CS 537 Midterm

Tien-Lung Fu

TOTAL POINTS

55 / 80

QUESTION 1

1 Entities responsible 2 / 2

+ **2 pts** OS, hardware, OS

✓ **+ 2 pts** **OS,hardware,hardware**

+ **0 pts** incorrect

QUESTION 2

2 System calls 1 / 2

+ **1 pts** trap or trap table index

✓ **+ 1 pts** **syscall index (via reg)**

+ **0 pts** both not mentioned

QUESTION 3

3 Base Register 0 / 2

+ **2 pts** can violate protection

✓ **+ 0 pts** **incorrect**

QUESTION 4

4 Threads Sharing 2 / 2

✓ **+ 0.5 pts** **sp: unique**

✓ **+ 0.5 pts** **code: shared**

✓ **+ 0.5 pts** **pt: shared**

✓ **+ 0.5 pts** **ip: unique**

+ **2 pts** unique,shared,shared,unique

+ **0 pts** none correct

QUESTION 5

5 RR scheduling 5 / 5

✓ **+ 2 pts** **Next scheduled process has PID = 21**

✓ **+ 3 pts** **Round Robin scheduling is based on order of arrival not the amount of time which a process has been running**

+ **0 pts** Incorrect PID of next scheduled job along with an incorrect or insufficient explanation of Round Robin scheduling rules.

QUESTION 6

6 MLFQ 5 / 5

✓ **+ 3 pts** **A promotion rule is needed to prevent starvation.**

✓ **+ 2 pts** **MLFQ starvation example scenario**

+ **0 pts** The need for the promotion rule in an MLFQ is not clearly stated and the scenario given does not demonstrate starvation in an MLFQ scheduler.

QUESTION 7

7 Deadlocks:Which 5 / 5

✓ **+ 1 pts** **i. Yes**

✓ **+ 1 pts** **ii. No**

✓ **+ 1 pts** **iii. Yes**

✓ **+ 1 pts** **iv. No**

✓ **+ 1 pts** **v. Yes**

+ **0 pts** No answer / Wrong answers

QUESTION 8

8 Deadlocks:Rewrite 5 / 5

✓ **+ 3 pts** **Describe one correct way for rewriting (e.g. apply some ordering, wrap with a meta lock, or use try-lock)**

- **1 pts** Partially incorrect or incomplete description

✓ **+ 2 pts** **Describe another correct way for rewriting (e.g. apply some ordering, wrap with a meta lock, or use try-lock)**

- **1 pts** Partially incorrect or incomplete description

+ **0 pts** No answer / Incorrect answers

QUESTION 9

9 RowMajor TLB 0 / 5

✓ **- 5 pts** **Wrong, both sequence and answer**

+ **5 pts** Correct

+ **3 pts** Correct Sequence

+ **4 pts** Almost correct answer like 7 but out of 15

QUESTION 10

10 ColumnMajor TLB 0 / 5

✓ **+ 0 pts Wrong answer**

+ 5 pts Correct

+ 3 pts Correct Sequence

+ 1 pts Correct Answer,Wrong reason,Lucky folks

+ 4 pts Almost correct but 15 total elements instead of 16

QUESTION 11

11 TLB improve 2 / 2

+ 0 pts Wrong

+ 2 pts Increase Page size, Quite a few people gave 64 Kb as answer.

✓ **+ 2 pts Increase Page Size to 64Kb**

+ 1 pts Increasing the size of TLB

QUESTION 12

12 Semaphore purpose 0 / 6

+ 2 pts Identified that mystery prevents starvation for write lock

+ 2 pts Example trace that could lead to starvation without mystery

+ 2 pts Example trace that shows how starvation is avoided with mystery

✓ **+ 0 pts Incorrect identification of purpose of mystery**

+ 1 pts Partially correct identification of mystery

+ 1 pts Partial description of execution that is prevented by mystery

- 1 pts Partially incorrect description of execution trace

QUESTION 13

13 Locks 4 / 4

✓ **+ 1 pts Saying lock implementation is not correct**

✓ **+ 1 pts Saying mutual exclusion could be violated**

✓ **+ 2 pts Example of how mutual exclusion could be violated**

+ 0 pts Not saying that implementation is incorrect

+ 1 pts Partial example of how mutual exclusion could be violated

+ 1 pts Other reasons like fairness / performance which are not related to correctness

QUESTION 14

14 Page size 2 / 2

+ 0 pts Correct

✓ **+ 2 pts Click here to replace this description.**

+ 1.5 pts Click here to replace this description.

QUESTION 15

15 Number of pages 2 / 2

+ 1.5 pts Correct

+ 0 pts Click here to replace this description.

✓ **+ 2 pts Click here to replace this description.**

QUESTION 16

16 Pagetable size 0 / 4

✓ **+ 0 pts .**

+ 2 pts only one inner page table is sufficient. 2^12 or 2^8 bytes, based on the assumption about whether or not invalid pages are considered.

+ 2 pts adding in page directory size of 1 entry in page directory -- 2^8 entries or 1 entry, based on the assumption about whether or not entries for invalid page tables are considered.

Observe: to accommodate this process only one inner page table is sufficient.

Answer: (1+2^8) or (2^8+2^12)
Two solutions based on the assumptions about whether or not entries for invalid page tables are considered and similarly with the invalid pages within the inner page table.

QUESTION 17

17 FIFO policy 5 / 5

✓ **+ 5 pts Correct**

+ 0 pts wrong hit rate and virtual page numbers

+ 2 pts wrong virtual page numbers

+ 3 pts wrong hit rate

QUESTION 18

18 LRU policy 5 / 5

✓ **+ 5 pts Correct**

+ 2 pts correct hit rate

+ 3 pts correct virtual page numbers

+ 0 pts wrong virtual page numbers and hit rate

QUESTION 19

19 OPT hit rate 2 / 2

✓ **+ 2 pts Correct**

+ 0 pts wrong hit rate

QUESTION 20

20 Print unbuffer 4 / 4

✓ **+ 2 pts ADFE**

✓ **+ 2 pts AFDE**

+ 0 pts Incorrect or empty

- 0.5 pts Incorrect output

QUESTION 21

21 Print fork fails 2 / 2

✓ **+ 2 pts Correct**

+ 0 pts Incorrect or Empty

- 0.5 pts Incorrect output

- 0.5 pts Incorrect output

QUESTION 22

22 Print exec fails 2 / 2

+ 0 pts Empty or incorrect

✓ **+ 1 pts AFBDE**

✓ **+ 1 pts AFDBE**

✓ **+ 1 pts ADFBE**

- 0.5 pts Incorrect output

QUESTION 23

23 Print buffered 0 / 2

✓ **+ 0 pts Incorrect or empty**

+ 2 pts AAFDE

+ 2 pts ADAFE

+ 2 pts AADFE

+ 2 pts AFADE

+ 2 pts FADE

- 0.5 pts Incorrect output

Any one of these
AAFDE
ADAFE
AADFE
AFADE
Or this (assuming If fork doesn't copy stuff)
FADE

Minus -0.5 for
ADFE
AFDE

If three or more incorrect responses, you get
0

CS 537 Midterm (Spring 2019)

Please write your FULL NAME on this page only

Tien-Lung Fu

Important notes

Please enter your answers in the space provided below each question.
You can use the reverse side to work out your answers

This test has 15 pages and 8 questions
Remember to read all of the questions carefully and good luck!

Grading

Qn	Maximum	Points
1. Warmup	8	
2. Scheduling	10	
3. Deadlocks	10	
4. TLBs	12	
5. Locks, Semaphores	10	
6. Paging	8	
7. Page Replacement	12	
8. What could be printed?	10	

Q1 Warmup [8 points]

1(a) Choose the entities which takes responsibility for doing these activities:

(1). modify the base register of a certain segment

(2). do the translation of address with the base register of a certain segmenth w

(3). do the translation of address according to the page table

The answers in each option are in sequence, i.e. first one is for (1), second for (2), third for (3). Circle the appropriate option. [2 points]

A. process, OS, hardware
B. OS, hardware, hardware
C. OS, hardware, OS
D. OS, OS, OS

1(b) System calls: Hardware provides two modes: user and kernel. While apps run in the user mode, the OS runs in the kernel mode. When an app needs to access a resource such as a disk or a network, it transitions into the kernel via a system call. How does the processor know which piece of code to execute when we make a system call? [2 points]

via system call number to know which system call to be excuted

1(c) Describe one shortcoming of using only a base register per process to do address translation. [2 points]

There may be lots of fragments since there may exists empty spaces between code, stack, and heap.

1(d) For each of the following, list if they are shared between threads or if each thread has a unique copy. [2 points]

Stack pointer F
Code section T
Page table T
Instruction pointer F

Q2 Scheduling [10 points]

The table below contains data about all processes which are currently running in an OS. The OS scheduler is implementing a simple Round Robin policy to share the CPU between the processes. Assume that the process with PID=15 just finished its turn using the CPU.

PID	Position in Process Table	Creation Time	CPU Time Used
21	8	2019-02-28 12:15:30	15
10	2	2019-02-28 04:15:30	3,000,000
15	6	2019-02-28 10:22:01	2,000
32	4	2019-02-28 06:30:00	500,000

(a) What is the PID of the next process which will be given access to the CPU? Why? [5 points]

Ans: PID 21,

Since PID 21 index is 8 in process table right behind index 6 of PID 15

(b) Consider a multi-level feedback queue scheduler. Why do we need a rule to promote a process to higher priority level periodically? Give an example scenario that could lead to a problem without this rule [5 points]

Given a process A that takes long running time, so it is finally demoted to the lowest priority queue. But when many short-running time processes coming, occupying the highest priority queue, CPU executes them one by one instead of A. So after waiting for a certain time, we want to promote A to higher priority to let A have chance to be executed.

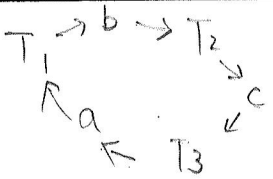
Q3. Deadlocks [10 points]

```
void foo(pthread_mutex_t *t1, pthread_mutex_t *t2, , pthread_mutex_t *t3) {
 pthread_mutex_lock(t1);
 pthread_mutex_lock(t2);
 pthread_mutex_lock(t3);

 do_stuffs();
 pthread_mutex_unlock(t1);
 pthread_mutex_unlock(t2);
 pthread_mutex_unlock(t3);
}
```

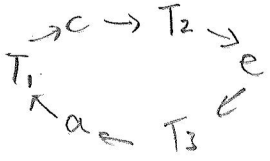
3(a) Thread 1, 2, and 3 are concurrently-executing threads that are running the above function. a, b, c, d, e, and f are six different mutexes. For each case below, indicate whether a deadlock can happen. [5 points]

	Thread 1	Thread 2	Thread 3
i.	foo(a, b, c)	foo(b, c, a)	foo(c, a, b)
ii.	foo(a, b, c)	foo(a, b, c)	foo(a, b, c)
iii.	foo(a, b, c)	foo(b, c, e)	foo(f, e, a)
iv.	foo(a, b, c)	foo(d, e, f)	foo(a, c, f)
v.	foo(a, b, c)	foo(c, d, e)	foo(f, d, b)

i 

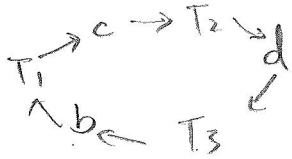
⇒ dead lock happen

ii $T_1, T_2, T_3 \Rightarrow a, b, c$ same order ⇒ not happen

iii 

⇒ dead lock happen

iv. do not have a loop ⇒ not happen

v. 

⇒ dead lock happen

atom

3(b) Describe two ways to rewrite this function so that deadlock will never happen for the above cases [5 points]

Way 1:

```
void foo( ){
  acquireMetaLock(); // grab all locks in atomic operation
  do_stuffs();
  releaseMetaLock();
}
```

Way 2:

```
void foo(pthread_mutex_t *t1, pthread_mutex_t * t2, pthread_mutex_t * t3 ){
  // C++ style
  vector<pthread_mutex_t*> locks = {t1, t2, t3};
  sort(locks.begin(), locks.end());

  for (int i=0; i< locks.size() ; i++)
     pthread_mutex_lock(locks[i]);

  do_stuffs();

  for (int i=0 ; i < locks.size() ; i++)
     pthread_mutex_unlock(locks[i]);
}
```

Q4: TLBs [12 points]

In C an array is stored in row-major format. For example, an integer array of shape 4x4 with data like the one is shown in Figure 2 in memory as shown in Figure 3

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Figure 2: Example 2D integer array

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Figure 3: Memory representation in row-major format

A standard way to access such an array is using pointer arithmetic. For example, the given function is going through the array row-by-row and printing contents separate lines

```
void print_array(int *arr, int cols, int rows){
        for (size_t i=0; i<rows; i++){
                for (size_t j=0; j<cols; j++){
                        printf("%d\n", arr[i*cols+j];
                }
}
```

Let's assume there is a tiny address space where each page is of 8 bytes and there are 16 such pages. We have a 4x4 array and the element a[0] is located at virtual address 20. (Virtual Page Number=02, offset = 04). Assume size of integer is 4 bytes

4(a). Ignoring variables i and j, what will be the TLB hit rate if the TLB can hold 4 entries with LRU replacement policy. [5 points]

$\frac{8 \text{ bytes}}{4 \text{ bytes}} = 2$ int a page can store

so the hit-miss pattern would be:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
m	h	m	h	m	h	m	h	m	h	m	h	m	h	m	h

hit rate = 50 %

4(b). Suppose we want to print the array in column order using the code

```
void print_col_array(int *arr, int cols, int rows){
        for(size_t i=0; i<rows; i++){
                for(size_t j=0; j<cols; j++){
                        printf("%d\n", arr[j*rows+i]);
                }
}
```

What will be the TLB hit rate with the same TLB size and replacement policy as before? [5 points]

same page

m [0 h 1]	m [2 h 3]		
m [4 h 5]	m [6 h 7]		
m [8 h 9]	m [10 h 11]		
m [12 h 13]	m [14 h 15]		

hit rate = 50 %

4(c). If you had control over page size how would you improve the TLB hit rate? [2 points]

change page size to be 4 bytes × 16 = 64 bytes

Thus a page can store 16 ints

⇒ only first access is miss

⇒ hit rate = $\frac{15}{16}$

Q5: Locks, Semaphores [10 points]

```
typedef struct __rwlock_t {
  sem_t lock;
  sem_t writelock;
  sem_t mystery;
  int readers;
} rwlock_t;

void rwlock_init(rwlock_t *rw) {
  rw->readers = 0;
  Sem_init(&rw->lock, 1);
  Sem_init(&rw->writelock, 1);
  Sem_init(&rw->mystery, 1);
}

void rwlock_acquire_readlock(rwlock_t *rw) {
  Sem_wait(&rw->mystery);
  Sem_wait(&rw->lock);
  rw->readers++;
  if (rw->readers == 1)
      Sem_wait(&rw->writelock);
  Sem_post(&rw->mystery);
  Sem_post(&rw->lock);
}
void rwlock_release_readlock(rwlock_t *rw) {
  Sem_wait(&rw->lock);
  rw->readers--;
  if  (rw->readers  ==  0)
      Sem_post(&rw->writelock);
  Sem_post(&rw->lock);
}

void rwlock_acquire_writelock(rwlock_t *rw) {
  Sem_wait(&rw->mystery);
  Sem_wait(&rw->writelock);
  Sem_post(&rw->mystery);
}
void rwlock_release_writelock(rwlock_t *rw) {
  Sem_post(&rw->writelock);
}
```

5(a) On the previous page you are given an implementation of a reader-writer lock. What is the role of the `mystery` semaphore that has been added to this lock? Give an example execution trace to explain its purpose. [6 points]

mystery is a lock to earlier mutual exclusive reader and writer.

So if reader hold the mystery lock, writer will wait

So do the case when writer hold the mystery lock, reader will wait.

If we do not have mystery,
the code will go to Sem_wait(&rw->writelock);
to check if a reader can get a writelock.
(late check
and waste 3 lines
to execute)

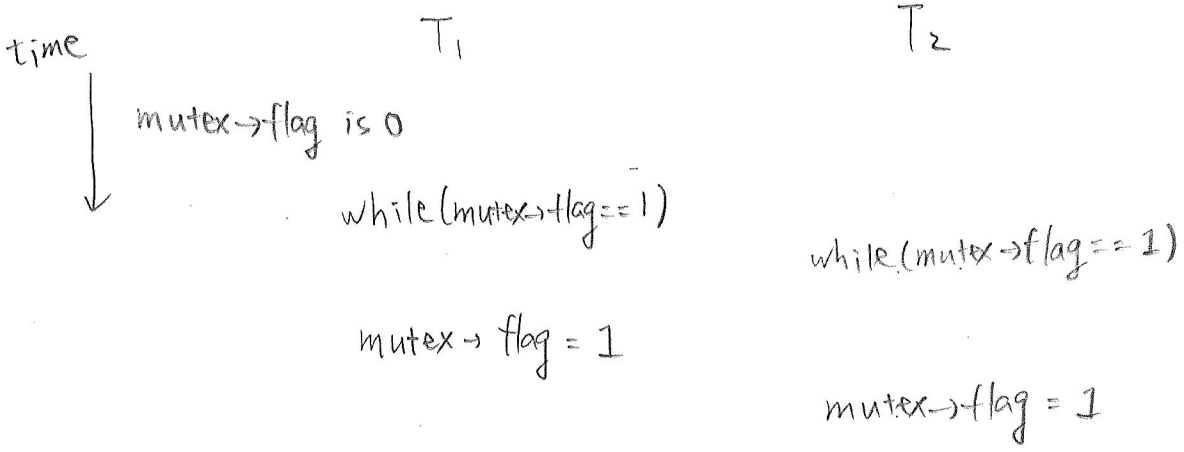
Consider the following implementation of a lock.

```
typedef struct __lock_t { int flag; } lock_t;
void init(lock_t *mutex) {
     // 0 -> lock is available, 1 -> held
     mutex->flag = 0;
}

void lock(lock_t *mutex) {
     while (mutex->flag == 1) // TEST the flag
     ; // spin-wait (do nothing)
     mutex->flag = 1; // now SET it!
}

void unlock(lock_t *mutex) {
     mutex->flag = 0;
}
```

5(b) Is this implementation of a lock correct? If not describe a scenario that could lead to undesirable behavior? [4 points]



The implementation is wrong, since T1 & T2 both get the lock.

Q6. Paging [8 points]

A computer system has 32 bit virtual address space and we subdivide the virtual address as follows: 8 bits for the page directory, 12 bits for the inner page and 12 bits for the offset. Assume each page table entry (PTE) and page directory entry (PDE) is 1 byte.
If a two-level paging with 1 byte for every entry in the page directory is used,

6(a) What is the page size in such a system? [2 points]

Page size = 2^{12} = 4KB

6(b) How many pages are there in the virtual address space of a process? [2 points]

pages = $2^8 \times 2^{12} = 2^{20}$

6(c) If the process size is 1MB allocated contiguously, what is the total size of the page table in bytes? Include the size of inner page tables and the page directory [4 points]

Q7 Page Replacement Policy [12 points]

Consider the following request sequence of virtual pages

3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4 tot = 12

Assume the number of physical frames is 4.

7(a) If the page replacement policy in use is FIFO, calculate the hit rate and the virtual page numbers remaining in physical memory at the end of this request sequence. [5 points]

m 3 3
m 2 3,2
m 1 3,2,1
m 0 3,2,1,0
h 3 3,2,1,0
h 2 3,2,1,0
m 4 2,1,0,4
m 3 1,0,4,3

m 2 0,4,3,2
m 1 4,3,2,1
m 0 3,2,1,0
m 4 2,1,0,4

final virtual page #

hit rate = $\frac{2}{12}$

7(b) If the page replacement policy in use is LRU, calculate the hit rate and the virtual page numbers remaining in physical memory at the end of this request sequence. [5 points]

m 3
m 2
m 1
m 0 0,1,2,3
h 3 3,0,1,2
h 2 2,3,0,1
m 4 4,2,3,0
h 3 3,4,2,0

h 2 2,3,4,0
m 1 1,2,3,4
m 0 0,1,2,3
m 4 4,0,1,2

final virtual page #

hit rate = $\frac{4}{12}$

7(c) Compute the optimal hit rate (i.e., the hit rate you get from OPT) [2 points]

3 2 1 0 | 3 2 4 3 2 1 0 4
h h m h h h m h

3 2 1 0
3 2 1 4
2 1 4 0

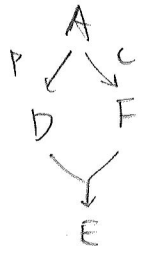
hit rate = $\frac{6}{12}$

Q8: What could get printed?

Assume the program /bin/true, when it runs, never prints anything

```
int main(int argc, char *argv[]) {
      printf("A");
      int rc = fork();
      if (rc == 0) {
            printf("F");
            char *my_argv[] = { "/bin/true", NULL };
            execv(my_argv[0], my_argv);
            printf("B");
      } else if (rc > 0) {
            printf("D");
            wait(NULL);
            printf("E");
      } else {
        printf("C");
      }
      return 0;
}
```

(a) If all calls to fork and exec succeed what are **all** possible outputs that could get printed assuming print statements are **not buffered** ? [4 points]



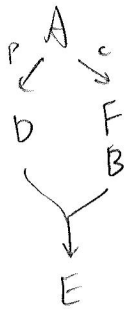
Ans: A D F E

A F D E

(b) If the call to fork fails list **all** possible outputs assuming print statements are **not buffered**? [2 points]

Ans: AC

(c) If the call to exec fails list **all** possible outputs assuming print statements are **not buffered**? [2 points]



Ans: A D F B E
A F D B E
A F B D E

(d) If all calls to fork and exec succeed, but if we have buffered print statements, what is **one** possible output that wasn't possible in part (a)? You can assume the buffer is FIFO within a process. [2 points]