Name: Amit Nijjar Pid: A11489111 email: a2nijjar@ucsd.edu

CSE 120 Homework #3
Spring 2016 (Kesden)

1. 1-level Page Table and Shared Memory

Consider two processes running on a UNIX-ish system that implements virtual address space but not demand paging. These processes communicate using a shared memory area mapped within the heap of each process.

Please complete the diagram on the following page that depicts the physical memory of the system, as well as the virtual memory and page table for each process.

You may locate the items in any legal location in physical memory. Please follow the virtual memory convention used in class (code at the bottom, stack at the top).

Please label the diagram to identify the contents of each page and frame and draw arrows depicting the relationship between each occupied virtual page, page table entry, and physical frame.

The system is configured as follows:

- Physical Address space: 64KB
- Virtual Address space: 32KB
- Page Size: 4KB

The processes are configured as follows:

Process 1:

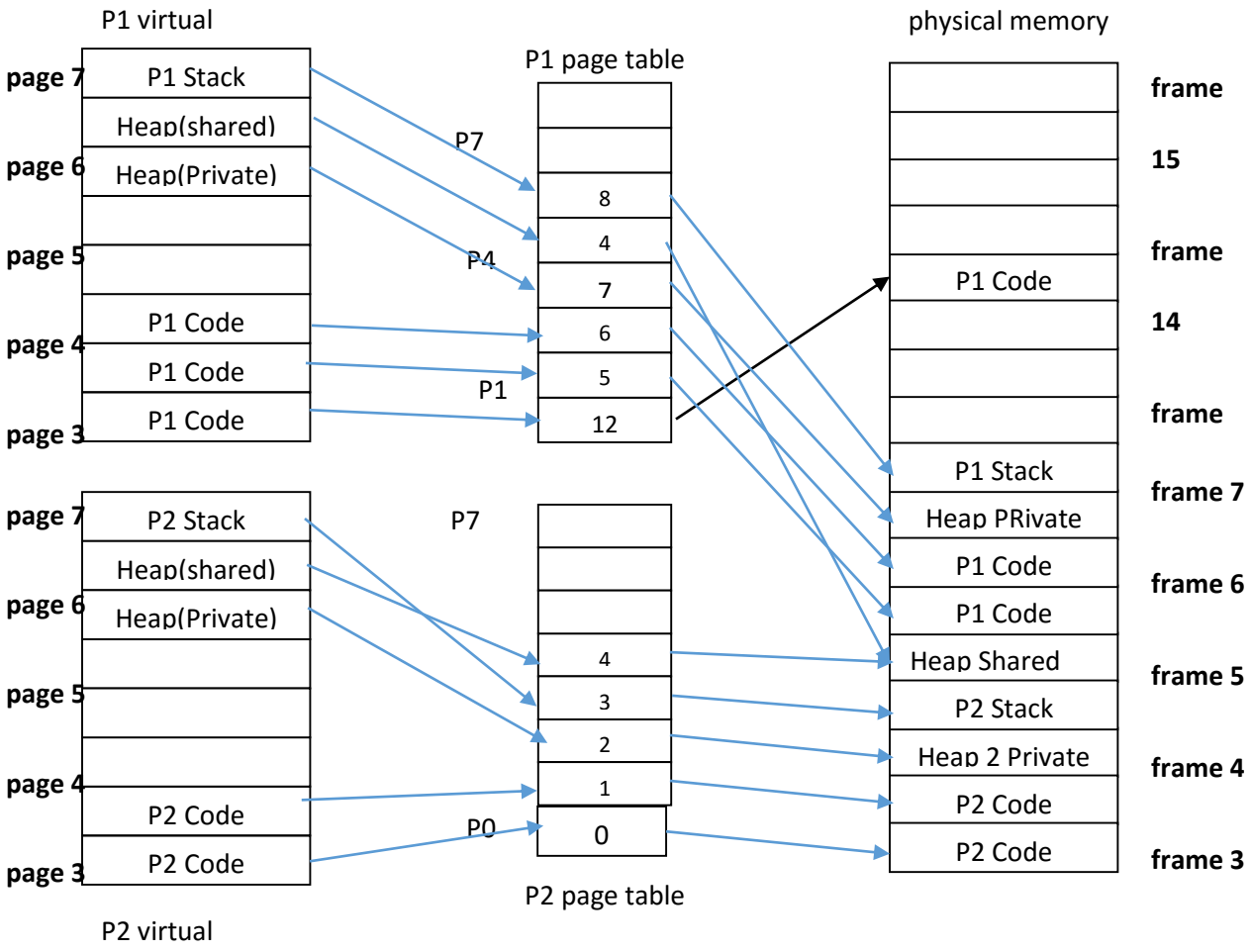
- Code: 12KB
- Heap (Shared): 1KB
- Heap (Private): 4KB
- Stack: 3KB

