Data Structures Start and Virtual Memory

Data Structures - Part 1

Primitive Types

1

Primitive Types

- The primitive data types include byte, int, long, short, float, double, and char. …
- Non-primitive are the more sophisticated and are built upon primitive types.
- Basically we build complex types on top of primitive types
- Primitive types are built from bits

2

Bit

- Tiniest data structure
- Can be on or off
  - 1 or 0
- Fundamental building block of all data types
- Bits are used to build binary numbers

3

Binary Numbers

- Build from a collection of bits
- Count the places from right to left
- We start counting from place zero
- Multiply each digit by 2 to the power of its place **number**
- For example:
  - If place #1 from the right is a 1, you would multiple 1 by 2 to the power of 1 and get 2
  - if place #2 from the right is a 1, you would multiply 1 by 2 to the power of 2 to get 4
- Example: 0b110
  - Place #0, is 0,
  - Place #1 is 1, so we multiple 1 by 2 to the power of 1 and get 2
  - Place #2 is 1, so we multiple 1 by 2 to the power of 2 and get 4
  - 4 + 2 = 6
- Example: b1000
  - Place # 3 is 1, so we multiple 1 by 2 to the power of 3 and get 8

- https://www.wikihow.com/Read-Binary

4

“Byte”

- 8 bits
- Bytes store numbers
- We use numbers to represent everything else
- Numbers are stored, ultimately in binary
- Example:
  00000001 is 1
  00000100 is 4
  00001100 is 12

5

Nybble / nyble/nibble

- Half a byte, 4 bits
- Image by sipa from Pixabay

6

Byte

- A byte can contain a number
- A byte can contain a character
- Byte = 25;
- Byte = ‘A’
- What if we want a word?

7

Array of bytes

char bytes[5] ;

bytes[0] = ‘M’

Bytes[1] = ‘I’

Bytes[2] = ‘K’

Bytes[3] = ‘E’

Bytes[4] = 0

(Remember , computers like to start counting from zero)

8

Big Numbers

- Integers
- What if we want a number bigger than the largest value a byte can hold
- Integer types:
  - Short - 2bytes (packaged as 1)
  - Int - (4 bytes packaged as 1)
  - Long (8 bytes packaged as 1)

9

Hex Numbers

Decimal	Binary	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

10

Ascii

- [extended ascii chart - Barta.innovations2019.org]

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~

11

Virtual Memory

SEIS-610

A small taste!

12

- Paging and Page Replacement
- Many of these slides came from
- **http://pages.cs.wisc.edu/~mattmcc/cs537/notes/Replacement.ppt**
- **Thank you to:**

Matt McCormick
1302 Computer Science Building
University of Wisconsin - Madison
1210 West Dayton Street
Madison, WI 53706

13

Paging

- In general..
  - Computer Memory is broken up into pages
  - A page can be in physical memory (a frame)
  - A page can be in persistent storage (our disk)
- Since somethings can be in physical memory and some things can be on a disk, we have the illusion of unlimited memory.

14

Paging

- If a page is not in physical memory (frame)
  - find the page on disk
  - find a free frame
  - bring the page into memory
- What if there is no free frame in memory?

15

Page Replacement

- Basic idea
  - if there is a free page in memory, use it
  - if not, select a *victim* frame
  - write the victim out to disk
  - read the desired page into the now free frame
  - update page tables
  - restart the process

16

Page Replacement

- Main objective of a good replacement algorithm is to achieve a low *page fault rate*
  - ensure that heavily used pages stay in memory
  - the replaced page should not be needed for some time
- Secondary objective is to reduce latency of a page fault
  - efficient code
  - replace pages that do not need to be written out

17

Virtual Memory -- Paging

- Remember –
  - Virtual Memory can give the illusion of large memory space.
  - Virtual Memory, the entire program does not need to be in volatile memory
- Once again, what if a page is not in physical memory???
  - find the page on disk
  - find a free frame
  - bring the page into memory

