Directions

- There are a total of 95 points on this exam.
- If you believe there is an error in the exam, notify a proctor. An announcement will be made if a significant error is found.
- It is your responsibility to properly interpret a question. **Do not ask questions regarding the interpretation of a question**; they will not be answered and you will only disrupt your neighbours. If there is a non-technical term you do not understand you may ask for a definition. If you are confused about a question, state your assumptions and proceed to the best of your abilities.
- If you require more space to answer a question:
  - use the back of that question's page, or
  - if there are blank pages at the end you may use them, but you must **clearly indicate** in the provided answer space that you have done so, or
  - If you need more paper beyond this, ask the proctors for blank pages. IT IS YOUR RESPONSIBILITY TO ENSURE THAT ANY EXTRA PAGES ARE STAPLED TO THIS EXAM.
- **Do not detach any pages and do not write on the QR codes**

1. **(10 Points)**

   (a) (2 points) In today's systems, can we eliminate the need for synchronization primitives by disabling interrupts?

   con: will not enforce mutual exclusion on multiprocessors
   con: ignoring timer interrupts has side effects (i.e., can cause problems)
   con: prevents all preemption, not just preemption that would threaten the critical section

   1 point per any correct answer

   (b) (3 points) Which of the following is **shared** between threads of the same process? Circle your answer(s).

   - integer and floating point registers
   - program counter
   - heap memory
   - stack memory
   - global variables
   - open files in the process

   - **heap memory**
   - **global variables**
   - **open files in the process**

   1 point per correct answer

   (c) (3 points) List **three** ways in which execution can switch from user space to kernel space.

   1- System calls, 2- Processor Exceptions, 3- Interrupts

   1 point per correct answer

   (d) (2 points) What, if any, is the difference between the a `trapframe` and a `switchframe`?

   used to save state coming from user to kernel mode, switch is used to switch between two threads in kernel

   1 point for explaining trapframe and 1 point for switchframe

2. **(14 Points)**

   (a) (4 points) Assume you have a system with a small virtual address space of size 64 KB that uses paging and that each page is of size 8 KB.

   i. How many bits are in a virtual address in this system?
   16 (1-KB of address space needs 10 bits, and 64 needs 6; thus 16).
   1 pt for answer correct answer

ii. Recall that with paging, a virtual address is usually split into two components: a virtual page number (VPN) and an offset. How many bits are in the VPN?

3. Only eight 8-KB pages in a 64-KB address space.

1 pt for answer correct answer

iii. How many bits are in the offset?

16 (VA) - 3 (VPN) = 13. Alternately: an 8KB page of course requires 13 bits to address each byte

1 pt for answer correct answer

iv. How many entries does this linear page table contain?

One entry per virtual page. Thus, 8.

1 pt for answer correct answer

(b) (4 points) Now assume you again have a small virtual address space of size 64 KB and that the system again uses paging, but that each page is of size 4 bytes (note: not KB!).

i. How many bits are in a virtual address in this system?
still 16. The address space is the same size.
1 pt for answer correct answer

ii. How many bits are in the VPN? 14.

1 pt for answer correct answer

iii. How many bits are in the offset?

Just 2 (4 bytes).

1 pt for answer correct answer

iv. How many entries does the page table contain?

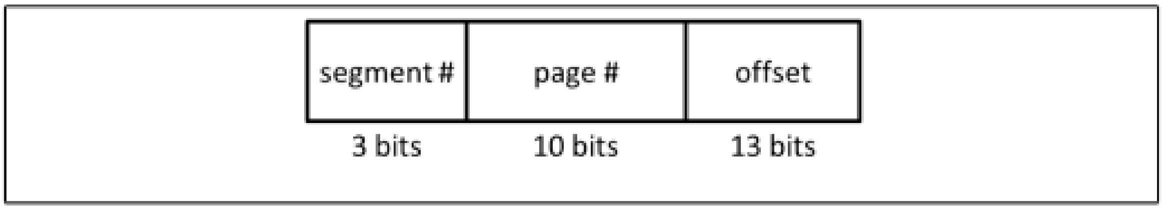
2^{14} or 16,384

1 pt for answer correct answer

(c) (6 points) Mr. Goose wants to implement a virtual memory system that uses segmentation combined with paging. The system will have the following parameters:

- Each process has 6 segments.
- Each segment can be up to 8MB in size.
- Each page is 8KB in size.

Draw a diagram showing the different fields of the virtual address in this system. Make sure to label each field and indicate how many bits it contains. Some useful information: 1K = 2^{10} and 1M = 2^{20}.



2 pt for each answer correct information in the vm address. 1 point for correct number of bit and 1 point for segment, page VPN or offset labels

3. **(10 Points)** Consider a solution to the producer/consumer problem, using semaphores for synchronization.

```
static volatile unsigned int empty;
static volatile unsigned int full;
static volatile unsigned int mutex;
empty = sem_create("empty",empty_initial_value);
full = sem_create("full",full_initial_value);
mutex = sem_create("mutex",mutex_initial_value);

void *producer(void *arg) { // core of producer
  for (i = 0; i < num; i++) {
    P(&empty);
    P(&mutex);
    add_to_buffer(i);   //adds item to buffer
    V(&mutex);
    V(&full);
  }
}

void *consumer(void *arg) { // core of consumer
  while (!done) {
    P(&full);
    P(&mutex);
    int tmp = get_from_buffer(i);
    V(&mutex);
    V(&empty);
    // do something with tmp ...
  }
}
```

(a) (6 points) What are the initial values of each sempahore and why?

- the `empty_initial_value` of sempahore `empty` is:
- the `full_initial_value` of sempahore `full` is:
- the `mutex_initial_value` of sempahore `mutex` is:

- the `empty_initial_value` of sempahore `empty` is:The semaphore empty can be initialized to 1 or a positive number N. If there is only a size=1 buffer, it is OK to initialize empty this way.
- the `full_initial_value` of sempahore `full` is:The semaphore full must be initialized to 0. Full tracks how many full buffers there are; at the beginning, zero
- the `mutex_initial_value` of sempahore `mutex` is:The semaphore mutex must be initialized to 1. Mutex provides mutual exclusion, and thus must be set to 1 at the beginning to work properly.

2 pt for each bullet, 1 point for correct semaphore init value and 1 point for justification

(b) (2 points) Why are the sempahores declared `volatile`?

shared varaibles should be declared as volatile to force fresh loads from memory

2 points total, 1 point for indicating shared data and 1 point to indicate volatile keyword forces loads from memory

(c) (2 points) What will happen if we remove lines 11, 13, 21 and 23 that pertain to the sempahore `mutex`?

unsafe/nondeterministic access to shared variable buffer

2 point for indicating that buffer access will not be sequentially accessed, that may cause overwrites

4. **(10 Points)** Suppose that two long running processes, P1 and P2, are running in a system. Neither program performs any system calls that might cause it to block, and there are no other processes in the system. P1 has 2 threads and P2 has 1 thread.

   (a) (3 points) What percentage of CPU time will P1 get if the threading model used is 1-1 (one is to one)? Explain your answer.

   (b) (3 points) What percentage of CPU time will P1 get if the threading model used is m:1 (many is to one)? Explain your answer.

   (c) (2 points) Consider the situation when a thread is waiting for an event to occur (e.g., waiting for a lock to be released), is it possible to achieve better performance by using a `spinlock` instead of a `lock`?

   (d) (2 points) Consider the two threads of P1, where one thread is waiting for a lock to be released by the second thread, explain why busy waiting can never perform better than blocking while waiting on single-CPU system.

   (i) If the threads are kernel threads, they are independently scheduled and each of the three threads will get a share of the CPU. Thus, the 2 threads of P1 will get 2/3 of the CPU time. That is, P1 will get 66percent of the CPU. (ii) If the threads are user threads, the threads of each process map to one kernel thread, so each process will get a share of the CPU. The kernel is unaware that P1 has two threads. Thus, P1 will get 50percent of the CPU. (iii) If the thread will wait for a short amount of time (compared to the time required for a context switch), then it is better to spin-wait. (iv) Blocking while waiting requires two context switches, one to block the thread and one to unblock it. If a thread t1 is waiting for an event in a single-CPU system (e.g., waiting for a lock to be released), then the thread t2 that will cause the event to happen (e.g., the thread that will release the lock) must be scheduled for t1's waiting to end. By spin-waiting, t1 is preventing t2 from being scheduled, and is therefore prolonging its own wait as well as wasting system resources. In a multi-CPU system, t2 would be scheduled on a different CPU.

   (i) and (ii)2 points each. 1 pt for correct answer and 1 point for correct explanation (iii) 2 points total, 1 point for explanation explanation of using spinlock will get better results if task is small and lock better if not expecting the task to be small (iv) 2 point for explanation

5. **(11 Points)**

Consider a system with two preemptively scheduled threads. One thread executes the `WriteA` function shown below. The other executes the `WriteB` function, also shown below. Both functions use `kprintf` to produce console output. The random function called by `WriteA` returns a randomly-generated nonnegative integer. `WriteA` and `WriteB` are synchronized using two semaphores, S_a and S_b. The intial value of both semaphores is zero. Assume that the individual calls to `kprintf` are atomic.

```
WriteA() {
    unsigned int n,i;
    while(1) {
    n = random();
    for(i=0;i<n;i++) {
      kprintf``(''A);
    }
    for(i=0;i<n;i++) {
      V(Sb);
    }
    for(i=0;i<n;i++) {
      P(Sa);
    }
  }
}//end of writeA

WriteB() {
  while(1) {
    P(Sb);
    kprintf``(''B);
    V(Sa);
  }
}//end of WriteB
```

(a) (8 points) Consider the following 10-character console output prefixes. Each prefix shows the first 10 characters printed to the console. Which of these prefixes could possibly be generated by the two threads running in this system? Write "YES" next to the output prefix if it could be generated, otherwise write "NO". You must *justify* your answer to receive the points.

i. ABABABABAB

A. ABABABABAB YES

1 pt for each correct answer

ii. `BABABABABA`

A. BABABABABA NO

1 pt for each correct answer

iii. `AAAAAAAAAA`

A. AAAAAAAAAA YES

1 pt for each correct answer

iv. `AAABABABAB`

A. AAABABABAB NO

1 pt for each correct answer

v. `AAABBBBAAB`

A. AAABBBBAAB NO

1 pt for each correct answer

vi. `AAAAAABBBB`

A. AAAAAABBBB YES

1 pt for each correct answer

vii. `AAABBBAABB`

A. AAABBBAABB YES

1 pt for each correct answer

viii. `BBBBBAAAAB`

A. BBBBBAAAAB NO

1 pt for each correct answer

(b) (3 points) Now, suppose that the initial value of semaphore S_b is 1, rather than 0 and consider the 10-character console output prefixes shown. Write “YES” next to the output prefix if it could be generated, otherwise write “NO”. You must *justify* your answer to receive the points.

i. `BABABABABA`

A. BABABABABA YES

1 pt for each correct answer

ii. `ABABBABABAB`

A. ABABBABABAB YES

1 pt for each correct answer

iii. `AAABBBABABB`

A. AAABBBABABB YES

1 pt for each correct answer

i. BABABABABA YES

ii. ABABBABABAB YES

iii. AAABBBABABB YES

1 pt for each correct answer

6. **(14 Points)**

Consider the OS/161 system with only two processes, ProcA and ProcB. ProcA is running and ProcB is ready. Suppose that ProcA causes an arithmetic overflow exception while it is running, and that the kernel's response to this exception is to terminate ProcA and allow ProcB to run.

(a) (7 points) List the sequence of events that will occur when ProcA causes the arithmetic overflow exception.
Create your list by choosing from the events shown below, using the event numbers to identify events. For example, a valid answer might be "7,5,4,8,5,2". Note that you may include an event more than one time in the sequence if it occurs more than once.

1. a syscall instruction occurs
2. privileged (kernel) execution mode is entered
3. unprivileged execution mode is entered
4. ProcA's application state is saved into a trapframe
5. ProcB's application state is saved into a trapframe
6. ProcA's application state is restored from a trapframe
7. ProcB's application state is restored from a trapframe
8. context switch to ProcB's thread
9. context switch to ProcA's thread
10. ProcB's application code starts executing
11. determine which type of exception occurred

Write your answer here:

2,4,11,8,7,3,10

1 pt for each correct answer, in the order shown

(b) (7 points) For each event in your list, indicate whether the event is a hardware or a software event. If an event occurs in software, name the function in OS/161 that corresponds to the software event and describe the purpose of that function.

