(VERSION WITH ANSWERS)
CORE EXAMINATION
Department of Computer Science
New York University
February 2, 2007

This is the common examination for the M.S. program in CS. It covers core computer science topics: Languages and Compilers, Operating Systems, and Algorithms. The exam has two parts. The first part lasts three hours and covers the first two topics. The second part, given this afternoon, lasts one and one-half hours, and covers algorithms.

Use the proper booklet or answer sheet for each question. Each booklet is marked with the Area and Question number, in the form PL&C1, PL&C2, PLC&C3, OS1, OS2, ALGS1 (this question has an answer sheet and not a booklet), ALGS2, ALGS3. Use the appropriate booklet for each question. DO NOT put your name on the exam booklet. Instead, your exam number must be on every booklet.

You will be graded according to your exam number, shown on the envelope containing the booklets. Remember your exam number: when grades are given out, they will be published according to this number, not by name.

Make sure your name and signature are on the envelope. This is the only place where your name appears. Please include all the booklets inside the envelope. You can keep the exam.

Good luck!

Programming Languages, Compilers and Operating Systems Questions

---

Question 1 – please use the Exam Booklet labeled PL&C1

Polymorphism in Java: Java's platform libraries support basic I/O through the `java.io` package. More specifically, input is modeled as reading from a stream, with character-based input streams all extending the abstract base class `Reader`. Output is modeled as writing to a stream, with character-based output streams all extending the abstract base class `Writer`. Concrete example classes include

- `FileReader` and `FileWriter` to access a file,
- `StringReader` and `StringWriter` to access a string,
- `BufferedReader` and `BufferedWriter` to access another `Reader` or `Writer` through a buffer.

Unless otherwise noted, keep your answers to 3 sentences or less. Longer answers will be ignored.

A) (2 Points) What is the primary benefit of using inheritance this way?

Answer: The common base classes `Reader` and `Writer` specify contracts that must be implemented by all concrete subclasses. As a result, any subclass of `Reader` can be used where a `Reader` is the declared class and any subclass of `Writer` can be used where a `Writer` is the declared class. □

B) (2 Points) Drawing on the classes listed above, give an example for this use of inheritance.

Answer: The constructors for `BufferedReader` take a `Reader` as their first argument and, consequently, can provide buffered access to `any` subclass of `Reader`. Similarly for `BufferedWriter`. □

C) (2 Points) Besides generics, Java provides another construct to express functionality shared between several concrete classes. In one word, what is that construct?

Answer: Interfaces. □

D) (2 Points) What are the primary two differences between abstract base classes and the second construct?

Answer: An abstract base class can implement methods, while an interface cannot. A concrete class can inherit only one abstract base class but many interfaces. □

E) (2 Points) Since `Reader` and `Writer` are abstract base classes instead of the second construct, what functionality cannot be expressed for input and output streams?

Answer: It is impossible to implement a class that is both an input and output stream, since a class can only inherit from one class. For example, `RandomAccessFile` provides methods to read and write a file but is neither an input nor an output stream. As a result, it cannot be used with other readers and writers. □

Question 2 – please use the Exam Booklet labeled PL&C2

Recall that a regular language is one that can be described by a regular expression. We can also use regular grammars to define regular languages. Recall that regular grammars have productions of the form $N \rightarrow t$ or $N \rightarrow tN'$ where N, N' are non-terminals and t is a terminal. A context-free grammar, which is more general than a regular grammar, can still generate a regular language. For example, the grammar

$$S \rightarrow a \mid b \mid SS$$

is not a regular grammar, yet it generates the regular language $(a + b)^+$. Consider the context-free grammars presented under items (A)—(E) below. For each of these grammars, please provide:

1. A verbal description of the language defined by the grammar, such as "all words of the form $a^i b^j$, such that $0 < i \leq j$".
2. An answer to the question "is the defined language regular?"
3. In case the language is regular, provide a regular expression, an automaton, or a regular grammar which defines the same language.

