Your grades for **Spring 2016 CS 350 Midterm** **Total** Score: **48.0**

Summary

Class scores distribution

Total (/score/3c8b8cff-227c-48a3-b5ce-1848a099d9f4)

Q1 (/score/3c8b8cff-227c-48a3-b5ce-1848a099d9f4/Q1)

Q2 (/score/3c8b8cff-227c-48a3-b5ce-1848a099d9f4/Q2)

Q3 (/score/3c8b8cff-227c-48a3-b5ce-1848a099d9f4/Q3)

Q4 (/score/3c8b8cff-227c-48a3-b5ce-1848a099d9f4/Q4)

Q5 (/score/3c8b8cff-227c-48a3-b5ce-1848a099d9f4/Q5)

Q6 (/score/3c8b8cff-227c-48a3-b5ce-1848a099d9f4/Q6)

Q7 (/score/3c8b8cff-227c-48a3-b5ce-1848a099d9f4/Q7)

Q8 (/score/3c8b8cff-227c-48a3-b5ce-1848a099d9f4/Q8)

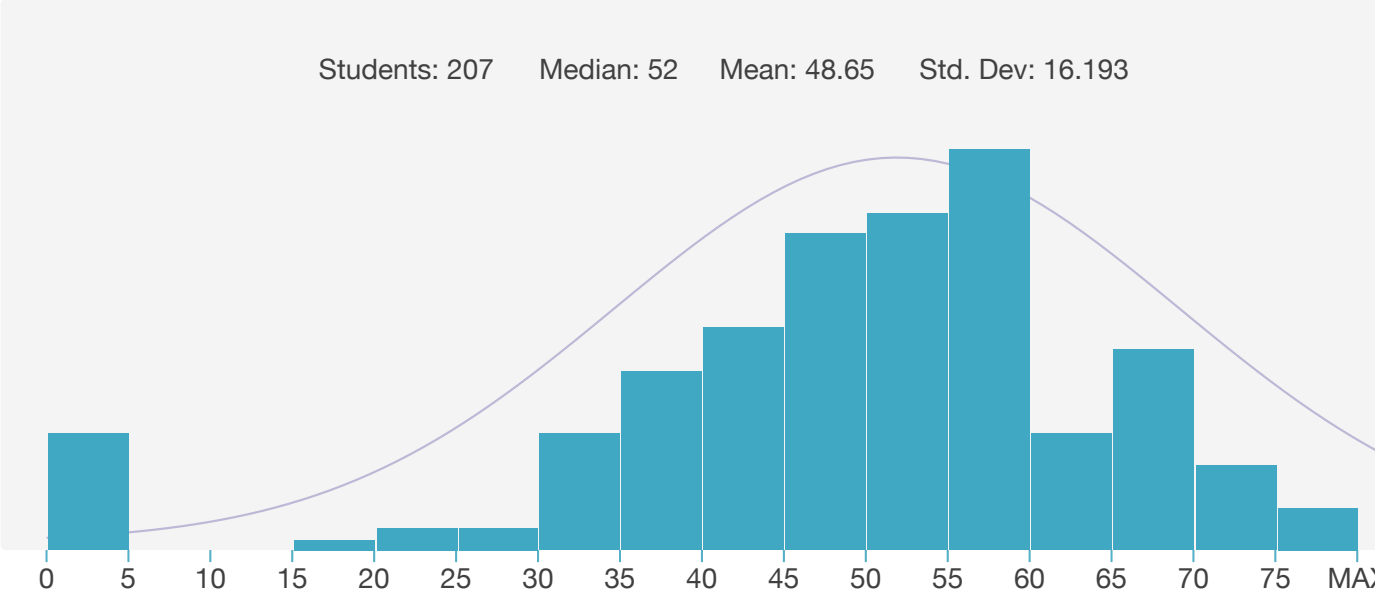
Q9-intro (/score/3c8b8cff-227c-48a3-b5ce-1848a099d9f4/Q9-intro)

Q9 (/score/3c8b8cff-227c-48a3-b5ce-1848a099d9f4/Q9)

Q10 (/score/3c8b8cff-227c-48a3-b5ce-1848a099d9f4/Q10)

extra (/score/3c8b8cff-227c-48a3-b5ce-1848a099d9f4/extra)

extra (/score/3c8b8cff-227c-48a3-b5ce-1848a099d9f4/extra)


Students: 207
Median: 52
Mean: 48.65
Std. Dev: 16.193
0
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
MAX

Q1 Score: **3.0**

If a question is worth 2 marks it means we are looking for an answer that only has a few words in it (such as a technical term, a phrase or a number). If it is worth 3 marks we are looking for roughly a sentence worth of explanation or for three points. Using point form rather than complete sentences is acceptable.

1a List three features of processor hardware that are required to implement a multitasking operating system. [3 marks]

1. Handle critical intersection.
2. Prevent starvation
3. Preempted ✓ interrupted.

Implement

1b In current systems, at least part of the exception handler is written in assembly language. Explain why by listing three capabilities that are provided by assembly language and that are typically missing from a high-level language. [3 marks]

1. jr Jump to a specific address, since we don't want processor interact with other processor, so we do this to disabling process get to other processor directly,
2. beq
3. bne

all same reason as, we want to prevent processes isolated!

1c How does a system call differ from a normal function call? Identify two distinct differences. [2 marks]

1. system call will get access into privillige mode while normal function call does not.
2. system call could communicate to hardware while normal function call can not.

Q2 Score: **11.0**

2a What is one advantage and one disadvantage of having larger pages sizes? [2 marks]

adv: Less pages need to track. ✓

disadv: More space will be wasted. ✓

2b Identify and briefly describe two of the methods (discussed in class) that an operating system could use to prevent deadlocks. [4 marks]

1. No hold and lock - A thread can't acquire more resource if it holds a related source. ✓ ✗

2. Ordering - give threads ordering that they could only acquired for higher level. So it won't form a deadlock. ✓ ✓

2c What does it mean for a synchronization mechanism to be *fair*. [3 marks]

Prevent following problem arise.

1. Deadlock

2. The strawation. ✓ ✓

3. Critical intersection crush.

2d List three different pieces of information about processes that are maintained by the kernel. [3 marks]

1. How much space should give to each process. ✗

2. TrapFrame information. ✓

3. Page Table information. ✓

Q3 Score: **9.0**

3a How does the relocation register facilitate Dynamic Relocation? [3 marks]

When we do Dynamic Relocation: Basically we calculate the offset of the vm first, then we add the offset and the relocation base address to form the physical address,
And if we do context switch, we need to change the relocation base address and the length of it too,

Then check if it within the length

3b What other information does the MMU track for Dynamic Relocation? What is it used for? [3 marks]

The relocation base address and the length,

Basically when we calculate the physical address, we need the relocation base address and the offset and add them up to get the physical address.
And we need the length to check if the offset is within the range. If yes, then its valid. Otherwise we prevent. ?

bound register - used to check whether the virtual address is larger than the register (raises exception if so) -2

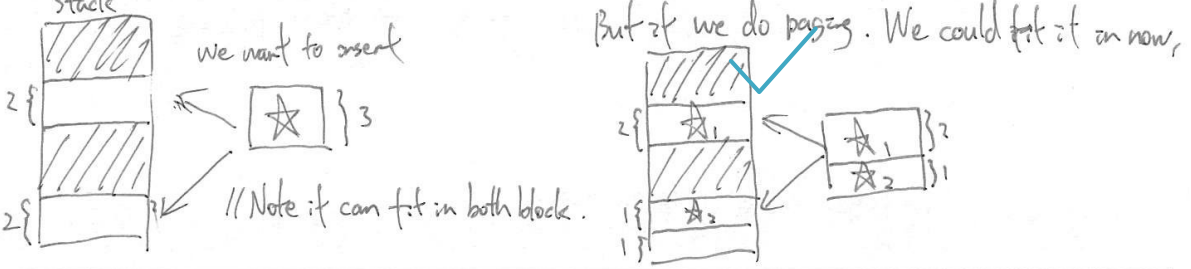
3c What is the purpose of the Translation Look-aside Buffer (TLB) in paging? [2 marks]

Speed up page searching.

So TLB have some page info stored and it a hardware.
When we want to search a page we look for TLB, and its fast!

3d What is an advantage of paging over segmentation? Explain briefly. [3 marks]

We could have pages in many different place, so it could save space,



Q4 Score: **4.0**

Consider a system with 28 bit virtual address and 28 bit physical address that uses simple paging with a page size of 1 Kb (1024 bytes). Assume that every user process has a code segment starting at virtual address 0x0 and stack segment ending at the largest possible address 0xFFFFFF, i.e. it has a sparse address space. The word size is 4 bytes and addresses must be word aligned, i.e. divisible by four. When convenient you may express your answer as a power of 2, i.e. 2^4, 2^8, etc.

4a How many bits are used to represent offset part of the virtual address? [2 marks]

24 bits

4b How many bits are used to represent the virtual page number? [2 marks]

4 bits

4c How many page table entries will a page table have? [2 marks]

$\frac{2^{28}}{2^{10}} = 2^{18}$ page table entries

since $2^{10} = 1024$

4d How many frames are occupied by a page table if each page table entry requires 4 bytes? [2 marks]

$\frac{2^{18} \times 4}{2^{10}} = 2^{10}$

2^{20} bytes or 2^{10} frames.

For the following question we will be using decimal numbers (rather than hexadecimal numbers) to describe memory. Suppose an operating system gives each process an allocation of 64000 bytes. Suppose further that we have a process that needs:

- 32860 bytes for the program code,
- 15420 bytes for the data, and
- 14680 bytes for the stack.

Note, that a page can hold code only, data only, or stack values only. It can never hold any mixture of code, data or stack.

5a Can this program be loaded into the allocation if the page size is 4000 bytes? Briefly explain your answer. [3 marks]

$\frac{64000}{4000} = 16$ pages // total page we got

$\lceil \frac{32860}{4000} \rceil = 9$ pages // we need for code

$\lceil \frac{15420}{4000} \rceil = 4$ pages // we need for data

$\lceil \frac{14680}{4000} \rceil = 4$ pages // we need for stack

$9+4+4 = 17$ // total we need

since total # of page we have is 16 and total # of page we needs 17. So No!

No.

5b Can this program be loaded into the allocation if the page size is 1000 bytes? Briefly explain your answer. [3 marks]

$\frac{64000}{1000} = 64$ pages

$\lceil \frac{32860}{1000} \rceil = 33$ pages

same stuff as a)

$\lceil \frac{15420}{1000} \rceil = 16$ pages

$\lceil \frac{14680}{1000} \rceil = 15$ pages

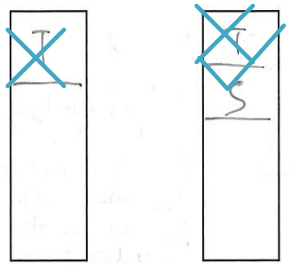
$33+16+15 = 64$ // = 64: we have

Since total # of page we have is 64 and total # of page we need is also 64. So we could load it!

Yes.

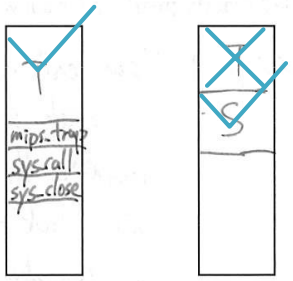
Consider an OS/161 system with a single core processor and two threads (T1 and T2). T1 is running, T2 is on the ready queue, having got there because it called `thread_yield`.

6a Draw a picture of the kernel stack for each thread. Write (T) for trapframe, (S) for switchframe and ignore the frames due to the calls to `mips_trap`, `syscall` or other kernel functions. Draw what is currently on each kernel stack in the diagrams below. The stack grows downward. [2 marks]



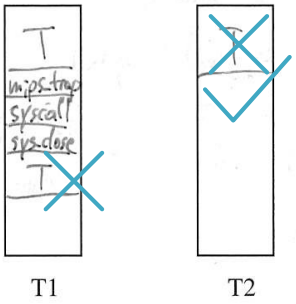
T1 T2

6b T1 makes a system call to close a file. What is on each kernel stack when T1 is executing the kernel function `sys_close`.? [2 marks]



T1 T2

6c During the system call there is a timer interrupt and T2 starts running. What is on each kernel stack now? [2 marks]



T1 T2

You are writing some code that forks a thread to set up some data structures for a new tab (in a browser) using the OS/161 kernel. You want to start a new thread with tab_init() and then wait until it has completed some initialization code before the main thread can continue. See the implementation below.

```
// global variables
// assume they are initialized properly
struct lock *init_lk;
struct cv *done;
```

```
void main( )
{
 thread_fork("init",NULL,0,tab_init,NULL);
 lock_acquire(init_lk);
 cv_wait(done, init_lk);
 lock_release(init_lk);
 // rest of code...
}
```

```
void tab_init (void *p, unsigned long n)
{
  // some initialization code ...
  lock_acquire(init_lk);
  cv_signal(done, init_lk);
  lock_release(init_lk);
  // rest of code ...
}
```

7a What could go wrong with this code? Briefly justify your answer. [3 marks]

This could form a deadlock: Main call fork -> it acquire lock init_lk, it wait for signal (done). Now tab_init acquire the init_lk too, and it will signal(done) after get the init_lk. Since the init_lk was hold in somewhere else, and wait for the signal to release the lock. So both won't give up and will stuck. (Deadlock

7b How would you fix it? [3 marks]

We could modify tab_init as following:

```
Void tab_init (void *p, unsigned long n)
{
  lock_release(init_lk);
  // s~~~
  lock_acquire(init_lk);
  CV_signal(done, init_lk);
  lock_release(init_lk);

  lock_acquire(init_lk);
  // rest of code }
```

We add two lines, so the tab_init will release init_lk before acquire it.

