Instructions:

1. No aids are permitted except non-programmable calculators with no persistent memory.
2. Turn off all communication devices. Communication devices must be stored with your personal items for the duration of the exam. Taking a communication device to a washroom break during this examination is not allowed and will be considered an academic offence.
3. Place all bags at the front or side of the examination room, or beneath your table, so they are inaccessible.
4. There are four (4) questions, some with multiple parts. Not all are equally difficult.
5. The exam lasts 150 minutes and there are 120 marks.
6. Verify that your name and student ID number is on the cover page and that your examination code appears on the bottom of each page of the examination booklet.
7. If you feel like you need to ask a question, know that the most likely answer is "Read the Question". No questions are permitted. If you find that a question requires clarification, proceed by clearly stating any reasonable assumptions necessary to complete the question. If your assumptions are reasonable, they will be taken into account during grading.
8. Do not fail this city.
9. After reading and understanding the instructions, sign your name in the space provided below.

Signature

Marking Scheme (For Examiner Use Only):

Q	Mark	Weight	Q	Mark	Weight	Q	Mark	Weight	Q	Mark	Weight
1a		11	2a		3	3a		6	4a		5
1b		14	2b		3	3b		4	4b		25
1c		3	2c		4	3c		10			
1d		4	2d		26						
Total											**120**

Question 1: Memory [32 marks total]

1A: Caching [11 marks]

A system has a cache that can contain three pages and it has pages 0 through 7. The cache starts out completely empty. Suppose the page references occur in this order: $1, 2, 3, 4, 2, 4, 1, 3, 4, 1, 2, 3, 7$ Complete the tables below to show the state of cache after each reference and sum up the total page faults.

First-In First-Out

Step	`cache[0]`	`cache[1]`	`cache[2]`	Fault?
1	1	—	—	✓
2	1	2	—	✓
3	1	2	3	✓
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
FIFO Page Faults:				

Least Recently Used

Step	`cache[0]`	`cache[1]`	`cache[2]`	Fault?
1	1	—	—	✓
2	1	2	—	✓
3	1	2	3	✓
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
LRU Page Faults:				

1B: Dynamic Memory Allocation [14 marks]

The diagrams below show a 32 KB block of memory that will be used to fulfill memory allocations in this question. Use shading to indicate allocated blocks and also show the size of each block. Grey dashed lines are guidelines in increments of 4 KB to assist you in drawing the blocks in the correct sizes. The initial state of the system is shown at the top of the diagram: an 8 KB free block, a 13 KB allocated block, and an 11 KB unallocated block.

Perform the following allocations and deallocations using the *best fit* algorithm on the left side and using the *next fit* algorithm on the right side. Assume the 13 KB block is the one most recently allocated. Remember to perform coalescence immediately when it is appropriate.

i. Allocate 8 KB

ii. Allocate 3 KB

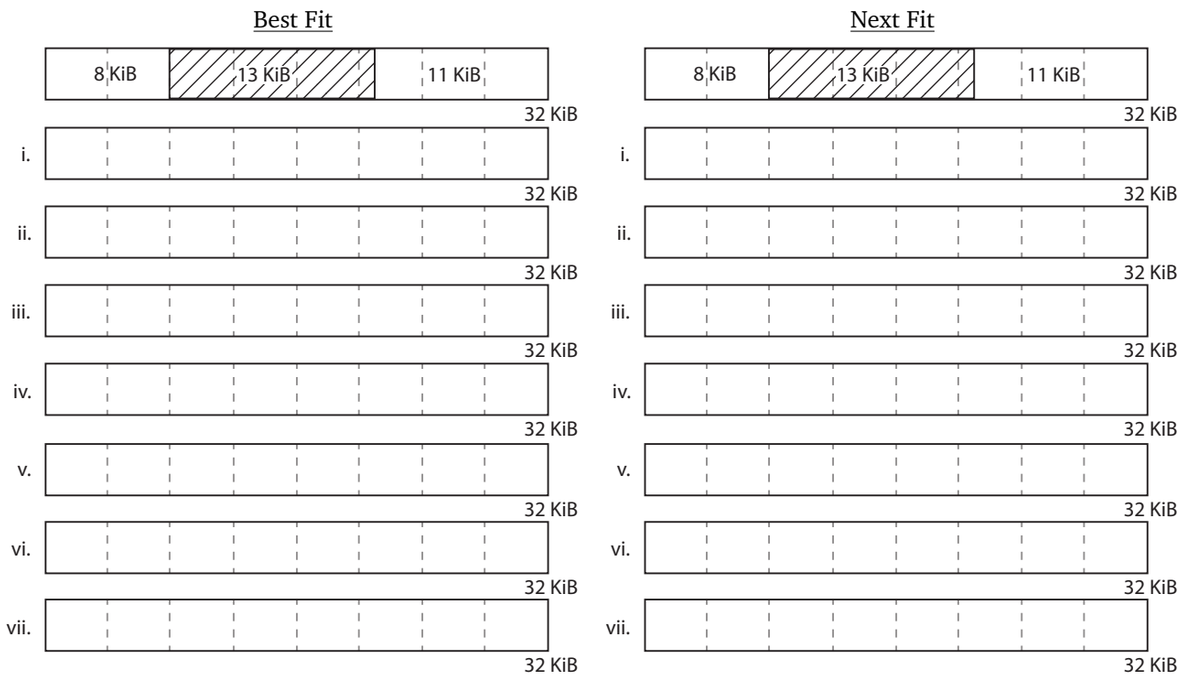
iii. Deallocate 13 KB

iv. Allocate 4 KB

v. Deallocate 8 KB

vi. Allocate 6 KB

vii. Allocate 2 KB

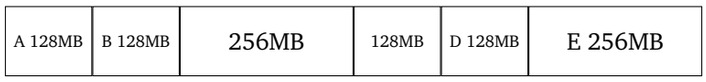


1C: Inverted Page Tables [3 marks]

Recall that in 32-bit computers we can have normal page tables, but in a 64-bit computer we use inverted page tables. Explain why, using numerical arguments, in a 64-bit computer where a page has a size of 4K (2^{12}) and each entry is 8 bytes, inverted page tables are necessary.

1D: Binary Buddy [4 marks]

The current condition of 1GB of memory allocated using the binary buddy system is shown below:



Show the state of this block of memory below after the memory marked as D has been deallocated.

Question 2: Scheduling [38 marks total]

2A: Round Robin Robin [4 marks]

Round-Robin schedulers normally maintain a list of all processes that are ready to run, with each process appearing once in the list. What would happen if a process appeared twice in the list? Think of two (2) things that would happen or that the operating system would need to do if this were permitted.

2B: Linux Scheduler [4 marks]

Suppose a Linux system is using the Constant Time ($O(1)$) scheduler. Describe how a process with priority level 124 will transition through the scheduler's data structures and run to completion, assuming it will run for multiple time slices and never be blocked. For full marks, there should be at least four items.

2C: Red-Line Overload? [4 marks]

You are working on a real time system that has the following periodic tasks:

Task	Worst Case Computation Time (ms)	Period (ms)
T_1	5	10
T_2	1	5
T_3	90	630
T_4	30	300

Using these values, compute the utilization U and indicate whether or not the system is overloaded. Remember that Utilization U is calculated as $\sum_{k=1}^{n} \frac{c_k}{\tau_k}$ where c_k is the computation time of task k and τ_k is the period of task k.

2D: Uniprocessor Scheduling [26 marks]

In this question, you will schedule the following set of processes, all of which have arrived in the ready queue at the same time, in the following order:

Process	Execution Time (ms)	Priority
P_1	10	5
P_2	2	1
P_3	7	3
P_4	7	2
P_5	9	4

This system, like UNIX, follows the rule that lower numbers indicate higher priority. If time slicing is necessary, one time slice equals 3 ms. Hint: turnaround time is the time of execution plus waiting time.

Complete the table below to fill in the execution order, turnaround time for each process, waiting time for each process, and then the average waiting time of all processes.

Scheduling Algorithm	Execution Order	Turnaround Times	Waiting Times	Average Waiting Time
Highest Priority Period		P_1. P_2. P_3. P_4. P_5.	P_1. P_2. P_3. P_4. P_5.	
First-In-First-Out		P_1. P_2. P_3. P_4. P_5.	P_1. P_2. P_3. P_4. P_5.	
Round Robin		P_1. P_2. P_3. P_4. P_5.	P_1. P_2. P_3. P_4. P_5.	
Shortest Process Next		P_1. P_2. P_3. P_4. P_5.	P_1. P_2. P_3. P_4. P_5.	

Question 3: Input/Output & Files [20 marks total]

3A: Disk Scheduling Algorithms [6 marks]

Consider a disk that has 500 cylinders labelled 0 through 499. Suppose the incoming disk reads waiting to be serviced are: `420, 283, 310, 170, 430`. The starting position of the disk read head is 355 and it is moving in the ascending direction. List the order the requests are serviced for the following disk scheduling algorithms:

Shortest Seek Time First (SSTF):

SCAN:

C-LOOK:

3B: File Allocation Methods [4 marks]

Explain, in four (4) points, the file allocation method of **indexed allocation**. Use point form.

