Q1 Address translation
25 Points

Consider the following 32-bit x86 page table setup.

CR3 holds 0x0.

The Page Directory Page at physical address 0x0 (the flags are PTE_P (present), PTE_U (user-accessible), and PTE_W (writable):

```
PDE 0: PPN=0x00001, PTE_P, PTE_U, PTE_W
PDE 1: PPN=0x00001, PTE_P, PTE_U, PTE_W
PDE 2: PPN=0x00002, PTE_P, PTE_U, PTE_W
... all other PDEs are zero
```

The Page Table Page at physical address 0x00001000 (which is PPN 0x00001):

```
PTE 0: PPN=0x00005, PTE_P, PTE_U, PTE_W
PTE 1: PPN=0x00006, PTE_P, PTE_U, PTE_W
... all other PTEs are zero
```

The Page Table Page at physical address 0x00002000 (which is PPN 0x00002):

```
PTE 0: PPN=0x00006, PTE_P, PTE_U, PTE_W
PTE 1: PPN=0x00005, PTE_P, PTE_U, PTE_W
... all other PTEs are zero
```

Find all virtual addresses that map to physical address 0x00005005

Q1.1
3 Points

Can you update the page table directory, i.e., write into it, while this page table is used? Explain your answer.

Q1.2 Physical pages
5 Points

What physical pages (provide physical page numbers) are mapped by this page table?

Q1.3 Virtual addresses
7 Points

What virtual addresses are mapped by this page table?

Q1.4 More page tables
10 Points

Using the same format for describing the page table as in the questions above construct a page table that maps the following virtual addresses [4MB; 8MB] to physical addresses [0; 4MB] (here we map up to 4MB boundary, i.e., the page 4MB and up doesn't have to be mapped). Note: you should use 4KB page tables.

Q2 Process organization
10 Points

Q2.1
5 Points

Imagine you would double the size of the stack for user-level processes in xv6, which lines of the xv6 you have to change? (explain your answer using relevant xv6 code)

Q2.2
5 Points

You want to maximize the amount of physical memory xv6 can handle. What is the max number you can change `PHYSTOP` to without crashing an xv6 system? Explain your answer.

Q3 Interrupts
15 Points

Q3.1
5 Points

An xv6 process executes a system call in the kernel, can it be preempted with a timer interrupt? (Explain your answer using relevant xv6 source code)

Yes

No

Relevant xv6 source code:

Explanation:

Q3.2
5 Points

If an xv6 process is preempted with a timer interrupt and is executing the timer interrupt handler in the kernel, can it be preempted with another timer

interrupt? (Explain your answer using relevant xv6 source code)

Yes

No

Relevant xv6 source code:

Explanation:

Q3.3
5 Points

You're executing line 2581 (i.e., just starting inside fork() in which lines the execution can be preempted with the timer interrupt? Explain your answer.

```
2579 int
2580 fork(void)
2581 {
2582   int i, pid;
2583   struct proc *np;
2584   struct proc *curproc = myproc();
2585
2586   // Allocate process.
2587   if((np = allocproc()) == 0){
2588     return -1;
2589   }
2590
2591   // Copy process state from proc.
2592   if((np->pgdir = copyuvm(curproc->pgdir, curproc->sz)) == 0){
2593     kfree(np->kstack);
2594     np->kstack = 0;
2595     np->state = UNUSED;
2596     return -1;
2597   }
2598   np->sz = curproc->sz;
2599   np->parent = curproc;
2600   *np->tf = *curproc->tf;
2601
2602   // Clear %eax so that fork returns 0 in the child.
```

```
2603   np->tf->eax = 0;
2604
2605   for(i = 0; i < NOFILE; i++)
2606     if(curproc->ofile[i])
2607       np->ofile[i] = filedup(curproc->ofile[i]);
2608   np->cwd = idup(curproc->cwd);
2609
2610   safestrcpy(np->name, curproc->name, sizeof(curproc->name));
2611
2612   pid = np->pid;
2613
2614   acquire(&ptable.lock);
2615
2616   np->state = RUNNABLE;
2617
2618   release(&ptable.lock);
2619
2620   return pid;
2621 }
```

Explanation:

Q4 System call arguments
15 Points

Q4.1
5 Points

Alice is confused about the checks in line 3618, specifically she argues that she can remove the following code || (uint)i+size > curproc->sz from that line.

```
3610 int
3611 argptr(int n, char **pp, int size)
3612 {
3613   int i;
3614   struct proc *curproc = myproc();
3615
3616   if(argint(n, &i) < 0)
3617     return -1;
3618   if(size < 0 || (uint)i >= curproc->sz || (uint)i+size > curproc->sz)
3619     return -1;
```

```
3620   *pp = (char*)i;
3621   return 0;
3622 }
```

Can you explain what can go wrong if she does this change?

Q4.2
10 Points

Alice wants to make system calls in xv6 a bit faster. Specifically she wants to pass the first 3 arguments to the system call in EDI, ESI, and EDX registers. What changes she needs to do to make it work (explain your answer using relevant xv6 code)?

Q5 A pipe under the sync
14 Points

Here's the source code of piperead().

```
6850 int
6851 piperead(struct pipe *p, char *addr, int n)
6852 {
6853   int i;
6854
6855   acquire(&p->lock);
6856   while(p->nread == p->nwrite && p->writeopen){
6857     if(myproc()->killed){
6858       release(&p->lock);
6859       return -1;
6860     }
6861     sleep(&p->nread, &p->lock);
6862   }
6863   for(i = 0; i < n; i++){
6864     if(p->nread == p->nwrite)
6865       break;
6866     addr[i] = p->data[p->nread++ % PIPESIZE];
6867   }
6868   wakeup(&p->nwrite);
```

```
6869   release(&p->lock);
6870   return i;
6871 }
```

Q5.1 Acquire what lock?
2 Points

What is the role of line 6855, i.e., `acquire(&p->lock);`?

Q5.2 Grind never rest
4 Points

What could happen if line 6861 `sleep(&p->nread, &p->lock);` is removed?

- Two readers reading from the pipe at the same time will read the same data out twice.
- Two writers writing to the pipe at the same time will overwrite each other's data.
- Both the first option and the second option. Both the readers and the writers can cause data races and lead to incorrect behaviors.
- Some programs might run slower.
- xv6 will freeze if it runs on a single-core CPU.
- None of the above.

Q5.3 Lock the door before sleep?
4 Points

Why does the `sleep` function in line 6861, `sleep(&p->nread, &p->lock);`, takes `&p->lock` as an argument?
"*Because sleep takes two arguments*" *is not an acceptable answer.*

Q5.4 Return what eye?
4 Points

Can the return value of piperead be greater than PIPESIZE?

Explanation: