Introduction to Operating Systems

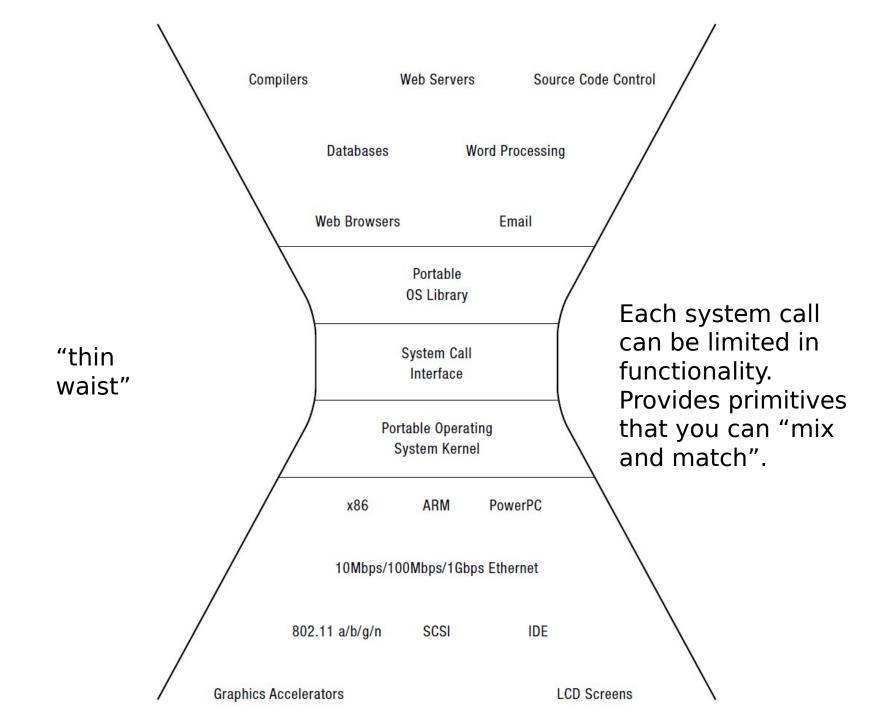
CPSC/ECE 3220 Summer 2018

Lecture Notes
OSPP Chapter 3

(adapted by Mark Smotherman from Tom Anderson's slides on OSPP web site)

OS Design Questions

- What functions should OS provide?
- Where should this functionality be located?
 - In library routines that run in user mode?
 - In the kernel?
 - In OS server processes that run in kernel mode?
 - In OS server processes that run in user mode?
 - (Note that clients and servers will need to make kernel calls to communicate)



UNIX Case Study

- Creating and managing processes
 - fork(), exec(), wait()

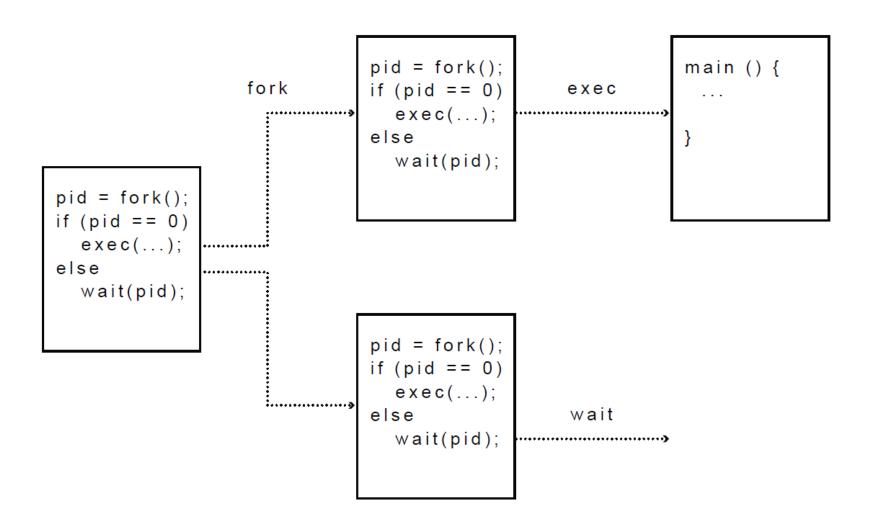
- Performing I/O
 - open(), read(), write(), close()

- Communicating between processes
 - pipe(), dup2(), select(), connect()

UNIX Process Management

- UNIX fork() system call to create a copy of the current process, and start it running – No arguments!
- UNIX exec() system call to change the program being run by the current process
- UNIX wait() system call to wait for a process to finish
- UNIX signal() system call to send a notification to another process

UNIX Process Management



Question: What does this code print?

```
int child pid = fork();
if (child pid == 0) {
                     // I'm the child
 process
  printf("I am process #%d\n", getpid());
  return 0;
} else {
                       // I'm the parent process
  printf("I am parent of process #%d\n",
 child pid);
  return 0;
```

Questions

Can UNIX fork() return an error?
 Why?

Can UNIX exec() return an error?
 Why?

 Can UNIX wait() ever return immediately? Why?

Implementing UNIX fork()

- Create and initialize the process control block (PCB) in the kernel
- Create a new address space
- Initialize the address space with a copy of the entire contents of the address space of the parent
- Inherit the execution context of the parent (e.g., any open files)
- Inform the scheduler that the new process is ready to run

Implementing UNIX exec()

- Load the program into current address space
- Copy arguments into memory in address space
- Initialize the hardware context to start execution at ``start'

(note that exec() does not create a new process)

Windows CreateProcess()

- Create and initialize the process control block (PCB) in the kernel
- Create and initialize a new address space
- Load the program into address space
- Copy arguments into memory in address space
- Initialize the hardware context to start execution at ``start''
- Inform the scheduler that the new process is ready to run

Windows CreateProcess() API (simplified)

```
CreateProcess(
              // No module name (use command line)
  NULL,
  argv[1],
              // Command line
              // Process handle not inheritable
  NULL,
              // Thread handle not inheritable
  NULL,
  FALSE,
               // Set handle inheritance to FALSE
  0,
               // No creation flags
  NULL,
              // Use parent's environment block
  NULL,
              // Use parent's starting directory
  &si,
              // Pointer to STARTUPINFO structure
              // Pointer to PROCESS INFORMATION structure
  &pi
```

UNIX I/O

- Uniformity
 - All operations on files, I/O devices, and pipes use the same set of system calls: open(), close(), read(), write()
- Open before use
 - Check permissions and set up internal housekeeping
 - open() returns a handle (file descriptor) for use in later calls on the file
- Byte-oriented interface
- Kernel-buffered read() and write()
- Explicit close()

UNIX File System Interface

- UNIX file open() is a Swiss Army knife:
 - Open the file, return file descriptor
 - Options:
 - if file doesn't exist, return an error
 - If file doesn't exist, create file and open it
 - If file does exist, return an error
 - If file does exist, open file
 - If file exists but isn't empty, nix it then open
 - If file exists but isn't empty, return an error

• ...

Interface Design Question

 Why not separate system calls for open()/create()/exists()?

```
if (!exists(name))
    create(name); // can create() fail?
fd = open(name); // does the file exist?
```

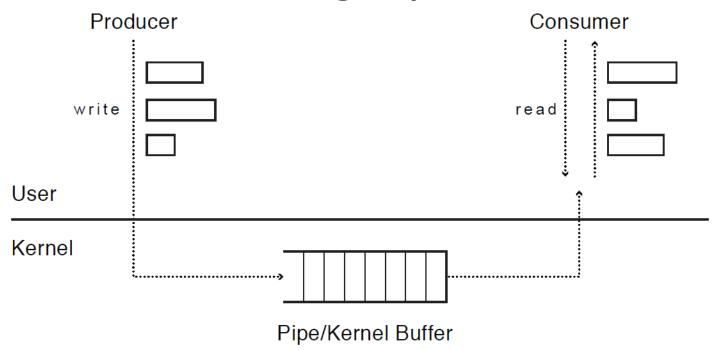
Command Interpreter ("Shell")

- Interactive interface to OS system calls
- Finds executable file associated with a command and creates a process (passing any parameters)
- Typically extended as a mini-language (e.g., control structures, macros)
- Shell scripts ("batch files" on Windows)
- Start-up files (e.g., .cshrc) and environment variables (predefined macros)
- Handles I/O redirection

Implementing a Shell

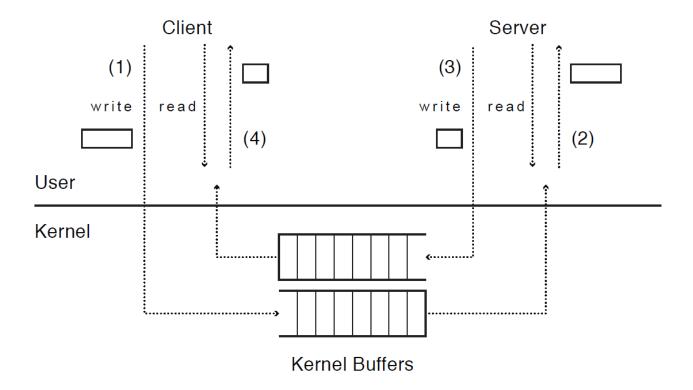
```
char *prog, **args;
int child pid;
// Read and parse the input a line at a time
while (readAndParseCmdLine(&prog, &args)) {
  child pid = fork(); // create a child process
  if (child pid == 0) {
    exec(prog, args); // I'm the child process. Run
  program
   // NOT REACHED
  } else {
    wait(child pid); // I'm the parent, wait for child
    return 0;
```

Interprocess Communication (IPC) Using Pipes



- Connect the I/O of two programs on the command line using a pipe ("|")
- Combine applications into complex structures

Client-Server IPC Using Pipes



 Server process can wait on input from multiple client processes using select()

OS Structure Design Choice

- Large kernel provides all or most system services
 - Sometimes called a "bloated kernel"
- Microkernel provides minimal services
 - Other services provided by user-mode servers

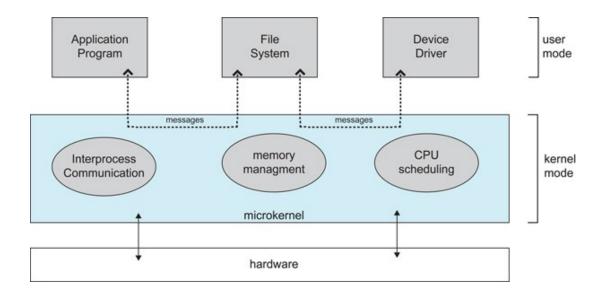


Diagram 2.14 from Silberschatz, Galvin, and Gagne, OS concepts, 9th ed.

Linux kernel map functionalities human interface processing networking system storage memory layers HI char devices processes memory access interfaces core files & directories sockets access user access 84,694 DVS. RIMITIVE Description and the space NA MINUS NA MATERIAL Decreased to NO. BURN Ast, String port, spe. yeary North, prigs (proc/test) interfaces NO. ROS spector special 565,768H Saff, No. Por tank aw COLUMN TWO Cooking for Min THE WORLDOOR NA LANCE N. Sept. color NA. rebod Virtual File System protocol families threads virtual memory MINES, N. --- security more was now NOT, Wife, propints DOMESTIC: manufactures. m print work upon Frits, Marr virtual **#345** FELONES. Tiroda. BOATS DANK HINE una Beste que Minusel, Streets. Inside permission Sed water Same Newari mouth, trade chale SHOW YOUR PAR OCO MUNTY, 198 un brief William Market deline new marche park a day on Man resident memory debugging synchronization States Street page cache networking socket bridges mapping na pina Mg/M splice storage sk rent ppd stress functional stress, see NAC SOLK In make brust, lacta, plos modas man has STREET AND DESIGNATION. NOTICE SHAPE week, when you Market Arrest No. World St. Two, Juga TAL APPLICATION Server pursers IN MAKE THAT Scheduler protocols HI subsystems logical system run logical memory file systems Link street logical BUT NAME Englishe Imprementations schoold Smanx **Amples** Miles Name of Aprillage real BOWN WILLIAM and in Page Allocator abstract devices generic HW access interrupts core block dévices network intertace **HID** class drivers device SCAPE, NA to for-Tell Shares -ALCOW! control nand desire BX person. of he ages NAME AND ADDRESS OF STATE STATE OF M. Sale of A Littlem | New York M.FQ-MAN oth, Monago, Money 100,100 48,313 HI peripherals device access CPU specific physical memory disk controller network March 14.70 device drivers and bus drivers operations device drivers hardware Street or say for half interfaces DESCRIPTION AND DESCRIPTION AN salt, first proper CHARL AND PROPERTY. enmudence. - std. Agency St SHEET, THE # 945, PM O'DOLD BOOK user peripherals I/O CPU disk controllers network controllers memory electronics

Diagram from http://www.makelinux.net/kernel_map/

Microkernel Evolution



First generation

Eg Mach ['87]

Memory Objects
Low-level FS,
Swapping
Devices
Kernel memory
Scheduling
IPC, MMU abstr.

- 180 syscalls
- 100 kLOC
- 100 µs IPC

Second generation

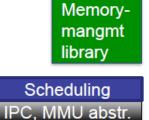
• Eg L4 ['95]

Kernel memory
Scheduling
IPC, MMU abstr.

- ~7 syscalls
- ~10 kLOC
- ~ 1 µs IPC

Third generation

seL4 ['09]



- ~3 syscalls
- 9 kLOC
- 0.2–1 µs IPC



Message Passing in Microkernels

- Typically define a message object for IPC
- Use send() and receive() operations
 - instead of file-like read() and write()
- Messages may be typed
 - For checking at compile time or run time
- Messages may be buffered in a queue
 - Also called a channel, endpoint, mailbox, port
- Zero-buffer approach is called rendezvous

```
// seL4 microkernel example from GitHub - seL4/libsel4/sel4 arch include/aarch32/sel4/sel4 arch/syscalls.h
LIBSEL4 INLINE FUNC void
seL4 Send(seL4 CPtr dest, seL4 MessageInfo t msgInfo)
  arm sys send(seL4 SysSend, dest, msgInfo.words[0], seL4 GetMR(0), seL4 GetMR(1), seL4 GetMR(2),
seL4 GetMR(3));
static inline void
arm sys send(seL4 Word sys, seL4 Word dest, seL4 Word info arg, seL4 Word mr0,
  seL4_Word mr1, seL4_Word mr2, seL4 Word mr3)
{
  register seL4 Word destptr asm("r0") = dest;
  register seL4 Word info asm("r1") = info arg;
  /* Load beginning of the message into registers. */
  register seL4 Word msg0 asm("r2") = mr0;
  register seL4 Word msg1 asm("r3") = mr1;
  register seL4 Word msg2 asm("r4") = mr2;
  register seL4 Word msg3 asm("r5") = mr3;
  /* Perform the system call. */
  register seL4 Word scno asm("r7") = sys;
                                                                       swi is ARM "software
  asm volatile (
    "swi $0"
                                                                     interrupt" instruction to
    : "+r" (destptr), "+r" (msg0), "+r" (msg1), "+r" (msg2),
                                                                         trap into the kernel
    "+r" (msg3), "+r" (info)
    : "r"(scno)
  );
```

Design Tradeoffs

- Performance concerns
 - Procedure call to library routine is fastest
 - Kernel call is slower
 - Passing messages to a server process is slowest
- Reliability concerns
 - Linux 4.11 has over 18M lines of code
 - half of which is reported to be device driver code
 - seL4 microkernel is 9,700 lines of C and 500 lines of assembler
 - but only implements interrupt handling, message passing, and scheduling; furthermore, it only runs on a limited number of platforms

Linus Shares His Opinion

I think microkernels are stupid. They push the problem space into *communication*, which is actually a much bigger and fundamental problem than the small problem they are purporting to fix. They also lead to horrible extra complexity as you then have to fight the microkernel model, and make up new ways to avoid the extra communication latencies etc.

from http://meta.slashdot.org/story/12/10/11/0030249/linus-torvalds-answers-yourquestions

Typical OS Design Approaches

- Hardware abstraction layer (HAL)
 - Portability across processors
 - Allows rest of OS to be written in a machineindependent manner
- Loadable device drivers
 - "Plug and Play"
 - The kernel does not need to be recompiled to work with new I/O devices
 - Trend to make device drivers run in user mode