Name: Key

Operating Systems
Fall 2015
Test 1
October 5, 2015

Closed books, notes, cell phones, PDAs, iPods, laptops, etc. No headphones, please. No calculator needed.

You have 75 minutes to solve 6 problems. You get 10 bonus points. As with any exam, you should read through the questions first and start with those that you are most comfortable with. If you believe that you cannot answer a question without making some assumptions, state those assumptions in your answer.

Partial credit will be offered only for meaningful progress towards solving the problems.

Please read and sign below if you agree with the following statement:

In recognition of and in the spirit of the academic code of honor, I certify that I will neither give nor receive unpermitted aid on this exam.

Signature:________________________________

0	10/10
1	/30
2	/5
3	/10
4	/10
5	/10
6	/15
Total	**/90**

1. [30 points] Answer the following multiple-choice questions. Circle all answers that apply. For these questions all 30 statements count equally.

[True statements in read]

A. In a typical modern OS, which of the following statements are true about user applications and the kernel?

1. The kernel runs at privileged level.
2. Applications run at privileged level.
3. Applications may directly invoke any function calls in the kernel.
4. The kernel frequently relinquishes control and must depend on the user application to allow it to regain control.

B. Which of the following instructions should be protected, i.e., can execute only when the processor is running in kernel mode?

5. MOV cr3, src; (move the value in src to control register CR3. CR3 contains the physical address of the base of the page table).
6. TRAP; (jump to kernel's syscall dispatcher with kernel privilege)
7. ADD; (add numbers)
8. JMP; (jump to a different instruction)
9. INB (contact I/O device).

C. Which of the following are elements of a typical process descriptor? (Recall that a process descriptor or process control block is the per-process state kept by the kernel.)

10. Process state (blocked/runnable);
11. Address space;
12. The state of the CPU registers;
13. Process ID.

D. Which of the following statements are true about processes?

14. Any process can access any other process's memory by default.
15. Timer interrupts are generally required for a kernel to correctly isolate processes.

16. The kernel must run one process to completion before it starts another process.

17. A user-level process can crash the kernel if not programmed carefully.

E. Which of the following statements are true when the kernel switches execution from the current process to another process?

18. The kernel must save the register values (including IP (Instruction Pointer), SP (stack pointer) etc.) of the current process to memory.

19. The kernel must save the content of the current process' memory on disk.

20. The kernel must close all the files opened by the current process.

21. The kernel must make the page table of the next process active.

F. Which of the following are true about memory virtualization:

22. In a multiprogramming system the main memory is generally shared among a number of processes.

23. A hardware mechanism is needed for translating virtual addresses to physical main memory addresses at the time of execution of the instruction that contains the reference.

24. In a modern multiprogramming environment the programmer knows at the time of coding how much space will be available and where that space will be.

25. A physical address is the location of a word relative to the beginning of the program and the processor translates that into a logical address.

G. Which of the following are true about memory systems based on segmentation or paging:

26. The placement policy determines where in real memory a process piece is to reside.

27. The smaller the page size, the greater the amount of internal fragmentation.

28. Segmentation does not eliminate external fragmentation.

29. All segments of all programs must be of the same length.

30. It is impossible to have both paging and segmentation in the same system.

2. [5 points] What happens if you run the following program on UNIX?

```
main () {
     while (fork() >= 0)
          ;
}
```

This is the so-called fork bomb. It creates processes until it fills the kernel's data structures in memory and the system becomes unresponsive. The only way to fix this is with a hard reboot (unless the OS has, by design, a way to prevent this situation from happening). Grading: full credit was given only for noting the effect on the system (memory overwhelming, etc.). The observation that it's an infinite loop is not sufficient for full credit.

3. [10 points] Assume that 5 processes arrive at the ready queue at the times shown below. The estimated next burst times are also shown. Assume that an interrupt occurs at every arrival time.

   a. What are the average turnaround time and average response time if the preemptive *shortest remaining time first* scheduling policy is used?
   b. What are the average turnaround time and average response time if the non-preemptive *shortest job first* scheduling policy is used?

Process	Arrival Time	Burst Time
P1	**0**	**7**
P2	**1**	**1**
P3	**3**	**2**
P4	**4**	**4**
P5	**5**	**3**

[This is not the full solution, but it's enough to see what it should be. A schedule is necessary, otherwise it's difficult for you to get the times and me to grade]

a) P1 runs from 0 to 1, then from 2 to 3, and finally from 12 to 17. The response and average turnaround times have to account for the arrival time—you should not, for example, include the time from 0 to 5 for P5 in the response time.
b) A common mistake was to provide the same schedule as in the previous case. This is not correct, as the problem specifies that SJF is non-preemptive. This means that P1, which starts at 0 since it's the only one to run at that time, will not be preempted (interrupted) until it's done, at time 7. The schedule in this case is: P1, P2, P3, P5, P4.

4. [10 points] Assume that the following C code

```
int array[1000];
...
for (i = 0; i < 1000; i++)
   array[i] = 0;
```

translates into the following assembly code (the hexadecimal number in front of each instruction shows its virtual address):

```
0x1024 movl $0x0,(%edi,%eax,4)
0x1028 incl %eax
0x102c cmpl $0x03e8,%eax
0x1030 jne 0x1024
```

The first instruction moves the value zero (shown as $0x0) into the virtual memory address of the location of the array; this address is computed by taking the contents of the %edi register and adding the content of the %eax register multiplied by four to it. Thus, %edi holds the base address of the array, whereas %eax holds the array index (i); we multiply by four because the array is an array of integers, each size four bytes.

The second instruction increments the array index held in %eax. The third instruction compares the contents of that register to the hex value 0x03e8, or decimal 1000. If the comparison shows that that two values are not yet equal (which is what the `jne` instruction tests), the fourth instruction jumps back to the top of the loop.

Assume the array is located at virtual address 40000 and thus, since it's 1000 x 4 bytes long, it is located between 40000 and 44000 (not including the last byte).

Assume a virtual address space of size 64 KB and a page size of 1 KB. What is the memory reference string generated by this code?

This problem is from the textbook, solution explained in Chapter 18.4.
Addresses referenced:

0x1024, 40000, 0x1028, 0x102c, 0x1030, 0x1024, 40004, 0x1028, 0x102c, 0x1030,
0x1024, 40008, 0x1028, 0x102c, 0x1030, 0x1024, 40012, 0x1028, 0x102c, 0x1030,
0x1024, 40016, 0x1028, 0x102c, 0x1030, 0x1024, 40020, 0x1028, 0x102c, 0x1030,
0x1024, 40024, 0x1028, 0x102c, 0x1030, 0x1024, 40028, 0x1028, 0x102c, 0x1030,
0x1024, 40032, 0x1028, 0x102c, 0x1030, 0x1024, 40036, 0x1028, 0x102c, 0x1030,
0x1024, 40040, 0x1028, 0x102c, 0x1030, …

Notice that you have both hexadecimal and decimal values above. If you got thus far and realize that you have to translate these addresses into virtual page numbers, you get full credit. The complete answer is below:

0x1024 = 0001 0000 0010 0100 => VPN = 000100 = 4

VPN for virtual address 40000 is 40000/1024 = 39

```
4 39 4 4 4      |
4 39 4 4 4      |        240 times
……              |
4 39 4 4 4      |

4 40 4 4 4      |
…               |        240 times
4 40 4 4 4      |

4 41 4 4 4      |
…               |        240 times
4 41 4 4 4      |

4 42 4 4 4      |
…               |        240 times
4 42 4 4 4      |

4 43 4 4 4      |
…               |        40 times
4 43 4 4 4      |
```

5. [10 points] Write the binary translation of the logical address 0001000010111110 under the following hypothetical memory management schemes, and explain your answer.

   a. A paging system with a 512-address page size, using a page table in which the frame number happens to be four times smaller than the page number.

   b. A segmentation system with a 2K-address maximum segment size, using a segment table in which bases happen to be regularly placed at real addresses: 22 + 4,096 x segment number.

a) The offset is represented on 9 bits (because the page contains 2^9=512 addresses) Thus, the offset is: **010111110,** the virtual page number is 0001000. Divided by 4 (means shifting 2 bits to the right) => frame number is **00010.**

The result: **00010010111110 $(1214)_{10}$**

b) Size of the segment is 2^{11}, thus offset within segment is given by the least representative 11 bits of the address: **00010111110 = 190**

Segment number is thus the remaining 00010 = 2. In the physical memory, base address for this segment is thus 22 + 4096 x 2 = 8214

The result: 8214+ 190 = **$(8404)_{10}$**

6. [15 points] A process has four frames allocated to it. (All the following numbers are decimal, and everything is numbered starting from zero.) The time of the last loading of a page into each frame, the last access to the page in each frame, and the virtual page number in each frame are as shown below (the times are in clock ticks from the process start at time 0 to the event).

Virtual Page Number	**Frame**	**Time Loaded**	**Time Referenced**
20	0	60	161
22	1	130	160
24	2	10	162
31	3	20	163

A page fault to virtual page 23 has occurred at time 164. Which **frame** will have its contents replaced for each of the following memory management policies?

The key here was to look at the two time columns and use the relevant information in the right algorithm. Also note that time increases, thus the lowest number for time is the earliest time (of course).

- FIFO (first-in-first-out)

  2

- LRU (least recently used)

  1

- Optimal. Use the following reference string: 23, 25, 23, 25, 20, 22, 31, 23, 24.

  2