Q8 Score: **0.0**

Create a function, `transfer`, that uses locks to synchronize the transfer of funds from one bank account to another. Assume that the locks are properly initialized and properly destroyed. Each lock is tied to a particular account so that `acct_lock(1234)` acquires the lock for account 1234, guaranteeing that no other thread can access this account until the thread calls `acct_release(1234)`. The use of these two functions has exactly the same semantics as `lock_acquire` and `lock_release`. I.e. you have two synchronization primitives at your disposal,

- `acct_lock`(unsigned int account_num) *// acquire the lock to a particular account*
- `acct_release`(unsigned int account_num) *// release the lock to a particular account*

You also have three functions to check and manipulate the funds that you will use to implement `transfer`, i.e. in order to transfer money you will `withdraw` from one account and `deposit` in the other.

- void `withdraw` (unsigned int account_num, unsigned int amount); *// remove amount from the account*
- void `deposit` (unsigned int account_num, unsigned int amount); *// add amount to the account*
- unsigned int `get_balance` (unsigned int account_num); *// return the account balance*

For simplicity assume that the amount is in cents, so there are no fractional amounts and that `withdraw` and `deposit` always work, i.e. no need to check a return value. However your function should either return the integer constant NOT_SUFFICIENT_FUNDS if there is not sufficient money to transfer from one account to the other or NO_ERROR if there is sufficient funds.

8. Implement the transfer function ensuring that it never causes deadlock. You may continue your function on the next page. [9 marks]

// transfer amount from src to dest account
int `transfer` (unsigned int src, unsigned int dest, unsigned int amount) {

```
- lock_acquire(src);
    if(get_balance(src)>=amount){
        withdraw(src, amount);
    }
    else{
-       lock_release(src);
        return NOT_SUFFICIENT_FUNDS;
    }
- lock_acquire(dest);
    deposit(dest, amount);
- lock_release(dest);
  return NO_ERROR;
}
```

if src == dst, self deadlock

must order locks, deadlock

not releasing before return

9. The following table lists the virtual and physical base addresses for the code segments (base 1), data segments (base 2) as well as the physical base addresses of the stacks, for two different processes. These values are to be used exactly as they are used in OS/161 except that here the stack size is only 8 pages (instead of 12). The page size is 4KB and the virtual and physical base addresses are all a multiple of 4K. Some of the values for Process 2 are missing.

4KB = 0x1000 x 8

Field	Process 1	Process 2
as_vbase1	0x0040 0000	0x0040 0000
as_pbase1	0x0100 0000	0x0400 0000
as_npages1	0x0000 0008	?
as_vbase2	0x1000 0000	?
as_pbase2	0x0200 0000	?
as_npages2	0x0000 0006	0x0000 0004
as_stackpbase	0x0300 0000	?

For the following questions your answer could be an address or *Seg Fault* if there is a segmentation violation. For addresses, add a space after every forth hex digit to make the hexadecimal numbers easier to read.

Question 9 continued on the next page …

Q9 Score: **6.0**

9a What is the physical address corresponding to Process 1's virtual address 0x0040 6100?

[2 marks]

0x0100 6100

9b What is the physical address corresponding to Process 1's virtual address 0x0200 4A00?

[2 marks]

Seg Fault

9c What is the virtual address corresponding to Process 1's physical address 0x0100 1600?

[2 marks]

0x 0040 1600

9d What is the virtual address corresponding to Process 1's physical address 0x0300 2C00?

[2 marks]

0x 7FFF D400

A 10
B 11
C 12
D 13
E 14
F 15

Q10 Score: **6.0**

10 Using the same table as the previous question, but considering Process 2, use the information below to calculate in the missing information in the table for Process 2.

The following are all valid physical addresses for Process 2

0x0400 51A4	0x0400 3000	0x0400 4A00	0x0400 1000	✗
0x0600 0F00	0x0600 3008	0x0600 7020	0x0600 4200	✗
0x0800 400C	0x0800 50A4	0x0800 3020	0x0800 2200	✓

The following are not valid physical addresses for Process 2

0x0400 6140	0x0402 1000	0x0401 0000	0x0400 C004
0x0600 CC00	0x0600 F100	0x0600 8800	0x0600 9100
0x0800 6C00	0x0800 1100	0x0800 8800	0x0800 AC00

Answer the following questions or put *Not Enough Info* if there is not enough information to answer the question with certainty.

10a What is as_npages1 for Process 2? [2 marks]

0x0000 0006 ✓

10b What is as_pbase2 for Process 2? [2 marks]

0x0800 2000 ✓

10c What is as_stackpbase for Process 2? [2 marks]

0x0600 0000 ✓

This page is for rough work or to continue answering a question. **If you want this work to be graded** clearly indicate which question you are answering and where your answer starts and ends.

This page is for rough work or to continue answering a question. **If you want this work to be graded** clearly indicate which question you are answering and where your answer starts and ends.