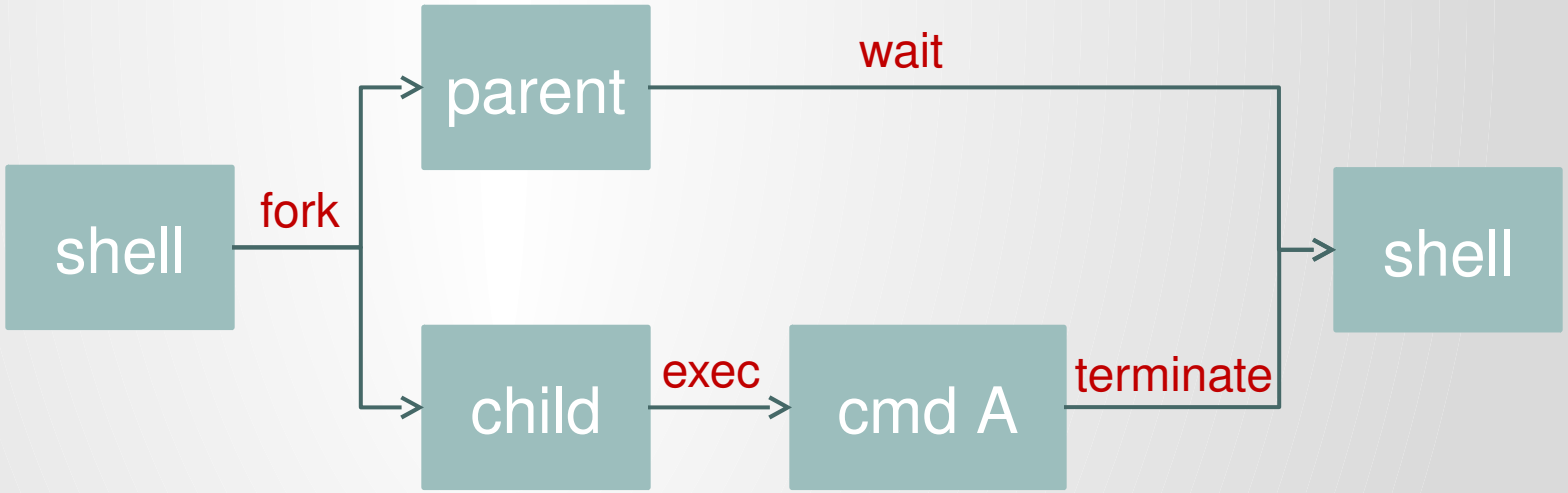
fork( )

- Create a new process
- Return twice if succeed
  - 0: child process
  - >0: parent process, the return number is the child's PID
- So we have a new process, then what?
  - Call exec*() in the child process

- Now the user's command is running, how can we wait until it finishes so that to accept next command?

wait*( )

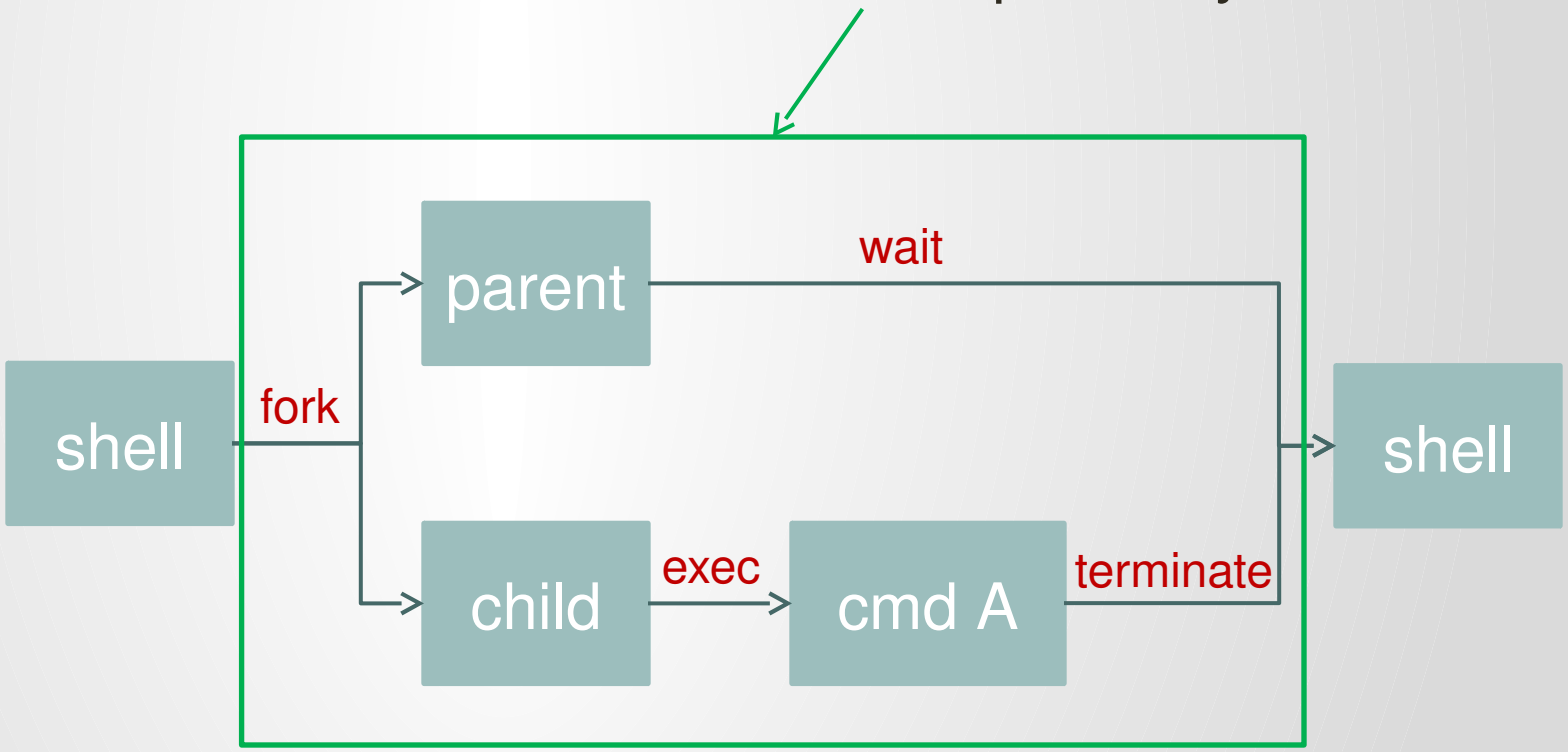
- Synchronize with children
  - Block current process until one of its children terminates
  - Return value is the PID of terminated child