3C: I/O Device Definitions [10 marks]

Define the following terms as they relate to I/O devices (2 marks each):

Device Driver

Buffering

Spooling

Block Device

Character Device

Question 4: Concurrency & Synchronization [30 marks total]

4A: Reusable Turnstile [5 marks]

Recall from lectures the reusable turnstile:

```
 1. wait( mutex )
 2. count++
 3. if count == n
 4.     signal( turnstile )
 5. end if
 6. signal( mutex )
 7. wait( turnstile )
 8. signal( turnstile )
 9. [critical point]
10. wait( mutex )
11. count--
12. if count == 0
13.     wait( turnstile )
14. end if
15. signal( mutex )
```

The semaphore `turnstile` is initialized to 0. This has a pattern that we have identified as a potential source of deadlock: `wait( turnstile )` is between the `wait( mutex )` and `signal( mutex )` calls. Explain why deadlock does not occur in the reusable barrier despite the nested wait calls and demonstrate that the barrier is reusable.

4B: Code Review [25 marks]

Your company has a standard coding problem they send to applicants. You have been asked by your manager to review the submission below from an applicant. Although it contains no syntax errors, it contains significant semantic errors. On the following pages, identify the line(s) of five (5) errors, what is wrong, and how it can be fixed. Your explanation may be written text or a code correction. Unlike a typical code review, do not make any comments about the code formatting. Assume a standard of at least C99 (so integers can be declared inside a for loop, for example).

```
1  #define NUM_CPUS 8
2  int index = 0;
3  int* results;
4  int max_val;
5  pthread_mutex_t results_lock;
6
7  int compare (const void * a, const void * b) {
8     return ( *(int*)b - *(int*)a );
9  }
10
11 int is_prime( int num ) {
12     if (num <= 1) {
13         return 0;
14     } else if (num <= 3) {
15         return 1;
16     } else if (num % 2 == 0 || num % 3 == 0) {
17         return 0;
18     }
19     for (int i = 5; i*i <= num; i+=6 ) {
20         if (num % i == 0 || num % (i + 2) == 0) {
21             return 0;
22         }
23     }
24     return 1;
25 }
26
27 void *find_primes( void *void_arg ) {
28     int *start = (int *) void_arg;
29     int *count = malloc( sizeof( int ) );
30
31     for (int i = *start; i < max_val; i += NUM_CPUS) {
32         if ( 1 == is_prime( i ) ) {
33             pthread_mutex_lock( &results_lock );
34             results[index] = i;
35             pthread_mutex_unlock( &results_lock );
36             ++index;
37             ++(*count);
38         }
39     }
40     free( void_arg );
41     pthread_exit( *count );
42 }
43 int main( int argc, char** argv ) {
44     max_val = atoi( argv[1] ); /* Read input as max val */
45     results = malloc( max_val * sizeof( int ) );
46     if ( results == NULL ) {
47         return -1;
48     }
49     pthread_mutex_init( &results_lock, NULL );
50
51     pthread_t threads[NUM_CPUS];
52     int global_sum = 0;
53
54     for ( int i = 0; i < NUM_CPUS; ++i ) {
55         int* start = &i;
56         pthread_create(&threads[i], NULL, find_primes, start);
57         pthread_detach( threads[i] );
58     }
59
60     void* temp_sum;
61     for ( int i = 0; i < NUM_CPUS; ++i ) {
62         pthread_join( threads[i], &temp_sum );
63         int *thread_sum = (int *) temp_sum;
64         global_sum += *thread_sum;
65         free( thread_sum );
66     }
67
68     printf("Found %d total primes less than %d.\n",
69         global_sum, max_val);
70
71     qsort(results, global_sum, sizeof(int), compare);
72
73     for (int i = 0; i < global_sum; ++i ) {
74         printf( "%d\n", results[i]);
75     }
76
77     pthread_mutex_destroy( &results_lock );
78     pthread_exit(0);
79 }
```

For your reference, some of the pthread functions:

```
pthread_create( pthread_t *thread, const pthread_attr_t *attributes,
                void *(*start_routine)( void * ), void *argument )
pthread_join( pthread_t thread, void **returnValue )
pthread_detach( pthread_t thread )
pthread_cancel( pthread_t thread )
pthread_exit( void *value )
pthread_mutex_init( pthread_mutex_t *mutex, pthread_mutexattr_t *attributes )
pthread_mutex_lock( pthread_mutex_t *mutex )
pthread_mutex_unlock( pthread_mutex_t *mutex )
pthread_mutex_destroy( pthread_mutex_t *mutex )
```

Problem 1 Line(s):

Problem 1 Description:

Problem 1 Solution:

Problem 2 Line(s):

Problem 2 Description:

Problem 2 Solution:

Problem 3 Line(s):

Problem 3 Description:

Problem 3 Solution:

Problem 4 Line(s):

Problem 4 Description:

Problem 4 Solution:

Problem 5 Line(s):

Problem 5 Description:

Problem 5 Solution: