**OSL Assignment 2B**

**Aditya Kangune**

**K 11**

**33323**

**Problem Statement:**

Assignment No. 2:

Process control system calls: The demonstration of FORK, EXECVE and WAIT system calls along with zombie and orphan states.

B. Implement the C program in which main program accepts an array. Main program uses the FORK system call to create a new process called a child process. Parent process sorts an array and passes the sorted array to child process through the command line arguments of EXECVE system call. The child process uses EXECVE system call to load new program which display array in reverse order.

1. **EXECV system call in C:**

The exec family of functions replaces the current running process with a new process. It can be used to run a C program by using another C program. It comes under the header file unistd.h. There are many members in the exec family which are shown below with examples.

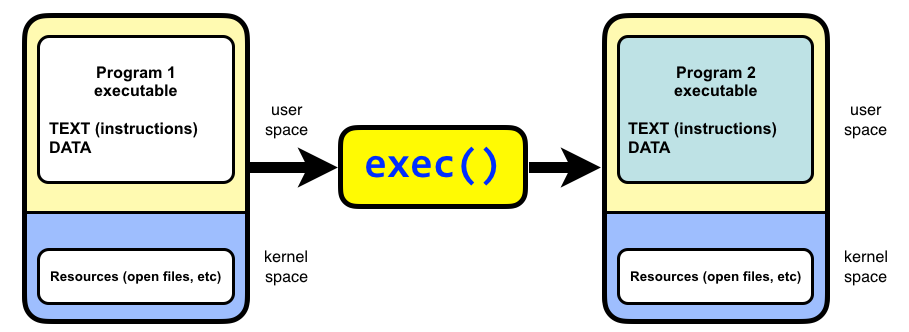
Using this command, the created child process does not have to run the same program as the parent process does. The exec type system calls allow a process to run any program files, which include a binary executable or a shell script.

**Syntax:**

**int execvp (const char \*file, char \*const argv[]);**

**file**: points to the file name associated with the file being executed.

**argv**: is a null terminated array of character pointers.



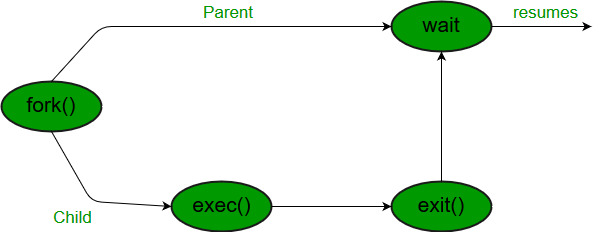
1. **Wait system call in C:**

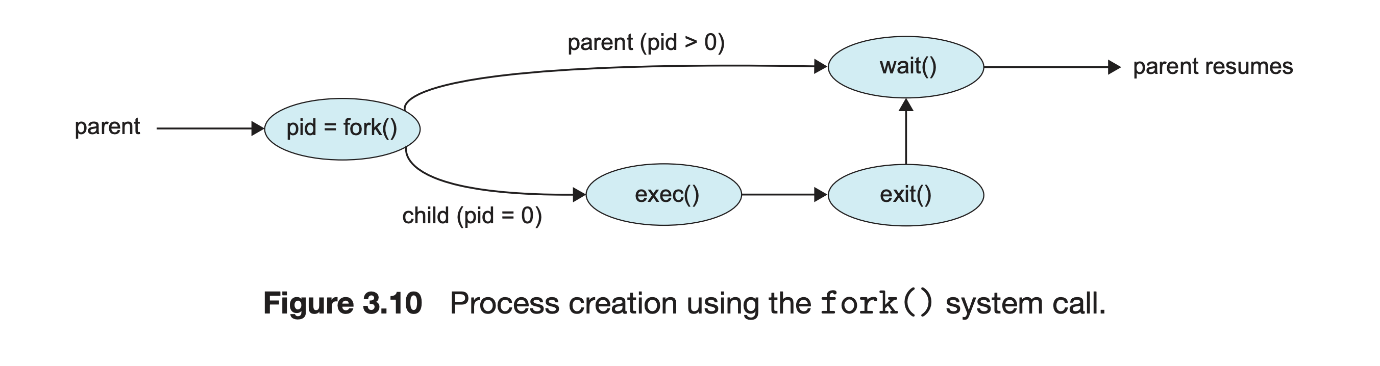
A call to wait() blocks the calling process until one of its child processes exits or a signal is received. After child process terminates, parent ***continues*** its execution after wait system call instruction.   
Child process may terminate due to any of these:

It calls exit();

It returns (an int) from main

It receives a signal (from the OS or another process) whose default action is to terminate.





