Name (Last, First): _____

This exam consists of 5 questions on 8 pages. Be sure you have the entire exam before starting. The point value of each question is indicated at its beginning; the entire exam is worth 100 points. Individual parts of a multi-part question are generally assigned approximately the same point value; exceptions are noted. For this exam you are allowed to bring a single page of notes (front and back). You may NOT share material with another student during the exam. Use of electronic devices is not allowed.

Be concise and clearly indicate your answer. Presentation and simplicity of your answers may affect your grade. Answer each question in the space following the question. If you find it necessary to continue an answer elsewhere, clearly indicate the location of its continuation and label its continuation with the question number and subpart if appropriate.

You should read through all the questions first, then pace yourself.

The questions begin on the next page.

Problem	Possible	Score
1	20	
2	20	
3	20	
4	20	
5	20	
Total	100	

1. (_____/20 points)
   Short answer questions

   (a) In C on a 32 bit machine how many bytes of memory are needed to store the string `"go usf\n"`?

   (b) Consider the following C code snippet:

   ```
   int *p;

   p = (int *) malloc(4 * sizeof(int));
   ```

   How many bytes of memory are allocated on the heap?

   (c) Explain why `fork()` must be implemented as a system call (not a user-level library function).

   (d) What three hardware mechanisms does a CPU (Processor) need to provide to allow an operating system to provide protection and isolation?

2. (_____/20 points)

UNIX tail

In most versions of UNIX, including Linux, there a is a command called `tail` that takes a filename as an argument and displays the last 10 lines of the specified file. If a file contains less than 10 lines, then the entire file is displayed. Your job is to implement the `tail` command in C. In order to help you out, here is given code for the `countlines` command that counts the number of lines in a file (note that `countlines` is the same as `wc -l`).

You only need to provide a single function called `tail_file(char *filename)`, that opens `filename` and displays the last 10 lines of the file if the file exists. You do not need to provide `main()`. Provide your implementation of the `tail`file()+ function on the next page. **Hint: Provide an English description of how your function works.** Hint: your solution does not have to be efficient.

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

int count_lines(char *filename)
{
    int count = 0, len = 0, fd, n;
    char c;

    fd = open(filename, O_RDONLY);

    if (fd < 0) {
        printf("file: %s does not exist\n", filename);
        exit(-1);
    }

    while((n = read(fd, &c, 1) > 0)) {
        if (c == '\n') {
            count += 1;
            len = 0;
        } else {
            len += 1;
        }
    }

    /* Count an incomplete line */
    if (len > 0) {
        count += 1;
    }

    close(fd);

    return count;
}

int main(int argc, char *argv[])
{
    int fd, count;
    char *filename;

    if (argc != 2) {
        printf("usage: countlines <filename>\n");
        exit(-1);
    }

    filename = argv[1];
    count = count_lines(filename);
    printf("count = %d\n", count);

    return 0;
}
```

Provide your solution to problem 2 here.

3. (_____/20 points)

Buffered I/O

Recall the buffered version of `get_line()` that we developed in class.

```
#define BUF_SIZE 32
char buffer[BUF_SIZE];
int buflen = BUF_SIZE;
int bufpos = BUF_SIZE;

int get_line_buffered(int fd, char *line)
{
    int count = 0;

    while (1) {
        if (buflen != BUF_SIZE && buflen == bufpos) { /* BBB */
            break;
        }

        /* Need to read a chunk? */
        if (bufpos == BUF_SIZE) {
            buflen = read(fd, buffer, BUF_SIZE);

            if (buflen < 0) {
                return buflen;
            }
            if (buflen == 0) { /* AAA */
                break;
            }
            bufpos = 0;
        }

        line[count] = buffer[bufpos];
        count += 1;
        bufpos += 1;

        if (buffer[bufpos-1] == '\n') {
            break;
        }

        if (count == MAX_LINE_LEN) {
            count = -1;
            break;
        }
    }

    line[count] = '\0';

    return count;
}
```

Answer the following questions.

(5 points) Will this code allow for lines to be read that are longer than `BUF_SIZE`?

(5 points) Assume you have a file with 100 bytes and 7 lines. If you read all of the lines of this file with this code (that is, you call `get_line_buffered()` repeatedly until no more characters are available), how many times is the `read()` system call invoked?

(5 points) Explain what the if statement is doing at `/* AAA */`.

(5 points) Explain what the if statement is doing at `/* BBB */`.

4. (_____/20 points)

Fork

Consider the following C/UNIX program.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    int id, c = 0;

    while(1) {
        id = fork();
        if (id == 0) {
            if (c == 5) {
                printf("c = %d\n", c);
                break;
            } else {
                c = c + 1;
            }
        } else if (id > 0) {
            wait(NULL);
            break;
        }
    }
    return 0;
}
```

What does this program do? That is, what does the value of `c` represent when it is displayed with `printf()`? Provide a picture of the different process(es) that are created and show the relationship between the processes at the point when `printf()` is called.

5. (_____/20 points)

Pipes

Consider the following C/UNIX program.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    int p1[2];
    int id;
    int r;
    char c;

    pipe(p1);

    write(p1[1], "abc", 3);

    id = fork();

    if (id == 0) {
        r = read(p1[0], &c, 1);
        printf("c1: %c\n", c);
        exit(0);
    }

    id = fork();

    if (id == 0) {
        r = read(p1[0], &c, 1);
        printf("c2: %c\n", c);
        exit(0);
    }

    r = read(p1[0], &c, 1);
    printf("p: %c\n", c);

    return 0;
}
```

What are the possible outputs of this program?

Continue your answers here if necessary.