A) (2 Points)

$$S \rightarrow b \mid aS \mid Sc$$

Answer: This grammar defines the regular language a^*bc^*. □

B) (2 Points)

$$S \rightarrow b \mid aScc$$

Answer: This grammar defines the non-regular language $\{a^n bc^{2n} \mid n \geq 0\}$. □

C) (2 Points)

$$\begin{array}{lll} E & \rightarrow & OO \\ O & \rightarrow & a \mid Ea \mid EaE \end{array}$$

Answer: This grammar defines the regular language $(aa)^+$. □

D) (2 Points)

$$\begin{array}{lll} E & \rightarrow & T \mid E + T \\ T & \rightarrow & D \mid T * D \\ D & \rightarrow & 0 \mid 1 \end{array}$$

Answer: This grammar defines the regular language $(0 \mid 1)((+ \mid *)(0 \mid 1))^*$. □

E) (2 Points)

$$\begin{array}{lll} A & \rightarrow & Aa \mid Ba \\ B & \rightarrow & Bb \mid b \end{array}$$

Answer: This grammar defines the regular language b^+a^+. □

Question 3 – please use the Exam Booklet labeled PL&C3

A) (5 Points) Consider the following functions in Scheme:

```
(define sum
  (lambda (x y)
    (+ x y)))

(define prod
  (lambda (x y)
    (* x y)))
```

Write a function called *itlist* that takes two arguments, the first of which is either *sum* or *prod* and the second of which is a list of integers. *itlist sum L* should return the sum of the integers in L and *itlist prod L* should return the product of the integers in L. If L is empty, then the result should be 0. For example:

```
=> (itlist sum '(1 2 3 4))
;Value: 10
=> (itlist prod '(1 2 3 4))
;Value: 24
=> (itlist prod '())
;Value: 0
```

Answer:

```
(define itlist
  (lambda (f l)
    (cond
      ((null? l) 0)
      ((null? (cdr l)) (car l))
      (else (f (car l) (itlist f (cdr l))))
    )
  )
)
```

B) (5 Points) Suppose we want to achieve something similar using inheritance in Java. We want to create two classes called *Sum* and *Prod*, each with a method called *apply* that takes an array of integers as its argument and computes the sum or product respectively. For example:

```
int[] nums = { 1, 2, 3, 4 };
Sum s;
Prod p;
int val;
val = s.apply(nums); // Computes 10
val = p.apply(nums); // Computes 24
```

Design *Sum* and *Prod* in such a way that they both inherit from a single class *Base* containing the method *apply* and an abstract method *f* that takes two integers and returns an integer. *Sum* and *Prod* should not contain any data members and should implement only the method *f*. Write the complete Java code for all three classes.

Answer:

```
abstract class Base {

   public abstract int f(int x, int y);

   public int apply(int[] array) {

      if (array.length == 0) return 0; // If array is empty return 0
      int result = array[0];           // Store the first member in the result

      // Perform the function f on the rest of the elemtents
      for(int i = 1; i < array.length; i++)
         result = f(result, array[i]);

      return result;                   // Return the result
   }
}

class Sum extends Base {
   public int f(int x, int y) {
      return x + y;
   }
}

class Prod extends Base {
   public int f(int x, int y) {
      return x*y;
   }
}
```

Question 4 – please use the Exam Booklet labeled OS1

Memory Management: Recall that

```
1B  = 1 byte.                          | 1s  = 1 second.
1KB = 1 kilobyte = 1024 bytes.         | 1ms = 1 millisecond = 1/1000 seconds.
1MB = 1 megabyte = 1024 kilobytes.     | 1us = 1 microsecond = 1/1000 milliseconds.
1GB = 1 gigabyte = 1024 megabytes.     |
```

Your numerical answers do not need to be simplified. For example an answer of $10^8/(2^{15}+2^7)$ is fine.

Consider an operating system with both segmentation and demand paging. The OS uses 40-bit virtual addresses and is running on a byte-addressable computer with 16GB of real memory. The page size is 16KB and each page table entry (PTE) is 4B. A process can have up to 512 segments and each segment table entry (STE) is 8B.

