16

3/14/17 CS 570 Exam 1 50 Name [redacted] Page 1

1: $\frac{0}{2}$ 2: $\frac{0}{2}$ 3: $\frac{-}{0}$ 4: $\frac{0}{2}$ 5: $\frac{0}{3}$ 6: $\frac{0}{8}$ 7: $\frac{6}{8}$ 8: $\frac{1}{2}$ 9: $\frac{1}{3}$ 10: $\frac{5}{7}$ 11: $\frac{0}{6}$ 12: $\frac{-}{3}$ 13: $\frac{-}{2}$ 14: $\frac{2}{2}$ Blank: $\frac{1}{0}$

Note: For any [entire] question you leave completely blank, you get 20% of the credit! This examination is scheduled to take 75 minutes. You may use additional paper if you do not have sufficient room for your answers on this exam. Put your name on this sheet now. Problem weights are shown above. Answer each question *concisely*.

1. Some languages (such as Java) use the concept of a 'monitor' to help cooperating processes coordinate their activities. In one or two sentences, briefly explain what the monitor concept does for you ('you' being the programmer...).

   Make sure that processes have ample room in memory.

2. *If the* `read()` *call on the next line is successful* (so <u>don't</u> tell me "-1" !!!), can
   `count = read(fd, buffer, nbytes);`
   return any value in `count` other than `nbytes`? Explain.
   (Note that this is `read()`, not `write()`.)

   open() does that

   Pg 2 late Tanenbaum

   Yes, it would return the file descriptor of the file..

3. A computer system has enough room to hold 4 programs in its main memory. These programs are idle (waiting for I/O) half the time. What fraction of time is the CPU busy? Show your calculations!

   1/16

   Pg 94.

   $\frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2}$

4. Your friend says: "When a page is brought into memory, the R bit is set to one. The paging daemon is therefore never going to find a page with the R bit set to zero, so I don't see how it can it possibly ever think there is any good candidate for page replacement." What would you tell your friend? That it wouldn't be looking in the page table where they are set to 1, but look in physical memory where they can be 0

   Pg 208

5. (a) Can a parent and child have a page that is in both their working sets at the same time? Explain.

   yes if there is ample space in memory

   #32 pg 254

   (b) Can two unrelated processes [for example, processes owned by two distinct users] have a page that is in both their working sets at the same time? Explain.

   yes, as long as there is ample room.

Something like it on final

pg 39 in notes

6. Consider the following attempt at solving Dining Philosopher's Problem, with N (N=5) forks and a single napkin-holder (in the center of the table, able to be acquired by the philosophers one at a time). As usual, the `take_fork()` and `take_napkin_holder()` routines will block until the resource becomes available.

```
#define N 5
void philosopher(int i)
{
    while (TRUE) {
/*1.*/  think();
/*2.*/  take_napkin_holder();
/*3.*/  take_fork((i+1) % N);
/*4.*/  take_fork(i);
/*5.*/  release_napkin_holder();
/*6.*/  eat();
/*7.*/  take_napkin_holder();
/*8.*/  release_fork(i);
/*9.*/  release_fork((i+1) % N);
/*10*/  release_napkin_holder();
    }
}
```

(a) If the above is an optimal solution, just say so. On the other hand:
If it does not allow maximum concurrency, explain why.
If deadlock/starvation can occur, state which one occurs, and outline the failure scenario.

~~No, napkin holds up the forks~~

(b) Now consider removing lines 7 and 10 (that is, do not try to guard the **release** commands). If this modified version is an optimal solution, just say so. On the other hand:
If it does not allow maximum concurrency, explain why.
If deadlock/starvation can occur, state which one occurs, and outline the failure scenario.

pg 81 of notes

Yes, is a ~~optimal~~ solution

(c) Now consider removing lines 2, 5, 7 and 10 (that is, do not use the napkin-holder at all). If this modified version is an optimal solution, just say so. On the other hand:
If it does not allow maximum concurrency, explain why.
If deadlock/starvation can occur, state which one occurs, and outline the failure scenario.

Yes, it is a ~~optimal~~ solution

7. In this problem, assume that the operating system uses demand paging and consider the following page replacement algorithms: *First-In-First-Out*, *Least-Recently-Used*, and *Optimal*. For each of the algorithms, assume that the replacement policy is local. Assume a process consists of six pages, `0 1 2 3 4 5`. Page `0` is automatically loaded when we start running the program. Other pages are loaded (as they are referenced) by the page fault mechanism. This process is allowed only ***THREE*** pages in memory at any one time.

0 1 2 5 4 3 0 1

(a) Invent a sequence of page references that this process can make (starting with 0, 2, 4, as shown below) that will allow the *First-In-First-Out* algorithm to perform as well as *Optimal*. [That is, find a sequence where *FIFO* has the same number of page faults as *Optimal* algorithm would have for that same sequence of page references.] Place an asterisk to the left of each entry that involves a page fault. **NOTE**: For part (a), **ignore** the last (LRU) column for now. Entries marked with an 'x' do not need to be filled in. That is, fill in the sequence of page references all the way down the first column to firmly establish the pattern, but you can stop doing the calculations after row 8.

After reference to page number	Pages in Memory					
		FIFO		Optimal		LRU
0	*	0	*	0	*	0
2	*	0 2	*	0 2	*	0 2
4	*	0 2 4	*	0 2 4	*	0 2 4
1	*	1 2 4	*	0 2 1	*	0 2 1
2		1 2 4		0 2 1		0 2 1
3	*	1 3 4	*	3 2 1	*	0 2 3
1		1 3 4		3 2 1	*	0 2 1
5	*	1 3 5	*	5 2 1	*	0 2 5
1	x	x x x	x	x x x	x	x x x
2	x	x x x	x	x x x	x	x x x
3	x	x x x	x	x x x	x	x x x

4 was most recently used, not least!

working set is number of pages you need to prevent page faults

(b) Now, for this exact same sequence of page references you have in column 1, fill in the rightmost column above, showing what pages would be in memory if the page replacement decisions were guided by the LRU algorithm.