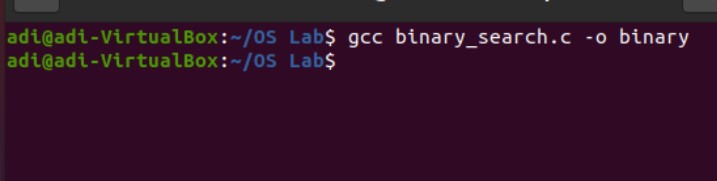
**How to get the output with both parent and child files:**

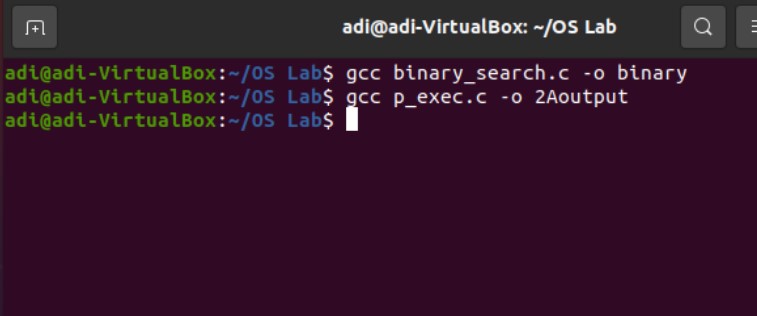
1. Compile child file.
2. Compile parent file. In the parent file use the output file of child file as the 0th index of buffer.
3. Run the parent file.

**Note:** No need to run child file explicitly.

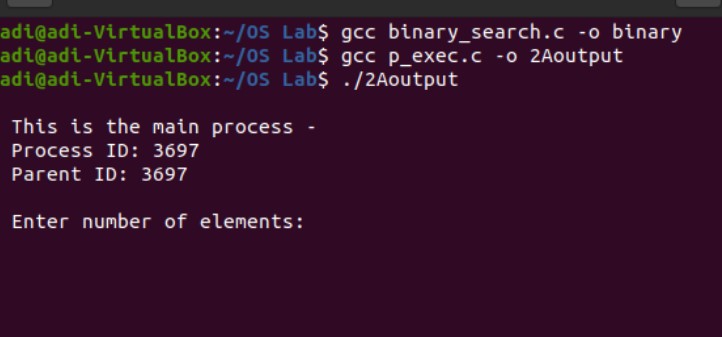
**Output screenshots:**

Compile the child file and store output in “binary” file:

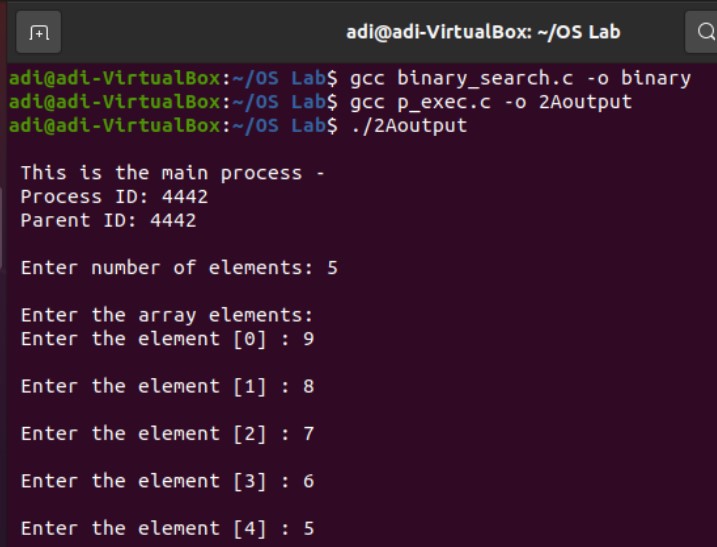


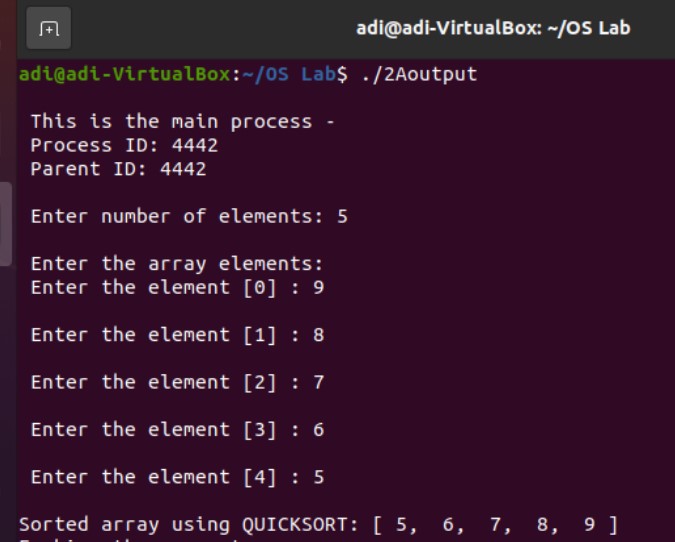
Compile the parent file with “binary” used in buffer: 