Process 2:

- Code: 8KB
- Heap (Shared): 1KB
- Heap (Private): 3KB
- Stack: 2KB

Name: Pid: @ucsd.edu email:

1. 1-level Page Table and Shared memory (cont.)



Name: Amit Nijjar Pid: A11489111 email: a2nijjar@ucsd.edu

2. 1-level vs. 2-level Page Tables

Consider a computer with 64Kbyte of physical memory and a frame size of 8Kbytes. The virtual memory has 8 pages.

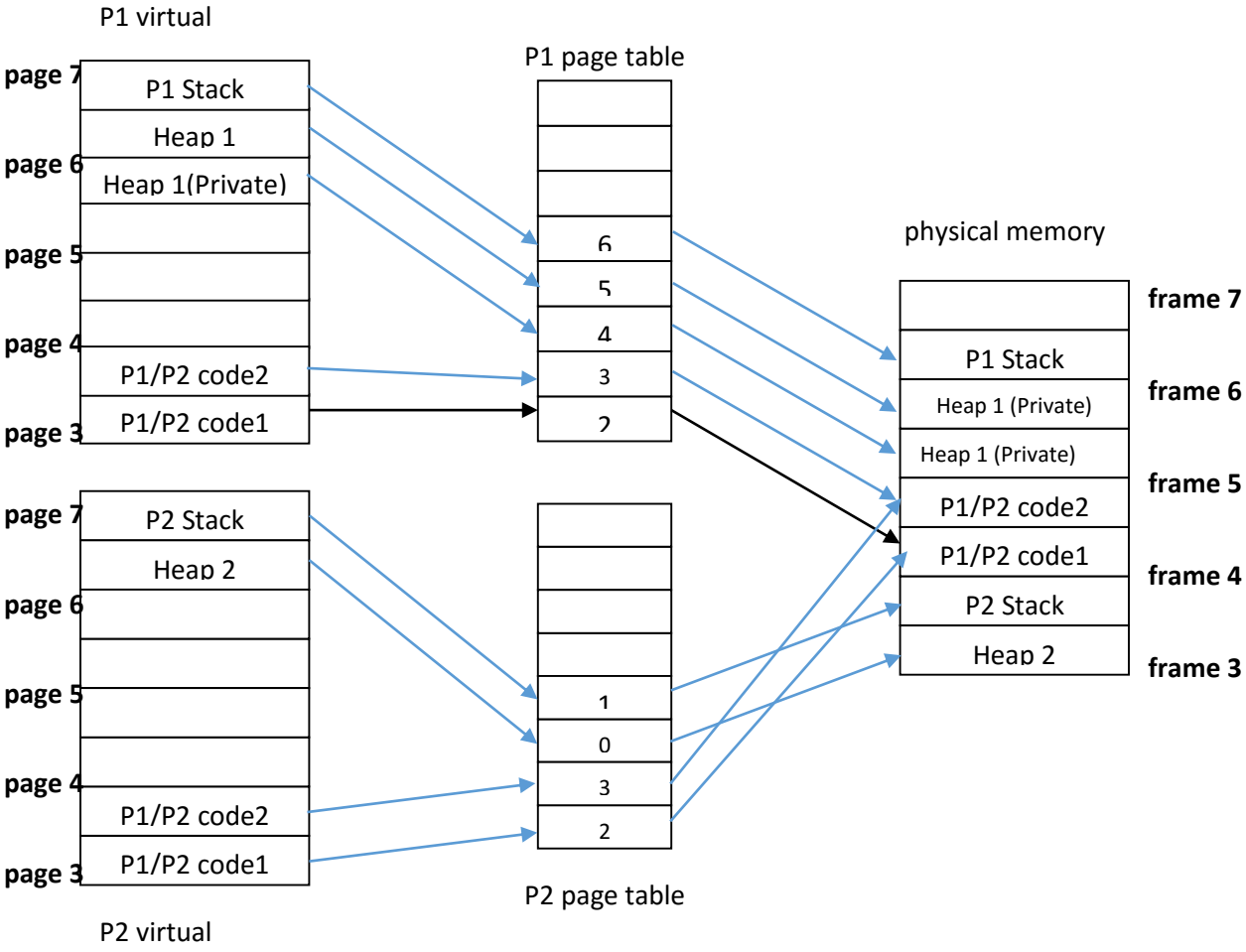
There are two processes (P1 and P2) running from the same code segments on this computer. P1's stack size is 7 Kbytes, its heap size is 12Kbytes, and its code size is 15Kbytes. P2's stack size is 8Kbytes and its heap size is 4Kbytes. The code segment is shareable between the processes.

