**Name: Brijesh Rohit**

**Admission no.: U19CS009**

**SYSTEM SOFTWARES - ASSIGNMENT -1**

**Fork and Getpid:**

**fork() creates a new child process of the previous parent process.**

**getpid() returns the process id of the calling function.**

**Code=>**

|  |
| --- |
| **#include<stdio.h>**  **#include<sys/types.h>**  **#include<unistd.h>**  **#include<stdlib.h>**  **int main()**  **{**  **if (fork() == 0)**  **{**  **printf("\nParent process is called!!");**  **printf("\nParent process pid : %d\n", getpid());**  **}**  **else**  **{**  **printf("\nChild process is called!!");**  **printf("\nChild process pid : %d\n", getpid());**  **}**  **printf("\n");**  **return 0;**  **}** |

**Output=>**



**Exec:**

**It creates and replaces the currently running process with another process**

**Code=>**

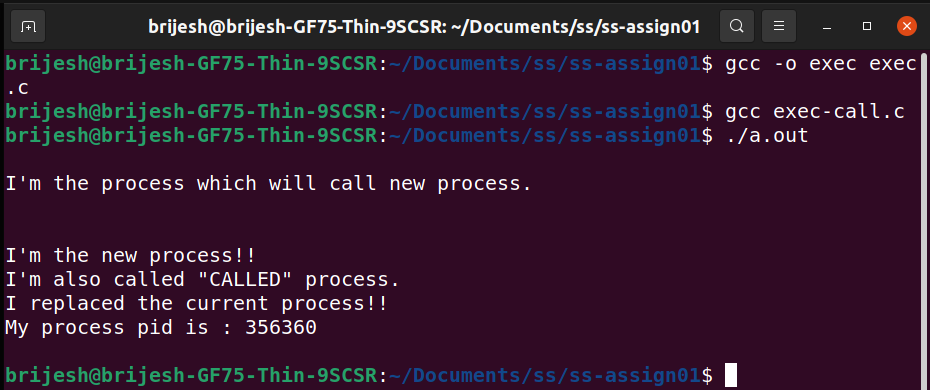
**exec.c [called file]**

|  |
| --- |
| **#include<stdio.h>**  **#include<sys/types.h>**  **#include<unistd.h>**  **int main()**  **{**  **//EXEC**  **printf("\n\nI'm the new process!!\nI'm also called \"CALLED\" process.");**  **printf("\nI replaced the current process!!");**  **printf("\nMy process pid is : %d", getpid());**  **printf("\n\n");**  **return 0;**  **}** |

**exec-call.c [calling file]**

|  |
| --- |
| **#include<stdio.h>**  **#include<sys/types.h>**  **#include<unistd.h>**  **#include<stdlib.h>**  **int main()**  **{**  **printf("\nI'm the process which will call new process.");**  **printf("\nI'm also called \"CALLING\" process.");**  **//EXEC**  **char \*args[] = {"./exec", NULL};**  **execvp(args[0], args);**  **printf("\nAfter calling, I'm running a print statement.");**  **printf("\nWhich will not be printed.\n");**  **return 0;**  **}** |

**Output=>**



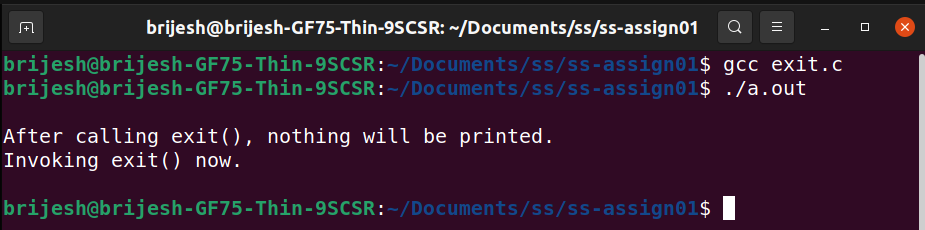
**Exit:**

**Terminates the execution of a program**

**CODE=>**

|  |
| --- |
| **#include<stdio.h>**  **#include<sys/types.h>**  **#include<unistd.h>**  **#include<stdlib.h>**  **int main()**  **{**  **printf("\nAfter calling exit(), nothing will be printed.");**  **printf("\nInvoking exit() now.\n\n");**  **exit(0);**  **printf("not printing because of exit()");**  **return 0;**  **}** |