system ( )

Keep your hands off!

- A naïve approach for our shell program is to just use the system() function
  - Because it takes care of this part for you



pipe

- An interprocess communication channel
  - Implemented as FIFO structures through the kernel
  - Resides in memory

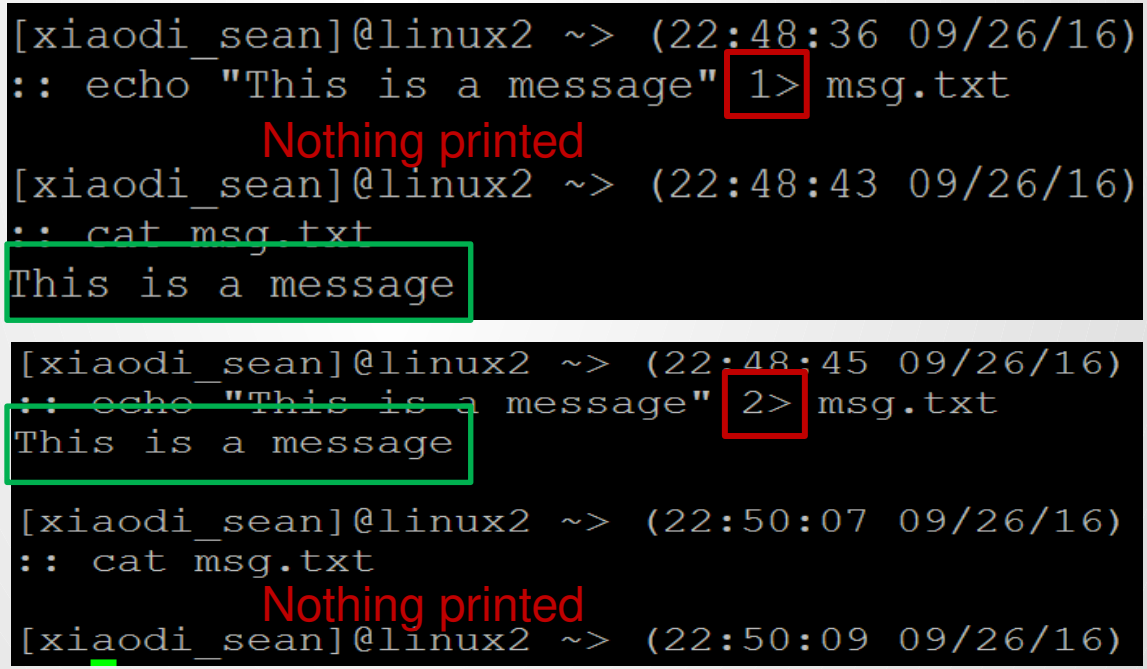
```
:: echo "This is a message" | wc -m
18
```

The output of echo becomes the input for wc, data is passed in RAM

- What is another naïve approach to such communication, particularly for our shell program?
  - Use a temporary file Don't

pipe

- File descriptor
  - An abstract indicator used to access a file or other I/O resource (pipe, network socket…)
  - A non-negative integer
  - 0 (stdin), 1 (stdout), 2(stderr)



pipe

- Use pipe(fd[2]) to create a pipe
  - fd[0] refers to the read end of the pipe
  - fd[1] the write end
- Terminal commands use stdin and stdout as I/O by default, we need to “tie” pipes to stdin/stdout
  - Use dup2(old, new) to make where new points to be the copy of where old points to, thus they refer to the same I/O port

Background command

- If the parent process doesn't wait, the child process will end up as a zombie after finish: Not good!
- Instead of using the blocking wait(), use waitpid() with option WNOHANG instead.
  - Need to periodically poll the status of the child process
  - Or capture the SIGCHLD signal

Special command

- cd
  - Call chdir() to do the job.
- exit
  - Should we wait for the background process to finish?
    - You can either wait for it or just kill it.

MP4

- Allow users to pipe the standard output from one command to the input of another an arbitrary number of times.
- Open a pipe
  - pipe()
- Set stdin and stdout to a pipe
  - dup2()
- Reference
  - man pipe
  - man dup2

MP4

- Allow users to specify whether the process will run in the background or foreground
  using an '&'. Backgrounding processes should not result in the creation of zombie
  processes.
  - Cannot use waitpid(pid, NULL, 0), otherwise your shell will not accept new command until the old one finishes.
  - Two options:
    - waitpid(pid, NULL, WNOHANG), a non-blocking version, poll is necessary.
    - Capture the SIGCHLD signal (a parent process will receive this signal whenever its child process exits)

MP4

- Allow your program to take an option ”-t” which will run your shell the same way,
  just without a prompt.
  - No prompt.
- (Bonus) Allow users to specify a custom prompt which supports printing the current
  directory, username, current date, and current time.
  - A customized prompt.
  - E.g. user:/home/user/svn$