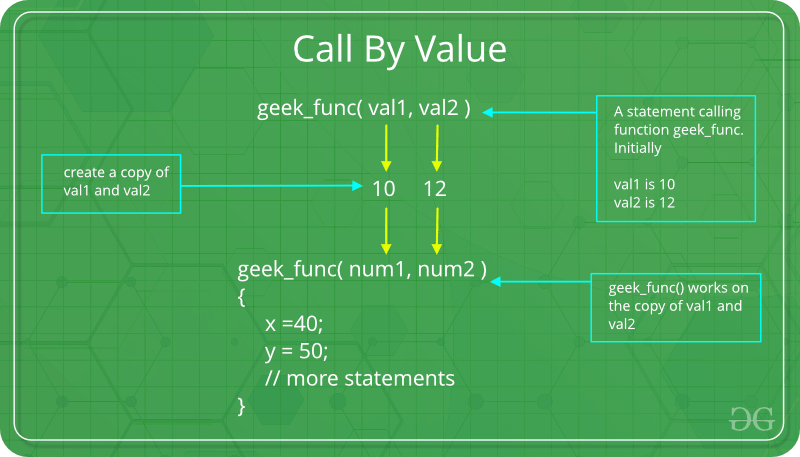
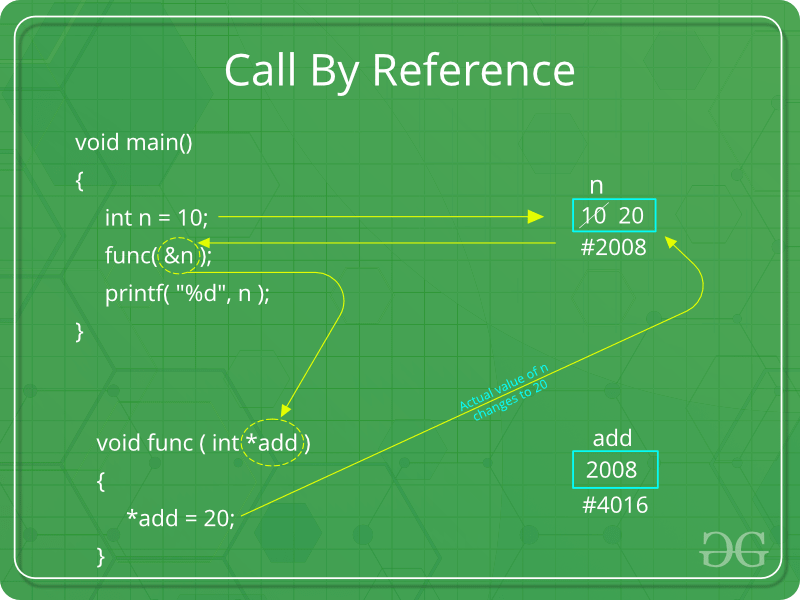
**Multiple Processes in C**

**In order to complete this lab, you have to use SSH tool to connect the server courses.brockport.edu. A portable version of Putty is** <https://the.earth.li/~sgtatham/putty/latest/w32/putty.exe>

**Create a folder under your login directory named csc414. For each lab, please create a separate folder to store the code.**

1. Parameter passing is one of important concepts in C/C++. Two techniques are involved in parameter passing: call by value and call by reference.





The subsequent code is about to swap value between two variables via passing parameters. Copy the below code to a file named pointer.c. Compile the code by using *gcc pointer.c –o POINTER*. You can run the program by type *./POINTER*. There are some errors in the code. After using a similar idea as the above example, please correct them so that the program should be able to swap the values. Paste the **screenshot** of the corrected code and explain why.

**#include <stdio.h>**

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

**{**

**void swap (int, int);**

**int a, b;**

**char str1 ="Please enter two integers: ";**

**printf ("%s", str1);**

**scanf ("%d %d", &a, &b);**

**swap (a, b);**

**printf ("a = %d b = %d\n", a, b);**

**return 0;**

**}**

**void swap (int x, int y)**

**{**

**int temp;**

**temp = x; x = y; y = temp;**

**return;**

**}**

**A computer screen shot of a program

Description automatically generated**

A screenshot of a computer code

Description automatically generated

Strings in c are arrays of chars. To pass a variable by reference insert & inbetween var type and var name. In function declarations insert \* in same location as &.

