CSE 451 Midterm Exam
May 13th, 2009

Your Name: _____

Student ID: _____

General Information:

This is a closed book examination. You have **50 minutes** to answer as many questions as possible. The number in parentheses at the beginning of each question indicates the number of points given to the question. There are **8 pages** on this exam (check to make sure you have all of them), and there is a total of **100 points** in all. Write all of your answers directly on this paper. Make your answers as concise as possible. If there is something in the question that you believe is open to interpretation, then please go ahead and interpret, but state your assumptions in your answer.

Problem 1: (25 points)

Which of the following require assistance from hardware to implement correctly and/or safely? For those that do, circle them, and name the necessary hardware support. For those that don't briefly explain why (one or two sentences at most).

System call: **requires hardware support, in particular to set the CPU's protection bit to run in protected kernel model**.

Process creation: **requires HW support. must do a system call, so need a protection bit. Also, need to manipulate address spaces (i.e., create new page tables for the new process).**

Thread creation: [ok if you argued that kernel threads require system calls, and hence need a protection bit, but user-level threads do not need assistance from HW]

Process context switch: **requires HW support. need a timer interrupt to prevent starvation, and also need to manipulate address spaces (usually by changing a register to point to the page directory for the new process, e.g., cr3 on x86)**

Thread context switch: [ok if you argued that kernel threads require system calls, but user-level threads do not need hw assistance]

Lock acquisition : **requires hw support. need support for atomic instructions (e.g., test and set), or support for disabling/enabling interrupts.**
[note: there are software-only solutions to the critical section problem – see problems 6.9 and 6.10 in the textbook -- and therefore it is possible to use those to implement a (very slow) lock. If you claimed you don't need HW for lock acquisition, and justified this by arguing for a software-only solution to CS, we gave you full credit.]

Lock release: [even if you require HW support to grab a lock, in many cases you don't need it to release the lock. See, for example, how test-and-set based locks works – you don't use the test-and-set to release the lock.]

Problem 2: (25 points)

Three processes P1, P2, and P3 have priorities P1=1, P2=5, P3=10. ("10" is higher priority than "1".) The processes execute the following code:

```
P1:  begin
     <code sequence A>
     lock(X);
     <critical section CS>
     unlock(X);
     <code sequence B>
     end
```

```
P2:  begin
     <code sequence A>
     lock(X);
     lock(Y);
     <critical section CS>
     unlock(Y);
     unlock(X);
     <code sequence B>
     end
```

```
P3:  begin
     <code sequence A>
     lock(Y);
     lock(X);
     <critical section CS>
     unlock(X);
     unlock(Y);
     <code sequence B>
     end
```

The X and Y locks are initialized to "unlocked", i.e., they are free. <*code sequence A*> takes **2** time units to execute, <*code sequence B*> takes **3** time units to execute, and <*critical section CS*> takes **4** time units to execute. Assume `lock()` and `unlock()` are instantaneous, and that context switching is also instantaneous.

P1 begins executing at time **0**, P2 at time **3**, and P3 at time **10**. There is only one CPU shared by all processes.

a) (**9 points)** Assume that the scheduler uses a priority scheduling policy: at any time, the highest priority process that is ready (runnable and not waiting for a lock) will run. If a process with a higher priority than the currently running process becomes ready, preemption will occur and the higher priority process will start running.

Diagram the execution of the three processes over time. Calculate the job throughput and the average turnaround time.

On your diagram, use:
"A" to signify that the process is executing <*code sequence A*>
"B" to signify that the process is executing <*code sequence B*>
"X" to signify that the process holds lock X and is in the critical section
"Y" to signify that the process holds lock "Y" and is in the critical section
"2" to signify that the process holds both locks and is in the critical section,
and leave blank space if the process is not executing (for any reason).

We've started your diagram for you on the next page.

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
P1	A	A	X			X	X	X																	B	B	B													
P2				A	A				2	2			2	2								B	B	B															
P3											A	A			2	2	2	2	B	B	B																			

Job throughput: 3 jobs in 27 time units = 1/9 jobs/time-unit

Average turnaround time: job 1: 27 time unit turnaround.
job 2: 21 time unit turnaround.
job 3: 11 time unit turnaround.

