

Linux Internals and Debugging

Greg Marsden and Bill Irwin

bill.irwin@oracle.com greg.marsden@oracle.com

December 7, 2004

Agenda.

- Linux != Solaris
- The Linux Kernel Group
- Understanding how Linux works
- Debugging on Linux

Linux != Solaris

- *truss* for solaris is *strace* for Linux
- resource <http://bhami.com/rosetta.html>
or <http://st-it.us.oracle.com:8080/knowledgebase/LinuxSo>

The Linux kernel group

- Linux is a GPL product, therefore all the code we write is O To protect Oracle's intellectual property, we are independent Oracle product development chain.

We work on OCFS and the Linux Kernel, both in a development and in the role of a platform specific OS DDR team.

Debugging Linux

- hardware bugs (bad RAM, stochastic errors)
- Linux kernel issues
- Oracle problems (ORA-600 and ORA-7445)
- Applications
- ...and occasionally Compiler (mostly on AS2.1)

Linux subsystems

- arch/
- fs/
- drivers/
- mm/
- net/
- kernel/

Loadable modules

- Separating off the source code
- Module license issue and EXPORT_SYMBOL()
- Support third-party modules and tainted kernels
- *modinfo*

The Linux Scheduler

- Improvements in the Linux scheduler
- reading *ps* output, process states, defunct processes
- *ps* and *top* are unreliable, now what?

The /proc filesystem

- Completely virtual file-based interface to kernel
- Changes are immediate and real-time
- /proc/PID/stat /proc/PID/statm

Threads under Linux

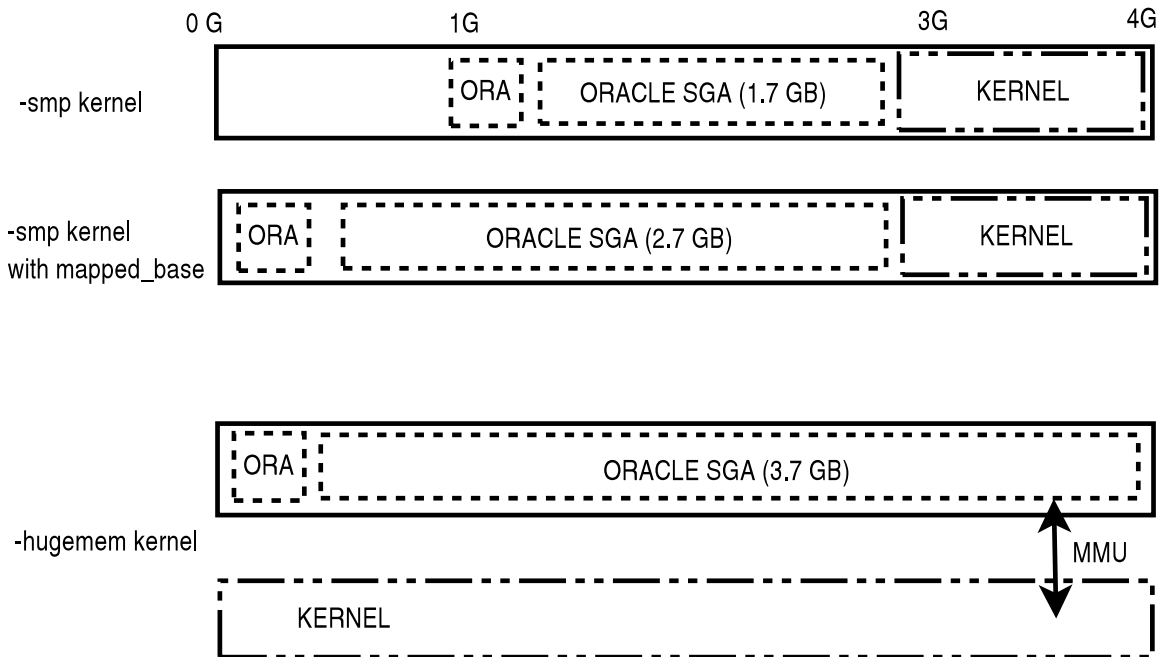
- pthreads
- nptl
- LD_ASSUME_KERNEL=2.4.18

Architectures

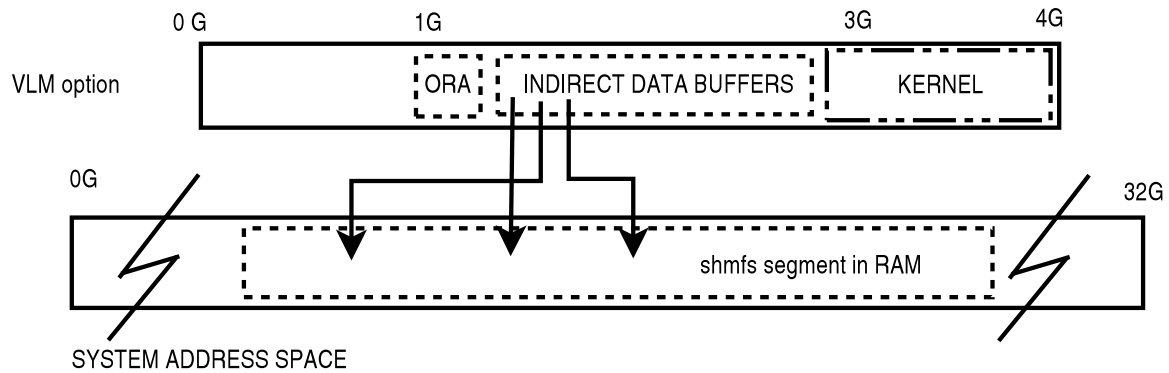
- x86, ia32
- x86_64, EM64T, ia32e, AMD64
- ia64, Itanium, IPF
- others...ppc, sparc, arm, etc...

Virtual Memory - Requirements

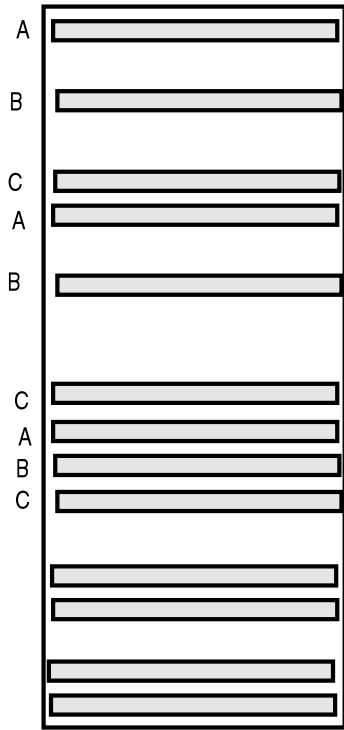
- Translating virtual addresses to physical addresses
- Protecting kernel data and other user's data from user access
- Software handling of presence and protection faults.



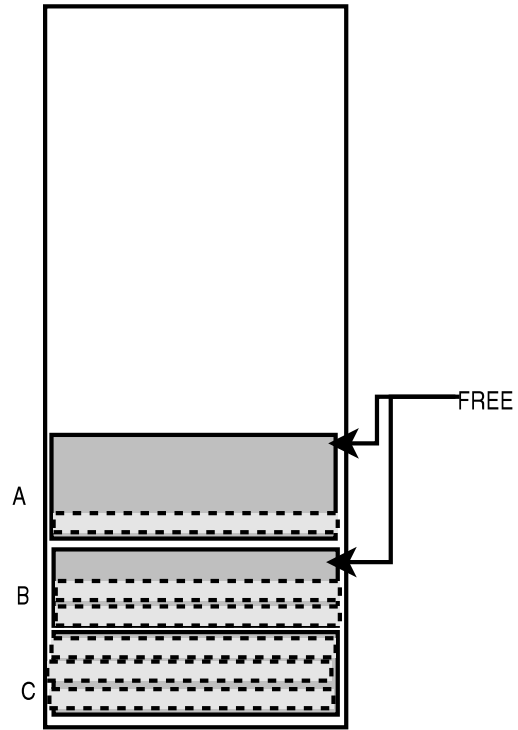
32-bit memory allocation schemes. Processes are allowed low memory, but not the reverse. Kswapd runs when processes are removed from lowmem to make room for kernel operations.



VLM_WINDOW_SIZE controls amount of shared memory mapped into shmfs to Oracle process space. Concepts *not related* to VLM include bigpages, etc.



AD HOC ALLOCATION



SLAB ALLOCATOR

LOWMEM ALLOCATION METHODS - PINNED DATA

Reading /proc/slabinfo

Large pages

- Large page sizes increase TLB coverage
- Large page sizes can't be used for everything
- RHAS2.1 and some other 2.4 distros call them "bigpages"
- More modern kernels use "hugetlbf", analogous to ramfs

Kernel memory footprint

- Boot-time allocations such as `mem_map[]`
- Runtime global allocations such as `dentries`
- Runtime per-process allocations
- Runtime per-thread allocations
- Which of these are “invisible” in RHAS2.1 et al

slabinfo - version: 2.0

# name	<active_objs>	<num_objs>	<objsize>	<objperslab>		
fat_inode_cache	0	0	384	10	1 : tunables	54
rpc_buffers	8	8	2048	2	1 : tunables	24
rpc_tasks	8	20	192	20	1 : tunables	120
rpc_inode_cache	0	0	448	9	1 : tunables	54
xfrm6_tunnel_spi	0	0	64	61	1 : tunables	120
fib6_nodes	5	119	32	119	1 : tunables	120
ip6_dst_cache	5	15	256	15	1 : tunables	120
ndisc_cache	1	20	192	20	1 : tunables	120

Linux Debugging - Tools

- /proc/meminfo, /proc/slabinfo
- shtrriage and bloatmon
- computing real process usage information
- dmesg, /var/log/messages
- netdump logs are for dumping oops outputs, not corefiles
- gdb on threads

Linux Debugging - Magic SysRq

- M : Memory
- P : Current processor (must run several times for SMP sy
- T : Kernel stack trace for all processes
- S,U,B : Key sequence to safely restart a hung system

Linux Debugging - Reading an Oops

- Looking at the EIP
- Difference from a Kernel Panic or a Kernel BUG()
- A Kernel Oops means the system encountered an bad condition. this does not necessarily mean the system stops running, it typically crash within a couple operations of encountering the bad condition. However, you may see a system trapped in a cycle of oops-ing.

Unable to handle kernel NULL pointer dereference at virtual address
printing eip:
00000000
*pde = 00000000

Oops: 0000

CPU: 0
EIP: 0010:[<00000000>]
EFLAGS: 00000246
eax: c107a960 ebx: c10670b8 ecx: c10670d4 edx: 00000000
esi: 00000000 edi: c17fc028 ebp: 00000000 esp: c16fdee4
ds: 0018 es: 0018 ss: 0018

Process bash (pid: 5, stackpage=c16fd000)

Stack: c0123dd4 c107a960 c10670b8 c16fc000 c107a960 c16fdf80 c16e7
c17b747c 00000000 00000000 c107a960 c16fc000 c0138b3d c107a
c16fc000 c107a960 00000001 00000000 c0135ee9 c107a960 c16fc

Call Trace: [<c0123dd4>] [<c0138a52>] [<c0138b3d>] [<c0135ee9>] [
[<c010956c>]

Code: Bad EIP value.