**Output=>**



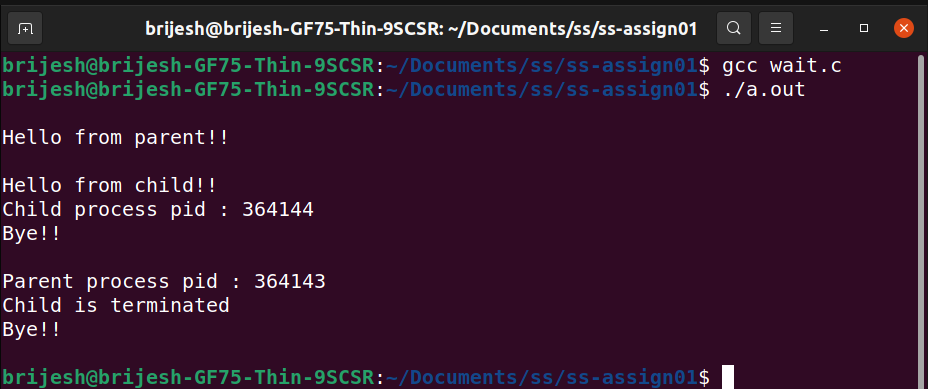
**Wait:**

**Wait() blocks the execution of parent process until a child process has finished executing or a signal is received. After child process ends parent will continue its execution.**

**CODE=>**

|  |
| --- |
| **#include<stdio.h>**  **#include<stdlib.h>**  **#include<sys/wait.h>**  **#include<unistd.h>**  **int main() {**  **int cpid;**  **if (fork()== 0)**  **{**  **printf("\nHello from child!!");**  **printf("\nChild process pid : %d", getpid());**  **}**  **else**  **{**  **printf("\nHello from parent!!");**  **printf("\nParent process pid : %d", getpid());**  **wait(NULL);**  **printf("\nChild is terminated");**  **}**    **printf("\nBye!!");**  **printf("\n\n");**  **return 0;**  **}** |

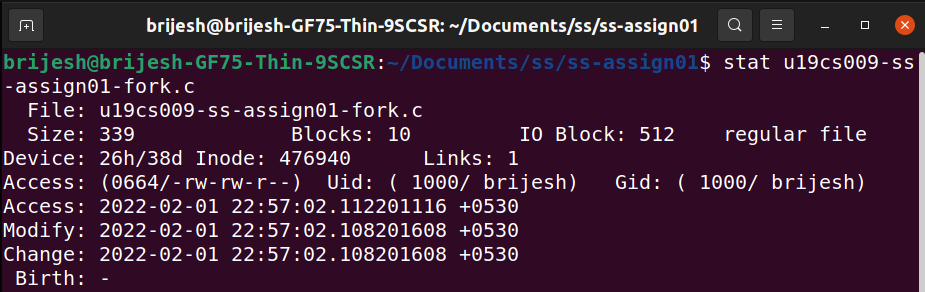
**Output=>**



**Stat:**

**It is used to display the status of a file (size, name, blocks used, etc.).**

**Output=>**



**Opendir:**

**It is used to open a directory stream corresponding to the directory name.**

**Closedir:**

**It closes the currently opened directory stream.**

**Readdir:**

**Reads the files and directories present in the opened directory stream.**

**Chdir:**

**It changes the current working directory to another one specified by user.**

**CODE=>**

|  |
| --- |
| **#include <stdio.h>**  **#include <stdlib.h>**  **#include <sys/types.h>**  **#include <unistd.h>**  **#include <dirent.h>**  **#include <errno.h>**  **int main()**  **{**  **struct dirent \*dir;**  **DIR\* d = opendir(".");**  **//READDIR**  **while ((dir = readdir(d)) != NULL)**  **printf("%s\n", dir->d\_name);**  **char s1[100];**  **printf("\nCurrent path : %s\n", getcwd(s1, 50)); //CHDIR**  **chdir("..");**    **printf("New path after using chdir : %s\n", getcwd(s1, 50)); //READDIR**  **while ((dir = readdir(d)) != NULL)**  **printf("%s\n", dir->d\_name);**    **//CLOSEDIR**  **closedir(d);**  **printf("\n\n");**  **exit(EXIT\_SUCCESS);**  **return 0;**  **}** |