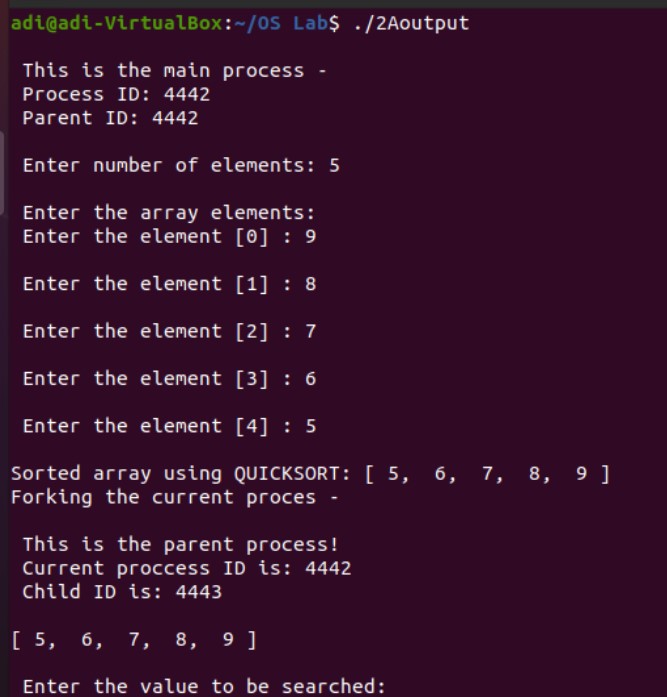
Run the parent file:

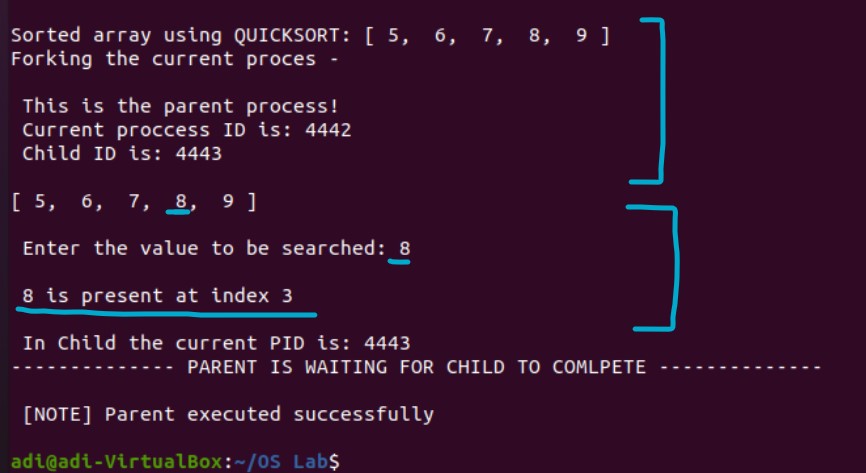


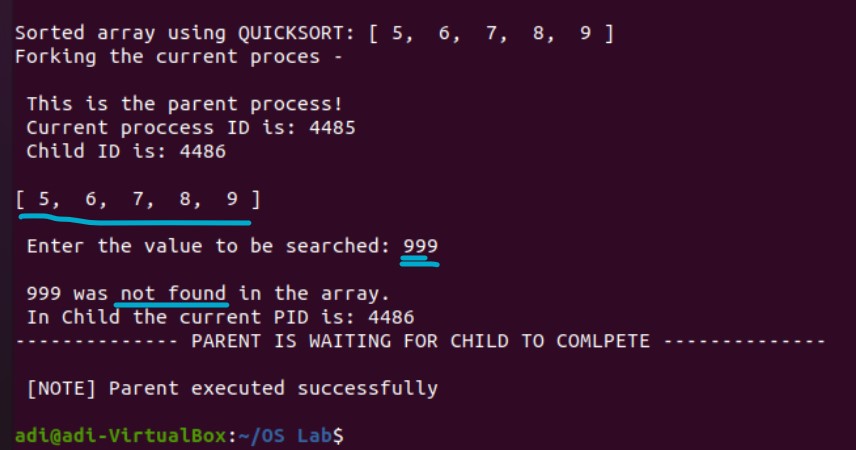
Enter array size and elements:



Output of parent process (QUICKSORT): 

After forking and execv system call: 

Output of child process if element is found: 

Output of child process if element is **NOT** found: 

**Final complete output:**

adi@adi-VirtualBox:~/OS Lab$ gcc binary\_search.c -o binary  
adi@adi-VirtualBox:~/OS Lab$ gcc p\_exec.c -o 2Aoutput  
adi@adi-VirtualBox:~/OS Lab$ ./2Aoutput  
  
 This is the main process -  
 Process ID: 5388  
 Parent ID: 5388  
  
 Enter number of elements: 5  
  
 Enter the array elements:  
 Enter the element [0] : 9  
  
 Enter the element [1] : 8  
  
 Enter the element [2] : 7  
  
 Enter the element [3] : 6  
  
 Enter the element [4] : 5  
  
Sorted array using QUICKSORT: [ 5,  6,  7,  8,  9 ]  
Forking the current proces -  
  
 This is the parent process!  
 Current proccess ID is: 5388  
 Child ID is: 5389  
  
[ 5,  6,  7,  8,  9 ]  
  
 Enter the value to be searched: 8  
  
 8 is present at index 3  
  
 In Child the current PID is: 5389  
-------------- PARENT IS WAITING FOR CHILD TO COMLPETE --------------  
  
 [NOTE] Parent executed successfully  
  
adi@adi-VirtualBox:~/OS Lab$