

```
#!/usr/bin/perl -w

#
# raid5 perl utility
# Copyright (C) 2005 Mike Hardy <mike [at] mikehardy.net>
#
# This script understands the default linux raid5 disk layout,
# and can be used to check parity in an array stripe, or to calculate
# the data that should be present in a chunk with a read error.
#
# Constructive criticism, detailed bug reports, patches, etc gladly accepted!
#
# Thanks to Ashford Computer Consulting Service for their handy RAID
information:
# http://www.accs.com/p\_and\_p/RAID/index.html
#
# Thanks also to the various linux kernel hackers that have worked on 'md',
# the header files and source code were quite informative when writing this.
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2, or (at your option)
# any later version.
#
# You should have received a copy of the GNU General Public License
# (for example /usr/src/linux/COPYING); if not, write to the Free
# Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
#

my [at] array_components = (
"/dev/loop0",
"/dev/loop1",
"/dev/loop2",
"/dev/loop3",
"/dev/loop4",
"/dev/loop5",
"/dev/loop6",
"/dev/loop7"
);

my $chunk_size = 64 * 1024; # chunk size is 64K
my $sectors_per_chunk = $chunk_size / 512;

# Problem - I have a bad sector on one disk in an array
my %component = (
"sector" => 2032,
"device" => "/dev/loop3"
);

# 1) Get the array-related info for that sector
# 2) See if it was the parity disk or not
# 2a) If it was the parity disk, calculate the parity
```

```
# 2b) If it was not the parity disk, calculate its value from parity
# 3) Write the data back into the sector

(
$component{"array_chunk"},
$component{"chunk_offset"},
$component{"stripe"},
$component{"parity_device"}
) = &getInfoForComponentAddress($component{"sector"}, $component{"device"});

foreach my $KEY (keys(%component)) {
print $KEY . " => " . $component{$KEY} . "\n";
}

# We started with the information on the bad sector, and now we know how it
fits into the array
# Lets see if we can fix the bad sector with the information at hand

# Build up the list of devices to xor in order to derive our value
my $xor_count = -1;
for (my $i = 0; $i <= $#array_components; $i++) {

# skip ourselves as we roll through
next if ($component{"device"} eq $array_components[$i]);

# skip the parity chunk as we roll through
next if ($component{"parity_device"} eq $array_components[$i]);

$xor_devices{++$xor_count} = $array_components[$i];

print
"Adding xor device " .
$array_components[$i] . " as xor device " .
$xor_count . "\n";
}

# If we are not the parity device, put the parity device at the end
if (!$component{"device"} eq $component{"parity_device"}) {

$xor_devices{++$xor_count} = $component{"parity_device"};

print
"Adding parity device " .
$component{"parity_device"} . " as xor device " .
$xor_count . "\n";
}

# pre-calculate the device offset, and initialize the xor buffer
my $device_offset = $component{"stripe"} * $sectors_per_chunk;
my $xor_result = "0" x ($sectors_per_chunk * 512);

# Read in the chunks and feed them into the xor buffer
for (my $i = 0; $i <= $xor_count; $i++) {
```

```
print
"Reading in chunk on stripe " .
$component{"stripe"} . " (sectors " .
$device_offset . " - " .
($device_offset + $sectors_per_chunk) . ") of device " .
$xor_devices{$i} . "\n";

# Open the device and read this chunk in
open(DEVICE, "<" . $xor_devices{$i})
|| die "Unable to open device " . $xor_devices{$i} . ": " . $! . "\n";
seek(DEVICE, $device_offset, 0)
|| die "Unable to seek to " . $device_offset . " device " . $xor_devices{$i} .
": " . $! . "\n";
read(DEVICE, $data, ($sectors_per_chunk * 512))
|| die "Unable to read device " . $xor_devices{$i} . ": " . $! . "\n";
close(DEVICE);

# Convert binary to hex for printing
my $hexdata = unpack("H*", pack ("B*", $data));
#print "Got data '" . $hexdata . "' from device " . $xor_devices{$i} . "\n";

# xor the data in there
$xor_result ^= $data;
}

my $hex_xor_result = unpack("H*", pack ("B*", $xor_result));
#print "got hex xor result '" . $hex_xor_result . "'\n";

#####
#####
# Testing only -
# Check to see if the result I got is the same as what is in the block
open (DEVICE, "<" . $component{"device"})
|| die "Unable to open device " . $component{"device"} . ": " . $! . "\n";
seek(DEVICE, $device_offset, 0)
|| die "Unable to seek to " . $device_offset . " device " . $xor_devices{$i} .
": " . $! . "\n";
read(DEVICE, $data, ($sectors_per_chunk * 512))
|| die "Unable to read device " . $xor_devices{$i} . ": " . $! . "\n";
close(DEVICE);

# Convert binary to hex for printing
my $hexdata = unpack("H*", pack ("B*", $data));
#print "Got data '" . $hexdata . "' from device " . $component{"device"} .
"\n";

# Do the comparison, and report what we've got
if (!( $hexdata eq $hex_xor_result )) {
print "The value from the device, and the computed value from parity are
inequal for some reason...\n";
}
else {
```

```
print "Device value matches what we computed from other devices. Score!\n";  
}
```

```
#####  
#####
```

```
# Given an array component, and a sector address in that component, we want  
# 1) the disk/sector combination for the start of its stripe  
# 2) the disk/sector combination for the start of its parity  
sub getInfoForComponentAddress() {
```

```
# Get our arguments into (hopefully) well-named variables  
my $sector = shift();  
my $device = shift();
```

```
print "determining info for sector "  
  . $sector . " on "  
  . $device . "\n";
```

```
# Get the stripe number  
my $stripe = int($sector / $sectors_per_chunk);  
print "stripe number is " . $stripe . "\n";
```

```
# Get the offset in the stripe  
my $chunk_offset = $sector % $sectors_per_chunk;  
print "chunk offset is " . $chunk_offset . "\n";
```

```
# See what device index our device is  
my $device_index = 0;  
for ($i = 0; $i <= $#array_components; $i++) {  
  if ($device eq $array_components[$i]) {  
    $device_index = $i;  
    print "This disk is device " . $device_index . " in the array\n";  
  }  
}
```

```
# Figure out which disk holds parity for this stripe  
# FIXME only handling the default left-asymmetric style right now  
my $parity_device_index = ($#array_components) - ($stripe %  
  $array_components);  
print "parity device index for stripe " . $stripe . " is " .  
  $parity_device_index . "\n";  
my $parity_device = $array_components[$parity_device_index];
```

```
# Figure out which chunk of the array this is  
# FIXME only handling the default left-asymmetric style right now  
my $array_chunk = $stripe * ($array_components - 1) + $device_index;  
if ($device_index > $parity_device_index) {  
  $array_chunk--;  
}
```

```
# Check for the special case where this device *is* the parity device and  
return special
```

```
if ($device_index == $parity_device_index) {  
$array_chunk = -1;  
}  
  
return (  
$array_chunk,  
$chunk_offset,  
$stripe,  
$parity_device  
);  
}
```