

# 05 VM - Virtual Memory Aneka Soal Ujian Sistem Operasi A. Wibisono (AW), C. BinKadal (CB) H. Kurniawan (HK)

© 2016 - 2023 — Rev: 44 - 23-Oct-2023. **URL:** https://rms46.vlsm.org/2/200.pdf. More can be accessed via https://os.vlsm.org/. This free document is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY, without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. You might change, reproduce, and distribute this document but not delete these provisions. This is the way!

## 1. 2016-1 (McGill Fall 1998)

### Asumsikan:

- i. Arsitektur komputer dengan ukuran halaman (page size) 1024 bytes.
- ii. Setiap karakter (char) menempati 1 alamat memori @ 1 byte.
- iii. Struktur data "matrix (baris,kolom)" untuk selanjutnya disebut "matrix".
- iv. Setiap 4 baris (berurutan) "matrix" berada dalam satu halaman (page).
- v. Setiap saat, maksimum ada 1 halaman (page) "matrix" dalam memori.
- vi. Saat awal eksekusi fungsi, tidak ada halaman (page) "matrix" dalam memori.

Lingkari atau beri silang huruf "B" jika betul, dan "S" jika salah.

- **B** / **S** Fragmentasi eksternal (external fragmentation) akan terjadi pada sistem berbasis halaman (paging systems).
- **B** / **S** Bingkai (frame) pada memori virtual (VM) dipetakan ke halaman (page) pada memori fisik.
- B / S Pengeksekusian program berbasis demand paging selalu menghasilkan page fault.
- **B** / **S** Sebuah fungsi, mungkin saja menempati lebih dari satu halaman (page).

```
011 void isiMatrix1 (){
012
       char matrix[256][256];
013
       int ii, jj;
014
       for (ii=0; ii<8; ii++) {
          for (jj=0; jj<256; jj++) {
015
016
             matrix[ii][jj] = 'x';
          }
017
       }
018
019 }
```

- B / S Setiap eksekusi baris 016, selalu akan terjadi "page fault" pada matrix.
- **B** / **S** Pada seluruh iterasi loop luar baris 014-018, akan terjadi 8 kali "page fault" pada matrix.
- ${\bf B} \ / \ {\bf S}$  Pada saat mengeksekusi fungsi isiMatrix1(), terdapat kemungkinan terjadi TOTAL¹ lebih dari 3 kali "page fault".

```
021 void isiMatrix2 (){
022
       char matrix[256][256];
023
       int ii, jj;
       for (jj=0; jj<256; jj++) {
024
025
          for (ii=0; ii<8; ii++) {
026
             matrix[ii][jj] = 'x';
027
          }
       }
028
029 }
```

- B / S Terdapat kemungkinan terjadi TOTAL 2 kali "page fault" saat mengeksekusi baris 026.
- B / S Pada setiap iterasi loop dalam baris 025-027, terjadi 2 kali "page fault" pada matrix.
- B / S Pada seluruh iterasi loop luar baris 024-028, terjadi 512 kali "page fault" pada matrix.

### 2. 2016-2 (Waterloo 2012)

Page Table.

Consider this following "structure addrspace" of a 32-bit processor.

```
struct addrspace {
  vaddr_t as_vbase1
                         = 0x00100000; /* text
                                                segment: virtual base addr */
  paddr_t as_pbase1
                         = 0x10000000; /* text
                                                segment: physical base addr */
  size_t as_npages1
                         = 0x20;
                                       /* text
                                                segment: number
                                                                   of pages */
  vaddr_t as_vbase2
                         = 0x00200000; /* data
                                                segment: virtual base addr */
                         = 0x20000000; /* data
  paddr_t as_pbase2
                                                segment: physical base addr */
                                                segment: number
  size_t as_npages2
                         = 0x20;
                                       /* data
                                                                   of pages */
  vaddr_t as_vbase3
                         = 0x80000000; /* stack segment: virtual
                                                                   base addr */
                         = 0x80000000; /* stack segment: physical base addr */
  paddr_t as_pbase3
  size_t as_npages3
                         = 0x10;
                                       /* stack segment: number
                                                                   of
                                                                       pages */
                         = 0x1000;
                                       /* virtual page size is 0x1000 bytes */
   int
           page_size
};
```

When possible, translate the provided address.

Possible	Virtual Address	Physical Address	Segment
YES	0x0010 0000	$0x1000\ 0000$	text
NO	$0x0030\ 0000$		
	$0\mathrm{x}0010~\mathrm{FEDC}$		
	0x $0$ 011 $0$ 000		
	0x7FFF FFFF		
		$0x2000\ 1234$	
		$0\mathrm{x}8000~\mathrm{FFFF}$	

2	20	1	7	1
3.	20	1	1-	· Т

(	a	Please	write	$\operatorname{down}$	your	student	ID (	(NPM)	):

|-----|-----|-----|-----|-----|

(b) Please write down the last 2 digits of your student ID (NPM):

\_\_\_\_|\_\_|

(c) Please convert the 2 decimal digits above into an unsigned 32-bit hexdecimal number.

Let's call that number INTEGER32: (HEX) |\_\_\_\_| |\_\_\_| |\_\_\_| |\_\_\_| |\_\_\_|

(d) Please add **INTEGER32** to 0080 0000 (HEX).

Let's call it Virtual Address **ADDRESS32**: (HEX) |\_\_\_\_| |\_\_\_| |\_\_\_| |\_\_\_| |\_\_\_|

- (e) ADDRESS32 with a 4 kbyte page size will have page offset space of \_\_\_\_\_ bits,
- (f) And the page number space of **ADDRESS32** is \_\_\_\_\_ bits.
- (g) Therefore, the page number of ADDRESS32 is (HEX) \_\_\_\_\_,
- (h) And, the page offset of ADDRESS32 is (HEX) \_\_\_\_\_.
- (i) The Page Table Entry (PTE) starts at Physical Address (PA) 001 000 (HEX). Each PTE consists of 4 hexadecimal digits (16 bits or 2 bytes) which is stored in BIG-ENDIAN form.

PA (HEX)	0	1	2	3	4	5	6	7	8	9	A	В	С	D	$\mathbf{E}$	F
001 000	71	00	71	03	71	05	71	07	71	09	71	0B	71	0D	71	0F
001 010	71	10	71	13	71	15	71	17	71	19	71	1B	71	1D	71	1F
002 000	71	20	71	23	71	25	71	27	71	29	71	2B	71	2D	71	2F
002 010	71	30	71	33	71	35	71	37	71	39	71	3B	71	3D	71	3F
003 000	71	40	71	43	71	45	71	47	71	49	71	4B	71	4D	71	4F
003 010	71	41	71	53	71	55	71	57	71	59	71	5B	71	5D	71	5D

The PTE of page number ADDRESS32 is: (HEX) \_\_\_\_\_.

(j) The first digit are the flags. The PTE is valid when the flags digit is not zero (0). The flags of the PTE above is (**HEX**) \_\_\_\_\_ which means the PTE is (VALID / NOT VALID).

(	k)	If the PTE is	VALID,	the next	three	digits are	the l	Physical	Frame	Number:	(HEX	)

(l) Thus, the physical address of ADDRESS32 is: (HEX) \_\_\_\_\_.

(m) Please put INTEGER32 into the physical address of ADDRESS32 (BIG-ENDIAN form).

-PA- (HEX)	0	1	2	3	4	5	6	7	8	9	A	В	С	D	E	F

#### 4. **2017-2**

(a) Please write down your student ID (NPM):



(b) Please write down the last 2 digits of your student ID (NPM):



(c) Please add those 2 decimal digits, convert it to hexadecimal, and then convert it to a 32-bit unsigned hexadecimal number. Let's call that number **INTEGER32**:

$$|-----|_{10} + |-----|_{10} = |-----|_{16} = |-----|_{-----} |_{-----} |_{-----} |_{-----} |_{16}$$

(d) Please add INTEGER32 to 0080 0000 (HEX). Let's call the 32-bit Virtual Address as ADDRESS32:

				i		

- (e) If the page size of **ADDRESS32** is 4 kbytes, the space of the offset will be \_\_\_\_\_ bits.
- (f) Therefore, the page number space of **ADDRESS32** will be \_\_\_\_\_ bits.
- (g) Therefore, the page number of ADDRESS32 is (HEX) \_\_\_\_\_,
- (h) And, the page offset of ADDRESS32 is (HEX) \_\_\_\_\_.
- (i) The Physical Address (PA) space is 44 bits. The Page Table Entry (PTE) starts at PA 0001 0000 000 (HEX). Each PTE consists of 8 hexadecimal digits (32 bits or 4 bytes) which is stored in BIG-ENDIAN form.

