Systems Programming – Homework 3: Memory & Processes

Course: Introduction to Systems Programming

Instructions: Write your answers clearly by hand on paper. Show all work and reasoning for partial credit. Assume a Linux/x86-64 environment.

Due: Submit a scanned PDF by the due date.

Problem 1: Memory Layout & Pointer Arithmetic (25 points)

Part A: Pointer Calculations

Given the following declarations:

```c
int data[5] = {50, 60, 70, 80, 90};
int *ptr1 = data + 2;      // Points to 70
int *ptr2 = &data[4];      // Points to 90
```

Calculate the values of:

1. `*ptr1`

2. `*(ptr1 - 1)`

3. `ptr2 - ptr1`

4. `*(ptr2 - 2)`

Show each step of your calculation.

Part B: Memory Diagram

Draw a complete memory diagram after this code executes to the comment `// HERE`. Include the stack, heap, and all pointer relationships.

```c
#include <stdlib.h>

void memory_example() {
    int x = 100;
    int *a = &x;
    int *b = malloc(2 * sizeof(int));

    b[0] = 200;
    b[1] = 300;

    int **c = &a;
    *c = b;  // a now points to heap

    // HERE
}
```

Label all variables, values, and arrows showing pointers.

Problem 2: Process Creation & File Descriptors (25 points)

Part A: Fork Analysis

Trace the execution of this program. Draw the process tree and determine all possible outputs.

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

```c
int main() {
    printf("A: PID=%d\n", getpid());

    if (fork() == 0) {
        printf("B: Child PID=%d\n", getpid());
        if (fork() == 0) {
            printf("C: Grandchild PID=%d\n", getpid());
        }
    } else {
        printf("D: Parent PID=%d\n", getpid());
    }

    printf("E: PID=%d exiting\n", getpid());
    return 0;
}
```

1. How many total processes are created?

2. List two different possible output orderings (due to scheduling).

3. Which print statement(s) could appear more than once? Why?

Part B: File Descriptor Manipulation

A process executes the following:

```c
int fd1 = open("input.txt", O_RDONLY);  // Returns fd 3
int fd2 = dup(fd1);                     // Returns fd 4
close(0);                               // Close stdin
dup2(fd1, 1);                           // Redirect stdout
close(fd1);
```

Draw the process's file descriptor table after all these operations. Show:

- File descriptor numbers (0-6)
- Connections to the kernel's open file table
- Reference counts
- What happens to standard input/output

Total: 100 points

Submission: Handwrite your answers, scan as a single PDF, and upload by the deadline. Show all work for full credit.