Average: (27 + 21 + 11) / 3 = 59 / 3 = 19 2/3

b) **(9 points)** "Deadlock" is a phenomenon that occurs when a process waits for a resource that will never become available.

Repeat a) with the same assumptions and priorities, but this time assume that:

- P1 begins executing at time 0, P2 begins executing at time 6, and P3 begins executing at time 3.
- P1 has priority 1, P2 has priority 10, and P3 has priority 5.

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
P1	A	A	X			X			X	X	B	B	B																											
P2							A	A																																
P3				A	A																																		

Job throughput: _____0_____

Average turnaround time: _____infinity!_____

c) **(7 points)** Propose and justify a solution to the problem you uncovered in b). Use no more than 3 sentences.

Change P3 to grab/release locks in the same order as P2

Problem 3: (25 points)

(a) **(9 points)** Find all of the logic bugs in the following piece of code. Modify the code (in place, or above the specific lines you'd like to change) to fix the bugs.

```
~~semaphore mutex = 0;~~    // BUG #1
semaphore mutex = 1;
// returns true if the item was added, false otherwise
bool add_item_to_queue(queue_t queue, item_t item) {
  P(mutex);
  if (is_full(queue)) {
     V(mutex);   // BUG #2 (this was missing)
     return false;
  }
  append_to_queue(queue, item);
  V(mutex);
  return true;
~~  V(mutex);~~  // BUG #3 - this should be before the return
}
```