PA (HEX)	0	1	2	3	4	5	6	7	8	9	A	В	С	D	E	F
0001 0000 000	00	02	00	00	00	02	00	01	00	02	00	02	00	02	00	03
0001 0000 010	00	02	00	04	00	02	00	05	00	02	00	06	00	02	00	07
0001 0001 000	00	02	04	00	00	02	04	01	00	02	04	02	00	02	04	03
0001 0001 010	00	02	04	04	00	02	04	05	00	02	04	06	00	02	04	07
0001 0002 000	00	02	08	00	00	02	08	01	00	02	08	02	00	02	08	03
0001 0002 010	00	02	08	04	00	02	08	05	00	02	08	06	00	02	08	07
						•				•	•					

The PTE of page number **ADDRESS32** is:

(j)	The PTE is valid if the PTE value is not zero (0). Therefore, PTE is ( $\mathbf{VALID}$ / $\mathbf{NOT}$ $\mathbf{VALID}$ ).
(k)	If the PTE is VALID, the Physical Frame Number is:
(1)	Thus, the 44-bit PA of <b>ADDRESS32</b> is:

(m) Please put **INTEGER32** into the 44-bit PA of **ADDRESS32** (BIG-ENDIAN form). One address (box) is for one byte (2 hexadecimal digits).

Physical Address (HEX)	0	1	2	3	4	5	6	7	8	9	A	В	С	D	E	F

### 5. **2018-1**

(a) Please write down your student ID (NPM):

- 1			 1	1	1	l .		
					l		l	

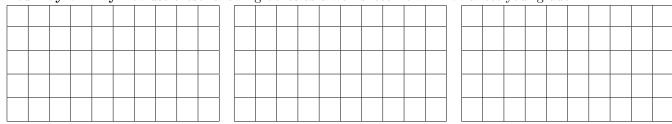
(b) Please write down the last digit of your student ID (NPM):

|\_\_\_\_|

(c) Consider the following page reference string: 1, 2, 3, 1, 4, 5, 3, 1, 6, 2. All frames are initially empty, so the first unique pages will always be a page fault. The frame allocation will depend on **the replacement** algorithm and **your last student ID (NPM) digit**. How many page fault(s) would occur:

		, , ,	, , , , , , , , , , , , , , , , , , ,
Please <b>circle</b> your last ID digit	Frame Allocation	Replacement Algorithm	Number of Page Fault(s)
0 1 2 3	3	FIFO	
4 5 6	4	TIFO	
0 1 2 3	4	LRU	
4 5 6	3	LICO	
0 2 4 6	3	Optimal	
1 3 5	4	Optimai	

You may or may not use these following boxes as a worksheet. It will not affect your grade!



## 6. 2018-2 (64%)

(a) Please write down the last digit of your student ID (NPM):

\_\_\_\_|

(b) Consider the following page reference string: 1, 2, 1, 3, 4, 5, 1, 2, 4, 1 All frames are initially empty, so the first unique pages will always be a page fault. The frame allocation will depend on **the replacement algorithm** and **your last student ID (NPM) digit**. How many page fault(s) would occur:

Please <b>circl</b> last ID d		Frame Allocation	Replacement Algorithm	Number of Page Fault(s)
0 1 2 3	3	3	FIFO	
4 5 6		4	1110	
0 1 2 3	3	4	LRU	
4 5 6		3	LICO	
0 2 4 6	5	3	Optimal	
1 3 5		4	Optimal	

You may or may not use these following boxes as a worksheet. It will not affect your grade!