Assume that all page tables and segment tables of all active processes must be memory resident. Since demand paging is supported, it is **not** necessary for the actual pages of the program to be memory resident.

Answer: PRELIMINARY REMARKS: Before answering any of the specific questions, I should note that with demand paging the size of a segment or process is limited by the available *virtual* memory and is not *directly* limited by the available physical memory. The *indirect* physical memory limitation is that the page and segment tables plus at least one page must fit in memory.

Since the page size is 16KB or 2^{14} bytes and the system is byte-addressable, the offset of a byte within a page requires 14 virtual address bits; thus 14 (typically low order) bits of the 40-bit virtual address give this offset.

Also note that support for $256 = 2^9$ segments requires that the segment number occupies 9 (typically high order) bits.

This leaves 40-(14+9)=17 bits for the third component of the virtual address, the page number within the segment, i.e., a segment can have up to 2^{17} pages.

My solution uses the simpler segmentation method, namely the one found in Multics; you could also use the more complicated method found in the Intel pentium. □

A) (6 Points)

What is the size, in bytes, of the largest possible segment?
What is the size, in bytes, of the largest possible page table?
What is the size, in bytes, of the largest possible segment table?
What is the **total** size, in bytes, of all segment and page tables used by a single largest-possible-process?
What is the maximum number of largest-possible-processes that can be active at one time?

Answer: From the above, each segment can have up to 2^{17} pages or $2^{17}2^{14} = 2^{31}$ bytes.

With segmentation and (demand) paging, each segment has a page table. Since the largest segment has 2^{17} pages, the largest page table has 2^{17} PTEs or $2^{17}4 = 2^{19}$ bytes.

Since a process can have up to $256 = 2^9$ segments, the largest segment table has 2^9 STEs or $2^98 = 2^{12}$ bytes.

The largest-possible-process has 2^9 segments each with 2^{17} pages. So we have 1 largest possible segment table and 2^9 largest possible page tables for a total size in bytes of

$$2^{12} + 2^92^{19} = 2^{12} + 2^{28}$$

The maximum number of these process that can fit is the floor of the quotient of the size of *physical* memory by the minimum total physical memory needed for one of these processes. The numerator is 16GB or 2^{34} bytes. The denominator is the size of the tables plus one page. So the answer is

$$\left\lfloor \frac{2^{34}}{2^{12} + 2^{28} + 2^{14}} \right\rfloor$$

□

B) (2 Points)

Assume that all I/O is performed by a disk that requires 10ms to start an I/O (this is the seek plus the rotational latency) and then transfers at a rate of 16MB/s. A large I/O of **contiguous** memory requires only one 10ms start up. Each page and segment table is contiguous, but two tables of the same process are not contiguous with one another. In order to suspend and later reactivate a process, all its page tables and segment tables must be written out and subsequently read back in.
How long would it take to suspend and reactivate a largest-possible-process?

Answer:

We have 256 PTs and 1 ST to write out *and* read back. Thus we need to start 2(256+1)=514 I/Os, which requires 514(10ms)=5.14s.

As computed above, the total table size, which we must write out *and* read back is $2^{12} + 2^{28}$ bytes. At a transfer rate of 16MB/s, this requires

$$2\left(\frac{2^{12} + 2^{28}bytes}{16MB/s}\right) = \frac{2^{13} + 2^{29}}{2^{24}}s = \left(32 + 2^{-11}\right)s$$

This is about 32.0005s. So the total time is about 5.14s+32.0005s = 37.1405s. □

C) (2 Points)

A new striped disk system is proposed that **increases** the start up time from 10ms to 15ms, but also increases the transfer rate by a factor of n, from 16MB/s to (16n)MB/s.
What is the smallest value of n for the new disk to be faster than the old when suspending a largest-possible-process?

Answer: At 15ms the start up time rises to (1.5)5.14s=7.71s. At (16n)MB/s the transfer time decreases to about $\frac{32.0005}{n}$s.

