**1.**

**Which version of System/161 and OS/161 are you using?**

sys161: System/161 release 2.0.8, compiled Jan 9 2017 17:17:19

os/161 base system version 2.0.3

**Where was OS/161 developed and copyrighted?**

Harvard College

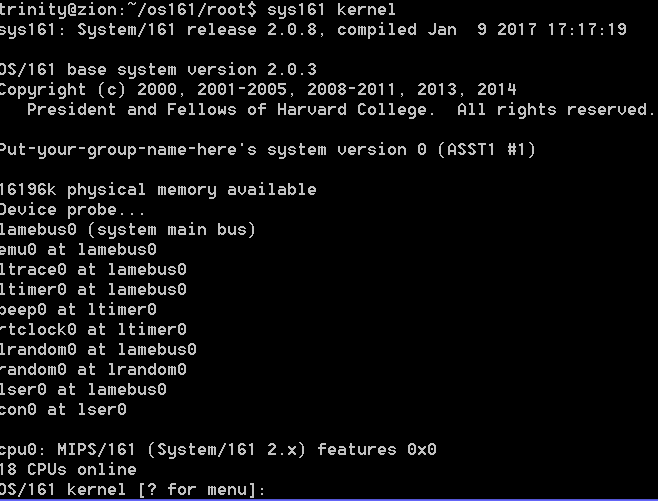
**How much memory and how many CPU cores was System/161 configured to use?**

16M and 8 cpu cores.

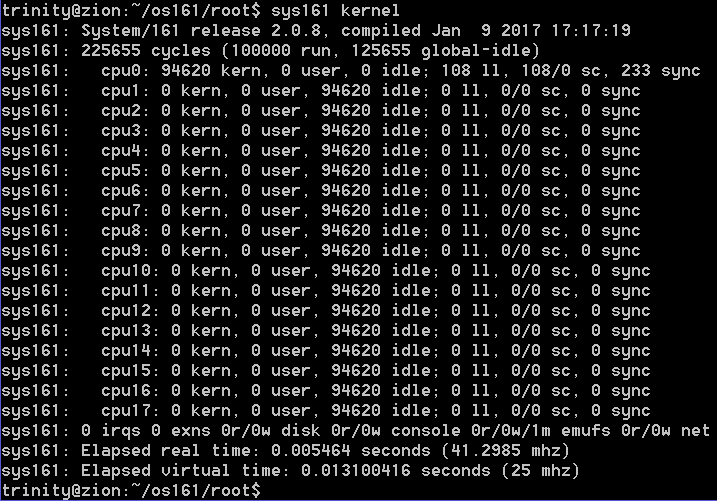
**Which configuration file was used by your running kernel?**

ASST1

**Boot your OS/161 kernel with 18 cores. What happens? provide a sample screenshot.**



**Try booting with 256k of memory. What happens? Provide a sample screenshot.**



**2. What are some of the details which would make a function “machine dependent”? Why might it be important to maintain this separation, instead of just putting all of the code in one function?**

A function could be machine dependent if it runs any assembly code. This separation can be used so that you can reuse the machine independent code, while writing machine dependent code to fill the gaps in functionality.

**3. What is the default compile option that we use for OS161’s virtual memory system?**

Bmake?

**4. In what file would you look to figure out how the various machine registers are labeled in OS161?**

/src/os161/kern/arch/mips/include/kern/regdefs.h

This file appears to define each of the registers.

**5. What is the difference between splhigh and spl0?**

Splhigh sets the priority level to the highest value, disabling interrupts, spl0 sets it to 0, enabling all of them. And splx(s) sets it to the specified level.

Found in /src/os161/kern/include/spl.h

**6. Why do you think types.h defines explicitly-sized types such as int32\_t instead of using the shorter int?**

The definitions are preceded by a line saying they are machine dependent, so I would assume that a basic int varies between architectures, just like the specific terms for each differently sized type.

**7. What is the interface to a device driver (ie. What functions must you implement to add a new device)?**

Devop\_eachopen, devop\_io, devop\_ioctl, as listed in /src/os161/kern/include/device.h

**8. What is the easiest way to add debug messages to your operating system?**

Run the kprintf(“text”) method.

/src/os161/lib/kprintf.c

**9. What synchronization primitives are defined for OS161?**

Semaphore, lock, condition variable, rwlock

/src/os161/kern/include/synch.h

**10. What is the difference between a thread\_yield and a thread\_sleep?**

Thread\_yield allows the next thread to use the cpu while keeping the current thread runnable.

Wchan\_sleep suspends the thread until awakened by someone else.

/src/os161/kern/include/thread.h

**11. What data structure do we use to keep track of the runnable threads in the system?**

A linkedlist of threads called threadlist /src/os161/kern/thread/threadlist.c

**12. What is a zombie?**

A thread that has exited but hasn’t been deleted yet. /src/os161/kern/include/thread.h

**13. What is the purpose of functions like copyin and copyout in copyinout.c? What do they protect against? Where might you want to use these functions?**

They act as a translation between user and kernel address spaces. They both run copycheck before running, protecting against direct control of the kernel and a wrapped around address.