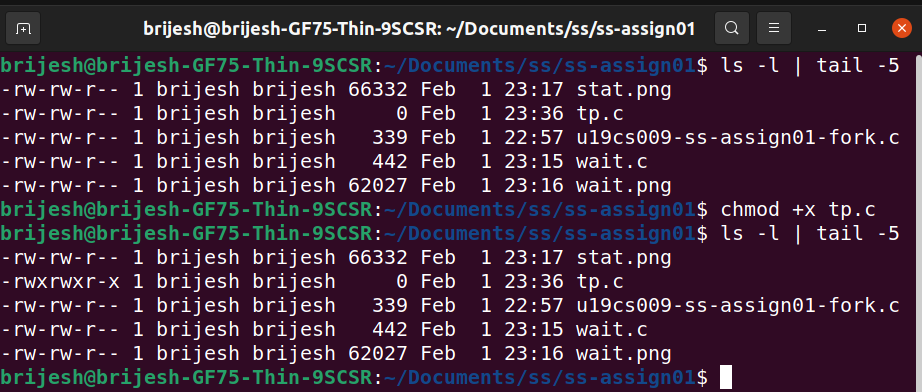
**Output=>**



**Chmod:**

**It is used change the access permissions or modes of a file-system.**

**OUTPUT=>**

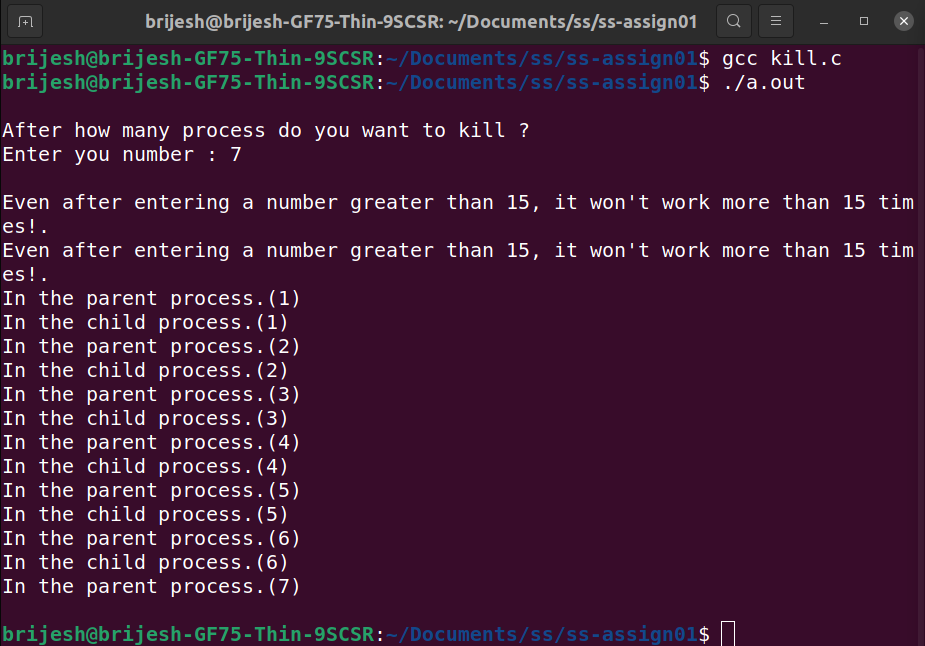


**Kill:**

**CODE=>**

|  |
| --- |
| **#include <stdio.h>**  **#include <stdlib.h>**  **#include <unistd.h>**  **#include <signal.h>**  **int main()**  **{**  **printf("\nAfter how many process do you want to kill ?");**  **printf("\nEnter you number : ");**  **int n;**  **scanf("%d",&n);**  **printf("\nEven after entering a number greater than 15, it won't work more than 15 times!.");**    **pid\_t retVal;**  **retVal = fork();**  **if(retVal > 0)**  **{**  **int i = 0;**  **while(i++ < n)**  **{**  **printf("\nIn the parent process.(%d)",i);**  **sleep(1);**  **}**  **//kill the child process**  **kill(retVal, SIGKILL);**  **}**  **else if (retVal == 0)**  **{**  **int i = 0;**  **//will not ever get to 15, because**  **//the parent process will kill it**  **while(i++ < 15)**  **{**  **printf("\nIn the child process.(%d)", i);**  **sleep(1);**  **}**  **}**  **else**  **{**  **//something bad happened.**  **printf("\nSomething bad happened.");**  **exit(EXIT\_FAILURE);**  **}**  **printf("\n\n");**  **return 0;**  **}** |

