Instructions:

1. No aids are permitted except non-programmable calculators.
2. Turn off all communication devices.
3. There are four (4) questions, some with multiple parts. Not all are equally difficult.
4. The exam lasts 150 minutes and there are 120 marks.
5. 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.
6. Do not fail this city.
7. 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		4	2a		3	3a		4	4a		5
1b		14	2b		3	3b		4	4b		25
1c		10	2c		4	3c		12			
1d		6	2d		26						
Total											**120**

Question 1: Memory [34 marks total]

1A: Memory Allocation in C [4 marks]

Remember that in C you can declare a two-dimensional array such as `int array[size1][size2]`. Hypothetically, you might expect that the two code fragments below are equivalent.

```
for( int j = 0; j < size2; ++i) {
    for (int i = 0; i < size1; ++j) {
        bar( array[i][j] );
    }
}
```

```
for( int i = 0; i < size1; ++i) {
    for (int j = 0; j < size2; ++j) {
        bar( array[i][j] );
    }
}
```

In reality, because C stores two dimensional arrays in row-major order, the fragment on the right is more efficient. Explain why this means the code on the right has better performance.

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 *first fit* algorithm on the left side and using the *worst fit* algorithm on the right side. Remember to perform coalescence immediately when it is appropriate.

i. Allocate 2 KB

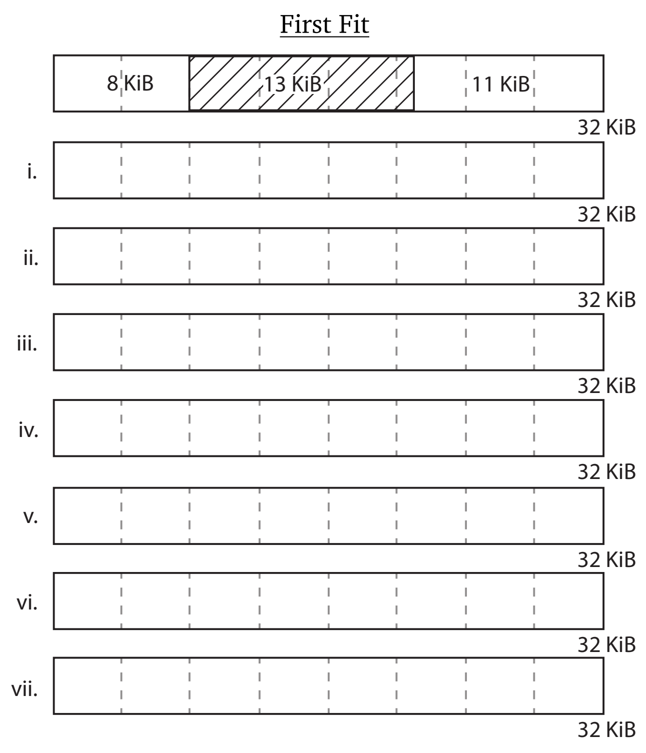
ii. Allocate 4 KB

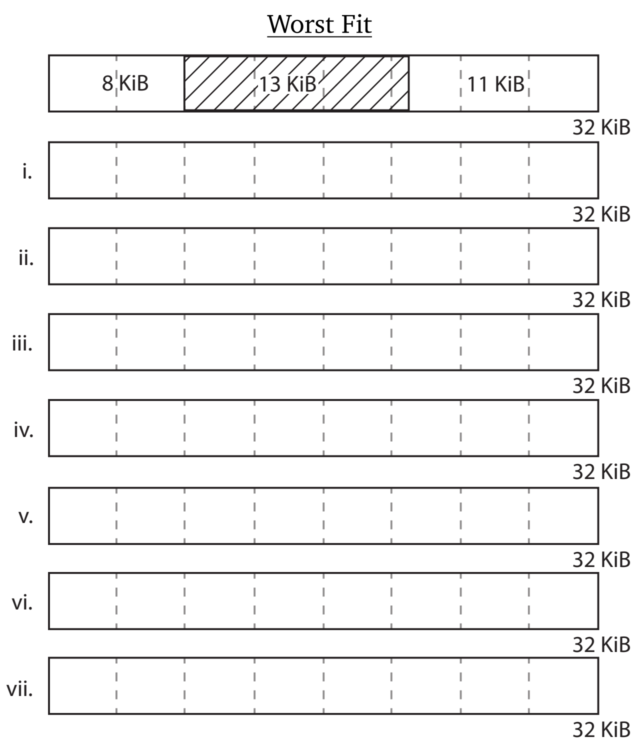
iii. Allocate 8 KB

iv. Deallocate 13 KB

v. Allocate 7 KB

vi. Allocate 3 KB



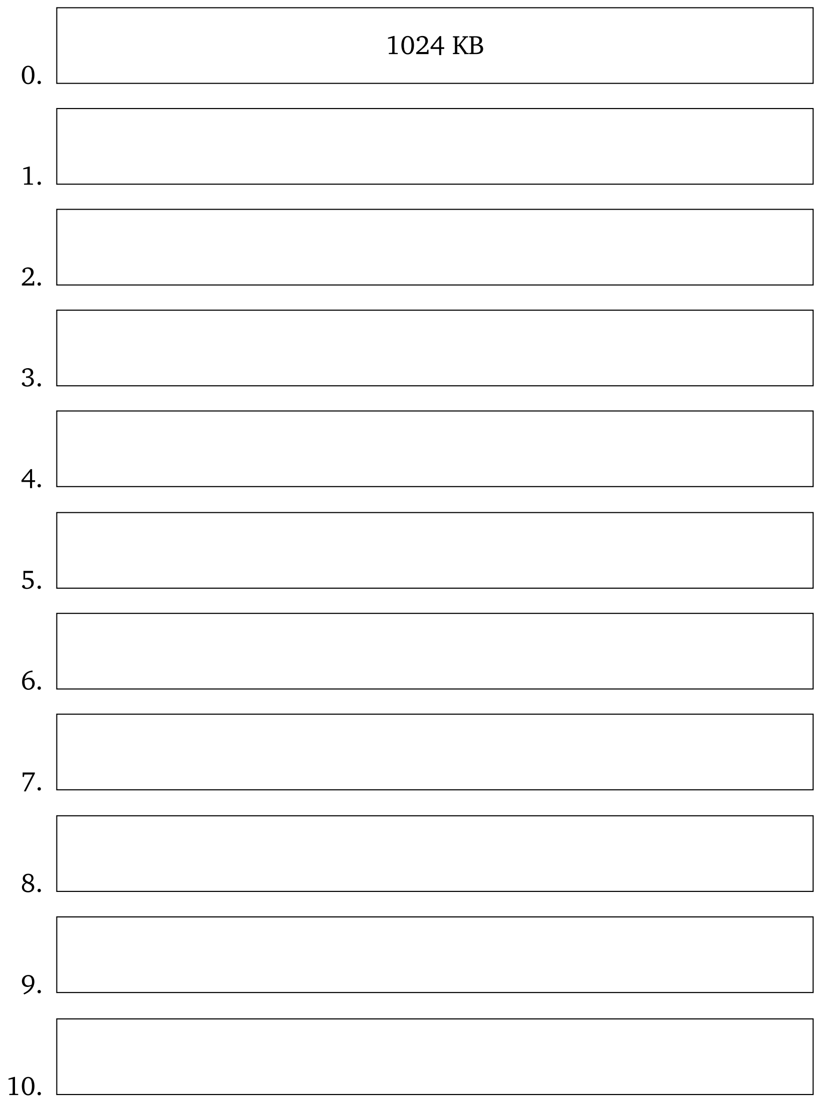


1C: Binary Buddy [10 marks]

In this 1024-KB segment, memory is allocated using the binary buddy system. Using the system shown in class, you will illustrate how the following requests are allocated. Clearly indicate the size of each block by writing its size, and if a memory allocation is present, write its letter as well. Assume the initial state is that the system is that no memory in this segment is allocated.

Complete the diagram below show the state of memory after each of the following operations. Remember to perform coalescence immediately when it is appropriate.

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



1D: Virtual Memory Page Size [6 marks]

Discuss the design tradeoffs related to the choice of virtual memory page size. Your discussion should consider three (3) different points.

Question 2: Scheduling [36 marks total]

2A: Interrupt Handling [3 marks]

Your co-worker has written an interrupt handler to update data and scheduled it to run every $4\ \mu s$ (microseconds). If you have a 100 MHz processor and the interrupt handler code takes 225 cycles to execute. What percentage of the CPU's cycles will be consumed handling this interrupt?

2B: Scheduling by Request [3 marks]

Remember the idea of priority inheritance: if a high-priority process is blocked, the priority of a lower priority process will be elevated. Suppose a process has the ability to elevate the priority of another process. Discuss a scenario in which a process would want to do that.