/src/os161/kern/vm/copyinout.c

**14. Look at how getch is implemented, It is the function for reading a character from the terminal. Which function in the kernel will the hardware call when a character is received from the terminal?**

Getch\_intr(struct con\_softc \*cs)

/src/os161/kern/dev/generic/console.c

**15. What happens when you do a read on /dev/null?**

It returns bytes containing the end of file character

Man null

**16. What lock protects the current working directory?**

Static struct lock \*vfs\_biglock; due to my inability to find anything lock related when searching for the keyword directory, I’m forced to assume that the general statement “the big lock for all FS ops” means this lock is also protecting the current working directory.

/src/os161/kern/vfs/vfslist.c

**17. What functions are used to enable and disable interrupts?**

Splhigh and spl0?

**18. What function is invoked to make an exact copy of an address space?**

As\_copy(struct addrspace \*src, struct addrspace \*\*ret);

/src/os161/kern/include/addrspace.h

**19. What function is called when a new thread is created? (You will need to find the function definition to fully answer this question.)**

Thread\_create(const char \*name)

/src/os161/kern/thread/thread.c

**20. What is the MIPS instruction that actually triggers a system call? (Answer this by reading the source in this directory, not looking somewhere else.)**

Void syscall(Struct trapframe \*tf)

/src/os161/kern/arch/mips/syscall/syscall.c

**21. What error should be generated if the user invokes a system call with an invalid argument?**

EINVAL, invalid argument

/src/os161/kern/main/main.c to see EINVAL returned

/src/os161/build/install/include/kern/errno.h to see it refers to invalid argument.

**22. Where is the first line of OS/161 code that is executed when a trap or an interrupt occurs? Where does control get transferred to from that point?**

Apparently, it is the j common exception line after either mips\_utlb\_handler: or mips\_general\_handler in /src/os161/kern/arch/mips/locore/exception-mips1.S

Control then gets transferred to the mips\_trap(struct trapframe \*tf) function in /src/os161/kern/arch/mips/locore.trap.c

**23. Which register number is used for the stack pointer (sp) in OS/161?**

Register 29 according to /src/os161/kern/arch/mips/include/kern/regdefs.h

**24. What is the highest interrupt level?**

1, as defined by /src/os161/kern/include/spl.h

**25.1. How frequently are hardclock interrupts generated?**

100 times a second, as explained in /src/os161/kern/include/clock.h

**25.2. What is the syscall number for execv?**

2 as found in /src/os161/kern/include/kern/syscall.h

**26. Which file contains the machine independent code for the reboot? Which file contains the machine dependent code that actually does the reboot?**

/src/os161/kern/main/main.c has the independent code

/src/os161/kern/arch/sys161/dev/lamebus\_machdep.c is where the machine dependent code to reboot would be, but apparently isn’t even implemented so it just shuts down instead.

**27. If I change a header file, which of the set of configure and install sequences must be rerun to get an up to date executable?**

Bmake, bmake install

**28. If I add a new header file, which of the set of configure and install sequences do I need to rerun?**

Bmake depend, bmake, bmake install

**29. At what point does the output generated by the kprintf statements in the first part of the boot function appears on the screen?**

Once mainbus\_bootstrap() runs.

/src/os161/kern/main/main.c

**30. Which functions call thread\_switch? For each function explain why it called thread\_switch with the first argument that it does.**

Thread\_exit - (S\_ZOMBIE,NULL,NULL) – mark thread as dead so it can be cleaned up properly

Thread\_yield – (S\_READY,NULL,NULL) – mark thread as runnable to it can be switched back to later

Wchan\_sleep – (S\_SLEEP, wc, lk) – mark thread as asleep so it stops until called

/src/os161/kern/thread/thread.c

**31. What (briefly) Is the purpose of userptr\_t?**

Based on the usage in /src/os161/kern/vm/copyinout.c it appears to be used as a marker for the memory that the user is trying to access, which is used in turn to ensure that the memory doesn’t overlap the kernel

**32. How many bytes is an instruction in MIPS? Provide justification. (Answer this by reading the syscall function for MIPS carefully, not by looking somewhere else.)**

4 bytes. The comments say this call is where the syscall function increments the program counter in the trapframe by one instruction so it doesn’t infinitely loop.

/src/os161/kern/arch/mips/syscall/syscall.c

**33. What bus/busses does OS161 support?**

LAMEbus, defined in /src/os161/kern/arch/sys161/include/bus.h

**34. What function forces the processor to switch into usermode? Is this function machine dependent?**

Void enter\_new\_process(int argc, userptr\_t argv, userptr\_t env, vaddr\_t stack, vaddr\_t entry)

Passing argc and argv may use additional stack space on other platforms apparently, so this function is machine dependent.

/src/os161/kern/arch/mips/locoer/trap.c

**35. When a user program exits, where is the return value of the program left?**

In register 0.

**36. Describe the steps you need to take to add a new system call.**

In /src/os161/kern/include/syscall.h create new prototype method

In /src/os161/kern/arch/mips/syscall/syscall.c add switch case for new method, implement method to call function dealing with machine code.

In /src/os161/kern/dev/lamebus/lamebus.c and lamebus.h add the method, and write it using the machine dependent code to achieve goal

Recompile using bmake, bmake install.

--following the path used in Sys\_reboot, avoiding making new files because im still unsure about what trouble adding a file would bring.

**37. Modify, Compile, Built and install the kernel so that you have your name in the output. A sample is given below. Please provide a sample screenshot from your build.**