System:
Physical Address Space: 64K
Virtual Address Space: 8K x 8 = 64K
Page Size: 8K

Process 1:
Code (Shared): 15K
Heap Private: 12K
Stack: 7K

Process 2:
Code (Shared): 15K
Heap Private: 4K
Stack: 8K

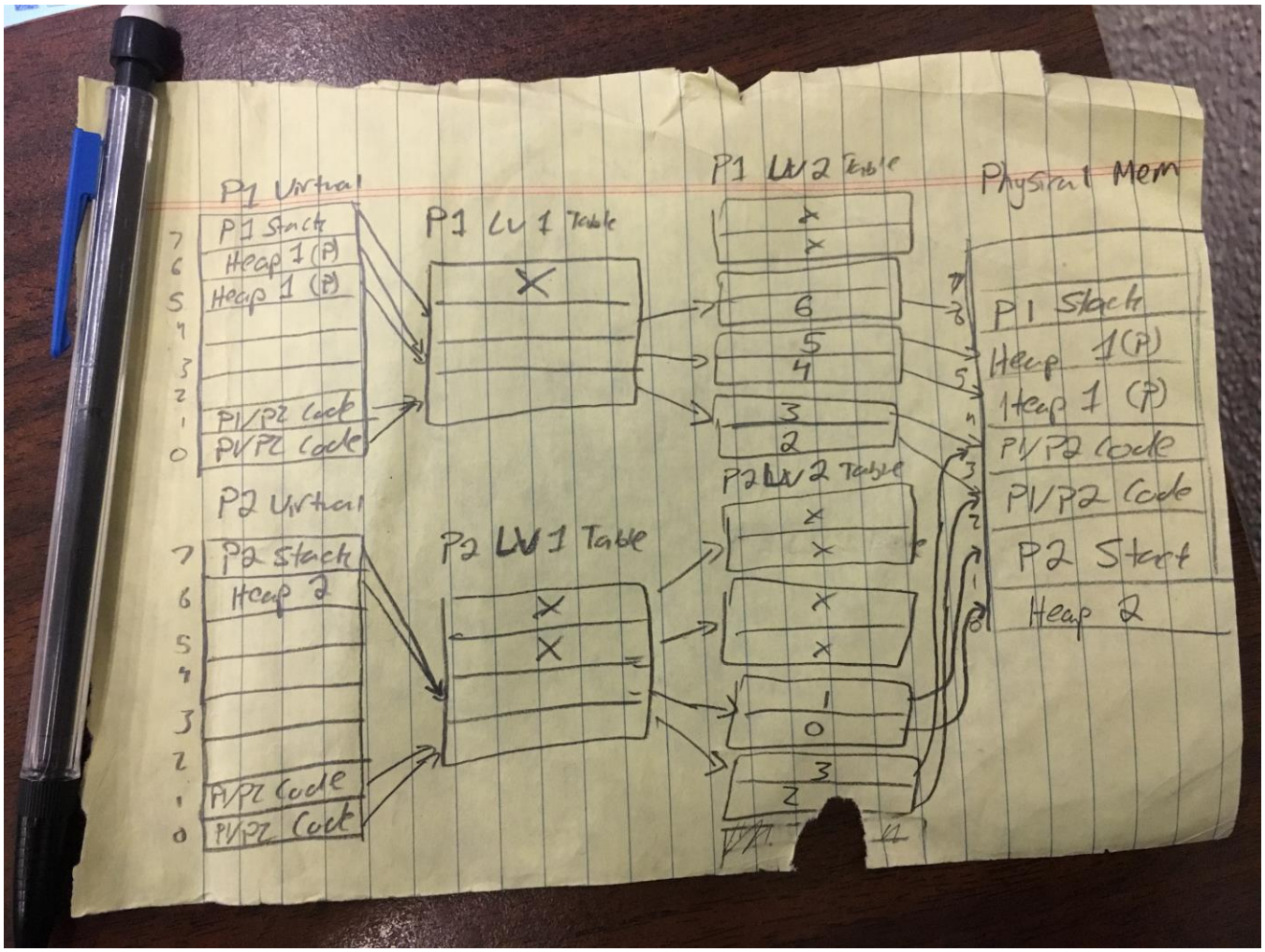
a) Assuming the memory architecture uses a single-level page table scheme, complete the schematic diagram showing the page tables of the two processes. Give appropriate values to the page table entries and draw arrows showing this association.



Name: Amit Nijjar Pid: A11489111 email: a2nijjar@ucsd.edu

2. 1-level vs. 2-level Page Tables (cont.)

b) Now assume the machine uses a 2-level paging structure, where the level-1 page table contains 4 entries. Complete the schematic diagram below. You need to draw the 2-level paging structure, fill out the page table, and sketch the arrows showing the association between virtual address and page table, and between page table and physical memory.



c) Suppose that P1's heap size increases by 3 Kbytes. Please show how your solution to part b would change. Please ensure that both the original answer and the modified answer are visible and labeled.

There would be no change. The pictures are the same because the heap will still fit on 2 pages, there is no need to add another page.

Name: Pid: @ucsd.edu email:

3. Bits vs Pages [Crowley via Voelker]

Suppose we have a computer system with a 44-bit virtual address, page size of 64K, and 4 bytes per page table entry.

a. How many pages are in the virtual address space? (Express using exponentiation.)
2^44 / 64K = 2^44 / 2^16 = 2^28

b. Suppose we use two-level paging and arrange for all page table pages (both master and secondary) to fit into a single page frame. How will the bits of the address be divided up?
First level: 64K/4 = 16K = 2^14 therefore first level paging length = 14
Second level: 64K/4 = 16K = 2^14 therefore second level paging length = 14
44-14-14 = 16 bits

c. Suppose we have a 4 GB program such that the entire program and all necessary page tables (using two-level pages from above) are in memory. (Note: It will be a *lot* of memory.) How much memory, in page frames, is used by the program, including its page tables?
Pages used by program 4G/64K = 2^16 page frames
Pages used by second page table 2^(16/14) = 4

Pages used by first level page 4 / 2^16 * 1

2^16 + 4 + 1

4. Memory Access

Consider a computer that uses a single-level paging scheme with no data cache. The hardware provides a 32 entry translation look-aside buffer (TLB), a.k.a. translation buffer (TB). The TLB read access time is 2 ns (nanoseconds) and TLB updates can be done in parallel with memory access. A memory access takes 60ns.

a) How long does a memory access take in the case of a TLB hit? A miss?