For this to be beneficial we need $7.71 + \frac{32.005}{n} < 37.1405$, which occurs when n is 2 (you could solve for the fractional value if you prefer). □

Question 5 – please use the Exam Booklet labeled OS2

Page Replacement Algorithms: A reference string S is a sequence of page numbers. Inputs to a page replacement algorithm are such reference strings, viewed as an abstraction of a sequence of memory references during a program execution. E.g., consider the reference string $S_0 = (701203042303)$, which we also write as

$$S_0 = (7012; 0304; 2303)$$

for ease of readng, using semicolons indicate substrings of length 4, with the last substring of length ≤ 4. Thus S_0 corresponds to a sequence of memory references, first to page 7, next to page 0, then page 1, etc.

A) (5 Points)

Suppose we have a page cache that can hold three frames. Initially, the frames hold no pages so the first reference is a page fault. How many page faults does the reference string S_0 incur, assuming each of the replacement algorithms below? You must show your calculations by displaying the contents of the frames after each page request.
(i) Assume the FIFO (first-in, first-out) algorithm.
(ii) Assume the LRU (least recently used) algorithm.
(iii) Assume the OPT (optimal page-replacement) algorithm.

Answer: (i) FIFO has 10 faults. We just list the pages in the 3 frames AFTER each page referene:

$$7, 70, 701, 201, -, 231, 230, 430, 420, 423, 023, -$$

The "$-$" indicates a HIT when the contents of the 3 frames are unchanged.

NOTE: there is a subtlety here. If a page is in the cache, and it is referenced (so we get a hit), do we regard the page as having a updated time of entering the cache? FIFO does not update (e.g., after the first hit in this example, the cache contains the pages $2, 0, 1$, but page 0 is still regarded as having been brought into cache before pages 2 and 1. Hence it was evicted in the next reference. If we use a updated time of entry, we would be doing LRU!

(ii) LRU has 9 faults.

$$7, 70, 701, 201, -, 203, -, 403, 402, 432, 032, -$$

(iii) OPT has 7 faults. Here are the pages stored in frame cache:

$$7, 70, 701, 201, -, 203, -, 243, -, -, 203, -$$

NOTE: there are four alternative answers for part(iii). For instance, the fourth snapshot (201) could have been replaced by 702 since we

could evict either page 1 or page 7 at this point. Similarly, the last two snapshots (203) could have been replaced by 043. the last frame cache of 203 may be replaced by 043.
□

B) (2 Points) What is Belady's Anomaly? Name an algorithm that exhibits Belady's Anomaly.

Answer: It is the phenomenon where increasing the number of size of the page cache can lead to more page faults. The FIFO algorithm for page replacement exhibits Belady's Anomaly.

Additional Remarks: Consider the page request sequence $S_1 = (0, 1, 2, 3; 0, 1, 4, 0; 1, 2, 3, 4)$ where the FIFO page replacement algorithm is used. If the FIFO queue has size 3, we get 9 page faults. If the FIFO queue has size 4, we get 10 page faults. Stack Algorithms (e.g., LRU) will not have this anomaly.
□

C) (3 Points) The OPT algorithm is un-implementable, and LRU is too expensive to implement in hardware. So in practice, we use some simple approximation of LRU algorithm by using hardware to keep track of two bits with each page: the M- and R-bits. First explain what these are. Then describe any page replacement algorithm that uses these bits.

Answer: The M-bit is the "modified bit". It is unset initially, but it set when a page in memory is modified. The R-bit is the "referenced bit", and is set whenever the page is referenced for reading or writing. When a page is first brought into memory, it's R-bit will be set. However, the R-bit can decay and is periodically unset by the hardware (typically at each clock interrupts).

One algorithm is to view (R,M) (not (M,R)!) as a binary number between 0 and 3. Thus all pages are put in one of four categories, and we evict a page randomly chosen from the smallest non-empty category. This is called the NRU (Not Recently Used) algorithm (Tanenbaum p.216). Another algorithm is the Second Chance Algorithm which is based on FIFO, but when the page at the front of queue has its R-bit set, we do not evict it. Instead, we give it a second chance by unsetting its R-bit, and putting it at the back of the queue. We then go to the next page in the queue to try to evict it, and so on. □