Name _____ Cal Poly Email _____

1. **[1 point]** Fill in the blanks: In a UNIX system, Everything starts in the directory called _root___, whose name is the single character: /.

2. **[2 points]** Given the C function len() and statements below, fill in the blanks that follow. You may assume that sizeof(char) is 1 byte, sizeof(int) is 4 bytes, and sizeof() any pointer is 4 bytes.

```
int len(char *s) {
    int l = 0;
    while (*s++ != '\0') {
        l++;
    }
    return l;
}
char str[] = "CSC/CPE 400";
char *course_num = &str[9];
```

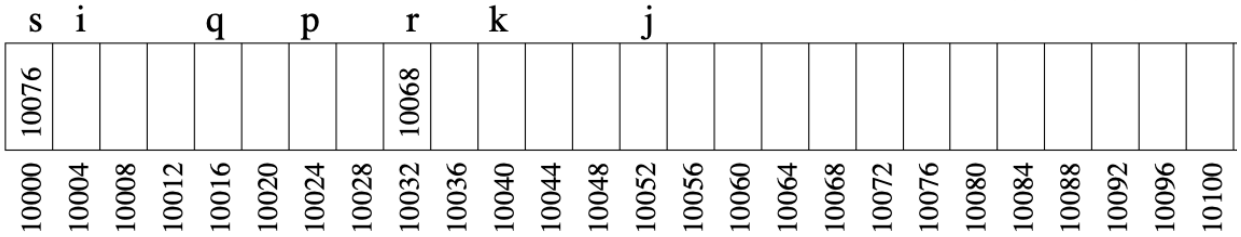
len(str) = 11 sizeof(*str) = 1

len(course_num) = 2 sizeof(course_num) = 4

3. **[3 points]** Write the C function strcat that takes two strings s1 and s2, copies s2 onto the end of s1 and returns the new string s1. You may assume that s1 points to the beginning of a char[] of sufficient size to hold the combined string; you do not need to allocate memory.

```
char *strcat(char *s1, char *s2) {
  int i = 0, j = 0;
  // Find the end of s1
  while (s1[i] != '\0') {
    i++;
  }
  // Copy characters from s2 to the end of s1
  while (s2[j] != '\0') {
    s1[i] = s2[j];
    i++;
    j++;
  }
  // Add the null terminator at the end
  s1[i] = '\0';

  return s1;
}
```

4. **[3 points]** The diagram below shows the contents of memory at some point in the execution of some program. The names at the top of the diagram are variable names. The numbers at the bottom of the diagram are memory addresses (in decimal). You may assume that integers and integer pointers each have a size of four (4) bytes.



At each point below, provide the contents of each specified variable.

```
int i, j, k;
int *p, *q, *r, *s;
i = 7;
j = 27;
k = 19;
p = &j;
q = &k;
```

p = 10052 q = 10040

```
*q = *p;
```

i = 7 j = 27 k = 27 p = 10052 q = 10040

```
*p = 42;
*q = 24;
```

i = 7 j = 42 k = 24 p = 10052 q = 10040

5. **[1 point]** What is the difference between the UNIX commands `"man printf"` and `"man 3 printf"`?

The command man printf typically opens the manual page for the printf command in section 1 (user commands), which is the shell's built-in printf function. The command man 3 printf specifically accesses the manual page for the printf function in section 3 (library functions), describing the printf function used in C programming.

CSC/CPE 357 - Fall 2024	10 points	Quiz 1 v2

Name: _____ Cal Poly Email: _____

1. **[1 point]** Which directory is not part of a standard UNIX file system?
   A. bin B. sys C. mnt D. proc

2. **[1 point]** What is not part of Memory Layout of a C Program?
   A. heap B. queue C. stack D. text

3. **[1 point]** Which format string is used for variable of double data type?
   A. %c B. %d C. %f D. %i

4. **[3 points]** Show the output of the following code in the box on the right.