## 7. 2019-1 (73.2%)

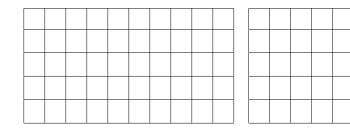
(a) (96%)Please write down the last digit of your student ID (NPM):

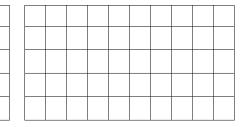
|\_\_\_\_

(b) Consider the following page reference string: 5, 4, 3, 2, 1, 2, 3, 4, 3, 2. All frames are initially empty, so every first unique page allocation will always causes a page fault. The frame allocation will depend on the **replacement algorithm and your last student ID (NPM) digit**. How many page fault(s) would occur:

Please <b>circle</b> your last ID digit	Frame Allocation	Replacement Algorithm	Number of Page Fault(s)
0 1 2 3	3	FIFO (78%)	
4 5 6	4	1110 (1070)	
0 1 2 3	4	LRU (68%)	
4 5 6	3	LITO (0070)	
0 2 4 6	3	Optimal (62%)	
1 3 5	4	Optimal (0270)	

You may or may not use these following boxes as a worksheet. It will not affect your grade!





## 8. **2019-2** (**50.9**%)

Consider **INTEGER64**, a 64-bit unsigned number; **ADDRESS64** a 64-bit Virtual Address; and **PA32**, the 32-bit Physical Address (PA) of ADDRESS64. Let **INTEGER64** = **ADDRESS64** = 0000 0000 0ABC (HEX).

- (a) (70%) If the page size of **ADDRESS64** is 4 kbytes, the space of the offset will be \_\_\_\_\_ bits.
- (b) (53%) Therefore, the page number space of **ADDRESS64** will be \_\_\_\_\_\_ bits.
- (c) (49%) The page number of **ADDRESS64** is \_\_\_\_\_\_(**HEX**).
- (d) (64%) And the page offset of **ADDRESS64** is \_\_\_\_\_\_(**HEX**).
- (e) (49%) The Page Table starts at **PA** ABCD E000 (HEX). Each Page Table Entry (PTE) consists of 6 hexadecimal digits (24 bits or 3 bytes) which is stored in **BIG-ENDIAN** form. A PTE is valid if the first bit of 24 is zero (0).

PA (HEX)	0	1	2	3	4	5	6	7	8	9	A	В	С	D	Е	F
ABCD E000	04	00	00	04	00	01	04	00	02	04	00	03	04	00	04	04
ABCD E010	00	05	04	00	06	04	00	07	04	00	08	04	00	09	04	00

The PTE of page number **ADDRESS64** is:

		1		П	1	i

- (f) (84%) The PTE is (VALID / NOT VALID).
- (g) (39%) If the PTE is VALID, the Physical Frame Number is:

1	I	1 1	П		ı
	l	ll	Ш	 	ı

(h) (33%) Thus, **PA32** of **ADDRESS64** is:

		11		- 1		1	ш	1		1
		11					ш			1
			l				ш			1
		11					ш			1

(i) (40%) Please put **INTEGER64** into the **PA32** of **ADDRESS64**. Each address box is for one byte (2 digits).

PA32 (HEX)	0	1	2	3	4	5	6	7	8	9	A	В	С	D	Ε	F

## 9. **2020-1**

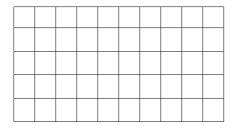
(a) Please write down the last digit of your student ID (NPM):

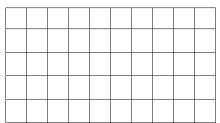
\_\_\_\_|

(b) Consider the following page reference string: 1, 2, 3, 4, 1, 2, 1, 5, 1, 2. All frames are initially empty, so every first unique page allocation will always causes a page fault. The frame allocation will depend on the **replacement algorithm and your last student ID (NPM) digit**. How many page fault(s) would occur:

Please <b>circle</b> your last ID digit	Frame Allocation	Replacement Algorithm	Number of Page Fault(s)
0 1 2 3	3	FIFO	
4 5 6	4	TIFO	
0 1 2 3	4	LRU	
4 5 6	3	LICO	
0 2 4 6	3	Optimal	
1 3 5	4	Optimai	

You may or may not use these following boxes as a worksheet. It will not affect your grade!





