System/161 2.x Software and Hardware Manual

David A. Holland
January 5, 2016

Introduction & Contents

System/161 is a synthetic (read: made up) hardware platform designed specifically for teaching operating systems and for experimental operating system hacking. It includes a simulated CPU, system bus, and bus devices that are realistic but simple and easy to work with.

New in System/161 2.x: support for multiprocessor and multicore system configurations.

The rest of this manual is divided into the following subsections:

Running System/161

There are two versions of System/161 that are built and installed: the normal one, `sys161`, and one compiled to be able to log information about what's happening and generally assist debugging, which is called `trace161`.

The general format for the command line for either of these is like this:

`sys161` [*System/161 options*] *kernel* [*kernel options*]

The *System/161 options* are:

–c *configfile*
: Specify alternate config file. Default is `sys161.conf`.

–C *slot:arg*
: Supply additional config argument to the device in the given slot. Applied after the config file is loaded, so values passed on the command line override values in the config file. Note that for now at least you cannot change the device

mappings on the command line, only adjust arguments. The most common use of this option is probably to adjust the system memory size and/or CPU count.

–D *count*
: Enable the doom counter. After the specified number of disk writes, the machine halts abruptly. (This is useful for testing file systems.) Individual disks can be marked "nodoom" in the config file, in which case writes to those disks do not count. (This is useful for your swap disk.)

–p *port*
: Listen for debugger connections on specified TCP port. The default is to use the Unix-domain socket `./.sockets/gdb` for debugger connections.
Note: because the remote gdb protocol does not support authentication, use this option only with caution.

–s
: Pass signal-generating characters (^C, ^Z, etc.) through to the running kernel instead of treating them as requests to sys161.

–w
: Wait for a debugger connection immediately on startup.

–X
: Do not hang waiting for the debugger; exit instead.

–Z secs
: Check for progress and first warn, then drop to the debugger if no progress occurs in the specified amount of simulator time. Progress is defined as successfully retiring an instruction in user mode. Don't use this option with in-kernel test workloads.

The following additional options control trace161's tracing and are ignored by sys161:

–f *tracefile*
: Set the file trace information is logged to. By default, stderr is used. Specifying –f– sends output to stdout instead of stderr.

–t *traceflags*
: Tell System/161 what to trace. The following flags are available:

d	Trace disk I/O
e	Trace emufs I/O
j	Trace jumps and branches
k	Trace instructions in kernel mode
n	Trace network I/O
t	Trace TLB/MMU activity
u	Trace instructions in user mode
x	Trace exceptions

Caution: tracing instructions generates huge amounts of output that may overwhelm smaller host systems.

The following option is also only available in trace161:

–P
: Collect a kernel profile and leave it in the file `gmon.out` for analysis by `gprof`.

The *kernel* is an operating system kernel to load and run. It should be an ELF–format executable for the same processor type as System/161 is compiled to support. For further information, see below.

Note that options found after the kernel name will be passed to the kernel and not interpreted by System/161.

As of version 2.0.5 the exit codes produced by System/161 are specified as follows:

0	Ordinary shutdown.
1	Crash shutdown caused by software failure.
2	Configuration, user, or runtime errors.
3	Explicit request: doom counter or debugger kill.

Note that in general software failures cannot reliably be distinguished from ordinary shutdowns. The crash shutdown exit code will be generated in the following cases:

- Exits taken when the –X option is in use, e.g. from illegal hardware operations or from a –Z timeout. (When –X is not in effect, these events stop and wait for a debugger connection.)
- System power–off, if at least one explicit debugger request was made via the trace device. With recent versions of OS/161 this normally means a kernel panic occurred; but note that nothing guarantees a panic will necessarily trigger such a request.

Manipulating disk images

The `disk161` tool can be used to manipulate disk images. It supports three actions: `create`, to create a new disk image; `info`, to print image information; and `resize`, to change the size of an image.

Create. Create a disk image like this:

```
disk161 create LHD0.img 5M
```

This creates an image called `LHD0.img` capable of holding 5 megabytes of data. The size is given in bytes, but as above can given one of the suffixes `G`, `M`, `K`, or `s` for gigabytes, megabytes, kilobytes, sectors (512–byte units) respectively. The size must be a unit number of sectors.

Info. After creating `LHD0.img` as above,

```
disk161 info LHD0.img
```

prints

```
LHD0.img size 5242880 bytes (10240 sectors; 5120K; 5M)
LHD0.img spaceused 8192 bytes (16 sectors; 8K; 0M)
```

meaning that the image holds up to 5M, but is currently only using 8K. (This is because the image is a sparse file by default; it only uses space for regions that have had data written to them. The only thing in a new image is the image header block, which is one sector rather than 8K; but here 8K is the host file system's block size.)

Resize. Change the size of a disk image like this:

```
disk161 resize LHD0.img 10M
```

which makes the image twice as big as before. You can also use + or – on the size argument to expand or shrink the image by a specified amount. Note that resizing the image will not resize any file system or other logical structure that may be stored on it. That must be done separately with other tools. Shrinking an image without doing this will destroy data.