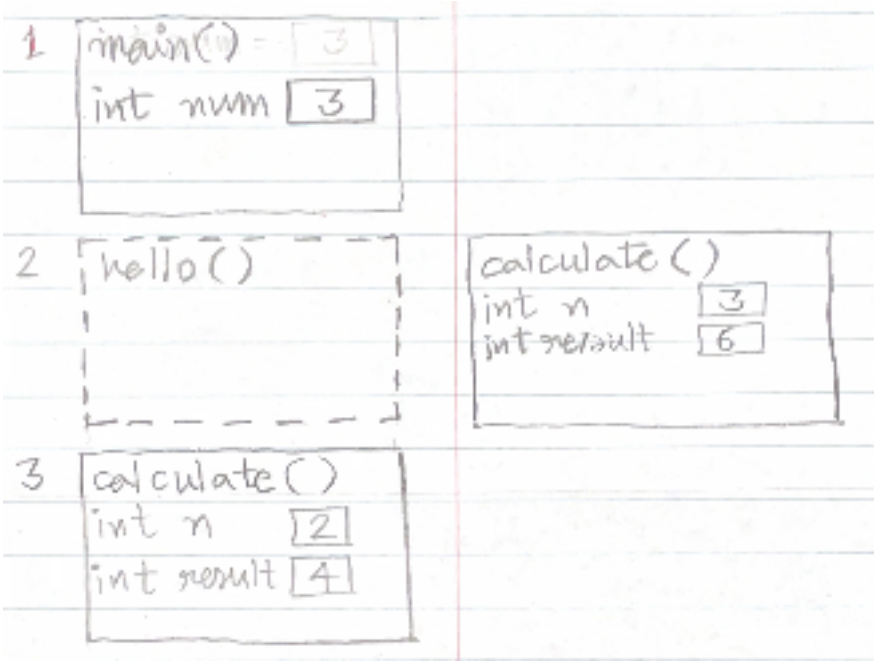
```
int main() {
    char str1[10] = "256";
    char str2[15] = "World";
    char str3[15] = "world";
    char str4[20] = "256.00";
    char result[20];
    int num1, num2;

    strcpy(result, str2);

    if (strlen(str2) == strlen(str3)) {
        strcat(result, str3);
    }

    int sum = atoi(str1) + strlen(str4);

    int comparison = strcmp(str2, str3);

    if (comparison != 0) {
        int len = strlen(result);
        result[len] = str3[0];
        result[len + 1] = '\0';
    }

    int length = strlen(result);

    printf("result: %s\n", result);
    printf("sum: %d\n", sum);
    printf("comparison: %d\n", comparison);
    printf("length of result: %d\n", length);

    return 0;
}
```

Output:

result: Worldworldw
sum: 262
comparison: -1
length of result: 11

5. [**3 points**] Given the following C program, draw the stack diagram at the moment when the **calculate()** function has been called for the second time (during recursion).

```
void calculate(int n) {
    int result = n * 2;
    printf("Result: %d\n", result);
    if (n > 1) {
        calculate(n - 1);
    }
}

void hello(){}

int main() {
    int num = 3;
    hello();
    calculate(num);
    return 0;
}
```



6. [**1 point**] What are the main differences between a Function Declaration and a Function Definition in C?

Function Declaration: Specifies the function's name, return type, and parameters without its implementation.

Function Definition: Includes the full implementation of the function.

CSC/CPE 357 - Fall 2024	10 points	Quiz 2 v1

Name: _____ Cal Poly Email: _____

1. **[1 point]** The getline() function reads a line from stream, delimited by the character
   A. \r B. \t C. \n D. \0

2. **[1 point]** Given: int a[10] an int *pa, what is equivalent to pa = a?
   A. pa = *a[0] B. pa = &a[0] C. &pa = a[0] D. *pa = &a[0]

3. **[1 point]** Which operation is not valid on structures?
   A. Incrementing Structure Variables B. Find address of structure with &
   C. Copy or assign as a unit D. Access individual members

4. **[3 points]** Consider the following C code for file operations:

```
int main() {
    FILE *file;
    char ch;

    file = fopen("example.txt", "w");    // Open a file in write mode
    if (file == NULL) {
        printf("Error opening file!\n");
        return 1;
    }

    fprintf(file, "Hello World!");    // Write data to the file
    fclose(file);

    file = fopen("example.txt", "r");    // Open the file in read mode
    if (file == NULL) {
        printf("Error opening file!\n");
        return 1;
    }

    while ((ch = fgetc(file)) != EOF) {     // Print file content to the screen
        putchar(ch);
    }
    fclose(file);
    return 0;
}
```

Assuming example.txt is in the same directory, what will be the output for each of the following cases?

Case 1: The code is run for the first time on a new example.txt file.

Output: Hello World!

Case 2: The file example.txt already contains the text "Goodbye" before running the code.

Output: Hello World!

Case 3: The code is modified to open the file in append mode ("a") instead of write mode ("w").

Output: GoodbyeHello World!

5. [**3 points**] Consider the following C code snippet involving pointers:

```
#include <stdio.h>

int main() {
    int x = 10, y = 20;
    int *p1, *p2;

    p1 = &x;
    p2 = &y;

    *p1 = *p1 + *p2;
    p2 = p1;
    *p2 = *p2 - 5;

    printf("x = %d, y = %d, *p1 = %d, *p2 = %d\n", x, y, *p1, *p2);

    return 0;
}
```

What will be the output for each of the following cases?

Case 1: The initial values are int x = 10, y = 20; (as given).

Output: x = 25, y = 20, *p1 = 25, *p2 = 25

Case 2: The initial values are changed to int x = 5, y = 15;.

Output: x = 15, y = 15, *p1 = 15, *p2 = 15

Case 3: The initial values are changed to int x = -10, y = -20;.

Output: x =-35, y= -20, *p1 = -35, *p2= -35

6. [**1 point**] What are the main differences between structures and unions in C?

In C, structures and unions both allow grouping of different data types, but they differ in memory allocation. In a structure, each member has its own memory space, so the total size is the sum of all members. In a union, all members share the same memory space, which means only one member can hold a value at a time, with the size determined by the largest member.

CSC/CPE 357 - Fall 2024	10 points	Quiz 2 v2

Name: _____ Cal Poly Email: _____

1. **[1 point]** For this array `int tda[3][4] = { {71,72,73,74}, {22,23,24,25}, {54,55,56,57} }`, what is the value of `tda[2][1]`?

   A. 73 B. 55 C. 24 D. Undefined

2. **[1 point]** Which function is not used for file operation?

   A. fscanf() B. getc() C. getchar() D. fprintf()

3. **[1 point]** Which of the following cannot be a valid member of a structure?

   A. A string B. A function C. A pointer D. An array

4. **[3 points]** Consider the following C code snippet:

```
int main() {
    FILE *file;
    char line[50];
    int count = 0;

    file = fopen("data.txt", "r"); // Open file in read mode to check for existing content
    if (file != NULL) {
        while (fgets(line, sizeof(line), file) != NULL) {
            count++;  // Count the number of lines already in the file
        }
        fclose(file);
    }
    if (count > 2) {    // Open file in write mode if it has more than 2 lines
        file = fopen("data.txt", "w");
    } else {    // otherwise in append mode
        file = fopen("data.txt", "a");
    }
    fprintf(file, "This is line %d\n", count + 1);    // Write a new line to the file
    fclose(file);

    file = fopen("data.txt", "r");    // Open the file in read mode to print its contents
    while (fgets(line, sizeof(line), file) != NULL) {    // Print file content to the screen
        printf("%s", line);
    }
    fclose(file);
    return 0;
}
```

   Assuming data.txt is in the same directory, what will be the output for each of the following cases?

   Case 1: The file data.txt is empty before running the code.

   Output: This is line 1

Case 2: The file data.txt initially contains one line:

```
Line 1
```

Output: Line 1
This is line 2

Case 3: The file data.txt initially contains three lines:

```
Line 1
Line 2
Line 3
```

Output: This is line 4

5. [**3 points**] Consider the following C code snippet involving pointers:

```
int main() {
    int x = 4, y = 7;
    int *p1, *p2;

    p1 = &x;
    p2 = &y;

    *p1 = *p1 * 2;  // Double the value pointed to by p1
    *p2 = *p2 + *p1;  // Add the value of *p1 to *p2
    p1 = &y;  // Make p1 point to y
    *p1 = *p1 - 3;  // Subtract 3 from the value pointed to by p1

    printf("x = %d, y = %d, *p1 = %d, *p2 = %d\n", x, y, *p1, *p2);

    return 0;
}
```

What will be the output for each of the following cases?

Case 1: The initial values are int x = 4, y = 7; (as given in the code).

Output: x = 8, y = 12, *p1 = 12, *p2 = 12

Case 2: The initial values are changed to int x = 10, y = 5;.

Output: x = 20, y = 22, *p1 = 22, *p2 =22

Case 3: The initial values are changed to int x = -2, y = 3;.

Output: x = -4, y = -4, *p1 = -4, *p2 = -4

6. [**1 point**] What are the basic rules for writing a makefile in C programming?

A Makefile consists of a sequence of rules to specifie:
- a target
- enumeration of files or other targets on which the target depends
- commands which define how to transform (ie. compile) the components into the target

Name Answer Key **Date** _____

1. **[3 points]** Given the variables below, determine whether the statements that follow are valid. Invalid statements are those that would lead (either immediately or later in the execution of the program) to unpredictable behavior, an error, or a segmentation fault.

```
int a[2] = { 3, 57 };
int *b = (int *) malloc(2 * sizeof(int));
int *c;
```

On each line, circle "Valid" or "Invalid" Briefly explain each "Invalid" response. Assume that these statements are run in the order below:

```
c = a;              (Valid) / Invalid
b[0] += 2;          (Valid) / Invalid
c = b + 3;          Valid / (Invalid)  b was allocated for 2 ints, b+3 points beyond
free(&a[0]);        Valid / (Invalid)  free should only used with malloc-allocated memory
b = realloc(b, 0);  (Valid) / Invalid  *May free memory
free(b);            (Valid) / Invalid
```

2. **[3 points]** Write a C program that takes command-line arguments representing two programs and their arguments. These two programs and their arguments will be separated by the `@` character. For example, if the program is called `one_of`, the following indicates that the program `ls` is to be run with the `-l` argument and the program `other` is to be run with `a`, `b`, and `c` as arguments.

```
% one_of ls -l @ other a b c
```

Your program must `exec` the first program with its provided arguments (up to, but not including the `@`). If that `exec` fails, then your program must `exec` the second program with its provided arguments. Report an error if both execs fail. Note that the use of `fork` is not required.

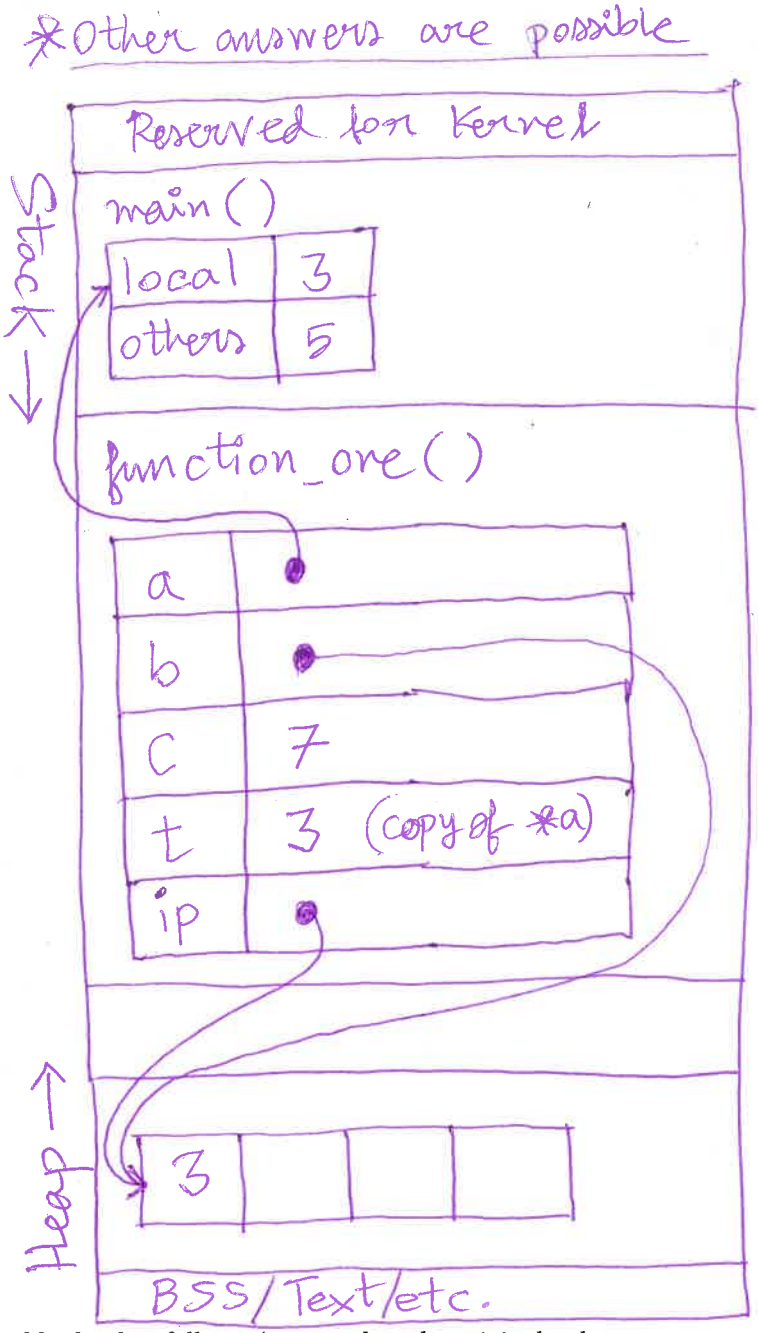
* Other answers are possible

```
// #includes omitted for space reasons, function prototypes for reference:
int execv(const char *path, char *const argv[]);
int execl(const char *path, const char *arg, ... /* (char  *) NULL */);
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
    if (argc < 2) {
        fprintf(stderr, "Usage: one_of program1 [args...] @ program2 [args...]\n");
        return 1;
    }

    int at_index = -1;
    for (int i = 0; i < argc; i++) {
        if (strcmp(argv[i], "@") == 0) {
            at_index = i;
            break;
        }
    }

    if (at_index == -1) {
        fprintf(stderr, "Error: '@' not found in arguments\n");
        return 1;
    }

    argv[at_index] = NULL; // Split the arguments

    if (execvp(argv[1], &argv[1]) == -1) {
        if (execvp(argv[at_index + 1], &argv[at_index + 1]) == -1) {
            perror("Both execs failed");
            return 1;
        }
    }

    return 0;
}
```

3. [**3 points**] Based on the C code below, draw a diagram of the stack and heap just before the `pause()` function is called. Your diagram must include the value for each variable. Pointers should be represented using an arrow to the appropriate location.

```
1  #include <stdio.h>
2  void start() {}
3  void pause() {}
4
5  void function_one(int *a, int *b, int c) {
6      int t = *a;
7      int *ip = (int *) malloc(sizeof(int) * 4);
8      *a = *b;
9      b = ip;
10     *b = t;
11     // draw stack and heap at this point
12     // (before pause() is called)
13     pause();
14 }
15 int main(void) {
16     int local = 3;
17     int other = 5;
18     start();
19     function_one(&local, &other, 7);
20     return 0;
21 }
```



4. [**2 points**] Given the C statements below, fill in the blanks that follow. Assume that the original values assigned to the variables are not changed by any of the expressions below.

```
#include <stdint.h>
uint8_t a = 3;
uint8_t b = 0x3;
uint8_t z = 0;
```

`a & z` = 0 `a | z` = 3 `a || z` = 1 `a && b` = 1

`a ^ b` = 0 `a & b` = 3 `b | z` = 3 `b >> 1` = 1

CSC/CPE 357 - Fall 2024	10 (+1 bonus) points	Quiz 3 v2

Name: _____ Cal Poly Email: _____

1. **[3 point]** Evaluate the validity of the following statements given the provided variables. Indicate whether each statement is valid or invalid, and for any invalid response, briefly explain why it is invalid. Assume the statements are executed sequentially:

```
int x[3] = { 10, 20, 30 };
int *y = (int *) malloc(3 * sizeof(int));
int *z;
```

On each line, circle "Valid" or "Invalid" Briefly explain each "Invalid" response. Assume that these statements are run in the order below:

z = x	(Valid / Invalid)	_____
y[1] *= 5	(Valid / Invalid)	_____
z = y - 2	(Valid / Invalid)	_____
free(&x[1])	(Valid / Invalid)	The free function is meant for heap memory. &x[1] points to memory on the stack
y = realloc(y, sizeof(int))	(Valid / Invalid)	_____
free(y)	(Valid / Invalid)	_____

2. **[3 point]** Write a C program that accepts command-line arguments describing two commands and their respective arguments, separated by the `#` character. If the program's name is `runner`, it should behave as follows:

```
runner cat file.txt # wc -l file.txt
```

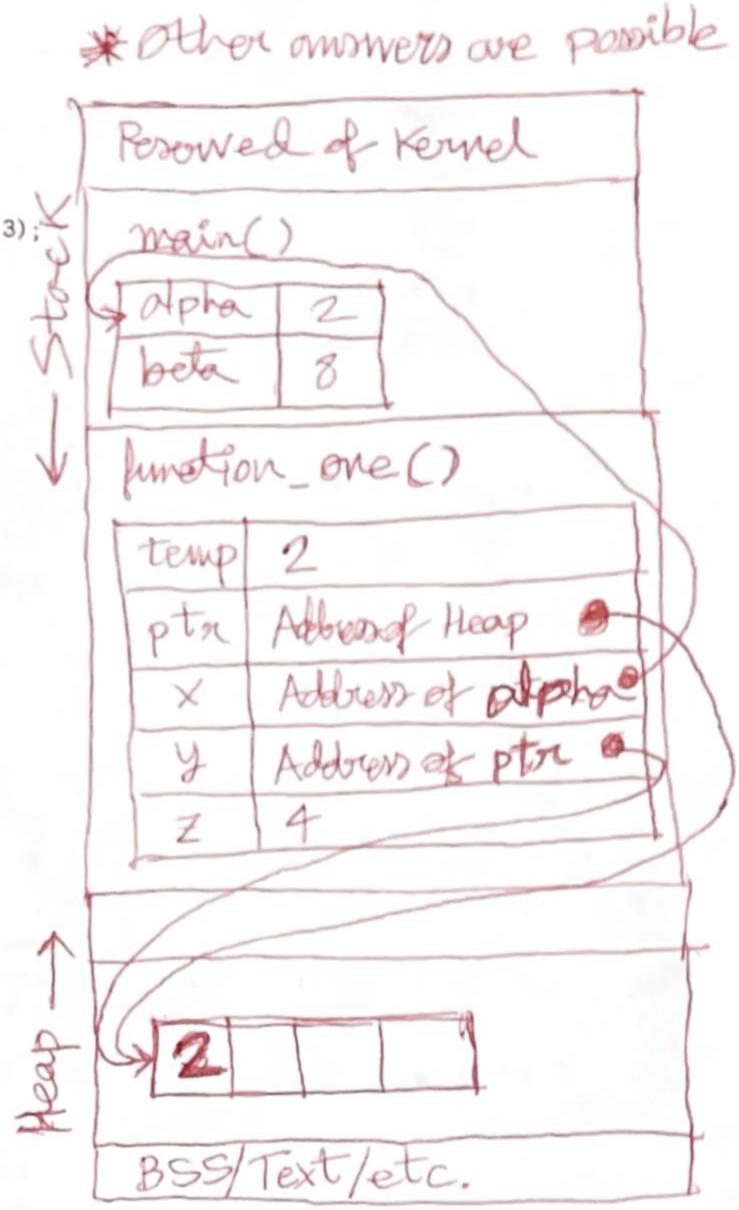
Your program must execute the first command with its arguments (everything before `#`). If the execution fails, run the second command with its arguments. Report an error if both executions fail. Note that the use of fork is not required.

```
// #includes omitted for space reasons, function prototypes for reference:
int execvp(const char *file, char *const argv[]);
int execlp(const char *file, const char *arg, ... /* (char *) NULL */);
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
    // Split the input into two commands using #
    char *commands = argv[1];
    char *cmd1 = strtok(commands, "#");
    char *cmd2 = strtok(NULL, "#");
    // Parse the first command and arguments
    char *args1[128]; // Assume a maximum of 128 arguments
    int i = 0;
    char *token = strtok(cmd1, " ");
    while (token) {
        args1[i++] = token;
        token = strtok(NULL, " ");
    }
    args1[i] = NULL;
    // Parse the second command and arguments
    char *args2[128];
    i = 0;
    token = strtok(cmd2, " ");
    while (token) {
        args2[i++] = token;
        token = strtok(NULL, " ");
    }
    args2[i] = NULL;
    // Execute the first command
    if (execvp(args1[0], args1) == -1) {
        perror("Error executing first command");
    } else {
        return 0; // First command executed successfully
    }
    // If first command failed, execute the second command
    if (execvp(args2[0], args2) == -1) {
        perror("Error executing second command");
        fprintf(stderr, "Both commands failed.\n");
        return 1; // Both commands failed
    }

    return 0; // Second command executed successfully
}
```

3. **[3 point]** Given the following C code, create a diagram of the stack and heap just before the `halt()` function is called. Include all variable values and show pointer references.

```
#include <stdio.h>
void begin() {}
void halt() {}

void perform(int *x, int *y, int z) {
    int temp = *x;
    int *ptr = (int *) malloc(sizeof(int) * 3);
    *x = *y;
    y = ptr;
    *y = temp;
    // Draw stack and heap at this point
    halt();
}
int main() {
    int alpha = 2;
    int beta = 8;
    begin();
    perform(&alpha, &beta, 4);
    return 0;
}
```



4. **[2 points]** Fill in the blanks based on the given C statements. Assume initial values of variables remain unchanged throughout the computations.

```
#include <stdint.h>
uint8_t p = 5;
uint8_t q = 0x5;
uint8_t r = 0;
```

A. $p \,\&\, r =$ 0 B. $p \mid r =$ 5 C. $p \,||\, r =$ 1 D. $p \,\&\&\, q =$ 1

E. $p \wedge q =$ 0 F. $p \,\&\, q =$ 5 G. $q \mid r =$ 5 H. $q >> 1 =$ 2

CSC/CPE 357 - Fall 2024	10 points	Quiz 4 v1

Name: Answer Key Cal Poly Email: _____

1. [**1 point**] Which of the following mechanisms is not intended for IPC on Unix?

   A. FIFOs B. Pipes C. Shared queues D. Semaphores

2. [**1 point**] Which statement best describes full-duplex communication?

   A. Data flows in one direction only at a time.
   B. Data flows in both directions, but only one direction at a time.
   C. Data flows in both directions simultaneously.
   D. Data flows in one direction, with acknowledgment sent in the other direction.

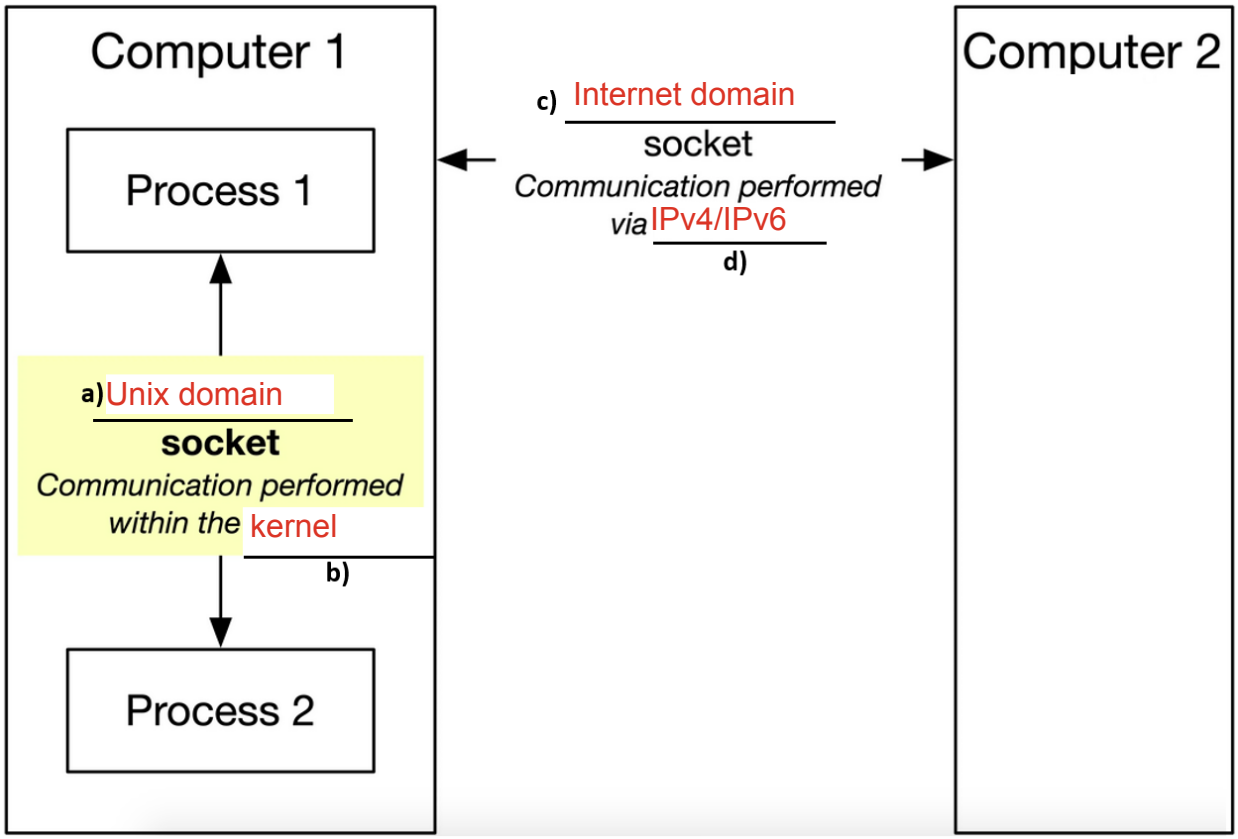
3. [**1 point**] Which socket type preserves message boundaries and ensures message delivery in the order sent?

   A. SOCK_DGRAM B. SOCK_SEQSTREAM C. SOCK_STREAM D. SOCK_SEQPACKET

4. [**4 points**] Complete the following C code by adding comments next to each // to explain its functionality.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(void) {
    int fd;

    printf("First line starts everything\n");

    // create a copy of the file descriptor (to restore later)

    fd = dup(STDOUT_FILENO);

    // Open file "dup.log" to write anything to stdout

    freopen("dup.log", "w", stdout);

    printf("Here comes the second line\n");

    // Force write all buffered data

    fflush(stdout);

    // Restore stdout, clean up temporary copy

    dup2(fd, STDOUT_FILENO);
    close(fd);

    printf("Third line is the charm\n");
    exit(0);
}
```

5. [**2 points**] Fill in the blanks (a, b, c, d) to complete the following figure:



6. [**1 point**] Explain the key differences between TCP and IDP protocols.

TCP:

Connection-oriented
Reliable
High protocol overhead
Uses stream socket

UDP:

Connectionless
Unreliable
Lower protocol overhead
Uses datagram

CSC/CPE 357 - Fall 2024	10 points	Quiz 4 v2

Name: _____ Cal Poly Email: _____

1. [**1 point**] Using FIFOs, what type of processes can exchange data?

   A. Named processes B. Unnamed processes C. Related processes (D.) Unrelated processes

2. [**1 point**] Which of the following is an advantage of UNIX domain sockets over Internet sockets?

   A. Ability to establish communication between computers on different networks.
   (B.) Enhanced performance due to the absence of protocol processing and network overhead.
   C. Built-in support for encryption and secure communication.
   D. Automatic generation of sequence numbers for reliable communication.

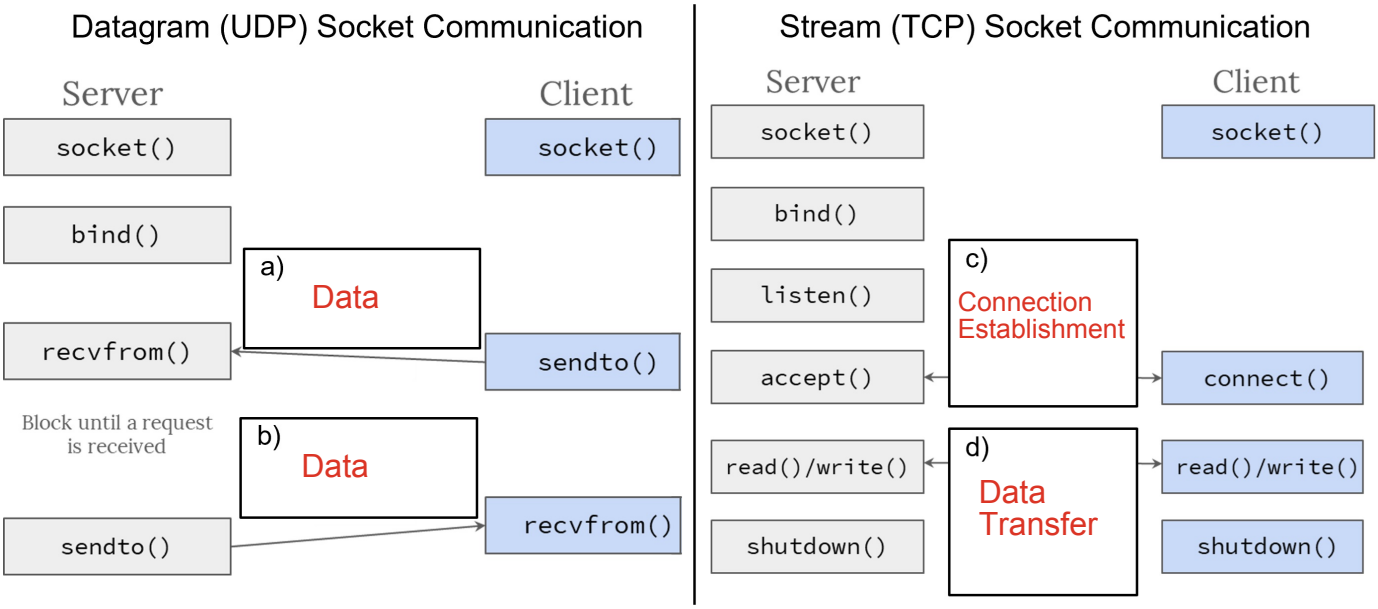
3. [**1 point**] What does the socketpair() function create?

   (A.) Full-duplex UNIX domain sockets B. Half-duplex network sockets
   C. Full-duplex TCP sockets D. Half-duplex UNIX domain sockets

4. [**4 points**] Write a C program demonstrating the functionality of the dup() and dup2() functions.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int f1(void);

// demonstrate dup() functions, redirect stdout to a file
int main(void) {
    int fd;

    printf("starting dup() example\n");

    fflush(stdout);

    // create a copy of stdout file descriptor (to restore later)
    fd = dup(STDOUT_FILENO);

    // open file "dup.log" and associate it with stdout stream
    // cause anything to be written to stdout to be sent to file "dup.log"
    freopen("dup.log", "w", stdout);

    printf("calling f1()\n");
    f1();

    fflush(stdout);

    // restore stdout, clean up temporary copy
    dup2(fd, STDOUT_FILENO);
    close(fd);

    printf("process complete\n");
    exit(0);
}

int f1(void) {
    printf("message from function f1()\n");
    return 0;
```

5. [**2 points**] Fill in the blanks (a, b, c, d) to complete the following figure:



6. [**1 point**] Explain the key differences between Unix domain socket and Internet domain socket.

Unix domain socket

Local communication between processes
No network protocol overhead
Higher performance for IPC

Internet domain socket

Can communicate over network
Include protocol overhead
Less feasible for IPC