1. Create a file named hellofork.c using the following code. Compile and run the below code and answer the following questions. Hint: gcc hellofork.c –o hellofork
2. How many times is “Hello World!” printed? (2) Why? Give the explanation on your answer.

**#include <stdio.h>**

**#include <unistd.h>**

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

**{**

**fork();**

**fork();**

**fork();**

**printf ("Hello World! My pid = %d and My parent pid = %d\n",**

**getpid(), getppid());**

**wait();**

**return 0;**

**}**

**The statement is printed 8 times because there are three forks. At each fork another fork is being done. Since there are 3 forks with 2 outcomes each time there are 2^8 processes and statements printed.**

1. Compile the below code and create an executable file named displayPID. Please (1) paste the screenshot to show the output, and (2) explain why this program can get those PIDs. Hint: What are stored in PCB?

**/\***

**Program to illustrate UNIX**

**system calls getpid and getppid**

**\*/**

**#include <stdio.h>**

**#include <sys/types.h>**

**#include <unistd.h>**

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

**{**

**pid\_t mypid, parentpid;**

**mypid = getpid();**

**parentpid = getppid();**

**printf("Hi! I am a simple process. In this birth,\n");**

**printf("My pid is %d\n", mypid);**

**printf("My parent's pid is %d\n", parentpid);**

**printf("I will terminate when you press Enter key\n");**

**getchar();**

**return 0;**

**}**

1. Compile and run the below program. (1) Describe what the output is. Paste a screenshot to show the output. (2) Explain how the code works.

**/\***

**Program to illustrate UNIX system calls**

**fork, exec, wait, getpid, and getppid**

**\*/**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <sys/types.h>**

**#include <unistd.h>**

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

**{**

**pid\_t childpid, mypid;**

**mypid = getpid();**

**printf("Hi! I am a process about to fork\n");**

**printf("My pid is %d\n", mypid);**

**printf("I will proceed when you press Enter key\n");**

**getchar();**

**childpid = fork();**

**if (childpid == -1) {**

**printf("Error in fork; program terminated\n");**

**exit (-1);**

**}**

**if (childpid == 0) {**

**/\* This code is exceuted only be the child \*/**

**execlp("./displayPID", "displayPID", (char \*) NULL);**

**}**

**else {**

**/\* This code is executed only by the parent \*/**

**int status;**

**/\* Wait for the child to terminate \*/**

**wait(&status);**

**mypid = getpid();**

**printf("Hi! I am the parent process after fork\n");**

**printf("My pid is %d\n", mypid);**

**printf("My child had pid %d\n", childpid);**

**}**

**return 0;**

**}**

1. The exec() family of system calls replaces the current process image with a new process image. There are several functions – execl, execlp, execle, execv, execvp, and execvpe. Function prototype for execlp is

**int execlp(const char \*path, const char \*arg, .. (char \*) NULL);**

|  |
| --- |
| Here are the descriptions for the family functions: |
| <https://manpages.ubuntu.com/manpages/xenial/man3/exec.3.html>  <https://www.ibm.com/docs/en/aix/7.1?topic=e-exec-execl-execle-execlp-execv-execve-execvp-exect-fexecve-subroutine> |

The first argument provides the path to the new executable; if only a file name is provided the PATH environment variable is used to search for the executable. The subsequent arguments are passed as command line arguments to the new executable. The systems call returns only if an error has occurred, in which case the return value is -1.

Please answer: (1) Does the printf statement that follows the invocation of “execlp” print out anything? Try it and observe what happens. Paste the screenshot to show the output. (2) Explain why?

**#include <stdio.h>**

**#include <unistd.h>**

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

**{**

**printf ("This will execute ls command\n");**

**printf ("Press Enter key to proceed\n");**

**getchar();**

**execlp ("/bin/ls", "ls", "-l", (char \*) NULL);**

**printf ("Successful completion of ls\n");**

**return 0;**

**}**