Hit => 2ns + 60ns = 62 ns

Miss => 2ns + 60ns (lookup) + 60 (read) = 122ns

b) What is the *effective memory-access time* if we have a TB hit ratio of 95%? Effective memory-access time indicates the average amount of time a memory access takes (not the average lookup penalty).

62(.95) + 122(.05) = 58.9 + 6.1 = 65ns

c) What is the minimal hit ratio that will guarantee the effective access time of at most 70ns?

62(x) + 122(1-x) = 70
62x + 122 – 122x = 70

-60 x = -52
X = 52/60
X = 86.667%

d) If the processor runs at 3Ghz and the average instruction takes 3 clock cycles, how many instructions could run in the time it takes to recover from a single TLB miss?

Simplify to 1ghz with 1 clock cycle per instruction. 122 ns for a single TLB miss with 1ghz processor with 1 cycle per instruction means 122 instructions could run in that time.

Name: Amit Nijjar Pid: A11489111 email: a2nijjar@ucsd.edu

5. [Crowley via Voelker] Access Time

Suppose we have an average of one page fault every 20,000,000 instructions, a normal instruction takes 2 nanoseconds, and a page fault causes the instruction to take an additional 10 milliseconds.

(a) What is the average instruction time, taking page faults into account?
1 instruction is 2 ns, 1 page fault is 10,000 ns, frequency of page fault = 1/20,000,000
Page fault time 10,000/20,000,000 = .5ns per instruction
2+.5 = 2.5 ns average instruction

(b) Redo the calculation assuming that a normal instruction takes 1 nanosecond instead of 2 nanoseconds.
Same calculations as above
1 + .5 = 1.5 ns average instruction

6. Behavior of Basic Page-Replacement Algorithms

Consider a system configured as follows:

- The hardware page size is 256 bytes.
- The main memory is 4 frames (physical pages) large
- The main memory is initially empty.

Given the above configuration, determine the behavior of *OPT* and *LRU* replacement for the provided trace, i.e. sequence of virtual memory references made by a process. The 16-bit virtual address of each memory reference is shown in hexadecimal. You should read the trace sequence from left to right, and then to the beginning of the next line.

0100 0204 0308 04a8 0524 0628 072c 02cc 05c0 04d4
0304 02c8 0120 02bc 05d0 0430 05c0 0620 07ac 030c
04ac 02c0

a) What is the *reference string* of this trace (the *reference string* is the list of page accesses vs. address, accesses)?

Reference String: 1 2 3 4 5 6 7 2 5 4 3 2 1 2 5 4 5 6 7 3 4 2

b) For the three page-replacement algorithms, fill in the table on the next page with the content of main memory after each page reference in the reference string.

Use the following notations to mark each cell:

- "*": the content of the frame is invalid (empty).
- "X": the reference causes a page fault (PF).
- numeric value: the resident page number in the frame.

TABLE ON NEXT PAGE

Name: Amit Nijjar Pid: A11489111 email: a2nijjar@ucsd.edu

6. Behavior of Basic Page-Replacement Algorithm, *cont.*

REF	OPT					LRU				
	Frames				PF	Frames				PF
1	1	*	*	*	X	1	*	*	*	X
2	1	2	*	*	X	1	2	*	*	X
3	1	2	3	*	X	1	2	3	*	X
4	1	2	3	4	X	1	2	3	4	X
5	5	2	3	4	X	5	2	3	4	X
6	5	2	6	4	X	5	6	3	4	X
7	5	2	7	4	X	5	6	7	4	X
2	5	2	7	4		5	6	7	2	X
5	5	2	7	4		5	6	7	2	
4	5	2	7	4		5	4	7	2	X
3	5	2	3	4	X	5	4	3	2	X
2	5	2	3	4		5	4	3	2	X
1	5	2	1	4	X	1	4	3	2	X
2	5	2	1	4		1	4	3	2	
5	5	2	1	4		1	5	3	2	X
4	5	2	1	4		1	5	4	2	X
5	5	2	1	4		1	5	4	2	
6	6	2	1	4	X	6	5	4	2	X
7	6	2	7	4	X	6	5	4	7	X
3	3	2	7	4	X	6	5	3	7	X
4	3	2	7	4		6	4	3	7	X
2	3	2	7	4		2	4	3	7	X

