ECE 254 S16 Final Exam Solutions

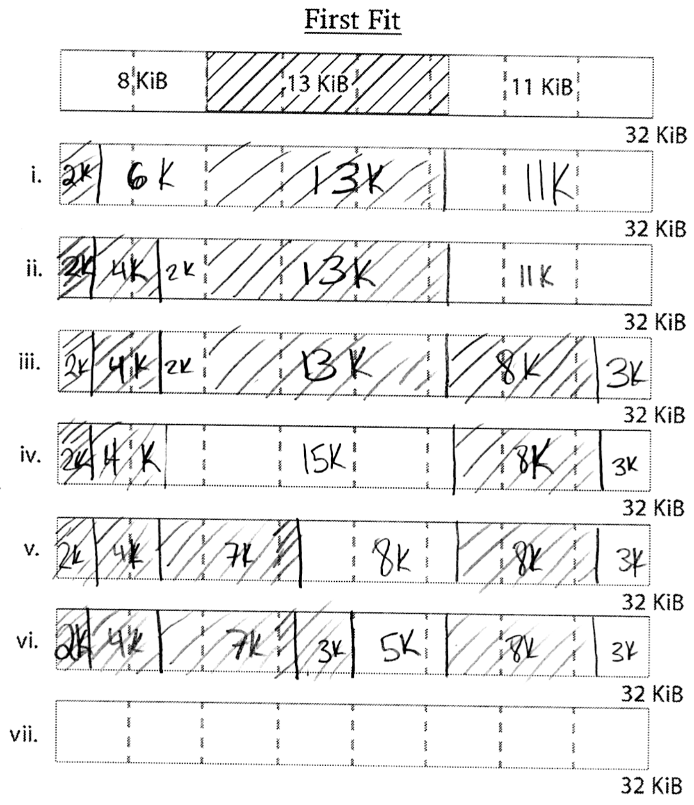
J. Zarnett

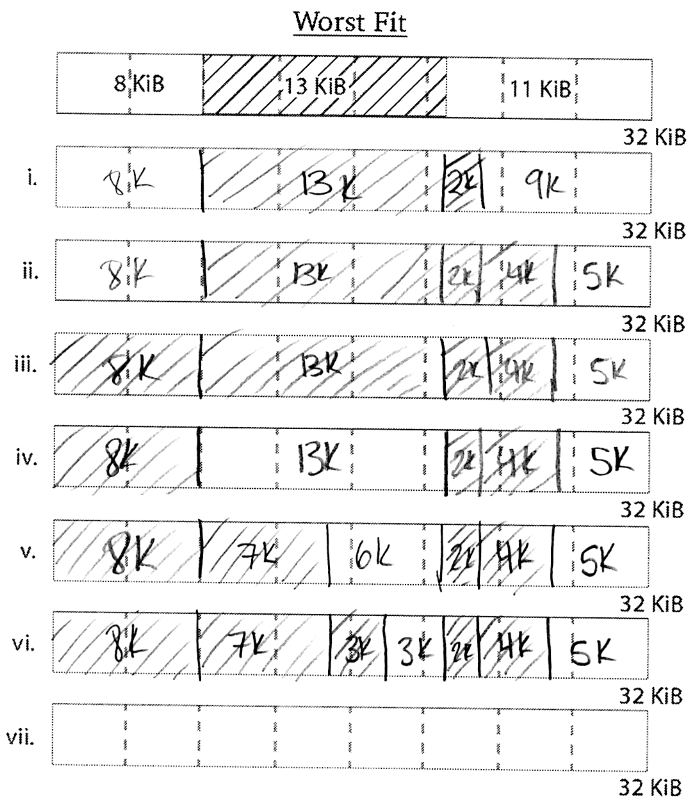
August 10, 2016

(1a) Because arrays in C are stored in row major order, the code on the left will have more page faults and cache misses, causing lower performance.

(1b)

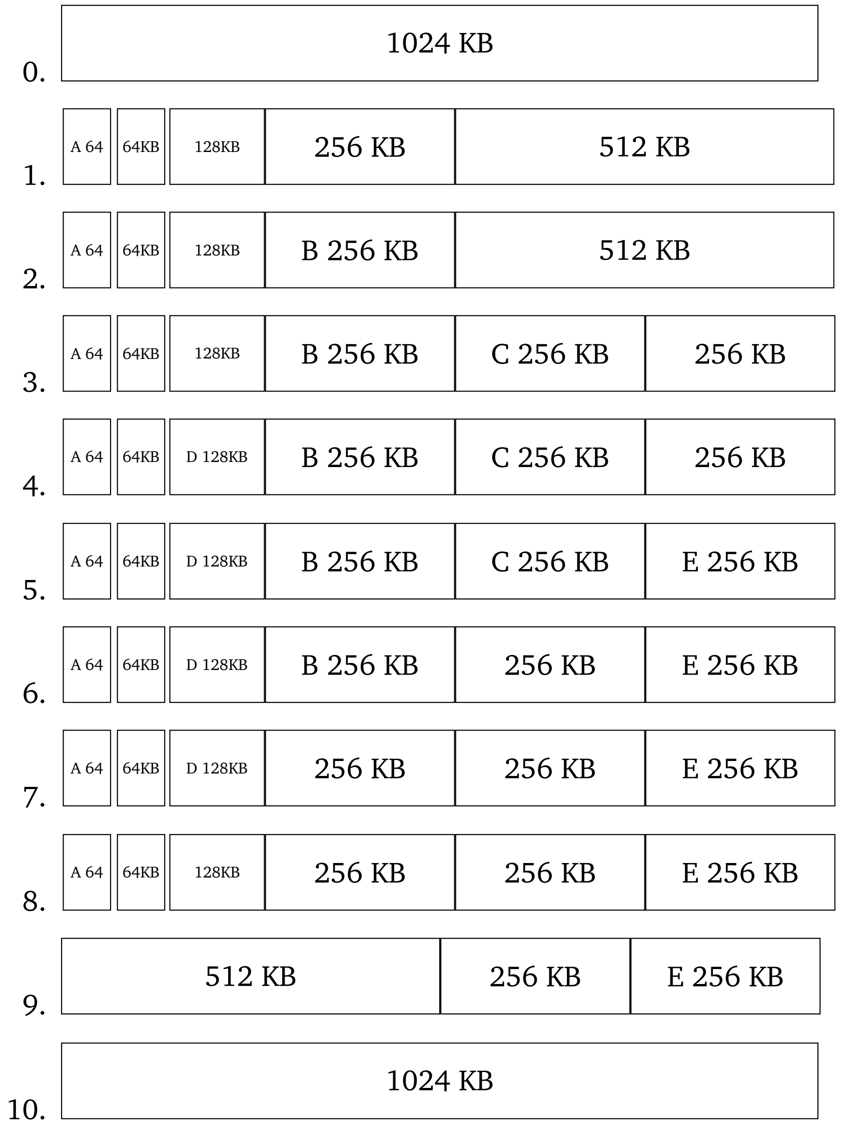
The instruction for item vii is missing from the exam. Whoops. Well, that item can be left blank or the same as vi. Well, as long as you answered the question and did not put something incorrect in the last row, you get the marks for that (free +2, yay!).





(1c)

1. Allocate A: 64 KB
2. Allocate B: 256 KB
3. Allocate C: 200 KB
4. Allocate D: 128 KB
5. Allocate E: 140 KB
6. Deallocate C
7. Deallocate B
8. Deallocate D
9. Deallocate A
10. Deallocate E


0. 1024 KB
1. A 64 | 64KB | 128KB | 256 KB | 512 KB
2. A 64 | 64KB | 128KB | B 256 KB | 512 KB
3. A 64 | 64KB | 128KB | B 256 KB | C 256 KB | 256 KB
4. A 64 | 64KB | D 128KB | B 256 KB | C 256 KB | 256 KB
5. A 64 | 64KB | D 128KB | B 256 KB | C 256 KB | E 256 KB
6. A 64 | 64KB | D 128KB | B 256 KB | 256 KB | E 256 KB
7. A 64 | 64KB | D 128KB | 256 KB | 256 KB | E 256 KB
8. A 64 | 64KB | 128KB | 256 KB | 256 KB | E 256 KB
9. 512 KB | 256 KB | E 256 KB
10. 1024 KB

Step 7: Important to NOT coalesce when B and C are deallocated. They're next to each other and the same size, but not binary buddies

Step 9: Deallocation of A triggers a whole bunch of coalescence in the first block.

(1d)

Sample answer (but anything reasonable is accepted):

- Smaller page size reduces internal fragmentation; larger pages increase it.
- Smaller page size requires more entries in the TLB to get a reasonable hit rate; larger pagers require fewer.
- Smaller page sizes means larger page tables (more entries), harder to search; larger pages means smaller page tables, less searching.

(2a)

An interrupt that runs every 4 microseconds will result in 250 000 interrupts per second (yikes!). If one interrupt is 225 cycles, 56 250 000 cycles are used for interrupt handling. 100 MHz is 1×10^8 cycles. 56 250 000 / (1×10^8) = 56.25%.

(2b)

Example scenario: a parent process will spawn a child process to do some work and then waits on that process. Since the parent process knows that it will be waiting for the child, it could choose to increase the priority of the child in the hopes of minimizing its wait time.

