ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

Project 1

Process, system call & Shell

World according to C

- Operating systems are large C programs consisting of many pieces written by many programmers
- C language
  - Data types, variables, control statements…
  - Header files: declaration, definition, macros…
  - For a large programming project

The Model of Run Time

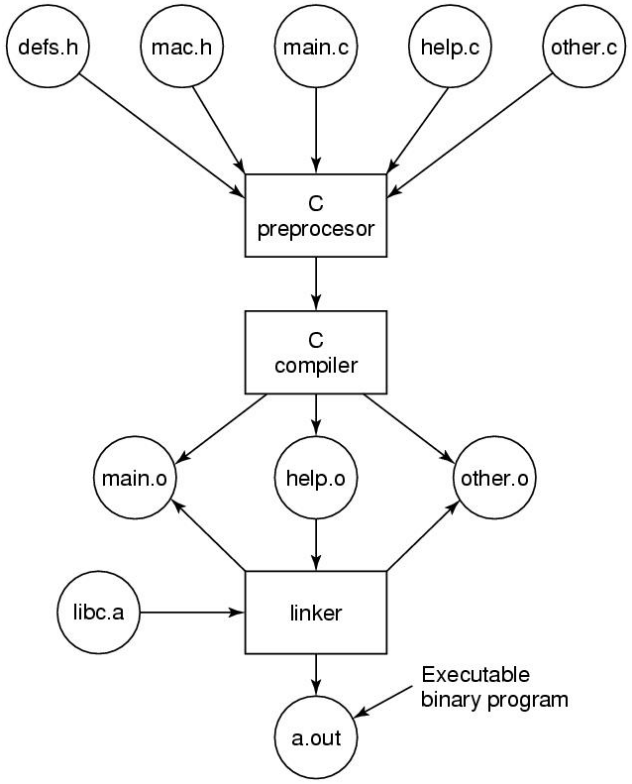

Figure 1-30. The process of compiling C and header files to make an executable.

Large programming projects

- C preprocessor:
  - Gets the header, expand macros, handling conditional compilation
- Compiler
  - .c -.o
- Linker
  - Combine all .o to an executable program; traditionally a.out

Gcc: preprocess-assemble-compile-link

- gcc -E hello.c –o hello.i
- gcc -S hello.i -o hello.s
- gcc -c hello.s –o hello.o
- gcc hello.o –o hello

- idd hello

- gcc -o hello hello.c

Processes

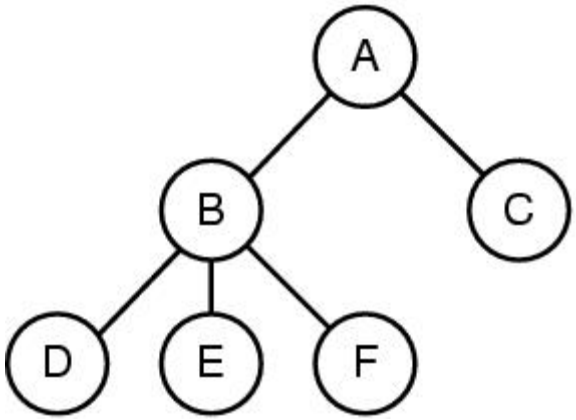

Figure 1-13. A process tree.
Process A created two child processes, B and C.
Process B created three child processes, D, E, and F.

Fork

- Pid=fork( )
  - Duplicates parent process completely Everything duplicated -data,registers,fd's
  - Fork returns a value (pid)
    - Zero in child
    - Child's PID in parent
    - Used to differentiate child from parent

Fork (Linux)

```
#include "stdio.h"
main(){
        int finish;
        int record = fork();
        if(record > 0){
                printf("I have a child: %d\n", record);
                int status = 0;
                finish = wait(&status);
                //finish = wait(0);
                printf("child %d is done\n", finish);
        }else if(record == 0){
                printf("child %d: existing\n", getpid());
        }else{
                printf("fork error\n");
        }
}
```

System calls

- System calls is the interface users contact with OS and hardware
- System calls vary from system to system, but the underlying concepts are similar
- Fork() is a system call

System calls

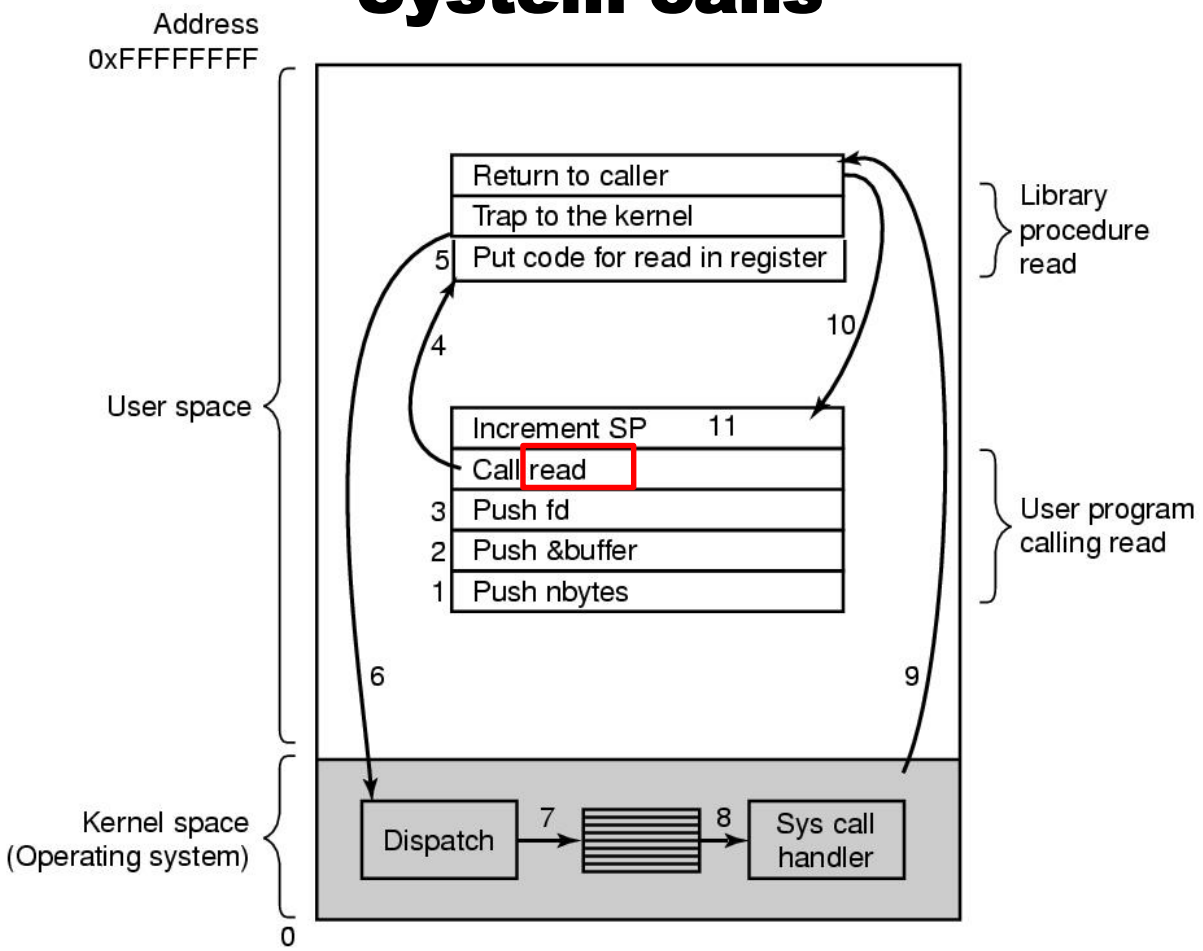

Figure 1-17. The 11 steps in making the system call read(fd, buffer, nbytes).

System calls for Process Management

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

Figure 1-18. Some of the major POSIX system calls.

