Operating Systems

Tutorial 1

Michael Tänzer

os-tut@nhng.de
http://os-tut.nhng.de

18th January 2011

Outline

True or False

- An alias is an virtual address that points to different physical addresses at different points in time

Describe the necessary steps to handle a page fault in an application's address space

1. Check whether access is legal (if not kill application)
2. Find and allocate a free frame (if there is none invoke page replacement algorithm which selects a frame and possibly writes it's contents to disk)
3. Fill the frame with the right contents (e. g. zero it out or retrieve previously swapped out page)
4. Adapt the page table (insert the new mapping and set the valid bit)
5. If using a software-managed TLB put the new mapping in there
6. Invoke the scheduler to decide which process to run next

In some systems the pager tries to always to have some free page frames in spare.

What is the basic idea behind such pagers?

- No need to write old contents to disk on page fault

⇒ Lower latency on a page fault

- Pager periodically writes some dirty pages to disk and marks them as clean

+ Bulk transfers to disk are more efficient

Global vs. local page replacement strategies

Global Page Replacement Algorithm

- Operates on all frames of all address spaces in the system
- When an application needs another frame it potentially gets one that belonged to another application

+ Adaptsto changing memory demands from applications

– Big set of frames to select from $\Rightarrow$ potentially slow

Local Page Replacement Algorithm

- Operates only on the frames of the current address space

+ Guaranteed number of frames available per application

– Difficult to select the optimal number of frames per application

What is thrashing?

When does it occur?

- Extremely high paging activity of a process (or the entire system)
- Occurs if the number of frames allocated to a process is too small even for it’s basic execution

Example

- Process with two pages, one for the stack and one for the code
- Only one frame assigned to the process and local page replacement strategy
- Every access to the stack causes the code page to be written to disk and stack page to be loaded
- On code access vice versa $\Rightarrow$ Process hardly makes any progress

What is the working set of a process?

How can it be used to prevent thrashing?

- Set of pages that had been accessed in the last Δ page references
- The OS needs to make sure the current working sets of all processes $\leq$ total number of frames
- If that condition doesn't apply some processes are selected and completely swapped out until it's possible to resume them again

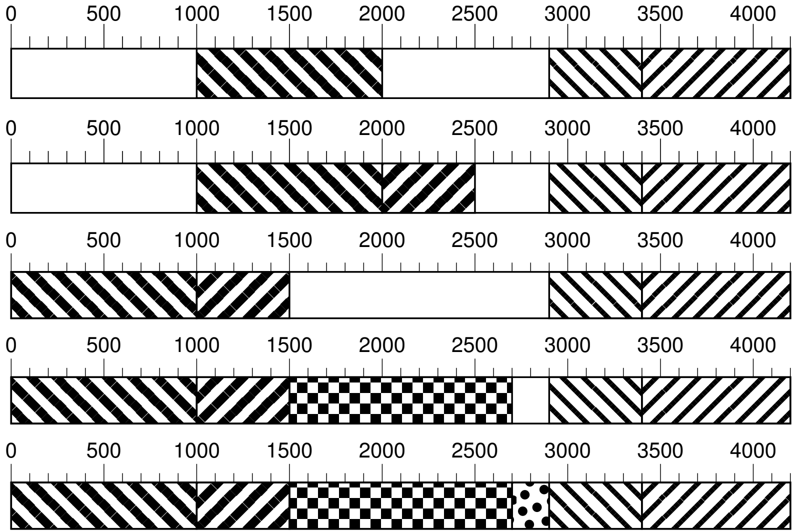
What page frame will be replaced if a page fault to page 4 occurs?

frame	virtual page	load time	access time	referenced	modified
0	2	60	161	0	1
1	1	130	160	0	0
2	0	26	162	1	0
3	3	20	163	1	1

Strategies

- First-In First-Out (FIFO)
- Least Recently Used (LRU)
- Clock, assume the circular buffer is ordered by loading time and next-frame points to frame 3
- Optimal algorithm, following references: 4, 0, 0, 0, 2, 4, 2, 1, 0, 3, 2

Allocate 500, 1200, 200 Blocks according to *best fit*



Linked lists vs. bit maps

- 128MB (2^{27} bytes) of physical memory
- Units of *n* bytes
- Each node in a linked list contains
  - Memory address
  - Length
  - Pointer(s) to the next (and previous) node
  - Each entry 32 bit

How many bytes are required for the three kinds of control structures (bit maps, singly and doubly linked lists) assuming the cases no memory allocated, all memory allocated and worst case?

