1: // Parent Process 2:

3: #include <stdio.h>

4:

5: #include <unistd.h>

6:

7: #include <sys/types.h>

8:

9: #include <sys/wait.h>

10:

11: #include <stdlib.h>

12:

13: #include <string.h>

17: // Function to sort an array using bubble sort 18:

19: void sortArray(int arr[], int n) { 20:

21: for (int i = 0; i < n - 1; i++) { 22:

23: for (int j = 0; j < n - i - 1; j++) { 24:

25: if (arr[j] > arr[j + 1]) { 26:

27: int temp = arr[j]; 28:

29: arr[j] = arr[j + 1];

30:

31: arr[j + 1] = temp;

32:

33: }

34:

35: }

36:

37: }

38:

39: }

43: int main(int argc, char \*argv[]) { 45: pid\_t pid;

47: int n;

51: // Input the number of elements

53: printf("Enter number of elements: "); 55: scanf("%d", &n);

59: // Allocate memory for the array dynamically 61: int \*arr = (int \*)malloc(n \* sizeof(int));

63: if (arr == NULL) {

65: perror("Memory allocation failed"); 67: return 1;

69: }

73: // Input array elements

75: printf("Enter %d elements: ", n); 77: for (int i = 0; i < n; i++) {

79: scanf("%d", &arr[i]);

81: }

85: // Sort the array 87: sortArray(arr, n);

91: // Create a child process 93: pid = fork();

97: if (pid < 0) {

99: printf("Fork failed!\n");

101: free(arr);

103: return 1;

105: }

109: if (pid == 0) {

111: // In child process: prepare arguments for execve

113: char \*args[n + 2]; // +1 for NULL and +1 for program name 115: args[0] = "./child"; // Name of the child program

119: for (int i = 0; i < n; i++) {

121: // Allocate memory for each number string

123: char \*num\_str = (char \*)malloc(10 \* sizeof(char)); // Assuming number won't excee 125: if (num\_str == NULL) {

127: perror("Memory allocation failed for number string"); 129: exit(EXIT\_FAILURE);

131: }

133: sprintf(num\_str, "%d", arr[i]); // Convert number to string 135: args[i + 1] = num\_str; // Store the string in the args array : 137: }

141: args[n + 1] = NULL; // Last element is NULL for execve : 145: // Load the child program using execve

147: if (execve(args[0], args, NULL) == -1) { 149: perror("execve");

151: exit(EXIT\_FAILURE);

153: }

157: // Free memory allocated for args in the child 159: for (int i = 1; i <= n; i++) {

161: free(args[i]);

163: }

165: } else {

167: // In parent process: wait for child to complete 169: wait(NULL);

171: printf("Parent process finished.\n"); 173: }

177: // Free dynamically allocated memory in parent 179: free(arr);

183: return 0;

185: }

187: //Child Process

188:

190: #include <stdio.h>

192: #include <stdlib.h>

196: int main(int argc, char \*argv[]) {

198: // Ensure that at least one argument is passed (excluding program name) 200: if (argc < 2) {

202: printf("Please provide integers as arguments.\n"); 204: return 1;

206: }

210: int n = argc - 1; // Number of elements (excluding the program name) 214: // Dynamically allocate memory for the array

216: int \*arr = (int \*)malloc(n \* sizeof(int)); 218: if (arr == NULL) {

220: printf("Memory allocation failed.\n"); 221:

222: return 1;

224: }

228: // Convert command-line arguments to integers and store in the array 230: for (int i = 0; i < n; i++) {

232: arr[i] = atoi(argv[i + 1]); // Convert string to integer 234: }

238: // Print the array in reverse order 240: printf("Array in reverse order: "); 242: for (int i = n - 1; i >= 0; i--) {

244: printf("%d ", arr[i]); 246: }

248: printf("\n");

252: // Free dynamically allocated memory 254: free(arr);

258: return 0; 259:

260: }

