CS 180 *Quiz #5* Name GOH WEI ZHE

DigiPen Institute of Technology November 20, 2020

7.5/25

Complete all questions. Total marks is 25.

0

1. Consider the following statements. Re-arrange the statements as to reflect how a page of logical memory is loaded into physical memory. Do not copy out the statements. Using the alphabets to represent the statements, sort the order of events that occur. Note that while you are not supposed to use every single statement (some of the statements are plainly false), it should be a series of steps as detailed as possible given the statements below. (2 marks)

   (a) User explicitly requests for page to be loaded

   4 (b) Interrupt handler locates the page of logical memory in the secondary storage.

   1 (c) address within a page is accessed

   2 (d) page table entry of the page indicates that the page is not present

   3 (e) MMU hardware issues a hardware page fault

   (f) MMU function call issues a software trap

   5 (g) After copying the page into a physical frame, the page table entry of the page is updated.

   Answer: A C D G c, d, e, b, g

0 1 2 3 4 5 6 7 8 9 A 10 B 11 C 12 D 13 E 14 F 15

0x64000 PTE 4B 4B 4B

$3 \times 4 = 12 = C$

$= 0x6400C$

0

2. In a one-level paging scheme, we have 4KB pages and 32-bit logical and physical addresses. Given that the logical address is 0x00003408 and that the page table register value is 0x64000. Assuming that each page table entry is 32 bits, What is the address of the page table entry corresponding to the logical address? [2 marks]

3

0x00003 | 408

page number offset

20 bit 12 bit

4 byte

$4K = 2^{12}$ $32 - 12 = 20$

00003 | 408

$1K = 2^{10}$ page # = 3 C E

   (a) 0x64034

   (b) 0x64340

   (c) 0x64003

   (d) 0x64030

   (e) 0x6400c

2

3. Given the logical segmentation address (1, 200) and the segment table below, what is the translated physical address? All addresses, bases and limits are given in hexadecimal. [2 marks]

Segment Number	Base	Limit	Present
0	100	300	1
1	500	800	0
2	1300	200	1
3	1600	300	0

100 - 400

500 - 1300

1300 - 1500

1600 - 1900

if 1, 500 + 200 = 700

if 0, fault

   (a) 300

(b) 700

(c) 1500

(d) 1800

(e) The translation will cause a fault.

E

0

4. Given the logical paging address 0x00401234 and the page table below, what is the translated physical address? All numbers in the table are given in base 16. The page table is partially shown below. You should note that the logical paging address is 32 bits. The first 20 bits refer to the page number while the last 12 bits refer to the offset. [2 marks] MSB is page number

each digit is 4 bits

Page Number	Frame Number	Present
...	...	...
4	00506	0
...	...	...
40	00608	0
...	...	...
400	01000	1
401	03456	1
...	...	...

0040|234
page # | offset 12 bits

(a) 506234

(b) 506123

(c) 6081234

(d) 608234

(e) 10001234

(f) 1000234

(g) 34561234

(h) 3456234

(i) The translation will cause a fault.

03456 (most significant 20 bit) + offset = 03456|234

c

h

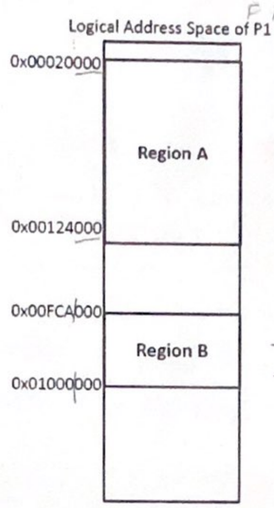
5. Consider the following diagram showing 2 memory regions that process P1 has in the logical address space. For this question, we assume that the size of physical memory address is 4 bytes, which is the same as the size of logical address, and the page size is 4KB. The page table entry includes the fields: frame number (20 bits, most significant bits), other fields (11 bits) and Present (1 bit, least significant bit).

12 bits offset

32 bits

0x104 = 0001/0000/0100

$2^8 + 2^2 =$



00124 − 00020 = 00104

0001 0000 0100

$2^8 + 2^2 = 260$

(a) How many pages are within region A? Please answer in base 10.[1 mark]

Answer: 26 ✗ 0 X124 − 0X20 = 0X104 = 260

(b) How many pages are within region B? Please answer in base 10. [1 mark]

Answer: ✗ 0x1000 − 0xFCA = 0x36 = 54

(c) Assuming it's 1-level paging, write down 1 page number each found in region A and region B respectively. Please answer in hexadecimal. [2 marks]

Answer: A: 0X0002$\cancel{0}$004 ✗ 0X00021 B: 0X00FCA004 0X00FCB

Any number between 0x20 − 0x124 (exclusive) and 0xFCA − 0x1000 (exclusive)

0X00021
0X00FCB

(d) Assuming it's 1-level paging, and the value of the page table register is 0x50000. Give the physical address of the page table entry of page 0x244B. Please answer in hexadecimal. [1 mark]

PTE +

Answer: 0X744B ✗ 0x50000 + 0x244B × 4 = 0x5912C

(e) Suppose that the MMU is able to translate the logical address 0x11B123 into 0x66123. What would be found in the most significant twenty bits of the page

0010 | 0100 | 0100 | 1011 | 00
2 4 4 B <<2
1001 | 0001 | 0010 | 1100
9 1 2 C = 5912C

logical physical

0x11B|123 → 0x66|123

page# frame #

0x66 = frame #

| frame # | other | P |

20B

PTE 20 bits = frame # ~~page #~~

table entry of page 0x11B? [1 mark]

0

Answer: Page number ✗ 0x00066

(f) Suppose during the translation of a logical paging address that the page table entry found is 0xFB1234. Comment on whether the page is found and why. [2 marks]

1

Present bit is 0. accessing

Answer: Not found. The page table entry is out of the page table ✗

6. Consider a memory manager managing a block of contiguous memory starting from address 1000 with 100 bytes size. Assuming that the memory manager accepts the following C functions and uses First Fit as it's allocation algorithm:

0xFB1|034 Present bit is 0

0010 0011 0100

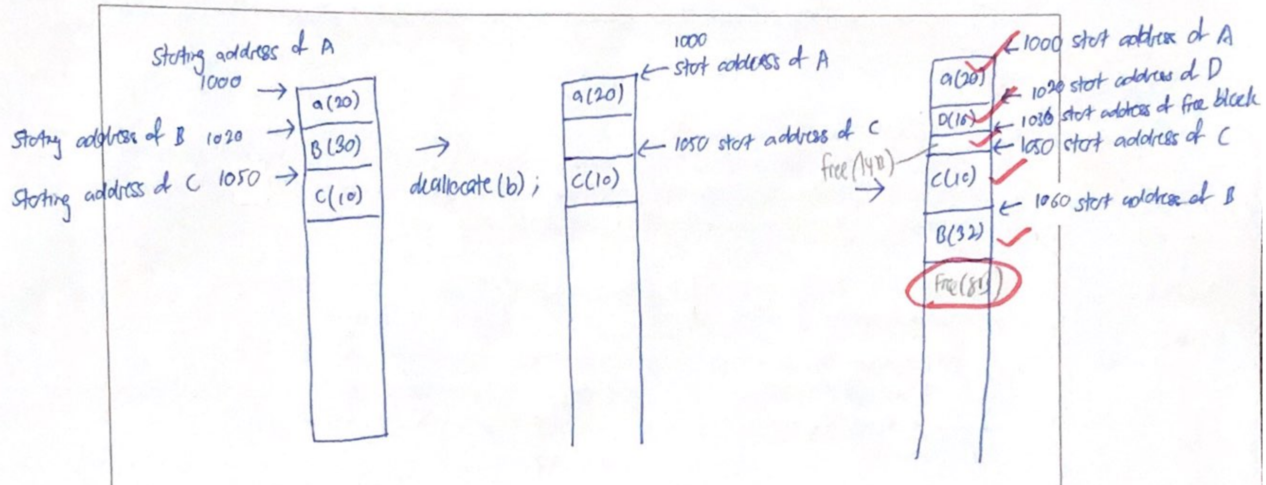
2 3 4

```
void *allocate(int num_of_bytes);
//returns the first address of the allocated block
void deallocate(void *addr);
//frees the allocated block starting at address addr.
```