- 2. privileged (kernel) execution mode is entered in HARDWARE
- 3. unprivileged execution mode is entered in HW
- 4. ProcA's application state is saved into a trapframe in SW common_exception
- 7. ProcB's application state is restored from a trapframe in S/W return from common_exception
- 8. context switch to ProcB's thread in SW thread_switch and, or switchframe_switch
- 10. ProcB's application code starts executing in SW
- 11. the kernel determines which type of exception occurred in SW in mips_trap using cause register

1 pt for each correct answer, in the order shown

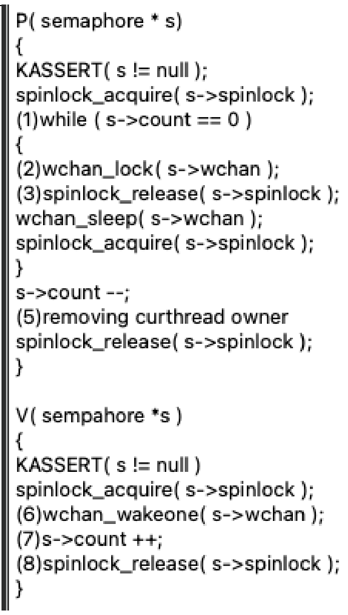
7. **(16 Points)**

Consider the following pseudocode implementation of a semaphore:

```
P( semaphore * s){

KASSERT( s != null );
spinlock_acquire( s->spinlock );

while ( s->count < 0 ){
spinlock_release( s->spinlock );
wchan_lock( s->wchan );
wchan_sleep( s->wchan );
spinlock_acquire( s->spinlock );
}
s->count --;
s->owner = curthread;
spinlock_release( s->spinlock );
}



V( sempahore *s ){
KASSERT( s != null )
spinlock_acquire( s->spinlock );
count --;
spinlock_acquire( s->spinlock );
}
```

Does this semaphore work? If yes, explain why. If no, correct it.

No, the solution is not correct the lines numbered 1 to 8 are the corrections that have to made



```
P( semaphore * s)
{
KASSERT( s != null );
spinlock_acquire( s->spinlock );
(1)while ( s->count == 0 )
{
(2)wchan_lock( s->wchan );
(3)spinlock_release( s->spinlock );
wchan_sleep( s->wchan );
spinlock_acquire( s->spinlock );
}
s->count --;
(5)removing curthread owner
spinlock_release( s->spinlock );
}

V( sempahore *s )
{
KASSERT( s != null )
spinlock_acquire( s->spinlock );
(6)wchan_wakeone( s->wchan );
(7)s->count ++;
(8)spinlock_release( s->spinlock );
}
```

2 points for each correct answer numbered 1-8 in the solution. There is no Point number 4, Points are shifted from 4 to 6. Therefore, point number 6 is 4 points.

8. **(5 Points)**

Consider a virtual memory system that uses paging. Virtual and physical addresses are both 32 bits long, and the page size is 4KB = 2^{12} bytes. A process P1 has the following page table. Frame numbers are given in hexadecimal notation. Recall that each hexadecimal digit represents 4 bits.

	Frame Number
0	0x00788
1	0x00249
2	0x0023f
3	0x00ace
4	0x00bcd

(a) (2 points) For each of the following **virtual addresses**, indicate the physical address to which it maps. If the virtual address is not part of the address space of P1, write NO TRANSLATION instead. Use hexadecimal notation for the physical addresses.

i. 0x00001a60
0x00249a60
1 point

ii. 0x000051ff
no translation
1 point

(b) (3 points) For each of the following **physical addresses**, indicate the virtual address that maps to it. If the physical address is not part of the physical memory assigned to P1, write NO TRANSLATION instead. Use hexadecimal notation for the virtual addresses.

i. 0x00249000
   0x00001000
   1.5 point

ii. 0x00000887
no translation
1.5 points

9. **(5 Points)**

   (a) (1 point)A process calls `execv`, which successfully executes. How many new processes were created? Explain your answer.

   0

   1 pt for correct answer and explanation

(b) (1 point) In the code below, are both parent and child processes executing the same program? Explain your answer.

```
int main(){
// assign progname and args
fork();
execv(progname, args);
}
```

answers can be either

i. yes, if parent calls execv before child, then parent and child will both be running same program

ii. no, they are executing different ”copies/instances” of the program because of fork

1 pt for either answer with correct explanation

(c) (3 points) Processes exist in a number of different states: Running, Ready, and Blocked.

Assuming you start observing the states of a given process at some point in time, not necessarily from its creation, but perhaps including that. Once you start observing the process, you will see ALL states it is in, until you stop sampling.

Which of these sequences of process states could you possibly observe? Write "YES" for a possible observation and "NO" otherwise. You must explain your answer.

i. Running, Running, Blocked, Blocked, Blocked, Running NO, since cannot go from blocked to running
   1 pt for correct answer and explanation

ii. Running, Running, Running, Ready, Running, Running, Running, Ready
Yes
1 pt for correct answer and explanation

iii. Ready, Ready, Ready, Ready, Ready
yes, not scheudled yet
1 pt for correct answer and explanation