**Output=>**



**Read:**

**It reads specified bytes of input indicated by the file descriptor in to the memory buffer.**

**Write:**

**It writes specified bytes from the memory buffer to the file indicated by the file descriptor.**

**Open:**

**Opens a file for reading, writing or both.**

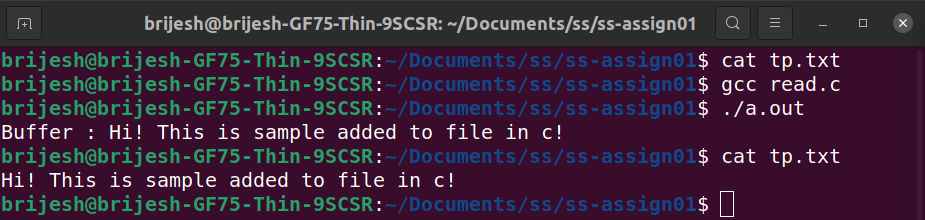
**Close:**

**Closes the file opened indicated by the file descriptor.**

**CODE=>**

|  |
| --- |
| **#include <stdio.h>**  **#include <stdlib.h>**  **#include <sys/types.h>**  **#include <sys/stat.h>**  **#include <fcntl.h>**  **#include <unistd.h>**  **#include <dirent.h>**  **#include <errno.h>**  **int main()**  **{**  **int f;**  **char buffer[100];**  **//OPEN**  **f = open("tp.txt", O\_CREAT | O\_WRONLY, 0600);**  **if (f == -1)**  **{**  **printf("Error opening the file");**  **exit(1);**  **}**  **//WRITE CLOSE**  **write(f, "Hi! This is sample added to file in c!\n", 39);**  **close(f);**  **//READ**  **f = open("tp.txt", O\_RDONLY);**  **if (f == -1)**  **{**  **printf("Error opening the file");**  **exit(1);**  **}**  **read(f, buffer, 39);**  **buffer[39] = '\0';**  **close(f);**  **printf("Buffer : %s", buffer);**  **return 0;**  **}** |

**Output=>**



**Lseek:**

**Used to change the read/write pointer of a file descriptor.**

**CODE=>**

|  |
| --- |
| **#include <stdio.h>**  **#include <stdlib.h>**  **#include <sys/types.h>**  **#include <sys/stat.h>**  **#include <fcntl.h>**  **#include <unistd.h>**  **#include <dirent.h>**  **#include <errno.h>**  **int main()**  **{**  **printf("\n");**  **int n, f;**  **char buff[6];**  **f = open("input.txt",O\_RDWR);**  **while(read(f,buff,6))**  **{**  **read(f,buff,6);**  **write(1,buff,6);**  **lseek(f,5,SEEK\_CUR);**  **read(f,buff,6);**  **write(1,buff,6);**  **}**  **printf("\n");**  **return 0;**  **}** |

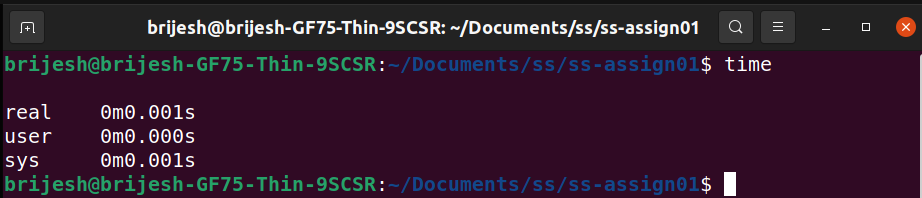
**Output=>**



**Time:**

**Returns the time since epoch.**

**Output=>**



**Mount:**

**It is used to mount the file-system found on a device to big tree structure(Linux file-system) rooted at '/'.**

**Chown:**

**It is used to change the owner and group of the file specified by the file descriptor or path.**