(c) Based on the paging behavior in your example above, what is the size of the working set for this process? $\boxed{5}$ 4?? ok What is the size of the resident set for this process? $\boxed{3}$

8. Tanenbaum enumerates per-process items (things that can be shared amongst threads in a single process) and per-thread items (items private to each thread).

(a) Does each thread need its own stack, or is this a per-process item? $\boxed{\text{per-thread}}$

(b) Tanenbaum asks: "Why is the register set listed as a per-thread item rather than a per-process item? After all, the machine has only one set of registers." Briefly discuss.

Q. 10 pg 170 Tanenbaum

Different threads can't share things such as program counter

9. Suppose the semaphore S has value 2 and empty queue. (Further assume that only this one process is accessing S.) What value is in the variable x at the time the following code causes a block? (Show your calculations!)

```
up(S);        S=3!
x = 0;
for (;;) {
   down(S);   S=1
   x++;
   down(S);   S=0
}             blocks!
```

$\boxed{x= \cancel{1}}$

10. Consider the simple UNIX shell written for this class.
   (a) One possible command line we could have fed to p2 might have been:
   `>&lower tr A-Z< upper a-z`
   You built an argument vector [perhaps called newargv]. In the boxes below, show what the first few entries of newargv would point to for this command line just prior to calling execvp() (write a single character, write NULL, or write ?? if the results might vary):

   not always

*newargv[0]= t	*newargv[1]= A	*newargv[2]= a	*newargv[3]= ??	*newargv[4]= ??

   (b) What was the point of using fflush()? Where was the most logical place to do that?
   Clear a stream, specifically stdout. Most logical place to put it is ~~before calling execvp().~~

   pg 80 in notes

   (c) Your friend says, "I put my fflush() calls right before execvp(), so that the child will report any command-line problems before it begins running a new binary. I don't need it anywhere else." Explain what is wrong with both of those assertions.
   ? fflush() would not report any command-line problems, it would be better if it was called before the fork.

   (d) A *zombie* is created whenever a parent does not wait for a child. Your p2 did not wait for background processes, creating zombies. Yet these zombies often disappeared while p2 was still processing commands, long before your p2 exited (**not** "just before", assuming you did things right). How did you arrange for them to go away?
   Created a while loop that reaped the processes in the process table until it encountered the child.

   while(some_pid != child_pid)
   wait(NULL);

   (e) I generally tested your shell with command such as `p2 < inputlines`, but `p2 inputlines` was supposed to react in exactly the same way. How did you make this happen? (In this answer, make sure you mention the system calls you used: you should say what they did, but don't worry if you don't recall their exact parameters.)
   In the beginning of main had an if block

```
if(strcmp(newargv[], "") != 0) { // Has an argument
   rstream(newargv[]);
}
```

   ↑ not correct name because I can't remember

   What it does is open a file where the name is the parameter passed, and executes each line as a command

   need dup2() as well

BE ON FINAL!

11. A particular old computer uses 16-bit physical RAM addresses, and is loaded with as much memory as it can support.

    (a) If the system only has room for 64 page frames, How **large** is each **physical** page **frame**? (In bytes; show your calculations.)

    2^4

    16 bits

    2^{16} bytes of RAM 6 | 10 – 2^{10} = 1024 bytes

    $\boxed{\cancel{1024}}$ 102

    (b) How many **bits** are needed to specify the offset within a **virtual** page? $\boxed{\cancel{1024}}$ 10

    (c) If the system uses 32-bit virtual addresses, how many virtual **pages** are available? Show your calculations.

    32

    22 | 10

    $\boxed{\cancel{64 \text{ million}}}$

    2^{22} = 4 million

    Pg 192

    (d) If virtual page 0 maps to physical frame 2, and virtual page 1 maps to physical frame 0, what physical address corresponds to virtual address 1024? Show your calculations.

    $\boxed{X}$

← doesn't matter

12. A computer with a 16KB page size, a 128MB main memory, and 32-bit virtual addresses uses an inverted page table to implement its virtual memory. The case of the computer is black and the power cord is 6 feet long. How big should the hash table be to ensure a mean hash chain length of <u>less</u> than 1? (Show your calculations!)

    $\boxed{2^{14}}$ 2^{13}

    2047

    1024

    1023

    0

    page 1

    0+0 – offset

    #19, Pg 250

    128MB

    $$\frac{2^7 \cdot 2^{20}}{2^4 \cdot 2^{10}} = 2^{27-14} = 2^{13}$$

    16KB

13. Suppose that a 32-bit virtual address is broken up into four fields, a, b, c, and d. The first three are used for a three-level page table system. The fourth field, d, is the offset. If you were only told the value of one of these four numbers, could you figure out the number of virtual pages the system can support? (If your answer depends on *which* of the four numbers you know, state which one, and explain your reasoning.)

    #13 Pg 250

    2^{32}

    32 – d → a+b+c

    | a | b | c | d | 32

    2^{a+b+c} pages

    offset – 2^d

14. One of the <u>process</u> scheduling algorithms we discussed was "Shortest Job First". If your pointy-haired boss asked your opinion on whether this method of scheduling should be used in the next release of your company's software, what would you tell him?

    Pg 40 drawer

    It would be a bad idea because if there are a lot of short processes, large ones get starved. Also it would involve magic fairies to try and calculate runtime.