(2c)

1a: 20t 1b: 15t

2a: 20t 2b: 19.5t

(Why? Alternation in the RR queue compared to the FIFO where one process runs to the finish and that's it).

(2d)

Highest Priority Period

Execution Order: P_2, P_1, P_4, P_5, P_3.

Turnaround times:
P_1: 3
P_2: 1
P_3: 20
P_4: 7
P_5: 12

Waiting times:
P_1: 1
P_2: 0
P_3: 12
P_4: 3
P_5: 7

Average waiting time: $(1 + 0 + 12 + 3 + 7)/5 =$ 4.6ms.

Alternatively, because P_1 and P_4 have equal priority, it can be in this order: P_2, P_4, P_1, P_5, P_3, which will obviously change the turnaround and waiting times.

FIFO

Execution Order: P_1, P_2, P_3, P_4, P_5.

Turnaround times:
P_1: 2
P_2: 3
P_3: 11
P_4: 15
P_5: 20

Waiting times:
P_1: 0
P_2: 2
P_3: 3
P_4: 11
P_5: 15

Average waiting time: 6.2 ms

RR

Execution Order: P_1, P_2, P_3, P_4, P_5, P_3, P_4, P_5, P_3, P_5, P_3.

Remember, this is FIFO with Time Slices of 2 ms.

Turnaround times:
P_1: 2
P_2: 3
P_3: 20
P_4: 13
P_5: 18

Waiting times:
P_1: 0
P_2: 2
P_3: $3 + 4 + 4 + 1 = 12$
P_4: $5 + 4 = 9$
P_5: $7 + 4 + 2 = 13$

Average waiting time: 7.2 ms

Shortest Process Next

Execution Order: P_2, P_1, P_4, P_5, P_3.

If you made the "right" assumption about whether P_1 or P_4 goes first, then this is the same as Highest Priority, Period, and you can save yourself some work!

Turnaround times:
P_1: 3
P_2: 1
P_3: 20
P_4: 7
P_5: 12

Waiting times:
P_1: 1
P_2: 0
P_3: 12
P_4: 3
P_5: 7

Average waiting time: 4.6ms.

(3a)

This sort of approach is simpler in terms of programming; there is no need to write into the code the open and close system calls and prevents the possibility of errors. However, if a file is referenced in a long-running program then a file appears to be "in use" for a very long time, even though that program may be finished with it. This may cause other processes to wait for a file to be available for a long time.

(3b)

Sequential: Media player (play the media from start to finish), grep (read the whole file, search for a given keyword).

Non-Sequential: accounting software (access different parts of the stored data depending on what users are doing), text editor (edit an arbitrary part of the file, scroll up or down as desired).

(3c)

Use of diagrams was appreciated in this answer.

Seek Time: time to position the arm of the hard drive.

Rotational Latency: the time to rotate the disk around so that the disk head is above the right position.

Track: a circular area of disk comprised of sectors.

Sector: a block of data stored on disk.

Cylinder: tracks on different disk platters stacked vertically.

Head crash: when the disk read/write head impacts the surface of the disk, likely resulting in damage and data loss.

(4a)

This solution assumes 0 means locked and 1 unlocked. Any alternative convention is okay as long as it's used consistently.

```
int lock;
void foo( ) {
    while( compare_and_swap(lock, 0, 1) == 1) {
        sched_yield();
    }
    /* Critical Section */
    lock = 0;

}
```

Many students used the compare-and-swap routine to set it back to 0 which his nice but not necessary (and not wrong either). But please don't read the value of lock in an if statement to check if this succeeded.

(4b)

There are many more than 5 problems in the code; but here are a few.

1. Variable `numBytes` undefined on line 19. Change to `num`.[1]
2. Double `free` of `void_arg` and `start` at lines 40 and 41; fix by deleting one of them (either is fine but not both).
3. Use of the returned pointer after `temp_sum` has been freed on line 68. Fix by moving that call to `free()` after the increase of `global_sum`.
4. Pthread mutex is initialized but not destroyed $\rightarrow$ memory leak. Fix this by adding call to `pthread_mutex_destroy(&result_lock)` after all threads are done and before the call to `pthread_exit` (so between line 70 and 80).
5. `pthread_join` called with wrong argument on line 66, should be `&temp_sum` (address OF the void pointer).
6. `max_val` and `global_sum` are in the wrong order in the `printf` statement on lines 71/72 $\rightarrow$ swap them.

[1]A good many students objected during the exam on the basis that this is a syntax error. It is not. A syntax error is something like `x += ;` . The expression is not correct because the right side of the assignment operator is missing. Here the syntax is valid: left side, comparison, right side. But the expression has no meaning because the right side is undefined. Just because the compiler finds the problem here doesn't make it a syntax error.