18

Page Replacement

- REMEMBER
  - Frames – Physical
  - Pages – Virtual
- Idea
  - if there is an open frame in memory, use it
  - if not, select a *victim* frame
  - write the victim out to disk
  - read the desired page into the now free frame
  - update page tables
  - restart the process

Reference String

- Reference string is the sequence of pages being referenced
- If user has the following sequence of addresses
  - 123, 215, 600, 1234, 76, 96
- If the page size is 100, then the reference string is
  - 1, 2, 6, 12, 0, 0
  - Divide by 100
- Pretending the page number is the first digits
- I will just give you reference strings, not addresses

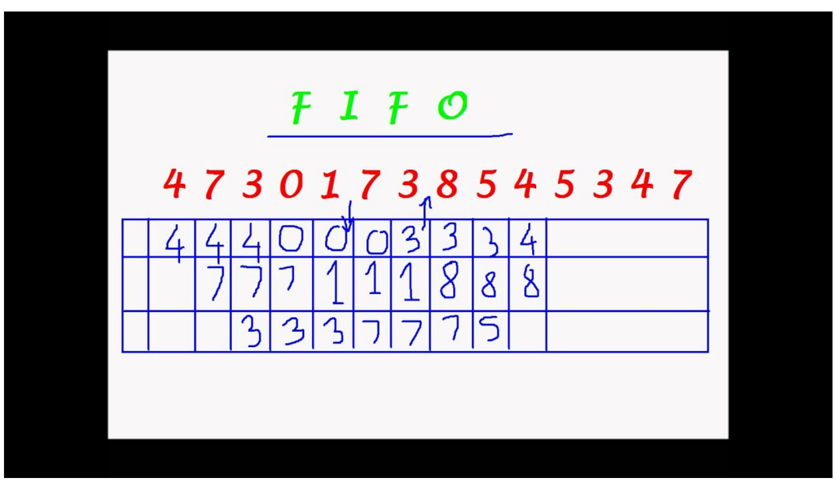
First-In, First-Out (FIFO)

- The oldest page in physical memory is the one selected for replacement
- Very simple to implement
  - keep a list
    - victims are chosen from the tail
    - new pages in are placed at the head

21

First-in, First Out Example

https://www.youtube.com/watch?v=KejyTiATz18



22

FIFO Issues

- Poor replacement policy
- Evicts the oldest page in the system
  - usually a heavily used variable should be around for a long time
  - FIFO replaces the oldest page - perhaps the one with the heavily used variable
- FIFO does not consider page usage

23

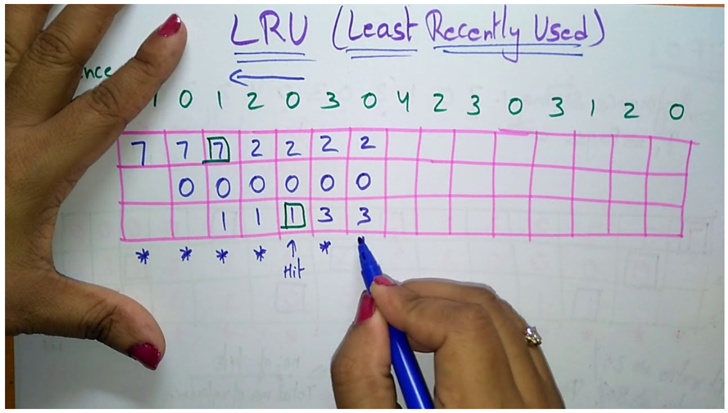
Least Recently Used (LRU)

- Basic idea
  - replace the page in memory that has not been accessed for the longest time
- Optimal policy looking back in time
  - as opposed to forward in time
  - fortunately, programs tend to follow similar behavior

24

Least Recently Used

https://www.youtube.com/watch?v=u23ROrlSK_g



25

Dirty Pages

- If a page has been written to, it is *dirty*
- Before a dirty page can be replaced it must be written to disk
- A *clean* page does not need to be written to disk
  - the copy on disk is already up-to-date
- We would rather replace an old, clean page than and old, dirty page

26

- The end

27