What are the basic methods for accessing a file?

Sequential Access

- File is accessed one record/byte after the other
- Read operation reads the next *n* records and avances the current position in the file accordingly
- Writes are appended to the end of the file

Direct Access

- Allows to read and write the records/bytes in any order
- Programmer has to specify explicitly which record to read/write

List two ways how the OS can determine which programme to run when clicking on a file icon

- ‘Remember’ which programme created the file
  - Can be stored in the file header or some other attached meta data store
  - Breaks when switching to another programme (e. g. Photoshop to GIMP)
- File type indicates the programme to run
  - File type can be determined by using file extensions (Windows) or magic numbers (Unix)
  - Look up suitable applications for this type in a table
  - If two applications use the same extension but a different file format almost certainly always the wrong one is started (Murphy’s law)

Relative vs. absolute path name

Absolute Pathname	starts at the root of the directory structure
Relative Pathname	starts at the current directory, the current directory is usually process specific

Symbolic vs. hard link

Symbolic Link

- A special kind of file which contains the path to the file it links to
- Path can be absolute or relative
- Can easily be identified as a link

Hard Link

- Directory entry that refers to the file data of the original file
- Indistinguishable from the original file
- Only possible within the same file system

What is an access control list (ACL)?

What problems can occur within ACLs, and how can they be solved?

- A list associated with a file which contains entries in the form `username: access rights`

+ Allows for fine-grained control which user is allowed to perform which operations on the file

– ACLs can become quite large especially on systems with many users

– High management overhead, when a new user is added to the system he has to be added to the ACL of *every* file he should have access to

$\Rightarrow$ Allow to groups in ACLs

Discuss kernel data structures required for a Unix-like handling of open files

- View on open files local to each process

⇒ On `open` a new entry is added to the process-local open file table in the PCB

- A new entry in a global open file table is allocated and assigned to the entry in the local open file table
- An entry in the global open file table contains mapping to a `vNode` stucture (virtual representation of the actual file) and meta data like seek position and access rights
- Every `open` crates a new entry in the global open file table even if they point to the same `vNode` (might for example have a different seek position)
- Multiple local open file table entries might point to the same global open file table entry (e. g. after `fork` or `dup`)

What's the name of the system calls that move the current file position in Windows and Unix

Unix:	`lseek`
Windows:	`SetFilePointer` and `SetFilePointerEx`

How do these syscalls differ?

`off_t lseek(int fd, off_t offset, int whence)`

- Sets seek position in the file referenced by `fd` to `offset` and returns the absolute seek position (or -1 on error)
- `whence` specifies how `offset` is interpreted

`SEEK_SET`	Absolute adressing
`SEEK_CUR`	`offset` is added to the current seek position
`SEEK_END`	`offset` is added to the end of the file

- It's possible to set the seek position after the end of the file
- If data is written to that point there will be a gap in between (no disk space occupied and reads will return zeros) until data is actually written to it

How do these syscalls differ? I

```
DWORD SetFilePointer(HANDLE hFile,
LONG lDistanceToMove,
PLONG lpDistanceToMoveHigh,
DWORD dwMoveMethod)
```

- Sets file pointer of the file referenced by `hFile` is to `lDistanceToMove` and returns the new value of the file pointer
- To specify bigger values than fit in a `LONG` `lpDistanceToMoveHigh` can contain a pointer to the high order 32 bit (the new file pointer high order bits are also returned via this pointer). Pass `NULL` if not used

How do these syscalls differ? II

- `dwMoveMethod` specifies the starting point for the file pointer move

`FILE_BEGIN`	Start of the file (like `SEEK_SET`)
`FILE_CURRENT`	Current file pointer (like `SEEK_CUR`)
`FILE_END`	End of the file (like `SEEK_END`)

- On error the return value is `INVALID_SET_FILE_POINTER` but this is a valid value

$\Rightarrow$ Need to call `GetLastError()` to determine whether an error occured

Discuss alternative implementations without such an additional system call

- Keep the state in the application instead of the kernel

$\Rightarrow$ Add a parameter to the access functions (`read()`, `write()`, etc.) in which the application passes the seek position for each call

+ Kernel needs to store less information

+ One syscall less $\Rightarrow$ less code and errors inside the kernel, less overhead on random access

– One more parameter passed to often called functions even though most files are accessed sequentially $\Rightarrow$ need to validate that parameter

– Each application needs to implement handling of that state $\Rightarrow$ could be done in a library

Is the syscall `open` absolutely necessary?

Consequences for not having it

- Again we could add additional parameters to the access methods

⇒ Same arguments apply

– Access methods have to do all the error checking which would normally be done by `open()`

⇒ On every access the kernel would have to look up the file in the directory structure and check the access rights

– If a file is renamed between the operations it can't be found

⇒ As the state can't be extracted as easily as with the seek position it only pays off when accessing many files and only doing few operations on the same file

What is the purpose of the VFS layer in an OS?

- Abstraction from specific file system implementations

⇒ Kernel subcomponents (and possibly applications) which use file services can use any of the VFS file systems without modification

- Doesn’t matter whether the file system is a normal file system (e. g. ExtX, ReiserFS) on a disk, ramdisk or a network file system (e. g. NFS, SMB)

Discuss potential drawbacks of using a VFS

- Lower performance through indirection. Instead of a direct call to a function of which the address can be determined at compile time the functions for the specific file system are called by following a function pointer in the VFS data structure (e. g. `vnode->vn_ops->vop_read()`)
- Special features and optimisations on some file systems (e. g. meta data channels) can't be used

⇒ To use them one could call these special functions directly ⇒ destroys portability, but at least the standard tasks don't have to be ported for every file system

Which layer (VFS or underlying FS) resolves hard respectively symbolic links?

Hard Links

- Are simply directory entries

⇒ They are implicitly resolved by the FS without it even "noticing" that it did

Symbolic Links

- Reference other files by their absolute or relative name

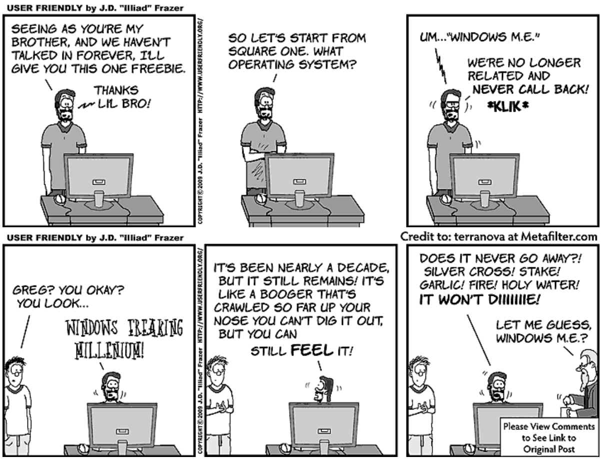
⇒ May cross file system borders

⇒ Has to be implemented at the VFS layer

Questions & Comments

Any questions or comments?

The End


USER FRIENDLY by J.D. "Illiad" Frazer
SEEING AS YOU'RE MY BROTHER, AND WE HAVEN'T TALKED IN FOREVER, I'LL GIVE YOU THIS ONE FREEBIE.
THANKS LI'L BRO!
SO LET'S START FROM SQUARE ONE. WHAT OPERATING SYSTEM?
UM..."WINDOWS M.E."
WE'RE NO LONGER RELATED AND NEVER CALL BACK!
KLIK
COPYRIGHT © 2009 J.D. "Illiad" Frazer HTTP://WWW.USERFRIENDLY.ORG/
USER FRIENDLY by J.D. "Illiad" Frazer
Credit to: terranova at Metafilter.com
GREG? YOU OKAY? YOU LOOK...
WINDOWS FREAKING MILLENIUM!
IT'S BEEN NEARLY A DECADE, BUT IT STILL REMAINS! IT'S LIKE A BOOGER THAT'S CRAWLED SO FAR UP YOUR NOSE YOU CAN'T DIG IT OUT, BUT YOU CAN STILL FEEL IT!
DOES IT NEVER GO AWAY?! SILVER CROSS! STAKE! GARLIC! FIRE! HOLY WATER! IT WON'T DIIIIIIE!
LET ME GUESS, WINDOWS M.E.?
Please View Comments to See Link to Original Post

from UserFriendly