This is a closed book (and notes) examination. Answer all questions on the exam itself. Take the number of points assigned to each problem and the amount of space provided for your answer as a measure of the length and difficulty of the expected solution. The exam totals 100 points.
1. (12 points) Indicate whether each of the following situations describe a short process, a long process or if the situation does not clearly describe a short or long process. Briefly explain as necessary.

(a) the idle process in an operating system

(b) a Web browser

(c) an application reading from a database stored in a file and processing the records

(d) process type favored by a preemptive shortest process next scheduling policy

2. (12 points) A philosophy is an overall strategy for an operating system design in managing resources. Two philosophies are 1) make efficient use of resources and 2) make fair use of resources. Consider these philosophies in the context of process scheduling.

(a) In process scheduling what is the resource being managed?

(b) Which process scheduling policy yields the most efficient use of this resource?

(c) Which process scheduling policy yields the most fair use of this resource?

(d) Is it possible to have a policy that is both fair and efficient? If so, describe one. If not, describe why not.
3. (8 points) One of the requirements for a solution to the mutual exclusion problem is *bounded waiting*. What is this requirement and why is it important?

4. (18 points) The Java language provides monitors by using the keyword *synchronized* to indicate a *guard procedure*.

   (a) What is the key property of a *guard procedure* in a monitor?

   (b) Consider the following Java class discussed in class. Assume that the Java Virtual Machine uses semaphores to ensure the property of a guard procedure. Show where semaphore operations would be inserted by the Java compiler as well as how semaphores would be initialized in the class constructor function.

   ```java
   public class Account {
       private int balance;

       public Account() {
           balance = 0;       // initialize balance to zero
       }

       // use synchronized to prohibit concurrent access of balance
       public synchronized void Deposit(int deposit) {
           int newbalance;   // local variable

           newbalance = balance + deposit;
           balance = newbalance;
       }

       public synchronized int GetBalance() {
           return balance;       // return current balance
       }
   }
   ```
5. (6 points) A *port* is used for interprocess communication between a client and server process. Is a port an example of a *direct* or *indirect* message passing mechanism?

6. (18 points) As shown below, processes can be in one of three states: running, ready and blocked. There are six possible state transitions (labeled 1-6). For each label, indicate whether the transition is *valid* or *invalid*. If valid, indicate when the transition is used for a process. If the transition is not valid then indicate why.

![State transition diagram]

State transitions:
(a) 1: Blocked to Running
(b) 2: Running to Blocked
(c) 3: Ready to Blocked
(d) 4: Blocked to Ready
(e) 5: Ready to Running
(f) 6: Running to Ready
7. (12 points) Explain the behavior of the following two programs in terms of concepts discussed in class.

(a) Consider the following code.

```cpp
#include <iostream.h>
main()
{
    int a, b, c;

    a = 5;
    b = 0;
    cout << "hello world ";
    c = a/b;
    cout << "The value of c is " << c << '\n';
}
```

When the above code is executed the output is:
Floating exception (core dumped)

(b) Consider the following code (a modified version of the previous code).

```cpp
#include <iostream.h>
#include <signal.h>
main()
{
    int a, b, c;

    void DivideByZero(int);

    signal(SIGFPE, DivideByZero);
    a = 5;
    b = 0;
    cout << "hello world ";
    c = a/b;
    cout << "The value of c is " << c << '\n';
}

void DivideByZero(int signum)
{
    cout << "Oops!\n";
    exit(0);
}
```

When the above code is executed the output is:
hello world Oops!
8. (14 points) In Project 2 you implemented various instances of the producer/consumer problem using semaphores. Consider the following situation where two producer processes separately produce and place values in two separate buffers. Each buffer can contain one value. A single consumer process takes values from either buffer. As in a traditional producer/consumer problem, a producer blocks if its buffer is full. In this problem the consumer should only block if both buffers are empty. If both buffers are full then the consumer should alternately take from each buffer. Show how this instance of a producer/consumer problem can be implemented with semaphores by sketching the code for producer and consumer processes with enough detail that it is clear you understand the solution. Also show how any semaphores are initialized. Assume the buffers are initially empty.