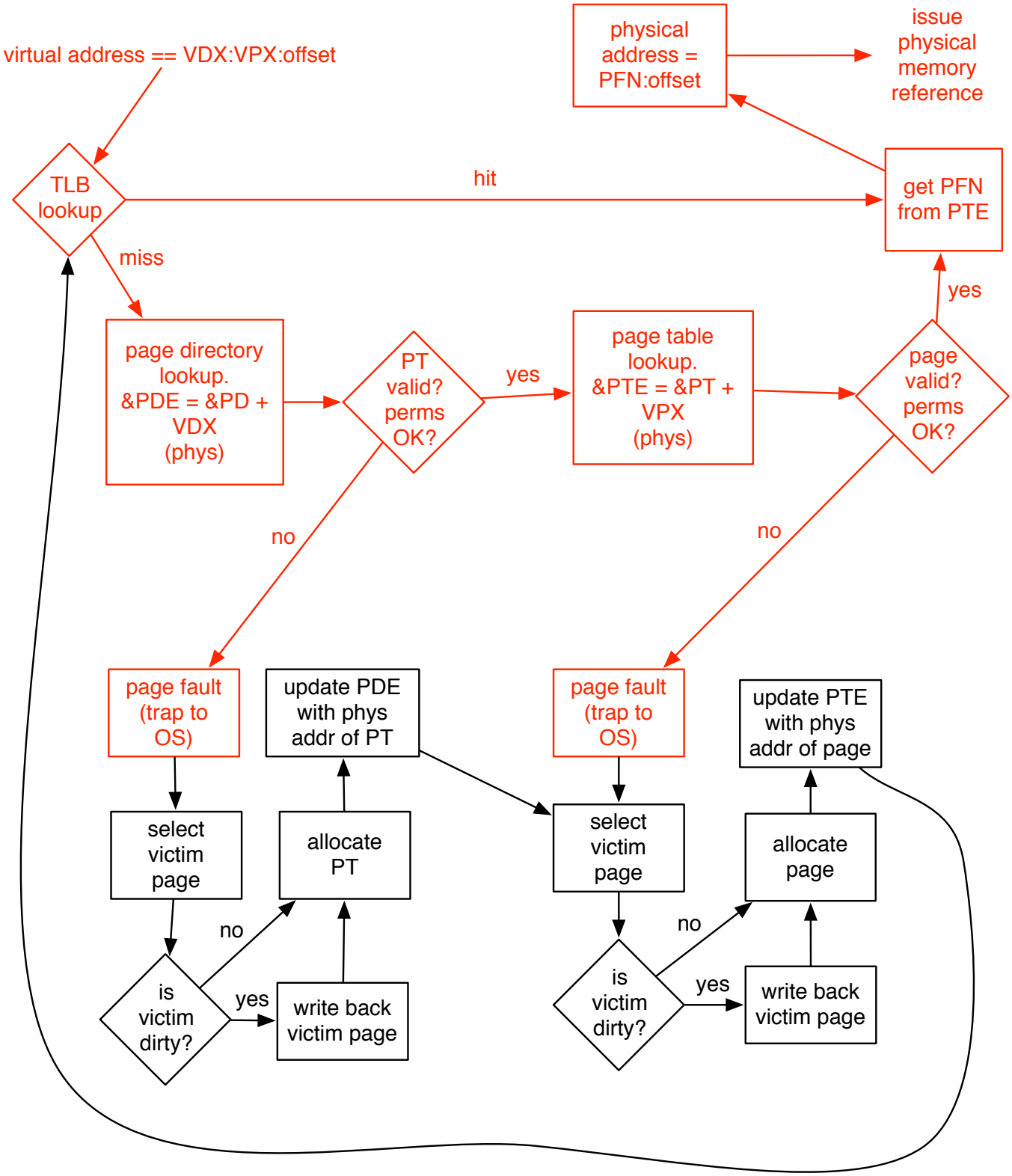
(b) **(16 points)** Find all of the logic bugs in the following piece of code. Modify the code (in place, or above the specific lines you'd like to change) to fix the bugs. If it is helpful, assume the existence of a thread-safe queue class that has add() and remove() functions.

```
~~semaphore thread_available = 0, work_available = 0;~~
semaphore thread_available = NUM_THREADS,
          work_available = 0;

~~functionptr_t dispatch_function;~~
queue dispatch_function_q;

thread_t thread_pool[NUM_THREADS];

void threadpool_start(int my_id);

// PUBLIC FUNCTION; initialize the thread pool
void threadpool_init() {
   for (int i=0; i<NUM_THREADS; i++) {
      thread_pool[i] = thread_create(threadpool_start, i);
   }
}

// PUBLIC FUNCTION; dispatch a worker to run function "f"
void threadpool_dispatch(functionptr_t f) {
  P(thread_available);

  ~~dispatch_function = f;~~
  enqueue(dispatch_function_q, f);

  V(work_available);
}

// PRIVATE FUNCTION: workers wait here until dispatched
void threadpool_start(int my_id) {
   printf("Worker thread %d is alive!\n", my_id);
   while(1) {
     P(work_available);

     x = dequeue(dispatch_function_q);
     x();

     V(thread_available);
   }
}
```

Problem 4: (25 points)

