Architecture et Système

Stefan Schwoon

Cours L3, 2015/16, ENS Cachan
January 5, 2016

Sharing and copy-on-write

Virtual memory isolates each process, which can therefore not influence other processes.

Sometimes, shared memory is still desirable, e.g. libraries used by multiple processes: OS can make several virtual pages point to the same real page.

Copy-on-write: multiple virtual pages with the same content: multiple virtual pages point to the same real page. When one of the pages is written to, the page will be duplicated in real memory and the virtual memory translation adapted. Applications: fork, calloc, . . .

Swap space

Virtual memory may be bigger than real memory (in fact, it usually is. . .)

Combined memory usage of all processes may exceed available real memory.

Solution: store “rarely” used pages on hard-drive, mark virtual page entries appropriately.

Access to a “swapped” page results in an interrupt/exception handled by the OS, which ensures that the page will be loaded.

Swapping

Conclusion from previous slides: Memory access is not “constant-time”!
Multiple levels of caching: L1, L2 (in the CPU), TLB (in the MMU), copy-on-write, swapping, . . .

$\Longrightarrow$ caching algorithms important on multiple levels.

Problem in general: store “frequently used” objects (from a large set) in limited space.

Algorithms: FIFO (bad!), Second Chance, NRU, LRU, . . .

FIFO caching

Assume a fixed number of slots available for storing pages.

When a page is requested that is already in the cache, return it directly.

Otherwise, load the requested page from (slower) memory, and replace the oldest slot in the cache by the newly requested page. (page fault)

Suffers from Belady's anomaly: More slots can mean more page faults.

> Example: Try the sequence 3 2 1 0 3 2 4 3 2 1 0 4 with three, then with four slots.

Second chance / Clock

Second chance: Modification/improvement of FIFO: Equip each slot with a “reference” bit that is set to 1 each time the page is requested.

On page fault: Throw out the oldest page if its reference bit is 0. Otherwise, set its reference bit to 0 and move it to the end of the list.

If all pages have been referenced, the oldest version will still get thrown out.

Clock: Implementation variant of Second Chance with circular list, with pointer to next free slot / oldest page.

LRU / NRU

Least recently used: Keep time of last access, on page fault throw out the page that was not accessed for the longest time.

Certain optimality results in terms of page fault ratio, but expensive.

Not recently used: Keep two bits with each page (referenced/modified); referenced bit is reset periodically.

Four classes of pages: 00 = not referenced/not modified, 01 = not referenced/modified etc, on page fault throw out a page from the lowest class possible.

Compromise between optimality and performance (w.r.t. LRU).

Shared memory

Unix provides a shared-memory mechanism, i.e. processes may share a segment of their memory with other processes.

Two steps (see example program):

Set up shared memory segment (using `shmget`).
Interface similar to file system, *key* acts as identifier.

Integrate the shared segment into virtual memory of the process (using `shmat`).

Communication over network

Network Communication: Crash course

We shall briefly discuss an example how to establish a TCP/IP connection over the network.

Model: Client/server structure

server establishes a services at a given port, waits for clients

(multiple) clients can connect to the port and communicate with the server

Adressing in IP: ports

In the IP protocol, an Internet address consists of a *machine address* and a *port*.

A port can be thought of as fine-grained addressing:
each machine can have 2^{16} of them.

Ports 0..1023 reserved for special use, certain ports pre-defined.

E.g., 80 for HTTP protocol.

Server

The server makes the following steps (see example program):

1. Create a TCP socket (using `socket` call).

2. Connect the socket to an IP port (using `bind`).
Note: The port can be freely chosen, but numbers up to 1024 are usually reserved for system services. The client must know which port to connect to.

3. Switch the socket to *listen* mode (using `listen`).

4. Wait for a client to connect (using `accept`).
Note: accept returns a file descriptor used for exchanging data with that client.

Note that step 4 can be repeated to accept multiple clients. For each client, a separate file descriptor is created.

Client

The client makes the following steps:

1. Create a TCP socket (like the server).

2. Connect the socket to the correct port on the machine where the server is running (using `connect`).

Connect returns a file descriptor for exchanging data with the server.

Serving multiple clients

How can a server communicating with several clients at a time?

Problem: The server does not know in advance which client will send the next piece of data.

`read` waits patiently till the next piece of data arrives - but the server will block if nothing arrives from that client!

Solutions:

Create a child process (or thread) for each client.

Use the `select` system call to find a file descriptor where data is available.