Final Exam

Quiz Instructions

Do all questions. Don't bother with includes for coding questions. Here are prototypes of various things used in this class.

```
JRB jrb_insert_str(JRB t, char *key, Jval val);
JRB jrb_insert_int(JRB t, int key, Jval val);
JRB jrb_insert_double(JRB t, double key, Jval val);

JRB jrb_find_str(JRB t, char *key);
JRB jrb_find_int(JRB t, int key);
JRB jrb_find_double(JRB t, double key);

JRB make_jrb();
JRB jrb_delete_node(JRB node);
void jrb_free_tree(JRB root);

Dllist new_dllist();
void free_dllist(Dllist);

void dll_append(Dllist, Jval);

int serve_socket(int port);
int accept_connection(int s);
int request_connection(char *hn, int port);

pid_t fork();
pid_t wait(int *stat_loc);
int execlp(const char *file, const char *arg0, ..., /*, (char *)0, */);
int dup(int fildes);
int dup2(int fildes, int fildes2);
int close(int fd);
int pipe(int fildes[2]);

typedef void (*sig_t) (int);
sig_t signal(int sig, sig_t func);

int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                   void *(*start_routine)(void *), void *arg);
int pthread_join(pthread_t thread, void **value_ptr);
int pthread_detach(pthread_t thread);
pthread_t pthread_self(void);
int pthread_mutex_init(pthread_mutex_t *mutex, NULL);
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);

int pthread_cond_init(pthread_cond_t *cond, NULL);
int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);
int pthread_cond_signal(pthread_cond_t *cond);
```

Question 1

8 pts

Behold the following procedure:

```
int fred(int x, int y)
{
  int j;

  j = x;
  y += j;
  return y;
}
```

Please give me the assembly code for this procedure.

Question 2

2 pts

Behold the following procedure:

```
int binky(int w, int x, int y, int z)
{
  return w+x * y+z;
}
```

What is the first instruction of the assembly code for this procedure?

Question 3

6 pts

Behold the following procedure:

```
int a(int b, int c)
{
  int d;

  d = 1;
  d = e(d+b+c);
  return d;
}
```

Suppose I call **a(5, 8)**. When I start the first instruction of a(), the stack and frame pointer both equal 0xfff60, and the program counter equals 0x4010. When I start the first instruction of e(), please answer the following questions:

1. What is the value at memory location 0xfff60?
2. What is the value at memory location 0xfff64?
3. What is the value at memory location 0xfff68?
4. What is the value at memory location 0xfff6c?
5. When we return from e() back to a(), what is the first instruction that is executed?
6. What is the next instruction?

Question 4

20 pts

You are working on a machine with 4-byte pointers. Below are 264 bytes of heap memory. All of these bytes are either allocated or free, and the free list is implemented as described in lecture. In your answers, you may answer in either hex or decimal. If your answer is hex, make sure it is preceded by "0x".

```
Freelist pointer: 0x809cd0

0x809c40     0x18   0x809c98     0x28   0x809cf0 0x809d04
0x809c44     0x30   0x809c9c 0x809ca4   0x809cf4 0x809d38
0x809c48 0x809c5c   0x809ca0 0x809cbc   0x809cf8     0x30
0x809c4c 0x809d08   0x809ca4 0x809c98   0x809cfc 0x809c58
0x809c50 0x809ca8   0x809ca8 0x809d08   0x809d00 0x809cd0
0x809c54 0x809c7c   0x809cac 0x809c98   0x809d04 0x809c48
0x809c58     0x18   0x809cb0 0x809c54   0x809d08 0x809d28
0x809c5c      0x0   0x809cb4 0x809ce8   0x809d0c 0x809d00
0x809c60 0x809cf8   0x809cb8 0x809c68   0x809d10 0x809c60
0x809c64 0x809c94   0x809cbc     0x30   0x809d14     0x28
0x809c68 0x809d04   0x809cc0 0x809c68   0x809d18 0x809cac
0x809c6c 0x809d24   0x809cc4 0x809cfc   0x809d1c 0x809c80
0x809c70     0x20   0x809cc8 0x809c7c   0x809d20 0x809c5c
0x809c74 0x809c64   0x809ccc     0x18   0x809d24 0x809cc0
0x809c78 0x809c80   0x809cd0     0x28   0x809d28     0x28
0x809c7c 0x809cac   0x809cd4 0x809cf8   0x809d2c     0x10
0x809c80 0x809ca0   0x809cd8      0x0   0x809d30 0x809c50
0x809c84 0x809c4c   0x809cdc 0x809cd0   0x809d34 0x809d1c
0x809c88 0x809d34   0x809ce0 0x809d18   0x809d38 0x809c80
0x809c8c     0x30   0x809ce4 0x809ce4   0x809d3c 0x809cf0
0x809c90     0x40   0x809ce8 0x809c74   0x809d40 0x809d24
0x809c94 0x809c8c   0x809cec 0x809c5c   0x809d44 0x809d28
```

Please answer the following questions:

1. How many bytes are in the first free chunk of memory?

2. What is the address of the second free chunk of memory?

3. How many bytes are in the second free chunk of memory?

4. What is the address of the third free chunk of memory?

5. How many bytes are in the third free chunk of memory?

6. There are four allocated chunks of memory. If we order them by their starting addresses, what is the address of the first one?

7. How many bytes are in the first one?

8. What is the address of the second one?

9. How many bytes are in the second one?

10. What is the address of the third one?

11. How many bytes are in the third one?

12. What is the address of the fourth one?

13. How many bytes are in the fourth one?

14. When the second chunk was allocated, the return value of the **malloc()** call was stored in an **(int *)** named a. What is ***(a+1)**?

Question 5

20 pts

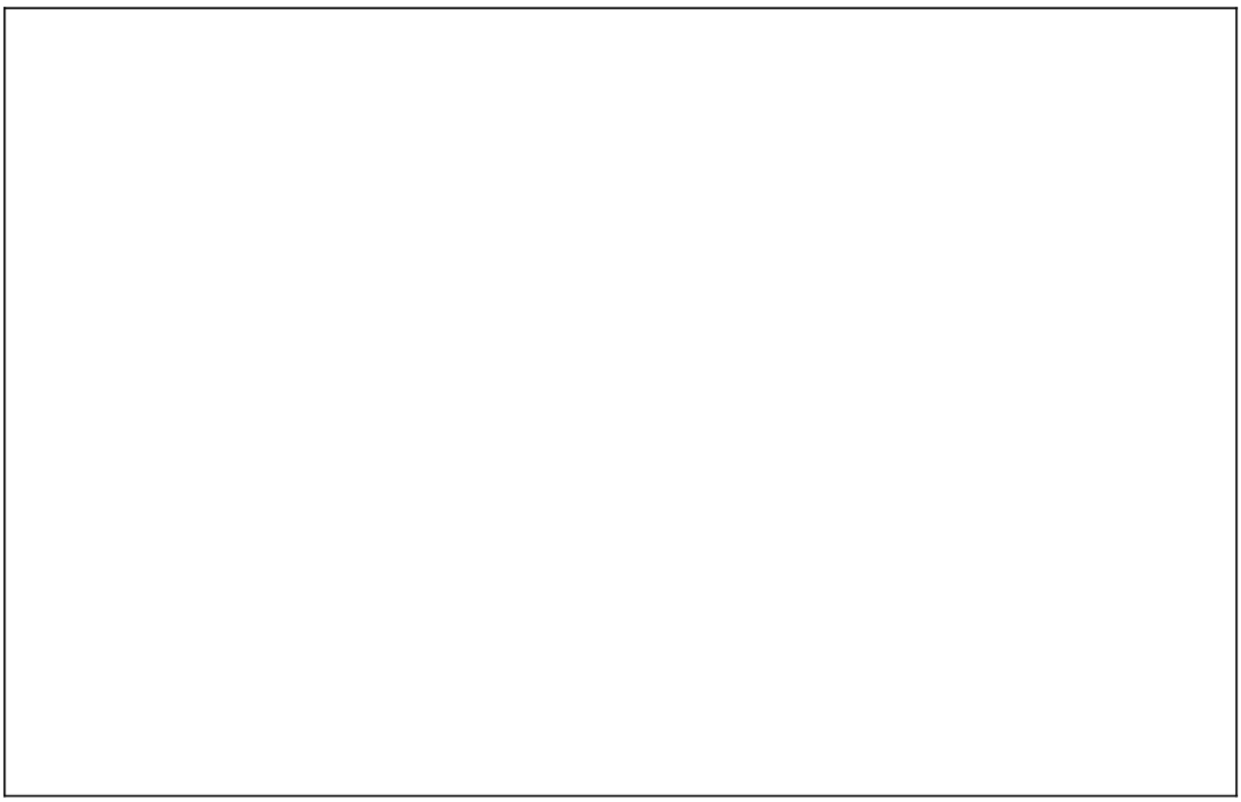
A co-worker has written code for an interactive server that interacts with multiple clients. This is a multithreaded program, and each client connection is serviced by its own thread. The clients can be programs, or they can be using a program like **telnet** or **nc**. There is a part of the code where a server thread reads two integers from a client connection, uses them to update a shared array, and then writes the array back to the client. The code is to the right.

This code has two potential problems related to performance.

Part A: Identify the two problems -- how they occur and how they compromise performance.

