CSE 351 Section 8 - Processes and Virtual Memory

Welcome back to section, we're happy that you're here ☺

Fork and Concurrency:

Consider this code using Linux's fork:

```
int x = 7;
if( fork() ) {
    x++;
    printf(" %d ", x);
    fork();
    x++;
    printf(" %d ", x);
} else {
    printf(" %d ", x);
}
```

Write all four of the different possible outputs (i.e. order of things printed) for this code?

Tip: try drawing a process graph for this program

Exercises:

- 1) Name three specific benefits of using virtual memory:
- 2) What should happen to the TLB when a new entry is loaded into the page table base register?
- 3) Fill in the formulas below using *descriptions*, not variables:

Page offset bits = $\log_2($ _______)

Virtual address bits = _______ + page offset bits

Physical address bits = physical page number bits + ______

Virtual page number bits = $\log_2($ ________)

Entries in a page table = _______

4) Fill in the following table:

VA width (n)	PA width (m)	Page size (P)	VPN width	PPN width	Bits in PTE (assume V, D, R, W, X)
32	32	16 KiB			
32	26			13	
	32		21		22
		32 KiB	25		26
64			48		29

5) **Processor:** 16-bit addresses, 256-byte pages

TLB: 8-entry fully associative with LRU replacement

• Track LRU (shown in decimal) using 3 bits to encode the order in which pages were accessed, with 0 being the most recent

At some time instant, the TLB for the current process is in the initial state given below.

Assume that all page table entries that are not in the initial TLB have read and write permissions, but no execute permission (i.e. R = 1, W = 1, X = 0).

• OS will assign new pages starting at PPN 0x20, with read and write permissions but no execute permission (i.e. R = 1, W = 1, X = 0).

Fill in the final state of the TLB according to the access pattern below. For each access, indicate if it leads to a:

a) TLB hit? b) TLB miss? c) Page fault? d) Protection fault?

Initial TLB:

TLBT	PPN	Valid	R	W	X	Dirty	LRU
0x01	0x11	1	1	1	0	1	0
0x02	0x18	1	1	0	0	0	6
0x10	0x13	1	1	1	1	1	1
0x20	0x12	1	0	1	0	0	5
0x00	0x00	0	0	0	0	0	7
0x11	0x14	1	1	0	0	0	4
0xAC	0x15	1	1	0	0	0	2
0x34	0x16	1	1	1	0	1	3

Page Table (partial):

	•	\1
VPN	Valid	PPN
0x0	0	0x00
0x1	1	0x19
0x2	1	0x18
0x3	1	0x17
0x4	0	_
0x5	0	_
0x6	1	0x1A
0x7	0	_

VPN	Valid	PPN				
0x8	1	0x1C				
0x9	1	0x1D				
0xA	0	0x1E				
0xB	1	0x1F				
0xC	0	_				
0xD	1	0x09				
0xE	0	_				
0xF	1	0x1B				

Access pattern:

1) Read 0x11F0 2) Write 0x0301 3) Write 0x20AE 4) Write 0x0532 5) Read 0x0E15 6) Write 0xACFF

Final TLB:

TLBT	PPN	Valid	R	W	Х	Dirty	LRU