Zago Ian
U88387934

1.

I tried to figure out this question.

For each task, we have a corresponding weight.

The instantaneous share $f_i(t)$ of process i at time t is

$f_i(t) = \frac{1}{n}$ since each task have equal weights.

The lag means the difference between the service time a process ~~is entitled to~~ should receive and the actual service time it receives.

Ziqi Tan
U88387934

2.(a) Logical Address:

Selector : offset

<0xBB88> : <0x3F0000>

⇓

1011|011|1000|000

⇓

Index 1011|011|1000|

⇓

Segment Num: 0x1771 ⇒ Base Address: 0x0001 0000 Limit: 0x0001FFFF Out of Limit

(b) <0xBB98> : <0x3FE00D>

⇓

1011|011|100|1000

⇓

Index 0x1773 ⇒ Base: 0x00030000 Limit: 0x003FFFFF Within the limit

Linear Address: 0x0042E00D

Page size: 4KB

0x0042E00D ÷ 4KB = 0x0000 042E ⇒ 1070

(C) 0x003FCFA0 ÷ 4KB = 0x0000 03FC ⇒ 1020

Frame number: 212

~~Assume frame number starts from 0~~

~~(212-1) x 4096 - 1 ~ 212 x 4096~~

~~212 x 4096 ~ (212+1) x 4096 - 1~~

212 x 4096 + 0xFA0 = 872352

Ziqi Tan
U88387934

2. (d)

guest virtual address
↓
page table 1 : 4 times
↓
page table 2 : 4 times
↓
guest physical address : host virtual address
↓
page table 1
↓
page table 2
↓
page table 3
↓
physical address

VM thought the page table as physical, but in fact it is not.

4 physical memory ~~address~~ references

Answer:
4+4+4 = 12 physical memory ~~address~~ references

If we consider GDT here,
it would be 5+5+5+5=20 references.

Zigo Tan
U88387934

3.(a)
```
semaphore lanterns_east = 9
semaphore lanterns_west = 0
semaphore ~~lantern~~key_east = 9
semaphore key_west = 0
mutex lock

procedure east_visitor() {

    wait(lanterns_east);
    wait(key_east);
    wait(key_east);   < if(key_east==0){signal(key_east);}

    // walking_through_house

    signal(lanterns_west);
    signal(key_west);
    signal(key_west);

}

procedure west_visitor() {
    wait(lanterns_west);
    wait(key_west);
    wait(key_west);   < if(key_west==0){signal(key_west));
    // walking_through_house
    signal(lanterns_east);
    signal(key_east);
    signal(key_east);
}
```

If a visitor holds a key, and there is not enough key for him/her to enter, then release the resource he/she holds, which can avoid visitors on one side to wait each other's keys.

(b) In this solution, any number of visitors on any sides will not lead to a deadlock.

Ziqi Tan
U88387934

3. (c)

pthread_cond_wait(&b->occupied_slot, &b->mutex)
and
pthread_mutex_unlock(&b->mutex) in the check id

may cause deadlock.

Consider this scenario.

① The buffer is almost full.

② Consumers cannot find their own items.
Consumers unlock the mutex.

③ Producers continue to produce.
Now the buffer is full.

④ The occupied_slot is signaled to wake up one of the consumers.
This consumer still cannot find its own item. and then return.

⑤ Now the producer cannot produce item and all the consumers are waiting for the occupied_slot to be signaled.
A deadlock occurs.

The solution is to let the consumers wake up each other when they cannot find their own items until the corresponding consumer get its item.

Add "pthread_cond_signal(occupied_slot)"
right after the "pthread_mutex_unlock(&b->mutex)"
and right before the "return Null".

Ziqi Tan

U88387934

4. It does not use LOCK to exclusively use the bus in a multicore system.
Fix: ~~Use~~ Add LOCK prefix before bts and btr.

Improve: prevent a CPU who ~~tries~~ attempts to acquire the lock continuously use the bus exclusively, which will cause a performance problem on other CPU cores.

```
spinlock_lock:
    LOCK bts $0, (mlock)
    jc   spin_wait
    ret
spin_wait:
    test (mlock), 1     # if the lock is free
    jnz  spin_wait      # not free
    jmp  spinlock_lock  # may be free

spinlock_unlock:
    LOCK btr $0, (mlock)
    ret
```

Ziqi Tan

U88387934

5. (a)

31	16	15	0
Base 0:15		Limit 0:15	

63 56	55 52	51 48	47 40	39 32
Base 24:31	Flags	Limit 16:19	Access Byte	Base 16:23

code segment descriptor: 0x0000FFFF
0x00CFFA00

data descriptor: ~~0x00CFF200~~
0x00CFF200

Access Byte:

Pr: present bit must be 1.

Privilege: 2 bits. Ring 3: 11

S: Descriptor type. 1 for code or data, 0 for system segments

Executable bit: 1 for code segment and 0 for data.

Direction/Conforming bit:

↓ data selectors

0: segment grows up. → limit > offset

1: segment grows down → limit < offset

→ code selectors

1: can be executed from an equal or Lower privilege level.

0: can only be executed on the ring set specified above.

RW: Readable for code selectors
Writable for data selectors.

Access bit: 0

Flags:

Gr: 0: byte granularity ; 1: 4kB Page size

Sz: 0: 16 bit protected mode ; 1: 32 bit protected mode

Zigu Tan

U88387934

5. (b) Idt Entry : 0X 0008 0000
0X FCC1 EE00

Reasoning : 31 16 15 0
Segment Selector Pointer
GDT RPL
0 00

63 48 47 46 45 44 43 40 39~32
Pointer P DPL S GateType 0

Present
GateType
0xE : 32 bit int gate
Descriptor privilege level
0 for int and trap gate

Specify which privilege Level the calling Descriptor minimum should have.

Pointer : FEC1 0000

[15..3] [2] [1:0]

Segment Selector : 0008 ⇒ Index : 0x01 GDT : 0 RPL : 00

5. (c) No ! Not safe !

~~Kernel can never~~

Kernel cannot trust the user-space code, in case that the user-level code is adverse.

eg: user level code call foo

foo :
any priviledged instruction
lgdt
ret

Kernel can never trust the user level code who wants to change the GDT. The execution flow should pass control to user space first and then execute the user level code ~~and then check privilege~~
User level does not have the priviledge to lgdt.

Ziqi Tan

U0638734

5. (d) ISR should use iret to return.

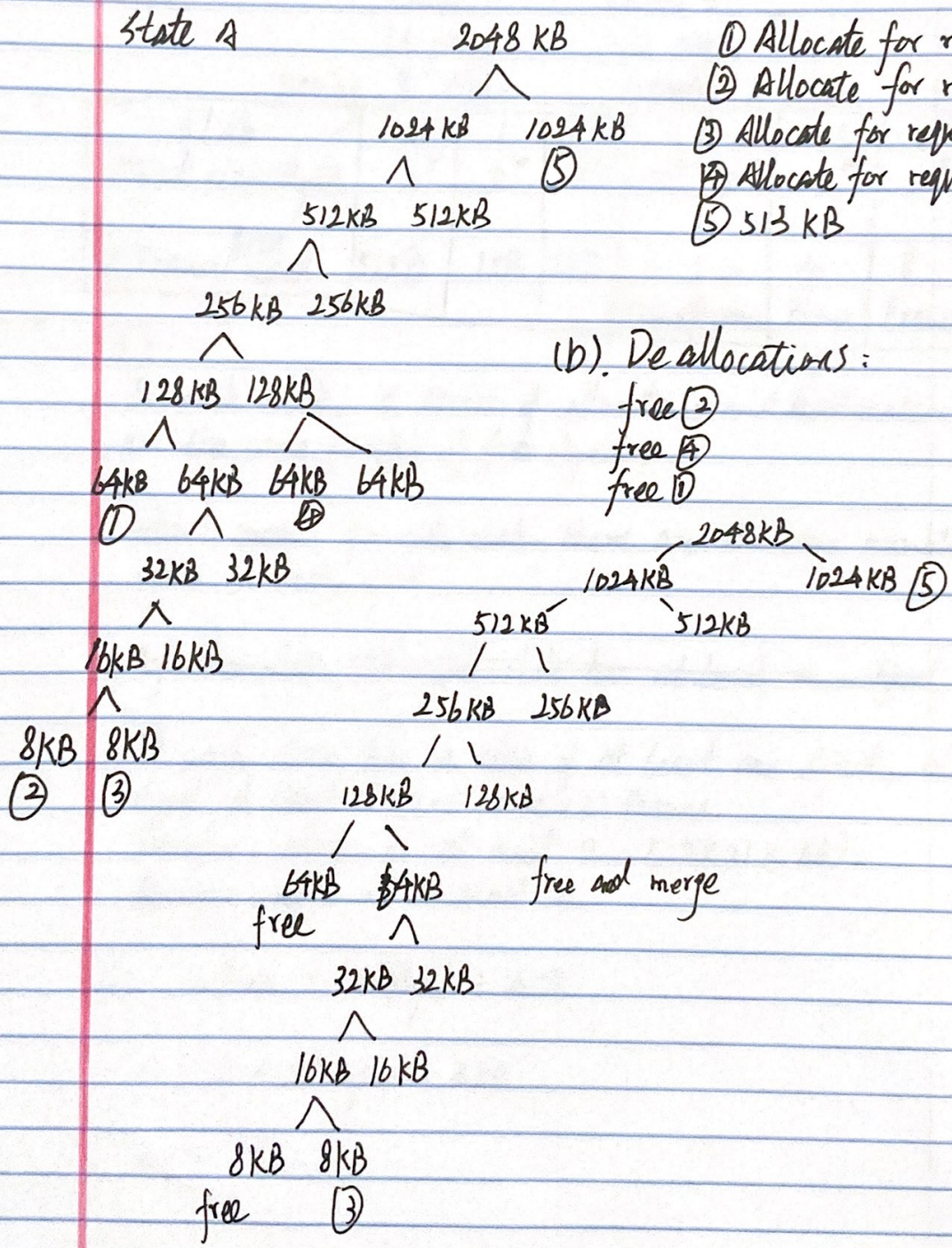
Although we ~~can~~ have tried to protect return address, iret also pops the IP, CS and flag register, which have been push when an interrupt occurs.

Zizi Tan

U81387934

6.(a) Due to limited space, I use a tree to represent the lists.

$2^9 \times 4KB = 2048KB$



(b). De allocations:

free ②
free ④
free ①

Ziqi Tan
U88387934

6. (c) Page size : 4kB

132 Pages

Cache 1: 66 pages having 8 slabs

Cache 2: 66 pages having 8 slabs

slab	0	1	2	3	4	5	6	7
Object size/Byte	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}
# Frames/page need	512B	1KB	2KB	1 Frames	2 Frames	4 Frames	8 Frames	16 Frames

i)

In each slab, a series of allocations and deallocations results in the same number of free objects,

which means ~~for each slab~~, there are the same numble of free space in each slab.

~~If the slabs~~ ~~If each slab has at least one object~~

If each slab has a size of at least one object, slab 3~7 ~~have~~ need at least $1+2+4+8+16 = 31$ frames.
Therefore, nfree is at most 2. ($2 \times 31 \leq 66$)
Answer: nfree is at most 2.

ii) nfree $\times$ 2^{16}Byte $\div$ 4KB

$= 2 \times 2^{16}$ Bytes $\div$ 4KB

= 32 Frames