7. [Voelker] Working Set

If many programs are kept in main memory, then there is almost always another program ready to run on the CPU when a page fault occurs. Thus, CPU utilization is kept high. If, however, we allocate a large amount of physical memory to just a few of the programs, then each program produces a smaller number of page faults. Thus, CPU utilization is kept high among the programs in memory.

What would the working set algorithm try to accomplish, and why? (*Hint*: These two cases represent extremes that could lead to problematic behavior.)

The working set algorithm is trying to keep the CPU busy. Both of these two cases keep the CPU utilization high, however, neither algorithm deals with thrashing. The algorithms are trying to efficiently handle multiple processes but does not handle thrashing well. The cpu will spend most of its time switching and working on page faults.

Name: Amit Nijjar Pid:A11489111 email: a2nijjar@ucsd.edu

8. Behavior of Basic Page-Replacement Algorithms

Consider the reference strings of the following processes. Assume that they are running concurrently on a multiprocessor system with 8 frames (physical pages) of RAM that uses global frame allocation, i.e. all processors and processes share the same main memory.

Please complete the table below showing how many pages you would allocate to each process at each point in time (memory reference). Please explain how you selected the initial number of frames to allocate to each process and how you determined when to steal frames from one process for use by another and how many to steal.

This process should be intuitive – we're not asking you to develop or apply any specific algorithm or policy. We're just looking for the optimal solution.

To reduce the amount of repetitive writing, you may choose to show the initial allocation to each process, and the allocation to each process only when the allocation changes for any process.

Reference Strings																											
P1	1	2	3	4	1	2	3	4	1	2	3	4	2	2	2	2	2	2	2	1	2	1	2	3	4	1	2
P2	5	6	5	6	5	6	5	6	5	6	5	6	5	7	8	9	7	8	9	7	8	7	8	7	8	7	8
P3	9	9	0	0	9	9	0	0	9	0	9	0	9	A	B	C	D	A	B	C	D	9	0	9	0	9	0
Frames Allocated																											
P1	1,2,3,4													2						1,2	1,2,3,4						
P2	5,6													7,8,9						7,8	7,8						
P3	9,0													A,b,c,d						A,b,c,d	D,9						

Name: Amit Nijjar Pid: A11489111 email: a2nijjar@ucsd.edu

10. [Silberschatz via Voelker] Performance Tuning

Consider a demand-paging system with the following time-measured utilizations:

- CPU utilization: 20%
- Paging disk: 97.7% (demand, not storage)
- Other I/O devices: 5%

For each of the following, say whether it will (or is likely to) improve CPU utilization. Briefly explain your answers.

a. Install a faster CPU

No, the page fault handler may run faster but the processes access data more often so CPU utilization may become worse

b. Install a bigger paging disk

No, more memory is needed to resolve the situation, not more storage to hold more pages

c. Increase the degree of multiprogramming

no, more processes makes the system thrash

d. Decrease the degree of multiprogramming

yes, closing applications is a solution to thrashing

e. Install more main memory

Yes, the best way to prevent thrashing

f. Install a faster hard disk, or multiple controllers with multiple hard disks

yes but it will only help a little bit, the disk is always slow compared to memory access

g. Add prepaging to the page-fetch algorithms

no, other processes are penalized for pre-paging. If pre-paging loads unnecessary pages, it makes the situation worse

h. Increase the page size

no, internal fragmentation aggravates thrashing. Small processes will take more time bringing in latger pages, thus wasting space