POSIX is Unix standardization by IEEE

```
pid = fork()            Create a child process identical to the parent

pid = waitpid(pid, &statloc, opts)
                        Wait for a child to terminate

s = wait(&status)
                        Old version of waitpid

s = execve(name, argv, envp)
                        Replace a process core image

exit(status)
                        Terminate process execution and return status

size = brk(addr)
                        Set the size of the data segment

pid = getpid()
                        Return the caller's process id

pid = getpgrp()
                        Return the id of the caller's process group

pid = setsid()
                        Create a new session and return its proc. group id

l = ptrace(req, pid, addr, data)
                        Used for debugging
```

MyCopy 1

- Open File Stream Object:
  - **FILE fopen(char** **filename*,**char** **mode*) for read and write.
    - "r": Read only.
    - "w+": Opens an empty file for both reading and writing. If the given file exists, its contents are destroyed.
  - **int fclose(FILE** **stream*) to close file stream object after use.
  - Check errors

MyCopy 2

- Read and write to a file
  - Read/Write a block of chars:
    - **size_t fread**(**void** *_buffer_, **size_t** _size_, **size_t** _count_, **FILE** *_stream_);
    - **fread** returns the number of full items actually read
    - **size_t fwrite**(**const void** *_buffer_, **size_t** _size_, **size_t** _count_, **FILE** *_stream_);
    - **fwrite** returns the number of full items actually written
  - Read an individual char:
    - **int fgetc**(**FILE** *_stream_);
    - **int fputc**(**int** _c_, **FILE** *_stream_);

Execve

- Replaces process' core image by the file named in its invocation
- Execve(name,arg,environp) has 3 parameters
  - Name of file (e.g. cp command)
  - Arg-pointer to argument array
  - Environp-pointer to environment array

Excve (example)

- `Cp f1 f2` is located by execve and parameters are passed to it by execve

  - Main(argc, argv, envp) is main prog in cp
    - Argc is #items on command line (=3 in cp)
    - Argv points to array (arg[0] points to cp, arg[1] points to f1,
    - Envp points to environment, an array of name=value with info like terminal type

A Simple Shell

```
#define TRUE 1

while (TRUE) {
        type_prompt( );
        read_command(command, parameters);

        if (fork( ) != 0) {
             /* Parent code. */
             waitpid(–1, &status, 0);
        } else {
             /* Child code. */
             execve(command, parameters, 0);
        }
}
```

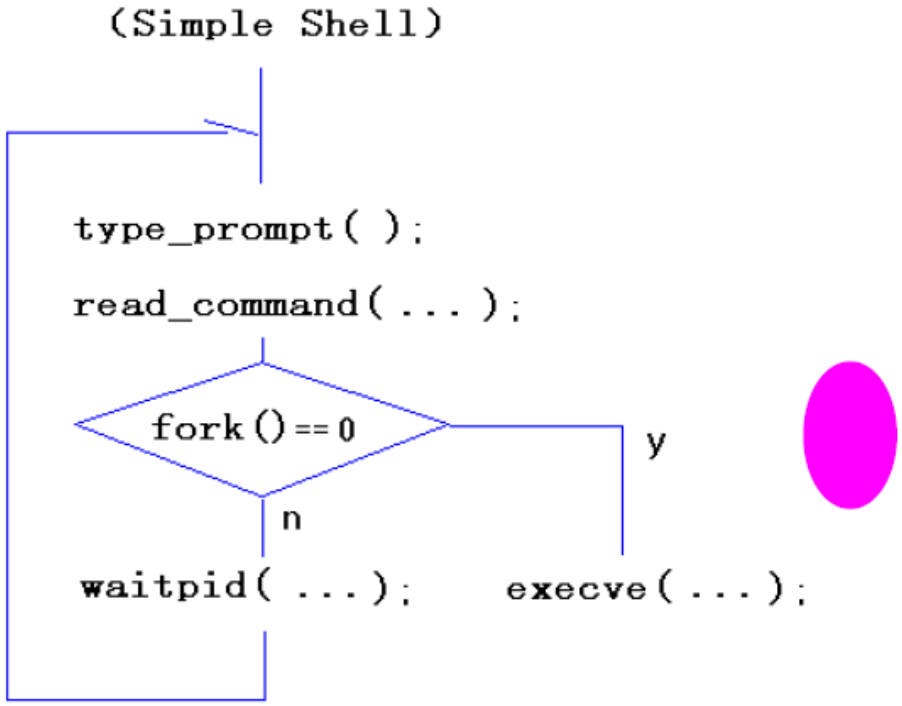

Figure 1-19. A stripped-down shell.