#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <unistd.h>

#include <sys/wait.h>

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

pid\_t processid;

int n = argc;

processid = fork();

if (processid == -1) {

perror("fork failed");

exit(1);

}

if (processid == 0) {

printf("\nChild process id: %d\n", getpid());

printf("Child Parent process id: %d\n", getppid());

// Copy and sort the arguments

int\* array = (int\*)malloc((n - 1) \* sizeof(int));

for (int i = 1; i < n; i++) {

array[i - 1] = atoi(argv[i]);

}

// Bubble sort the array

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

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

if (array[j] > array[j + 1]) {

int temp = array[j];

array[j] = array[j + 1];

array[j + 1] = temp;

}

}

}

// Print the sorted elements

printf("The elements stored in array (after sorting) are:\n");

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

printf("%d\n", array[i]);

}

// Execute a new program with reversed arguments

char\* execArgs[n + 2];

execArgs[0] = "reversed\_array\_program"; // The name of the new program

for (int i = 1; i <= n; i++) {

execArgs[i] = argv[i];

}

execArgs[n + 1] = NULL;

execvp(execArgs[0], execArgs);

perror("execvp failed");

exit(1);

} else {

int status;

wait(&status);

if (WIFEXITED(status)) {

printf("Parent process id: %d\n", getpid());

printf("Parent parent process id: %d\n", getppid());

}

}

return 0;

}