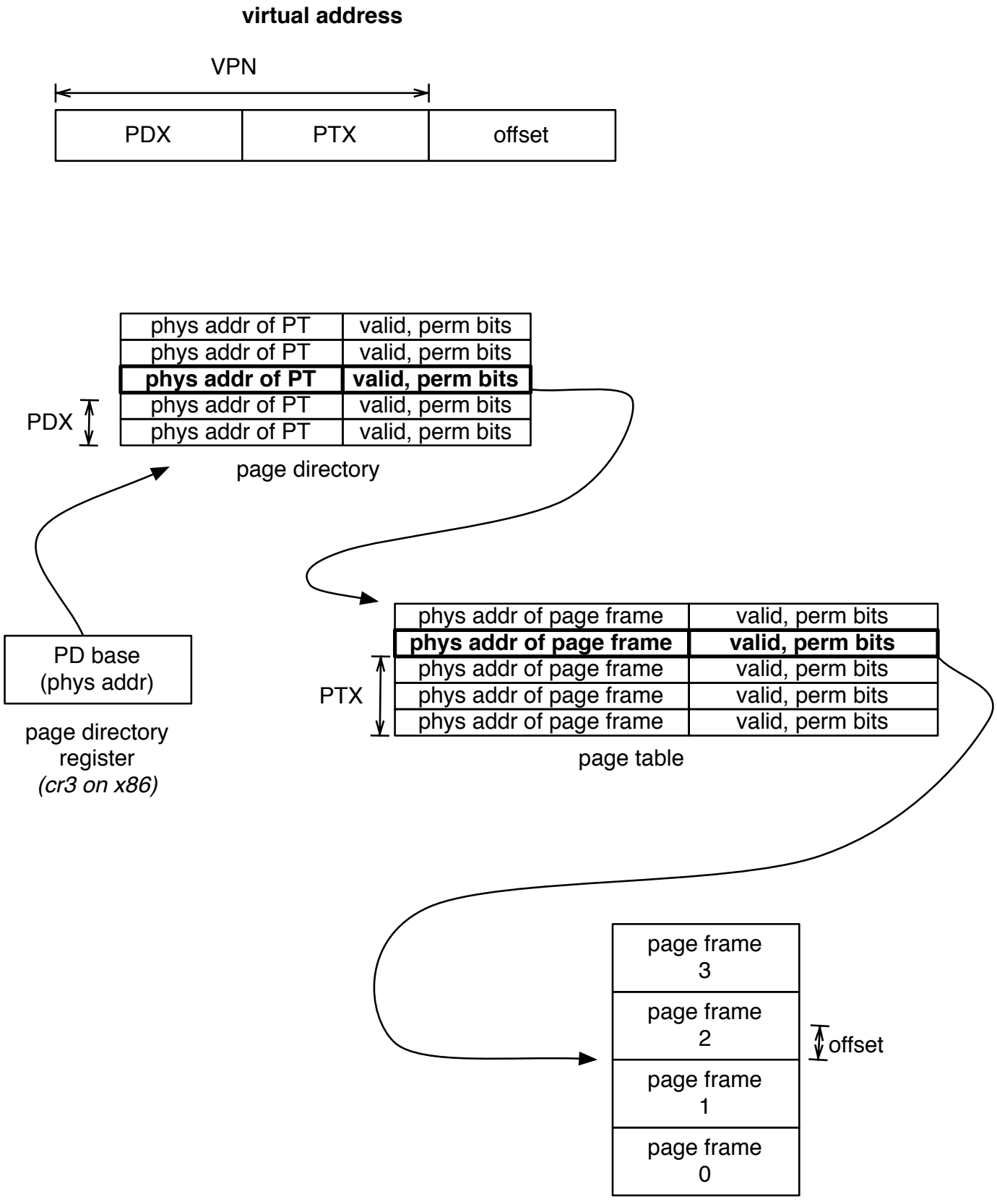
Consider an operating system that uses paging in order to provide virtual memory capabilities; the paging system employs a TLB and a two-level page table. Assume that page tables are "wired" (pinned) in physical memory, and assume that the TLB is hardware-loaded (i.e, the CPU walks page tables to load the TLB on a miss).

Draw a flow chart that describes the logic of handling a memory reference. Your chart must include the possibility of TLB hits and misses, as well as page faults. Be sure to mark which activities are accomplished by hardware, and which are accomplished by the OS's page fault exception handler. Also be sure to clearly identify addresses as either virtual or physical.



RED = HARDWARE

BLACK = OS


virtual address
VPN
PDX
PTX
offset
phys addr of PT
valid, perm bits
phys addr of PT
valid, perm bits
phys addr of PT
valid, perm bits
phys addr of PT
valid, perm bits
phys addr of PT
valid, perm bits
PDX
page directory
PD base
(phys addr)
page directory
register
(cr3 on x86)
phys addr of page frame
valid, perm bits
phys addr of page frame
valid, perm bits
phys addr of page frame
valid, perm bits
phys addr of page frame
valid, perm bits
phys addr of page frame
valid, perm bits
PTX
page table
page frame
3
page frame
2
page frame
1
page frame
0
offset