The following shows a sequence of allocate and deallocate. Draw a diagram to indicate the allocated and free regions after this sequence of code is executed. The entire block is free at the beginning. Indicate the addresses clearly. You may write the addresses in base 10. [5 marks]

```
void *a, *b, *c, *d;
a = allocate(20);
b = allocate(30);
c = allocate(10);
deallocate(b);
b = allocate(32);
d = allocate(16);
```

4,5



Answer:

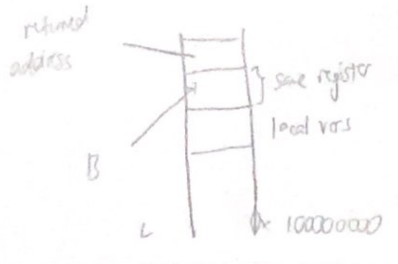
7. For the following questions, please refer to this C code. This program is compiled and executed on a paging system where each page is 4K in size. Both physical addresses and logical addresses are 64 bits in width.

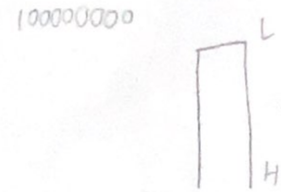
```
#include <stdio.h>

#define SIZE 0x10000
int A[SIZE];

int main()
{
  char B[SIZE];
  char *C;
  C = (char*) malloc(SIZE);

  printf("Address of A is %p\n", A);
  printf("Address of B is %p\n", B);
  printf("Address of B[0xFFFF] is %p\n", &B[0xFFFF]);
  printf("Address of C is %p\n", C);
  B[0x10005] = 19;
  free(C);
}
```

The printout of a process executing this program is the following:

```
Address of A is 0x1004071a0
Address of B is 0xfffecc10
Address of B[0xFFFF] is 0xffffcc0f
Address of C is 0x6000005d0
```

(a) Given that the stack of the process begins at logical address 0x100000000, what is the *minimum* number of pages we need to reserve for the stack? (In order to





store all the local variables required by the program.) [1 mark]

0x13|4f0 round up

0x14

0

Answer: 3 page ✗ 0x100000000 - 0xfffecc10 = 0x134f0 ⇒ 20 pages

(b) Given that the heap of the process begins at logical address 0x600000000, what is the *minimum* number of pages we need to allocate for the heap? (In order to store all the dynamically allocated) [1 mark]

$0x134f0 \div 2^{12}$

$= 0x13 + 1$

0

Answer: 3 page ✗ 0x6000005d0 + 0x10000 - 0x600000000 = 0x125d0 17 pages

(c) Explain why in the given program, when we overrun the buffer by a little (e.g., `B[0x10005]=19`) in the C code above, the program does not crash immediately because of array out of bounds access. Explain in terms of a paging system. [2 marks]

0

It does not crash immediately as ^it is accessing^ memory is located in physical address, however will have a segmentation fault.

The program will crash immediately if it is try to access a memory that is not allocated in the first place. ✗

Answer:

B[0x100005] is at a higher address than that of B[0xFFFF]. Address of B[0xFFFF] is 0xfffeccf and Address of B[0x10005] is 0xfffecd5. They are all on the page for the stack. Thus the program does not crash at all as eventually there will be physical frame to contain the whole page via demand paging