**1. Program with a processes P and resources R. Enter the MAX matrix, Allocation matrix for processes.**

#include <stdio.h> int main()

{

int n;

printf("Enter t al number of processes: "); scanf("%d", &n);

int bt[n], wt[n], tat[n];

printf("Enter Process Burst Time:\n");

for (int i = 0; i < n; i++)

{

printf("P[%d]:", i + 1);

scanf("%d", &bt[i]);

}

wt[0] = 0;

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

{

wt[i] = 0;

for (int j = 0; j < i; j++) wt[i] += bt[j];

} printf("nProcess\tBurst Time\tWaiting Time\tTurnaround Time"); for (int i = 0; i < n; i++) {

tat[i] = bt[i] + wt[i];

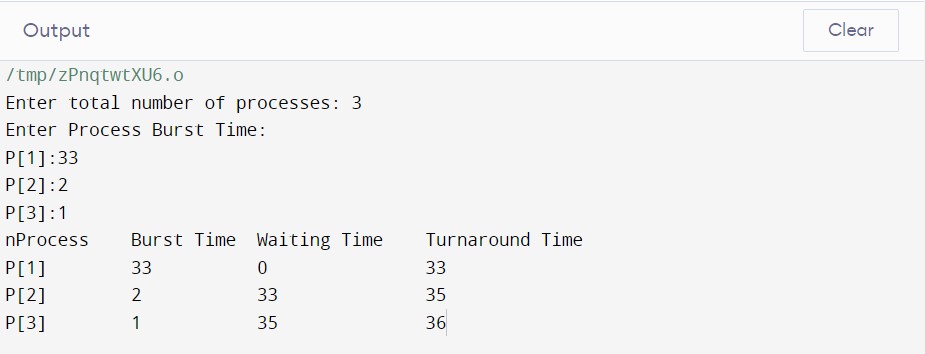
printf("\nP[%d]\t%d\t%d\t%d", i + 1, bt[i], wt[i], tat[i]);

}

return 0;

}

**Output:**



**2.Write a shell program to check if the given number is even or odd.**

echo "Enter a Number" read num e=`expr $num % 2` case $e in 0) echo "EVEN";; 1) echo "ODD";; esac

output:



**3. Program to demonstrate FORK() System Call**

1. **Single Fork:**

#include<stdio.h> #include<unistd.h>

int main()

{

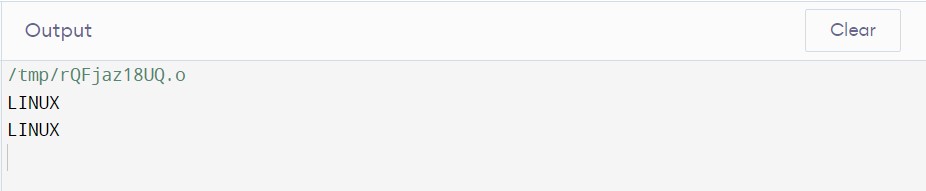
fork();

printf("LINUX\n");

return 0;

}

**Output:**



1. **Multiple fork() :**

#include<stdio.h> #include<unistd.h> int main(){

fork(); //CALLING FORK TO CREATE A CHILD PROC S printf("LINUX\n");

fork(); //CALLING FORK TO CREATE A CHILD PROC S printf("UNIX\n");

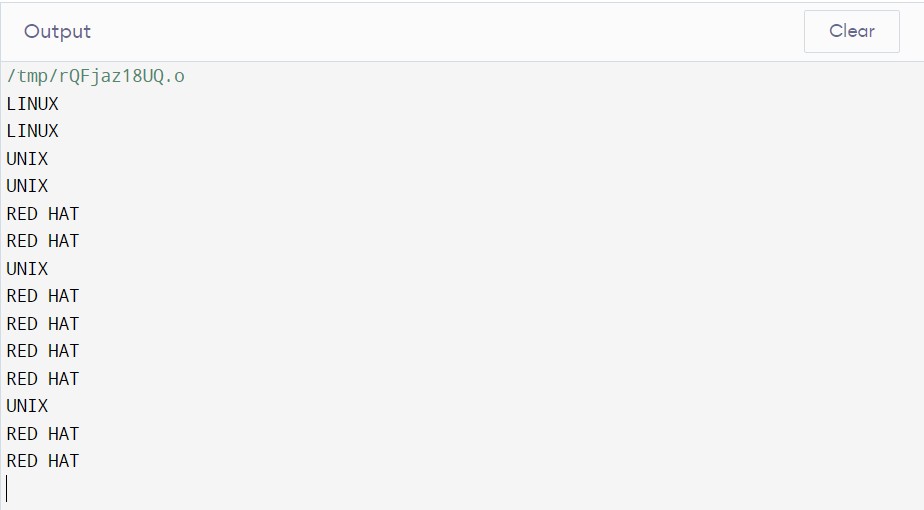
fork(); //CALLING FORK TO CREATE A CHILD PROC S

printf("RED HAT\n");

return 0;

}

**Output:**



**4. Program to demonstrate Parent Process that computes the SUM OF EVEN and Child Process that computes the sum of ODD NUMBERS using fork().**

#include <iostream> #include <unistd.h> using namespace std;

int main()

{

int a[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }; int sumOdd = 0, sumEven = 0, n, i;

n = fork(); if (n > 0) { for (i = 0; i < 10; i++) { if (a[i] % 2 == 0) sumEven = sumEven + a[i];

}

cout << "Parent process: ";

cout << "Sum of even no. is " << sumEven << endl;

}

else { for (i = 0; i < 10; i++) { if (a[i] % 2 != 0) sumOdd = sumOdd + a[i];

}

cout << "Child process: ";

cout << "Sum of odd no. is " << sumOdd << endl;

}

return 0;

}

**Output:**



**5. Program to demonstrate WAIT () System Call.**

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

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

int main()

{

pid\_t pid; pid=fork(); if(pid==0){ printf("I m Child\n");

exit(0);

}

else{ wait(NULL); printf("I m Parent\n");

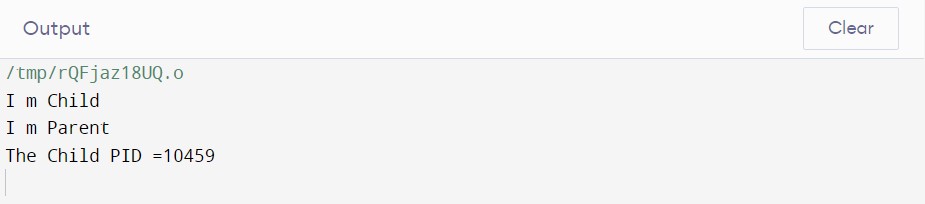
printf("The Child PID =%d\n", pid);

}

return 0;

}

**Output:**



**6. Program to implement ORPHAN PROC S & ZOMBIE PROC S:**

**(i) Orphan Process:**

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

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

int main()

{

pid\_t pid;

pid=fork(); //CALLING FORK TO CREATE A CHILD PROC S if(pid==0)

{

sleep(6);

printf("\n I m Child. My PID = %d And PPID = %d",getpid(),getppid());

}

else

{

printf("I m Parent. My Child PID = %d And my PID =%d",pid,getpid());

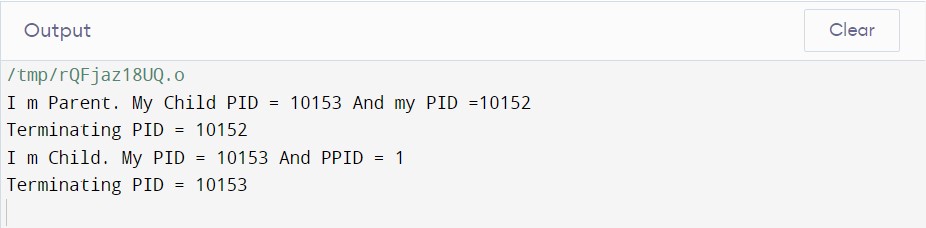
}

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

return 0;

}

**Output:**



**(ii) Zombie Process:**

#include<stdio.h>

#include <sys/wait.h> #include <unistd.h> int main()

{

int pid=fork();

if (pid==0)

{

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

}

else if(pid>0)

{

sleep(20);

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

}

else

printf("Process N Created");

return 0;

}

**Output:**