2C: Linux Real Time Scheduling [4 marks]

Recall that Linux has two real-time scheduling queues `SCHED_FIFO` and `SCHED_RR`. In this question, there are two identical processes, A and B. Both processes are equal in priority, and will run for $10t$ seconds, where t is the length of a timeslice in the system. Assume they arrive at the same time and will not get blocked during execution for any reason. Fill in the values below if they are scheduled according to:

1. `SCHED_FIFO`
   (a) Total time to complete:
   (b) Average time to complete:
2. `SCHED_RR`
   (a) Total time to complete:
   (b) Average time to complete:

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	2	2
P_2	1	1
P_3	8	4
P_4	4	2
P_5	5	3

This system, like UNIX, follows the rule that lower numbers indicate higher priority. If time slicing is necessary, one time slice equals 2 ms. Hint: turnaround time is the time of execution plus waiting time.

Complete the table on the next page 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 [20 marks total]

3A: Opening and Closing Files [4 marks]

Some systems automatically open a file when it is referenced for the first time, and automatically close the file when the process terminates. What are the advantages and disadvantages of this approach compared to the UNIX approach with the `open` and `close` system calls?

3B: File Access Patterns [4 marks]

Provide two examples of applications that are likely to access the content of a file in a sequential manner:

Provide two examples of applications that are likely to access the content of a file in a non-sequential manner:

3C: Hard Drive Terminology [12 marks]

Define the following terms related to hard drives:

(a) Seek Time:

(b) Rotational Latency:

(c) Track:

(d) Sector:

(e) Cylinder:

(f) Head Crash:

Question 4: Concurrency & Synchronization [30 marks total]

4A: Compare and Swap [5 marks]

In lectures we went over the “test-and-set” instruction that is implemented in an atomic hardware instruction. An alternative to this instruction is the “compare-and-swap” instruction, which will also be an atomic hardware operation. The following code describes, using C, the semantics of this instruction:

```
int compare_and_swap( int *var, int expected, int newval ) {
    int oldval = *var;
    if ( oldval == expected ) {
        *var = newval;
    }
    return oldval;
}
```

Apply this construct to the code below, to protect the critical section (shown as `/* Critical Section */`). For your convenience, a variable `lock` has been defined as a global. If your attempt to swap is unsuccessful, call `sched_yield()` to put the current thread to sleep temporarily before trying again. Otherwise, proceed to the critical section and remember to indicate the code is exiting from the critical section afterwards so that another call to `foo` will work.

```
int lock;
void foo(  ) {




    /* Critical Section */





}
```

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.

```
 1  int index = 0;
 2  int* results;
 3  int max_val;
 4  pthread_mutex_t results_lock;
 5
 6  int compare (const void * a, const void * b) {
 7     return ( *(int*)b - *(int*)a );
 8  }
 9
10  int is_prime( int num ) {
11
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 <= numBytes; 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              ++index;
36              pthread_mutex_unlock( &results_lock );
37              ++(*count);
38          }
39      }
40      free( void_arg );
41      free( start );
42      pthread_exit( count );
43  }
```

```
44  int main( int argc, char** argv ) {
45      max_val = atoi( argv[1] ); /* Read input as max val */
46      results = malloc( max_val * sizeof( int ) );
47
48      if (results == NULL) {
49          return -1;
50      }
51
52      pthread_mutex_init( &results_lock, NULL );
53
54      pthread_t threads[NUM_CPUS];
55      int global_sum = 0;
56
57      for ( int i = 0; i < NUM_CPUS; ++i ) {
58          int* start = malloc( sizeof( int ) );
59          *start = i;
60          pthread_create(&threads[i], NULL, find_primes, start);
61          pthread_detach(&threads[i]);
62      }
63
64      void* temp_sum;
65      for ( int i = 0; i < NUM_CPUS; ++i ) {
66          pthread_join( threads[i], temp_sum );
67          int *thread_sum = (int *) temp_sum;
68          free( temp_sum );
69          global_sum += *thread_sum;
70      }
71      printf("Found␣%d␣total␣primes␣less␣than␣%d.\n",
72                          max_val, global_sum );
73
74      qsort(results, global_sum, sizeof( int ), compare);
75
76      for (int i = 0; i < global_sum; ++i ) {
77          printf( "%d\n", results[i]);
78      }
79
80      pthread_exit(0);
81  }
```

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: