Memory Management in Linux

Orr Dunkelman

orrd@vipe.technion.ac.il

Why to manage the memory?

The Memory Management system is a very important part of the kernel. Without a good memory management, even simple tasks as opening a telnet connection would take several hundreds of seconds.

There are two basic blocks in the memory management system - Virtual Memory (which allows usage of more memory than there really is), and the Paging Daemon (which makes sure that only the needed data is in the fast layer of memory.

Memory Layers

Memory comes in varying sizes, costs and ...speeds. The cost-speed tradeoff is one of the more known in the computers area, and affects many of the strategies used in this context.

The fastest memory which is available is the registers. Allocated inside the processor, the access time to them is one clock cycle. This makes them the fastest memory usable. However, as they are very close to the processor, and very fast, they cost much more than any other memory layer.

Memory Layers (cont'd)

The Layer after the registers is the cache layers. This layer resides near the processor, and is also very fast (1-4 clock cycles access time). This memory is still bounded in size (most systems up to 512 KB), and thus in a multi-user environment (or when many processes are activated) the cache is fully used.

Note that in RISC systems, the processor can not access directly a memory cell (just in load/store operations he can), therefore, the processor needs to load the data into the registers.

Memory Layers (cont'd)

Next layer is the RAM - Random Access Memory. This layer respond in speeds of 10-200 clock cycles, and is usually large enough for keeping most of the programs (in your home PC). However, in very large systems, or when performing big tasks - not enough.

Question: Where can you get more memory?

Memory Layers (cont'd)

Answer - Hard Disk.

Hard disks are very slow system, they work in the area of $10^6 - 10^7$ clock cycles. Thus approaching them should be done as little as possible.

Still the Hard Disks contain any size of data usable (well, at the end you keep the data on the hard disk, so it should be big enough to contain it all). Also this is the cheapest device, thus using more of it, is cheap (but slow!).

Uses of Memory Management

Large Address Space Allows the process to think he has bigger address space, and use more of it (one of the operating system bluffs)

Protection Protect each process memory environment from others to interfere (assume someone get to play with root's shell memory ... br...)

Fair Physical Memory Allocation Justice for all! Anyone deserves some memory.

Shard Virtual Memory Why to keep two copies of the same thing? Share it! This is used to lower the memory requirements, and sometimes also as a meeting point for mutual communication

Model of Virtual Memory

The processor executes commands he finds in the memory in the place the PC register (Program Counter) points to. Note that the command also need to reside and memory and to be fetched to the processor.

In a Virtual Memory environment, address are not real address, but a pointer to some real address (through some translation technique). The Virtual address is converted to actual (physical address) by the processor, according to table kept in the Operating System.

Model of Virtual Memory (cont'd)

To make the translation easier virtual and physical memory are divided into handy sized blocks called *pages*. There pages are all in the same size, this is not a must, but rather a simplifying rule.

The size of the page, is based on the machine, and in Linux the default values are 8KB on ALPHA, and 4KB on i386. Each page has it's own number called PFN (Page Frame Number). In this model each address is combined of offset and a virtual page frame number. The offset size is the number of bytes in a page (in order to access a byte in a page), where the rest of the address is the VPFN.

Model of Virtual Memory (cont'd)

Each time the processor encounters a virtual address it extracts the offset and translate the VPFN into PFN and only then he can access the memory.

The translation phase is done using *Page Tables*. Each entry in the page table contain the following data:

- Valid Flag - Is the data in this entry is valid
- The PPFN (Physical PFN)
- Access control information - restrictions and protections (who may read/write)

Model of Virtual Memory (cont'd)

After the processor finds the VPFN, it looks for it in the page table. If the entry corresponding is valid, the processor combine the PPFN with the offset and access the memory. However, sometimes this entry is invalid, thus the Operating System is called with interrupt - Page Fault.

If no page fault had occurred, the processor takes the PPFN, multiplies by the page size, adds the offset and voilà here is the real address.

Note that the above scheme may map memory not according to any order.

Demand Paging

If the OS would not use the physical memory wise, than soon enough it would run out. A good method is to keep only the currently needed pages in memory. This method is also named *Demand Paging*.

When a process attempts to access a virtual address that is not currently in memory, than a page fault would occur, thus for the specific VPFN, there would be no valid PPFN. In this case the OS take over. It first check that the address exist somewhere, otherwise the process has tried to access not existent address, which require special treatment.

Demand Paging (cont'd)

If the address exist (and just not mapped), the OS load the page needed to the physical memory (from hard-disk), and then change the VPFN table so the pointer would agree with the reality.

As this scheme is quite efficient, Linux uses it.

Question: But what if there is no free physical memory?

Swapping

If there is not enough physical space, the page fault should “throw” away old page in order to bring in a new one. The problem with that, is that the system might throw out a page which will soon be needed, thus creating another page fault.

If the algorithm who do the swapping (A.K.A. *page Daemon*) is not good one, there would be lots of page faults, which will slow down the system.

Swapping (cont‘d)

Linux's page Daemon uses a swapping method named *Least Recently Used*, or more accurately an approximation for that method.

Any page in the system has the time he last was accessed, and whenever we need to throw away a page, we throw the one which was accessed the the longest time ago.

This algorithm was heavily researched, and it found to be in practice very close to the optimal solution, though in the theoretical plane, it can be proved to be not so optimal.

Sharing Virtual Memory

If two process access the same binary, why to keep two copies of it in the memory? We just make sure that the two processes' page table will point to the same PPFN ...

This way we use less physical memory, and can use the rest of the memory for other purposes.

Physical and Virtual Addressing Modes

Question: Why not to put the system itself in Virtual Memory?

Sounds quite silly, doesn't it? most of the multi-purpose processors allow the system to run in two modes - Virtual addressing (using page tables) and Physical addressing (when you really need to access a physical address without too many complications.

Note that in the Alpha AXP processor, there is not such a mode. There is a special area (also called KSEG) which is where the kernel resides.

References

[1] D. Rusling, *The Linux Kernel*, pp. 15-33, LDP.

[2] Galvin, Silberschatz, *Operating Systems Concepts*, pp. 239-329, pp.647-741, Adisson-Wesley.