Read all of the following information before starting the exam:

- Please keep your written answers brief; be clear and to the point. You do not need to fill the whole space provided for answers.

- This test has 6 problems and 9 pages (including the cover and a blank page). Make sure that you have all of the pages!

- This is a **closed book** exam. No additional material is allowed.

- Good luck!

Question	Marks	Avg	Avg %	Max
1	12	7.78	64.81	12
2	12	7.24	60.37	12
3	12	8.02	66.85	12
4	12	8.82	73.52	12
5	14	8.63	61.63	14
6	18	15.67	87.04	18
Total	**80**	56.16	70.20	77

Problem 1. (*12 points*)

a. (*4 pts*)   What resources does a thread share with other threads of the same process?

Answer: Program code, global variables, heap, and any resources assigned by the OS such as files or I/O devices. [i.e. everything in the address space. So technically, the stack as well.]

b. (*4 pts*)   What resources are associated with each thread that it does not share with other threads (of the same process)?

Answer: CPU registers, program counter, stack pointer, and the stack (though see note above).

c. (*4 pts*)   Give **two** examples of how a thread can go directly from *running* state to *ready* state.

Answer:

- Time quantum expired

- Preempted by a higher priority process

- Calls *thread_yield*

Problem 2. (*12 points*)

a. (*3 pts*)   What are the requirements that must be satisfied for shared access to a *critical section*?

Answer:

 i Mutual Exclusion

 ii Progress

 iii Bounded waiting

b. (*4 pts*)   Consider the following solution to the critical section problem involving only two threads. Would it work? Explain your answer.

```
/* shared variable */
boolean flag[2]                 /* Shared. Initially false */

...

flag[i] = true;                 /* Declare intent to access */
while (flag[1-i]) {}            /* Busy wait */

  < Critical Section >

flag[i] = false;
```

Answer: It will not. Two threads can both set the flag and wait indefinitely. Violates *progress*.

c. (*5 pts*)   Suppose we have the following two techniques to implement mutual exclusion:

 i Disabling Interrupts

 ii Special hardware instruction (e.g. test-and-set)

State **one** advantage of each over the other. Which one is used by OS/161?

Answer:

 i Disabling Interrupts does not require busy-waiting if the critical section cannot be entered immediately.

 ii Special hardware instruction will work on multiprocessors.

OS/161 uses interrupt disabling.

Problem 3. (*12 points*)

a. (*4 pts*)   There are only two processes in the system P_A and P_B containing one thread each. Assume that P_A is running and P_B is on the ready to run queue. Describe the steps required to save and restore or manipulate the processes context (including MMU) if P_A performs a non-blocking system call and it has not used up all of its quantum. Be sure to explain where the context is saved to and restored from. Provide a general description, you dont need to give specific register names.

Answer:

- On the trap into the kernel all of the processors registers are saved onto P_A's trap frame (onto the kernel stack for P_A). This is P_A's user-mode context.

- After executing the system call P_A's user-mode context is retreived from the trap frame in its kernel stack and restored to the processor registers just prior to returning from the exception to continue running in user mode.

b. (*8 pts*)   There are only two processes in the system P_C and P_D containing one thread each. Assume that P_C is running and P_D is on the ready to run queue. Describe the steps required to save and restore or manipulate the processes context (including MMU) if P_C performs a blocking system call and it has not used up all of its quantum. Be sure to explain where the context is saved to and restored from. Provide a general description, you dont need to give specific register names.

Answer:

- On the trap into the kernel all of the processor's registers are saved onto P_C's trap frame (onto the kernel stack for P_C). This is P_C's user-mode context.

- When P_C blocks, we save its kernel-mode context onto its kernel stack.

- We then restore P_D's context from its kernel stack. This is its kernel-mode context.

- Flush/Invalidate the contents of the TLB.

- Then P_D's user-mode context is retreived from the trap frame in its kernel stack and restored to the processor registers just prior to returning from the exception to continue running in user mode.

Problem 4. (*12 points*)

a. (*2 pts*)   Why is it important to have a separate kernel stack for each thread?

Answer: Otherwise a thread will be able to access kernel data in user mode and change it.

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

Answer:

- System call

- Interrupt

- Exception

c. (*3 pts*)   Consider the following factors:

- Size of the page table

- Internal fragmentation

- I/O overhead

Which of these factors may argue for a large page size and which could be used to argue for a smaller page size? Explain (**one sentence each**).

Answer:

- Larger page size means the number of pages is smaller.

- Smaller page size will waste less memory in the last allocated page.

- I/O is more efficient when done in larger chunks (i.e. larger page size).

d. (*4 pts*)   Explain in brief how OS uses the *timer interrupt* for scheduling.

Answer: The OS uses the timer interrupt to measure the passage of time and determine when the quantum of a running process has expired, at which point the process is preempted. The timer interrupt also ensures that the kernel will get the CPU back from a running process so that it can make scheduling decisions.

Problem 5. (*14 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. We have a TLB that can hold **eight** entries and during execution of a process P_1, it has the following entries.

Virtual Page#	Physical Frame#	Valid	Dirty
AC	2CD	1	1
DA	5DF	1	1
0	1F	1	0
29	6C1	0	1
21	32	0	0

Note: All numbers in this question are in **hexadecimal**. Also consider the *Dirty* bit has the same meaning as in an OS/161 TLB entry.

   a. (*8 pts*)   If possible, explain how the MMU will translate the following virtual addresses into physical address. If it is not possible, explain what will happen and why. **Show and explain how you derived your answer.**

4 KB page size = 2^{12}, and so 3 hex digits for offset.
32-12 = 20 bits, and so 5 hex digits for virtual page number.

Load from virtual address = 0x000A CF91.

Answer:
Offset = F91
Virtual Page = AC
Valid bit = 1, so the TLB entry is valid.
Load means dirty bit doesnt matter.
Translation occurs.
Frame number = 2CD
Phyical address = 0x002C DF91

Store at virtual address = 0x0000 03F8.

Answer:
Offset = 3F8
Virtual Page = 0
Valid bit = 1, so the TLB entry is valid.
Dirty bit = 0, so the page is read-only.
Translation fails.
Trying to write on a read-only page. Exception EX_MOD is raised.

b. (*6 pts*) Now consider we have the same entries in the TLB and the following is part of the page table for process P_1.

	Physical Frame#	Read/Write
...	...	...
1	3D2	1
...	...	...
13	1F6	1
...	...	...
132	259	1
...	...	...
1328	981	1
...	...	...
13289	FFC	1
...	...	...

Note: Consider the *Read/Write* flag in page table entry has the same meaning as the *Dirty* flag in the TLB entry.

If possible, explain how the MMU will translate the following virtual addresses into physical address. If it is not possible, explain what will happen and why. **Show and explain how you derived your answer.**

Load from virtual address = 0x0001 3289.

Answer:
Offset = 289
Virtual Page = 13
Not in TLB. Translation fails.
TLB miss on load. Exception EX_TLBL is raised.
Exception handler loads a new TLB entry

Virtual Page#	Physical Frame#	Valid	Dirty
13	1F6	1	1

Exception handler returns. Instruction is executed again (PC did not advance).

Offset = 289
Virtual Page = 13
Valid bit = 1, so the TLB entry is valid.
Load means dirty bit doesnt matter.
Translation occurs.
Frame number = 1F6
Phyical address = 0x001F 6289

Problem 6. (*18 points*) This question uses the following notation (as used in the course notes) to describe resource allocation in a computer system:

- D_i : demand vector for process P_i

- A_i : current allocation vector for process P_i

- U : unallocated (available) resource vector

Given the scenario below, fill in the details for the resource allocation graph, **indicate if the system is deadlocked, and justify your answer**.

PLEASE USE SOLID LINES WITH ARROWS FOR ALLOCATION EDGES AND DASHED OR DOTTED LINES WITH ARROWS FOR REQUEST EDGES.

$U = (0, 0, 0, 0, 1)$
$D_1 = (0, 1, 0, 0, 1), D_2 = (0, 0, 1, 0, 1), D_3 = (0, 0, 0, 0, 1), D_4 = (0, 0, 0, 0, 1)$
$A_1 = (1, 0, 1, 1, 0), A_2 = (1, 1, 0, 0, 0), A_3 = (0, 0, 0, 1, 0), A_4 = (1, 0, 0, 0, 0).$
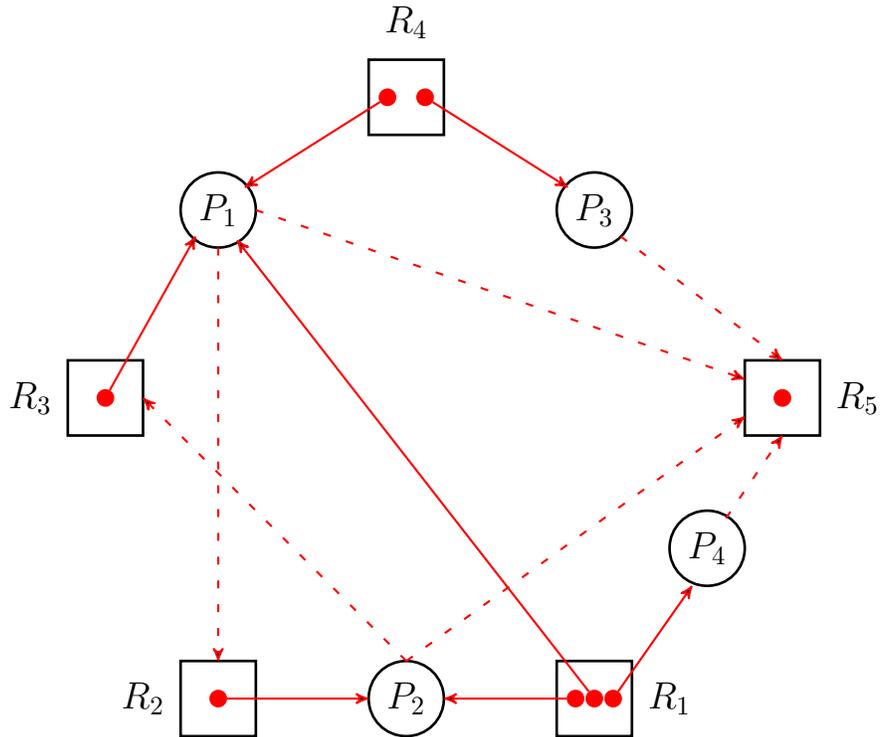
Space for **Problem 6**. Please mention if you answer other questions here.

$D_1 = (0, 1, 0, 0, 1), D_2 = (0, 0, 1, 0, 1), D_3 = (0, 0, 0, 0, 1), D_4 = (0, 0, 0, 0, 1)$
$A_1 = (1, 0, 1, 1, 0), A_2 = (1, 1, 0, 0, 0), A_3 = (0, 0, 0, 1, 0), A_4 = (1, 0, 0, 0, 0).$

Using the Deadlock Detection algorithm,

$R = U = (0, 0, 0, 0, 1)$
$D_1 \leq R$? No
$D_2 \leq R$? No
$D_3 \leq R$? Yes
$R = R + A_3 = (0, 0, 0, 1, 1)$

$D_1 \leq R$? No
$D_2 \leq R$? No
$D_4 \leq R$? Yes
$R = R + A_4 = (1, 0, 0, 1, 1)$

$D_1 \leq R$? No
$D_2 \leq R$? No

So the system is deadlocked. P_1 is allocated R_3 and it needs R_2 to finish and P_2 is allocated R_2 and it needs R_3 to finish.