Part B: Tell me in sufficient detail how you'd fix each problem. "Sufficient detail", means enough to communicate that you know what the solutions are -- if you give me vague wording with buzz-words, then you won't get full credit

```
typedef struct {
  int data[4096];
  int counter;
  pthread_mutex_t *lock;
} Shared_Info;

typedef struct {
  FILE *fin;
  FILE *fout;
  Shared_Info *s;
} Private_Info;

void interact(Private_info *pi)
{
  int x;
  Shared_Info *si;

  si = pi->s;

  if (fscanf(pi->fin, "%d", &x) != 1) exit(1);

  pthread_mutex_lock(si->lock);
  si->data[si->counter%4096] = x;
  si->counter++;

  if (fscanf(pi->fin, "%d", &x) != 1) exit(1);
  si->data[si->counter%4096] = x;
  si->counter++;

  fwrite(si->data, sizeof(int), 4096, pi->fout);
  fflush(pi->fout);

  pthread_mutex_unlock(si->lock)
}
```



Question 6

16 pts

You are writing the code for a matchmaking service called "2022mingle.com." You don't wish to be politically incorrect, so rather than match people by gender, you're going to match them by keyword. You've written your IOS and Android apps, which clients will use, and you've filmed your commercial -- "2022 Mingle. Can you hear your heartstrings tingle?"

Time to write the server. The server is going to serve a socket and fork off a thread for every client. Each thread is going to do some initialization, and then call the procedure **find_match()** with the following prototype:

```
void find_match(Person *me, char *keyword);
```

A **Person** struct has all of the contact information for a person, plus:

- A pointer to a condition variable in the field **cond**. This condition variable is unique to the person (and it is initialized properly).
- A field called **match**, which is of type **Person ***. This is initialized to be NULL.

When **find_match()** is called, the variable **me** is pointing to the contact information for the client that is calling it. You need to find a match for this client. To help you, you get to use two global variables: A JRB named **tree**, and a pointer to a mutex called **lock**. You may assume that they have both been initialized correctly.

When **find_match()** is called, you are going to search the tree for a match. If you find one, you'll need to set the **match** field for both clients to each other, delete the one client from the tree, and have both clients return. If you don't find a match, you should insert the client into the tree and have him/her wait for a match.

Part A: Go ahead and write **find_match()**.

Part B: Is it possible in your code for the following to occur?

- Clients A and B match on a keyword.
- Clients C and D subsequently match on the same keyword.
- Clients D returns from **find_match()** before client A does.

If so, tell me how that happens, in terms of mutexes and condition variables and the order in which operations occur. If not, convince me that it cannot

Question 7

20 pts

You procrastinated when assembling a team for your CS402 design project, and got stuck with Kim, Khloe and Kourtney designing a data processing application. You were surprised with how proactive they were, because they finished their components within a couple of weeks. But then, you realized that they were leaving the country to record their new reality show, "The Kardashians Yodel in Switzerland," and they stuck you with the task of putting their pieces together. Here's what happened:

- Kim wrote a program called **kim**, which reads from standard input and writes on standard output. It does the initial processing of the data.
- Khloe wrote a program called **khloe** that processes the output of Kim's program. It reads that on standard input, and writes on standard output. It works well for a while, but Khloe got bored and had her program exit before it was finished reading the input.
- Kourtney wrote a program called **kourtney** that starts reading from where Khloe's program stops, and produces the rest of the output. Thank you, Kourtney.

Since they are paranoid about intellectual property theft, they've taken the source code to Switzerland with them -- all you have are the executables.

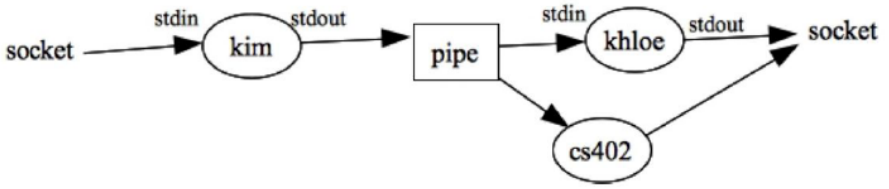
Now, what your group's program is supposed to do is serve a connection on port 90210, connect to one client, then read input from the client, process it, and write output back to the client.

Fortunately, you took CS360, so you can make this work. Here's what you are going to do. You're going to write a program **cs402** which does the following. It's going to serve a socket and wait for a connection. Then it's going to launch **kim** and make **kim**'s standard input come from the socket.

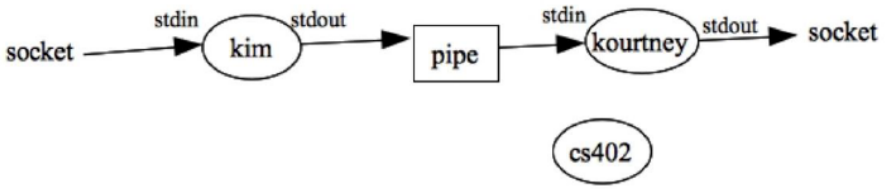
It's also going to set up a pipe so that **kim**'s standard output can go to a **khloe** process' standard input. **khloe**'s standard output will go to the socket.

Now, you're not going to close your file descriptor to the read end of the pipe or the socket connection, because when **khloe** exits, you're going to launch a **kourtney** process and have it take over for **khloe** -- it will read from the pipe and write to the socket.

Write this program. I feel like I'm being too nice here, but I'd like for you to get this right. Here's how you should set it up after you get the connection:



and here's how it should look after Khloe exits:



Your program should exit when everyone is done.