OS - Practical - 4

Name: Rishabh Jain

Branch & Sem: CSE - IV

Sec: A

Roll no.:54

AIM : Demonstrate process control system calls:  
fork, vfork, exec, wait and sleep, getpid and getppid also understand the concept of fork bomb, zombie states and orphan states.  
Note: All the terminal outputs are given in the form of Screenshots.

PROGRAMS:

1. Simple fork example

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main(void)

{

fork();

printf("Hello \n");

fork();

printf("bye\n");

return 0;

}

OUTPUT:

1. Program showing process pid and ppid.

Creating child process using fork sys call.  
#include <stdio.h>

#include <unistd.h>

int main()

{

    int id;

    printf("Hello, World!\n");

    id = fork();

    if (id > 0)

    {

        printf("Parent Process.\n");

        printf("Process id: %d.\n", getpid());

    }

    else if (id == 0)

    {

        printf("Child Process.\n");

        printf("Process id: %d.\n", getpid());

    }

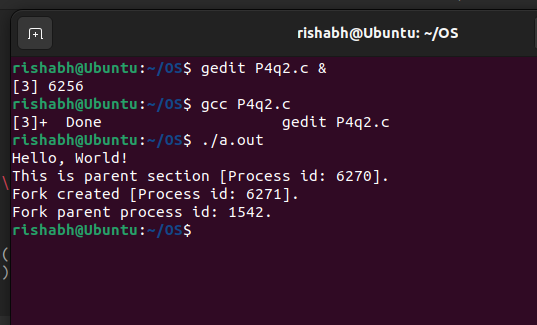
    else

    {

        printf("Fork Creation Failed\n");

    }

    return 0;

OUTPUT:

3)Program to check if a number is prime and sorting numbers.   
If after fork pid == 0 then it will go to child process which is checking if a number is prime or not. Else it will go to parent process where it will first sleep for 5 sec then asks user to input 10 numbers and then sorts it.

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

void main()

{

    int pid, n, a[10], i, t, j, flag = 0;

    pid = fork();

    printf("Pid=%d\n", pid);

    if (pid == 0)

    {

        printf("Enter a number to check whether prime or not\n");

        scanf("%d", &n);

        if (n == 1)

        {

            printf("Number is prime\n");

        }

        else

        {

            for (j = 2; j < (n / 2); j++)

            {

                if (n % j == 0)

                {

                    flag == 1;

                    printf("Not a Prime\n");

                    break;

                }

            }

            if (flag == 0)

                printf("Prime Number\n");

        }

    }

    else

    {

        sleep(5);

        printf("Enter 10 numbers to sort\n");

        for (j = 0; j < 10; j++)

            scanf("%d", &a[j]);

        for (i = 0; i < 9; i++)

        {

            for (j = 0; j < 9 - i; j++)

            {

                if (a[j + 1] < a[j])

                {

                    t = a[j];

                    a[j] = a[j + 1];

                    a[j + 1] = t;

                }

            }

        }

        printf("Sorted Elements are:\n");

        for (i = 0; i < 10; i++)

        {

            printf("%d ", a[i]);

        }

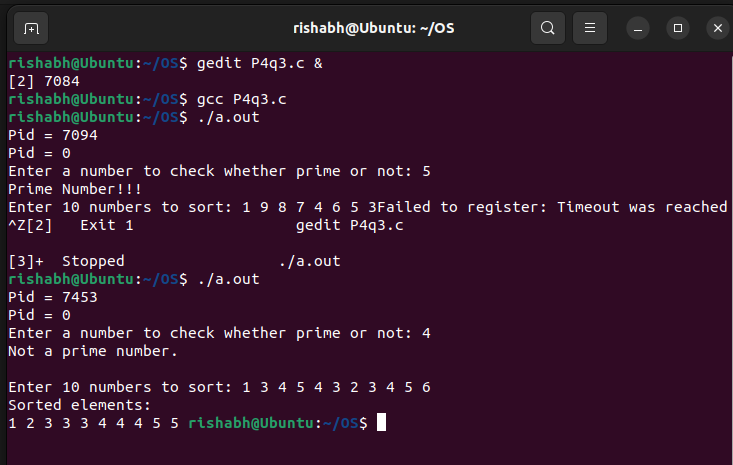
        printf("\n");

        printf("PID; %d\n", getpid());

        printf("PPID; %d\n", getppid());

    }

}

OUTPUT:

4.1) Prints the pid and ppid of parent process.

#include <stdio.h>

void main()

{

    int pid1, pid, ppid;

    pid1 = fork();

    if (pid1 > 0)

    {

        printf("I am the parent process\n");

        printf("Parent pid is %d\n", getpid());

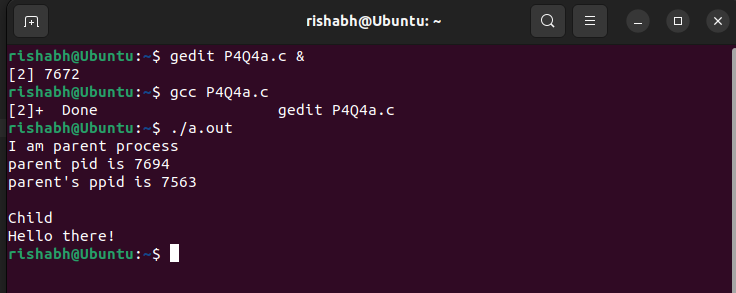
        printf("Parent ppid is %d\n", getppid());

        printf("\n");

        sleep(10);

    }

}

OUTPUT:

4.2) Program that prints numbers from 1 to 30 but after printing one number process sleeps for 1 second and the continues the loop.  
 After Executing the child process it goes to parent process where it sleeps for 10 sec then prints I am the Parent Process then waits for 0 sec and prints I am the Parent Process again.

#include <stdio.h>

void main()

{

    int pid1, i;

    pid1 = fork();

    if (pid1 == 0)

    {

        printf("I am the child process\n");

        for (i = 0; i < 30; i++)

        {

            printf("%d\n", i);

            sleep(1);

            // system("ps -el");

        }

        exit(0);

    }

    else

    {

        wait(0);

        sleep(10);

        printf("I am the Parent Process\n");

        wait(0);

        // sleep(10);

        printf("I am the Parent Process\n");

        // wait(0);

    }

}

OUTPUT:



4.3) Program to verify pid and ppid

#include <stdio.h>

void main()

{

    int pid1, pid, ppid;

    pid1 = fork();

    if (pid1 == 0)

    {

        printf("I am the child process\n");

        printf("Child pid is %d\n", getpid());

        printf("Child ppid is %d\n", getppid());

        printf("\n");

        system("ps -el");

        sleep(5);

        printf("Now child pid is %d\n", getpid());

        printf("Now child ppid is %d\n", getppid());

        system("ps -el");

    }

    if (pid1 > 0)

    {

        sleep(2);

        printf("I'm the parent process\n");

        printf("Parent pid is %d\n", getpid());

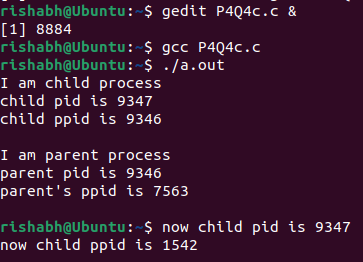
        printf("Parent ppid is %d\n", getppid());

        printf("\n");

    }

}

OUTPUT:



Here, Child PPID and Parent PID is same, but when the parent process gets completed and child process is still running then child process will have a random PPID of system (i.e. here 1576).

5.1) Program executing wait syscall

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

int main()

{

    int status;

    pid\_t fork\_return;

    fork\_return = fork();

    if (fork\_return == 0) /\* child process \*/

    {

        sleep(2);

        printf("I'm the child!\n");

        exit(0);

    }

    else /\* parent process \*/

    {

        printf("Parent");

        wait(&status);

        printf("\nI'm the parent!");

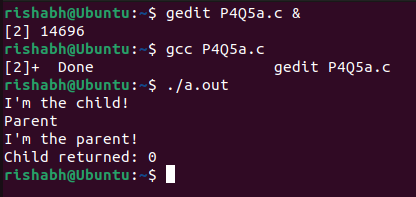
        if (WIFEXITED(status))

            printf("\nChild returned: %d\n", WEXITSTATUS(status));

    }

}

OUTPUT:



5.2) Program executing waitpid

#include <stdio.h>

#include <stdlib.h>

#include <sys/wait.h>

#include <unistd.h>

void waitexample()

{

    int i, stat;

    pid\_t pid[5];

    for (i = 0; i < 5; i++)

    {

        if ((pid[i] = fork()) == 0)

        {

            sleep(1);

            exit(100 + i);

        }

    }

    // Using waitpid() and printing exit status of children.

    for (i = 0; i < 5; i++)

    {

        pid\_t cpid = waitpid(pid[i], &stat, 0);

        if (WIFEXITED(stat))

            printf("Child %d terminated with status: %d\n", cpid,

                   WEXITSTATUS(stat));

    }

}

// Driver code

int main()

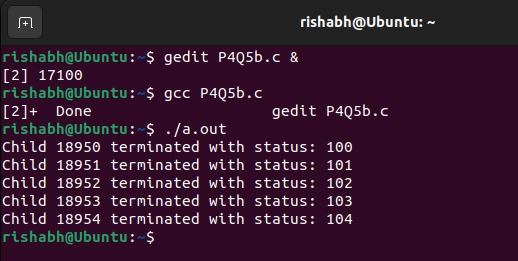
{

    waitexample();

    return 0;

}

OUTPUT:



6) Program for orphan process

#include <stdio.h>

#include <unistd.h>

int main()

{

    pid\_t p;

    /\* create child process \*/

    p = fork();

    if (p == 0)

    {

        /\* fork() returns Zero to child \*/

        sleep(5);

    }

    printf("The child process PID is %d\nAnd Parent PID is %d\n\n", getpid(),

           getppid());

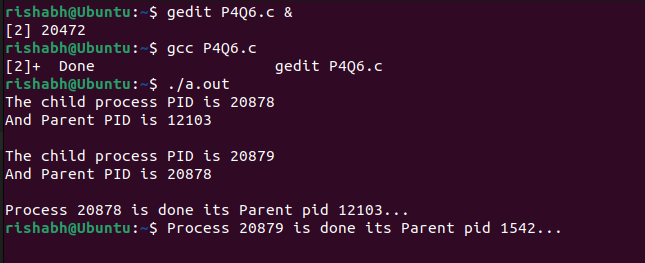
    /\* parent/child waits for 20 secs and exits \*/

    sleep(10);

    printf("Process %d is done its Parent pid %d...\n", getpid(), getppid());

    return 0;

}

OUTPUT:  


7.1) Program demonstrating fork()

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

#include <string.h>

#include <errno.h>

int main()

{

    int i, value;

    int status;

    pid\_t f;

    value = 0;

    i = 0;

    status = 1;

    f = fork();

    if (f < 0)

    {

        fprintf(stderr, "Error : %s - fork() < 0 (%d)\n", strerror(errno), f);

    }

    else if (f > 0)

    {

        printf("\n =====Begin Parent=====\n\n");

        printf(" fork() = % d\n", f);

        printf(" getpid() = % d\n", getpid());

        while (i < 10)

        {

            printf(" Parent-value = % d\n", value);

            ++value;

            ++i;

        }

    }

    else

    {

        printf("\n =====Begin Child=====\n\n");

        printf(" fork() = % d\n", f);

        printf(" getpid() = % d\n", getpid());

        while (i < 10)

        {

            printf(" Child - value = % d\n", value);

            ++value;

            ++i;

        }

        printf(" status = % d\n", status);

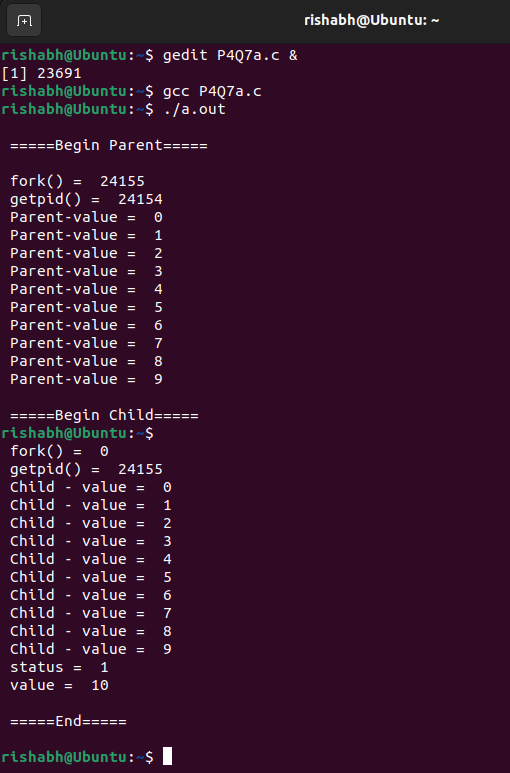
        printf(" value = % d\n\n", value);

        printf(" =====End=====\n\n");

        return 0;

    }

}

OUTPUT:  


7.2) Program Demonstrating vfork()

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

#include <string.h>

#include <errno.h>

int main()

{

    int i, value;

    int status;

    pid\_t f;

    value = 0;

    i = 0;

    status = 1;

    f = vfork();

    if (f < 0)

    {

        fprintf(stderr, "Error : %s - fork() < 0 (%d)\n", strerror(errno), f);

    }

    else if (f > 0)

    {

        printf("\n=====Begin Parent=====\n\n");

        printf(" fork() = %d\n", f);

        printf(" getpid() = %d\n", getpid());

        while (i < 10)

        {

            printf(" Parent-value = %d\n", value);

            ++value;

            ++i;

        }

    }

    else

    {

        printf("\n =====Begin Child=====\n\n");

        printf(" fork() = %d\n", f);

        printf(" getpid() = %d\n", getpid());

        while (i < 10)

        {

            printf(" Child - value = %d\n", value);

            ++value;

            ++i;

        }

        \_exit(status);

    }

    printf(" status = %d\n", status);

    printf(" value = %d\n\n", value);

    printf("=====End=====\n\n");

    return 0;

}

OUTPUT:



7.3) Another Program to demonstrate vfork().

To count number if vowels.

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

#include <string.h>

#include <errno.h>

int main()

{

    int j, n, a, i, e, o, u;

    char str[50];

    a = e = i = o = u = 0;

    pid\_t pid;

    if ((pid = vfork()) < 0)

    {

        perror(" FORK ERROR");

        exit(1);

    }

    if (pid == 0)

    {

        printf("Counting Number of Vowels using VFORK\n");

        printf("--------------------------------\n");

        printf("Enter the String : \n");

        gets(str);

        \_exit(1);

    }

    else

    {

        n = strlen(str);

        for (j = 0; j < n; j++)

        {

            if (str[j] == 'a' || str[j] == 'A')

                a++;

            else if (str[j] == 'e' || str[j] == 'E')

                e++;

            else if (str[j] == 'i' || str[j] == 'I')

                i++;

            else if (str[j] == 'o' || str[j] == 'O')

                o++;

            else if (str[j] == 'u' || str[j] == 'U')

                u++;

        }

        printf("Vowels Counting\n");

        printf("--------------\n");

        printf("Number of A: %d\n", a);

        printf("Number of E: %d\n", e);

        printf("Number of I: %d\n", i);

        printf("Number of O: %d\n", o);

        printf("Number of U: %d\n", u);

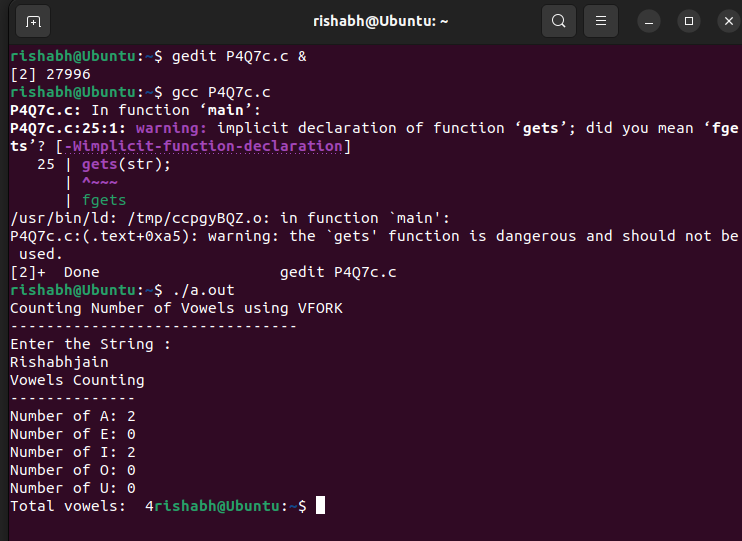
        printf("Total vowels: % d", a + e + i + o + u);

        exit(1);

    }

}

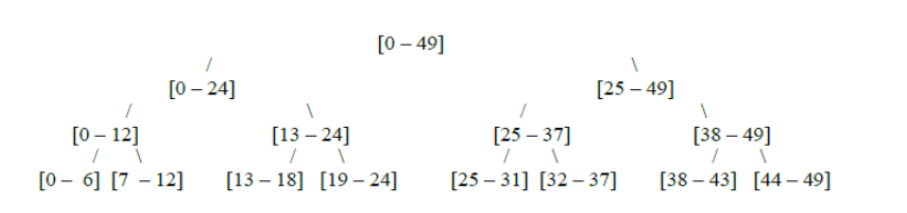
OUTPUT:



**Additional Question:**  
 In this, you work with the fork(), wait() and the exec\*() family of functions in order to find the maximum in an array of integers.   
**Part 1**Write a C program parmax.c that creates a tree of processes in order to recursively compute the maximum in an array of integers. The process at the root of the tree reads the count n of integers in the array. An array A of size n is then populated with randomly generated integers of small values (in the range 0–127). The initially unsorted array is printed by the root process.

Any process in the tree handles a chunk of the array A. The chunk is delimited by two indices L and R. For the root process, L = 0 and R = n – 1. Any process P in the tree (including the root) first counts the number of integers in the chunk it has got. If that count is less than 10, the process P itself computes the maximum element in its chunk, prints it, and exits. If the chunk size of P is 10 or more, then P creates two child processes PL and PR which handle the chunks [L, M] and [M + 1, R] in A respectively, where M = (L + R) / 2. P waits until the two child processes PL and PR exit. It then computes the maximum of the two maximum values computed by PL and PR, prints this maximum, and exits. Every non-root process returns to its parent (via the exit status) the maximum value for its chunk. During the printing of the maximum computed by a process P, the PID and the parent PID of P are also printed.

For n = 50, the ranges of the chunks handled by different processes in the tree are shown below.



It is expected that your code will handle values of n in the range 50 – 100. Compile your code, and generate an executable file with the name parmax.  
**Part 2**   
Write a separate C code wrapper.c to run the executable parmax created in Part 1. When parmax exits, your wrapper function should also exit. Submit the two C source files parmax.c and wrapper.c.

**Code:**

Parmax.c  
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

int findmax(int \*A, int L, int R)

{

    int max = A[L];

    for (int i = L + 1; i <= R; i++)

    {

        if (A[i] > max)

        {

            max = A[i];

        }

    }

    return max;

}

void process(int \*A, int L, int R)

{

    if (R - L + 1 < 10)

    {

        int max = findmax(A, L, R);

        printf("Process Id : %d (Parent Process Id : %d) - Max: %d\n",

               getpid(), getppid(), max);

        exit(max);

    }

    else

    {

        int M = (L + R) / 2;

        int left\_child, right\_child;

        int left\_status, right\_status;

        left\_child = fork();

        if (left\_child == 0)

        {

            process(A, L, M);

        }

        else

        {

            right\_child = fork();

            if (right\_child == 0)

            {

                process(A, M + 1, R);

            }

            else

            {

                waitpid(left\_child, &left\_status, 0);

                waitpid(right\_child, &right\_status, 0);

                int max\_l = WEXITSTATUS(left\_status);

                int max\_r = WEXITSTATUS(right\_status);

                int max;

                if (max\_l > max\_r)

                    max = max\_l;

                else

printf("Process Id : %d (Parent Process Id : %d) - Max:

%d\n", getpid(), getppid(), max);

exit(max);

            }

        }

    }

}

void main()

{

    int n = 50;

    int A[n];

    printf("Array is : \n");

    for (int i = 0; i < n; i++)

    {

        A[i] = rand() % 128;

        printf("%d ", A[i]);

    }

    printf("\n");

    process(A, 0, n - 1);

}

**Wrapper.c**  
#include <stdlib.h>

#include <stdio.h>

int main()

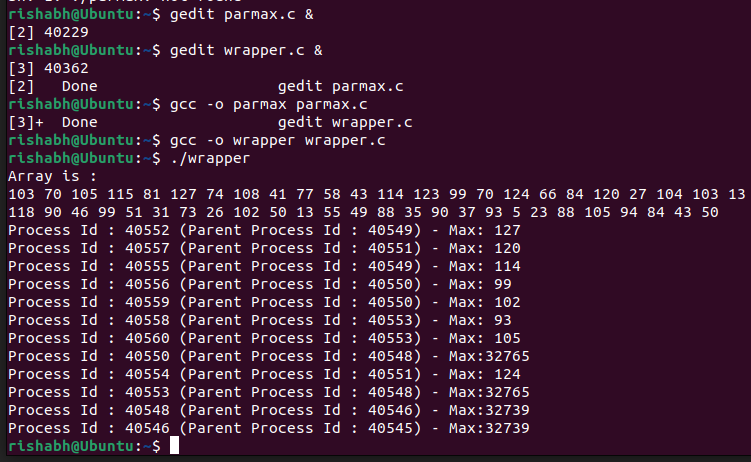
{

    system("./parmax");

    return 0;

}

**OUTPUT:**

**  
  
Result:**   
Process control system calls has been studied and Linux C programs on them has been implemented.