## 10. **2022-2** (46.8%)

Here is a retro problem that is adapted from chapter 10 Silberschatz's book. Your answer should be based the last digit your student ID# (NPM):

Last ID# Digit	N	umber of Frame	es
Last ID# Digit	LRU	FIFO	OPTIMAL
0	1	3	5
1	2	4	6
2	3	5	7
3	4	6	1
4	5	7	2
5	6	1	3
6	7	2	4

(a) Please write down the last digit of your student ID# (NPM) (100%):

1

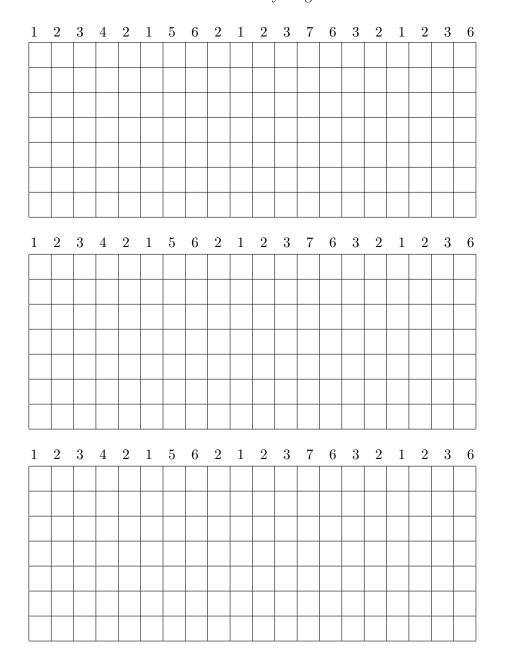
(b) Consider the following page reference string:

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

How many page faults would occur for the following replacement algorithms, assuming one, two, three, four, five, six, and seven frames? Remember that all frames are initially empty, so your first unique pages will cost one fault each.

Last ID# Digit	I	Number of Page Faul	S
Last ID# Digit	LRU (36%)	FIFO (50%)	OPTIMAL (36%)

You **may or may not** use these following boxes as a worksheet. It will **not** affect your grade!



11. <b>2023-2 (CB:35</b> %)
-----------------------------

Consider INTEGER64, a 64-bit unsigned number; ADDRESS64 a 64-bit Virtual Address; and	<b>PA32</b>	, the
32-bit Physical Address (PA) of ADDRESS64. Let INTEGER64 = ADDRESS64 = 0000 0000	0000	1234
(HEX).		

- (a) If the page size of **ADDRESS64** is 4 kbytes, the space of the offset will be \_\_\_\_\_ bits (91%).
- (b) Therefore, the page number space of **ADDRESS64** will be \_\_\_\_\_ bits (45%).
- (c) The page number of **ADDRESS64** is \_\_\_\_\_\_(HEX) (27%).
- (d) And the page offset of **ADDRESS64** is \_\_\_\_\_\_(**HEX**) (27%).
- (e) The Page Table starts at **PA** ABCD E000 (HEX). Each Page Table Entry (PTE) consists of 6 hexadecimal digits (24 bits or 3 bytes) which is stored in **BIG-ENDIAN** form. A PTE is valid if the first bit of 24 (most left) is zero (0).

PA (HEX)	0	1	2	3	4	5	6	7	8	9	A	В	С	D	$\mathbf{E}$	F
ABCD E000	04	00	00	04	00	01	04	00	02	04	00	03	04	00	04	04
ABCD E010	00	05	04	00	06	04	00	07	04	00	08	04	00	09	04	00
				•	•											

The PTE of page number **ADDRESS64** is (18%):



- (f) The PTE is (VALID / NOT VALID) (82%).
- (g) If the PTE is VALID, the Physical Frame Number is (18%):



(h) Thus, **PA32** of **ADDRESS64** is (21%):

l 1	1 1	1	1	1	1	l I	1	1 1	
		1				ı .	1		

(i) Please put **INTEGER64** into the **PA32** of **ADDRESS64**. Each address box is for one byte (2 digits) (18%).

PA32 (HEX)	0	1	2	3	4	5	6	7	8	9	A	В	$\mathbf{C}$	D	$\mathbf{E}$	F