## Problem 1: Concepts (Please be concise)

1) Consider the use of threads vs. the `select()` system call for asynchronous I/O. Name one advantage of each approach.

2) Consider a dynamically allocated linked-list where items are added and deleted over time and the list is unordered. Each node of the list has a unique virtual address and the process is allocated a fixed number of memory pages. By far, the most common operation is a linear scan through all elements of the list. If you were concerned about virtual memory performance for this operation, how would you organize this list?

3) In socket programming, it is good practice to put `hton*` calls around port numbers and IP addresses. Why?

4) Suppose the system offers two types of locks, a spinlock (which “blocks” a thread by spinning in a while-loop) and a mutexlock (which blocks a thread by putting it to sleep). When might you prefer to use a spinlock?

5) Some compilers do in-lining of functions before generating the assembly code. In-lining replaces a function call with the body of the function. In the example below, `foo` is in-lined within `bar`.

   ```c
   void foo (int a, int b) { return (a+b); }
   void bar () {int a,b,c; c = foo (a,b); …} becomes
   void bar () { int a,b,c; c=a+b; …}
   ```

   In-lining may be helpful with virtual memory. Why?

6) Sketch a solution to this problem using semaphores. Synchronize a group of threads that each wish to connect to a DB server by completing `DB_thread` below and giving the semaphore an initial value. At most K connections to a DB server are allowed at any one time. You should use the semaphore functions `down (sem_t *)` and `up (sem_t *)`. These functions are also sometimes called P/V or wait/post respectively.

   ```c
   sem_t s = ____;
   void DB_thread (...) { // this is the function all threads call
   DB.connect (...);
   ```

7) Suppose that when threads finish an activity they execute a simple counter increment statement `Counter++;` (where `Counter` is a global with an initial value of 0). Suppose this statement is executed concurrently by 1000 threads. What are the possible values of `Counter` at the end? Why
Problem 2: Virtual Memory

Consider the code fragment below. Suppose the resulting process is allocated $K$ memory frames, the page size/frame size is 2K, and arrays are stored by row in the virtual address space. Only consider the memory needed for the array.

```c
int A[2048][2048];

for (i=0; i<2048; i++)
    for (j=0; j<2048; j++)
        A[i][j] = 0; // Line #1
```

a) How many page faults will this process suffer at Line #1? Why?

b) Replace $A[i][j] = 0$ with $A[j][i] = 0$. How many page faults will this process suffer at Line #1? Why?
Problem 3: Synchronization

Note: this is for practice and is likely harder than I would put on exam.

Threaded-merger. Write a program that merges data from multiple input streams into a single output stream.

- The main program takes a set of filenames and starts a reader thread for each one
- The yellow buffer is like a ring buffer
- Readers exit when they see EOF on their input files
- Writer must distinguish between buffer empty and input done
Problem 4: Network Programming

One problem with Internet servers is that the client must know the machine or address name of the server. Suppose instead that a client wishes to discover a server based on what service it provides, e.g. news, weather, and so on. We call this the service name. Suppose a discovery service maintains <service_name, struct sockaddr> pairs (the latter is the socket address of service_name); and the discovery service is running on a server in the Internet. The discovery service has several operations that a client may use but we will focus on LOOKUP <service_name> which returns the associated address if the service exists.

Request message (string format):

```
LOOKUP
news
<blank_line>
```

Assume the reply to LOOKUP is formatted not as a string but as an encoded data type (reply_t):

```c
typedef struct {
    int exists; // does service exists: 1 yes, 0 no
    struct sockaddr the_addr; // the addr of the service
} reply_t;
```

You write a simple client that uses the discovery service to lookup and connect to a service named news.

1) What crucial piece of information did I omit that you need in order to write this simple client?

2) Assuming the presence of the information from (1) above, write this client (using socket code). No error checking code is needed.
System call signatures (not all may be needed):

Sockets:

- int socket(PF_INET, SOCK_STREAM, 0);
- int bind(int s, struct sockaddr *addr, int len);
- int accept(int s, struct sockaddr *addr, int *len);
- int connect(int s, struct sockaddr *name, int len);
- int listen(int s, int backlog);

- struct sockaddr {
  sa_family_t sin_family;  // AF_INET
  u_int16_t sin_port;      // port: network byte order
  struct in_addr sin_addr; // IP_addr: network byte order
}

- uint32_t htonl(uint32_t hostlong);
- uint16_t htons(uint16_t hostshort);
- uint32_t ntohl(uint32_t netlong);
- uint16_t ntohs(uint16_t netshort);

Synchronization:

- int sem_init(sem_t *S, 0, int init_val);
- int sem_wait(sem_t *S); // like P
- int sem_post(sem_t *S); // like V (or signal)

- pthread_t thread = PTHREAD_MUTEX_INITIALIZER;
- pthread_mutex_lock(pthread_mutex_t *mutex);
- pthread_mutex_unlock(pthread_mutex_t *mutex);

- pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
- int pthread_cond_signal(pthread_cond_t*cond);
- int pthread_broadcast(pthread_cond_t*cond);
- int pthread_cond_wait(pthread_cond_t*cond,
  pthread_mutex_t*mutex);

Misc:

- void fork();
- pid_t wait(int *status);
- int pipe(ends[2]);
- size_t sizeof(<type> or <var>);
- ssize_t read(int fd, void *buf, size_t count);
- ssize_t write(int fd, const void *buf, size_t count);