CS 342302 Operating Systems

Fall Semester 2021

Prof. Pai H. Chou

Weekly Review 3

The questions here serve the purpose of reviewing concepts from the lecture, and expect the concepts to be tested on the midterm and final. However, they are by no means exhaustive. Anything covered in the lecture and projects can be tested.

1. Definitions and Short Answers - week 3 (9/27 lectures)

1. A shell can (1)implement all of the commands that it supports, as opposed to (2)relying on external programs that are not part of the shells. What are the advantages and disadvantages?

(1)

advantages: self-contained, efficient per command

disadvantages: to add commands need to modify the shell

(2)

adv: smaller shell, expandable

dis: heavier weight per command

1. Does the unix-style shell (such as bash, csh, etc) implement all of the commands that it can execute or not? Why is it a good idea?

no, expandable

1. Does the shell (GUI, CLI, etc) run in user mode or kernel mode? Why?

user mode, it is a user interface

1. What is the purpose of a **system call**?

switch to kernal mode

function calls to request OS services

1. Why doesn't a system call instruction take the target address of the routine, unlike a jump or call instruction? How does it indicate **what service to request**?

not supposed to know the address, number

1. What are ways **parameters** can be passed to a system call?

registers

table: stores values in table in memory and pass table address in reg

stack: user code push param on stack, os pop it off on return

1. Are the functions in standard-I/O library <stdio.h> all system calls? For example, is printf() a system call? If not, how does it perform I/O operation? What actual system call does printf() call?

no, no, write to buffer, write( )

1. Do all **system calls** execute in kernel mode?

yes

1. How are the functions in the **POSIX API** related to **system calls**?
2. Why would an application programmer prefer programming using an API than making system calls directly?

simple, portable, efficient

1. Do all <stdio.h> functions **make one or more system calls** in their implementation? Why or why not?

no, mode switch cost time and resources

1. Does MS-DOS **create a new process** to run a user program? What happens to the shell when the user program is running and when it exits?

no, user program replace most of the shell, shell gets reloaded again

1. At what point does FreeBSD **start the shell**? Is there just one kind of shell?

when user logins, many kinds

1. How does a shell on FreeBSD **start a process**? What does the shell do when the process is executing?

fork( ), wait until the process terminates or continue with user commands

1. What are the purposes of **system programs**?

Convenient for program development and execution

defines most users view of OS

1. What is the difference between **policy** and **mechanism**? What is the principle for the separation between policy and mechanism?

policy: what will be done? what is allowed? (parameterizable)

mechanism: how to do it? (implementation)

allows max flexibility if policy are to be changed later

1. MS-DOS has a simple structure consisting of BIOS and DOS device drivers, resident system program, and application program. What are the advantages and disadvantages with this minimal structure?

adv: use the least space

dis: unsafe, difficult to enhance

1. Do **system programs** in Unix including shells, compilers, and interpreters run in kernel mode or user mode?

user mode

1. Is traditional Unix a **two-layer** or **N-layer** structure? What are its pros and cons?

2-layer

pros: the entire OS kernel runs in 1 address space

cons: difficult to scale complexity

1. In an N-layered OS structure, what layer is **hardware**? user interface? What is the dependency between a lower and an upper layer?

layer 0, layer n-1

lower independent of upper, upper use services only of lower

1. What are the advantages of layered approach? What is a successful example use? What are the disadvantages compared to the 2-layer, tightly coupled structure?

adv: easier debugging and maintenance

ex. TCP/IP protocol stack

dis: less efficient, difficult to define layers

1. What is the key idea with **microkernels** compared to **monolithic** kernels?

dont want a user program/os to crash another user program/part of os

core service done in kernel-defined user space

1. Which of the following functions of a microkernel is done in kernel mode? in user mode?
   1. network driver
   2. device driver
   3. graphic driver
   4. interprocess communication
   5. CPU scheduling
   6. memory management
   7. file system
   8. application program
2. How can a microkernel run an OS service in **user mode** while also **protecting** the rest of the system?

make system call into OS, do core service in kernel-defined user space

since it is separated out, any crash would not crash the OS

1. Which of the following is easier or more efficient to do on a microkernel or a monolithic kernel? Why?
   1. port to a new architecture
   2. add new features
   3. OS overhead

microkernel: a, b (since easier extending) (but more message passing)

monolithic: c (due to shared memory)

1. What are the two main models for **interprocess communication**?

message-passing, shared-memory

1. How does a modular OS divide its functionality? What are the advantages of this organization?
2. What are **loadable kernel modules** (LKM)? Do they run in the same **address space** as the kernel or different?

api interface that can be loaded when needed, yes

1. How would you characterize Linux and Solaris? monolithic or microkernel? combined LKM or hardcoded services in the kernel?

monolithic + LKM

1. Is Darwin a monolithic or microkernel? What are the advantages of kexts in this context?

microkernel + layered + loadable

Mach microkernel + BSD Unix(POSIX)

can load in IO kit and external extensions

1. What is the purpose of the BSD Unix subsystem in Darwin?

CLI, file system, networking

1. Why does Android replace glibc with Bionic?

smaller than glibc, bypass Gnu Public License

1. Since Android uses Linux kernel, does it mean it can also run executable programs for desktop and laptop Linux? Why or why not?
2. What is a **core dump** file? How is it different from a **log file**?

memory content dump by os when program crashes, debugger can read it

core dump file: captures memory of the process

log file: error info

1. What is a **crash dump** file and how is it different from a **core dump**?

similar idea but it is for OS, it contains kernel memory

2. EdSim51 and 8051 - week 3 (9/29 lecture)

1. What is the interrupt vector (address of ISR) of UART on 8051?

0023H

1. To use UART interrupt on the 8051, why is it necessary to lay out the code memory this way:  
    ORG 0H  
    JMP Main  
    ORG 23H  
    JMP Serial\_ISR  
   Main: ...  
   Serial\_ISR: ...

dont want everything in the same place

1. The UART uses Timer-1 to generate its timing. What do the following bit registers do?

a. TR1

b. ET1

c. TF1

a.

b. enable timer interrupt

c. timer 1 flag

1. What is SDCC?

small device c compiler

1. What is the size (in terms of bits or bytes) of the following types in SDCC? and are they signed or unsigned?

.

a. bool: 1 bit, unsigned

b. char: 1 byte, signed

~~b. byte~~ (use #include<stdint.h> and uint8\_t instead) (refer to Q&A)

c. int: 2 bytes, signed

d. long: 4 bytes, signed

e. float: 4 bytes, signed

1. What does SDCC do if you invoke it with the following arguments?

a. sdcc file.c

b. sdcc -S file.c

c. sdcc -c file.c

d. sdcc file.rel -o file.ihx

a. compile file.c

b. compile to assembly, dont assemble/link

c. compile and assemble but dont link, create relocatable object file

d. name output file as file.ihx instead of default name

1. Why does main in .c file get translated into \_main in the .asm file?

the compiler inserts another \_ in front when generating assembly code

1. The .asm output from SDCC is different from Intel assembly syntax and therefore cannot be loaded directly into EdSim51 and run. So what file generated by SDCC (or related tools) in order to run on EdSim51?

packihx main.ihx > main.hex

1. In SDCC's <8051.h> header file, how is the GPIO port P1 **declared** so that it refers to the special-function register at memory address 90H?
2. Why does SDCC require that you declare an ISR with the \_\_interrupt keyword and a number such as (4) as in  
   void Serial\_ISR(void) \_\_interrupt(4) {  
    ...  
   }

tells the compiler to treat it as ISR, return with RETI

1. What is on the stack when an ISR is invoked?

the return address of the following line

1. Why do we NOT recommend **calling a function** such as DisplayLED(char num) from within the ISR?

ISR should do just enough, leave the other to user code

1. By observing SDCC's assembly output for the ISR that calls a function (slides 24 and 26), what is the purpose of all the push (bits, acc, b, dpl, dph, 0+7, 0+6, … psw) instructions and all the corresponding pop instructions before the reti instruction?

save and restore registers

1. What is the purpose of calling main() in the function definition  
   void \_sdcc\_gsinit\_startup(void) { main(); }

jump to main after everything start up

since \_sdcc\_gsinit\_startup in assembly will jump to \_main

1. What happens if you don't define the following routines to empty when compiling and linking with SDCC?  
   void \_mcs51\_genRAMCLEAR(void) {}  
   void \_mcs51\_genXINIT(void) {}  
   void \_mcs51\_genXRAMCLEAR(void) {}

might get some unnecessary junk code

1. What clock frequency should be used when running the UART at 4800 baud?

11.0592MHz

1. How do you define a lookup table in SDCC to put data into CODE memory for mapping digits (0-9) to the LED segment to light up?

\_\_code

1. On slide 21, the \_sdcc\_gsinit\_startup() function contains inlined assembly code  
   \_\_asm  
    mov sp, #0x57  
   \_\_endasm;  
   Can it be rewritten as C? How?

yes, sp = 0x57

3. Short Assembly Programs

1. Try running the C program on slide 11. Make sure it compiles using the command on slide 15 and load the .ihx file into EdSim51 ("Load" button)
2. Try running the two-file example (LEDtest0.c, LEDseg.c) shown on slide 16, compile and link with the commands on the slide, and load into EdSim51 to run.
3. Try running the Serial Echo example on slide 18
4. Try running the Polling UART example on slide 19
5. Try running the intrLED.c example on slide 20. What is the difference between the use of Main() in previous examples and main() in this example?