



# CSN-361

Assignment 2

25.07.2019

VISHAL GARG

17114076

## Problem Statement 2:

Write a C program to demonstrate both Zombie and Orphan process..

```
//C program to demonstrate orphan process
//17114076
//Vishal Garg
#include <bits/stdc++.h>
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main() {
    if (fork() != 0) { //call fork to get a child
        sleep(2);
        printf("PARENT [%d] is going to die[%d]\n", getpid(), getppid()); //terminates before
        child rendering it orphan
    }
    else {
        sleep(1);
        printf("CHILD [%d] having original parent, [%d]\n", getpid(), getppid());
        sleep(3); //orphan process gets adopted by init/systemd process
        printf("CHILD [%d] having original parent, [%d]\n", getpid(), getppid());
    }
    return 0;
}
```

```
//C program to demonstrate zombie process
//17114076
//Vishal Garg
#include <bits/stdc++.h>
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main() {
    if (fork() != 0) { //fork to produce a child
        sleep(10); //parent is sleeping hence wont call wait() for child
        printf("PARENT [%d] is terminating now\n", getpid());
    } else {
        sleep(1);
        printf("CHILD [%d] is going to exit but parent [%d] won't wait()
it\n", getpid(), getppid());
        exit(0); //child has sent exit but signal hasnt be recieved by parent thus it stays
in process table
    }
    return 0;
}
```

## Data Structure And algorithm

No complex algorithm has been used.

The screenshot shows a Linux desktop environment with the following components:

- Visual Studio Code:** A C++ program is open, showing a `main()` function that forks a child process. The code is as follows:
 

```
1 //C program to demonstrate orphan process
2 //17114076
3 //Vishal Garg
4 #include <bits/stdc++.h>
5 #include <stdio.h>
6 #include <sys/types.h>
7 #include <unistd.h>
8
9 int main()
10 {
11     if (fork() != 0) { //call fork to get a child
```
- System Monitor:** A window showing a list of processes. The 'systemd' process is highlighted, showing it is owned by 'vader' and has a PID of 1809. The table below shows the processes listed in the window:
 

Process Name	User	Priority	% CPU	ID	Memory	Disk read
systemd	root	Normal	0	1	2.7 MiB	
systemd-journald	root	Normal	0	309	1.4 MiB	
systemd-udev	root	Normal	0	332	3.6 MiB	
systemd-resolved	systemd-resc	Normal	0	1058	1.7 MiB	
systemd-timesyncd	systemd-time	Normal	0	1060	556.0 KiB	
dbus-daemon	messagebus	Normal	0	1065	2.6 MiB	
systemd-logind	root	Normal	0	1096	840.0 KiB	
systemd	gdm	Normal	0	1252	1.6 MiB	
dbus-daemon	gdm	Normal	0	1275	796.0 KiB	
systemd	vader	Normal	0	1809	1.8 MiB	
dbus-daemon	vader	Normal	0	1842	5.0 MiB	
gnome-system-monitor	vader	Normal	3	12987	13.9 MiB	26.0
- Terminal:** A terminal window showing the execution of the C++ program. The output is as follows:
 

```
root@vader-Inspiron-5567:~/home/vader/Desktop/c++$ sudo -l
root@vader-Inspiron-5567:~# ./a.out
-bash: ./a.out: No such file or directory
root@vader-Inspiron-5567:~# vim

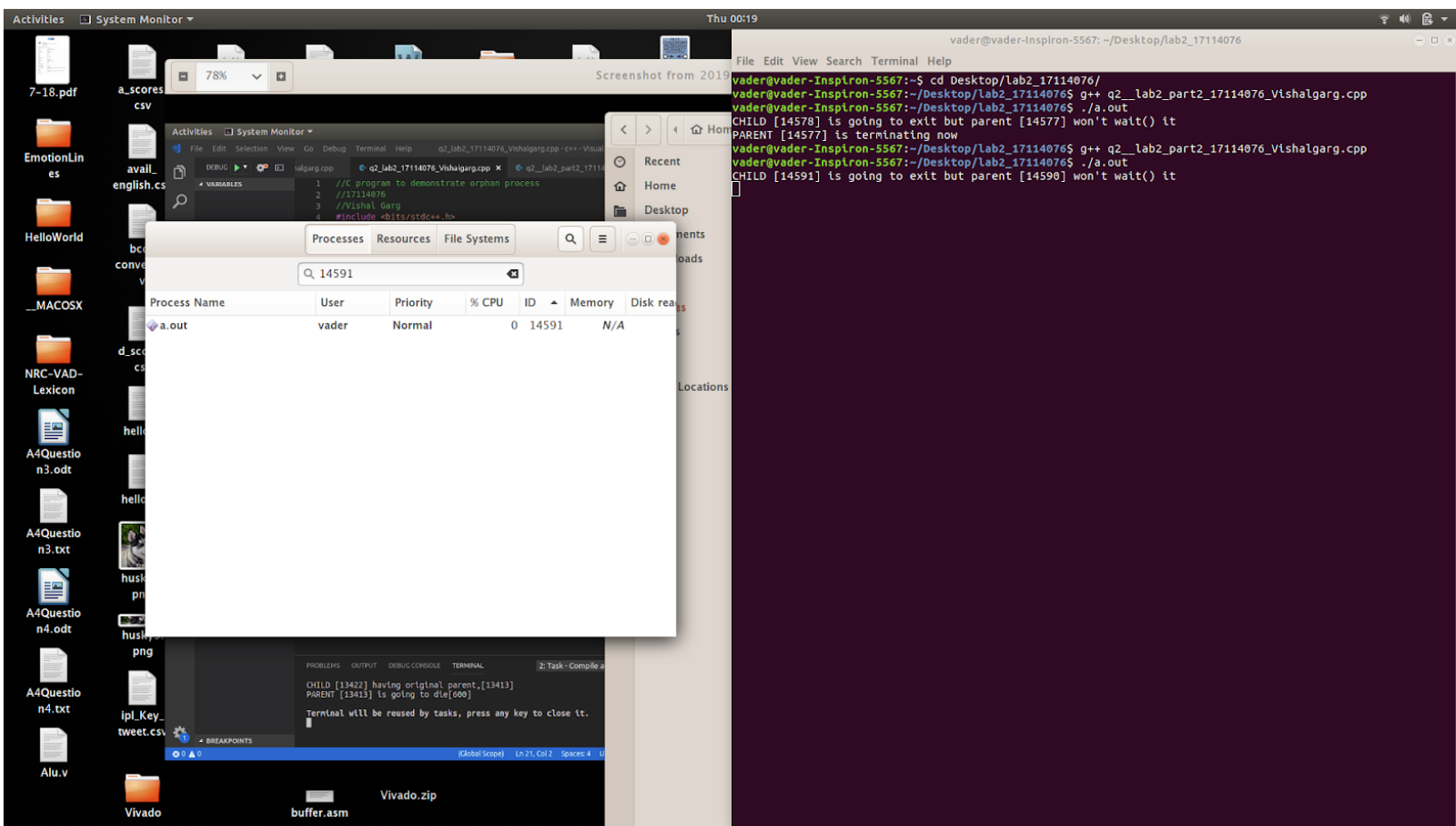
Command 'vim' not found, but can be installed with:

apt install vim
apt install vim-gtk3
apt install vim-tiny
apt install neovim
apt install vim-athena
apt install vim-gtk
apt install vim-nox

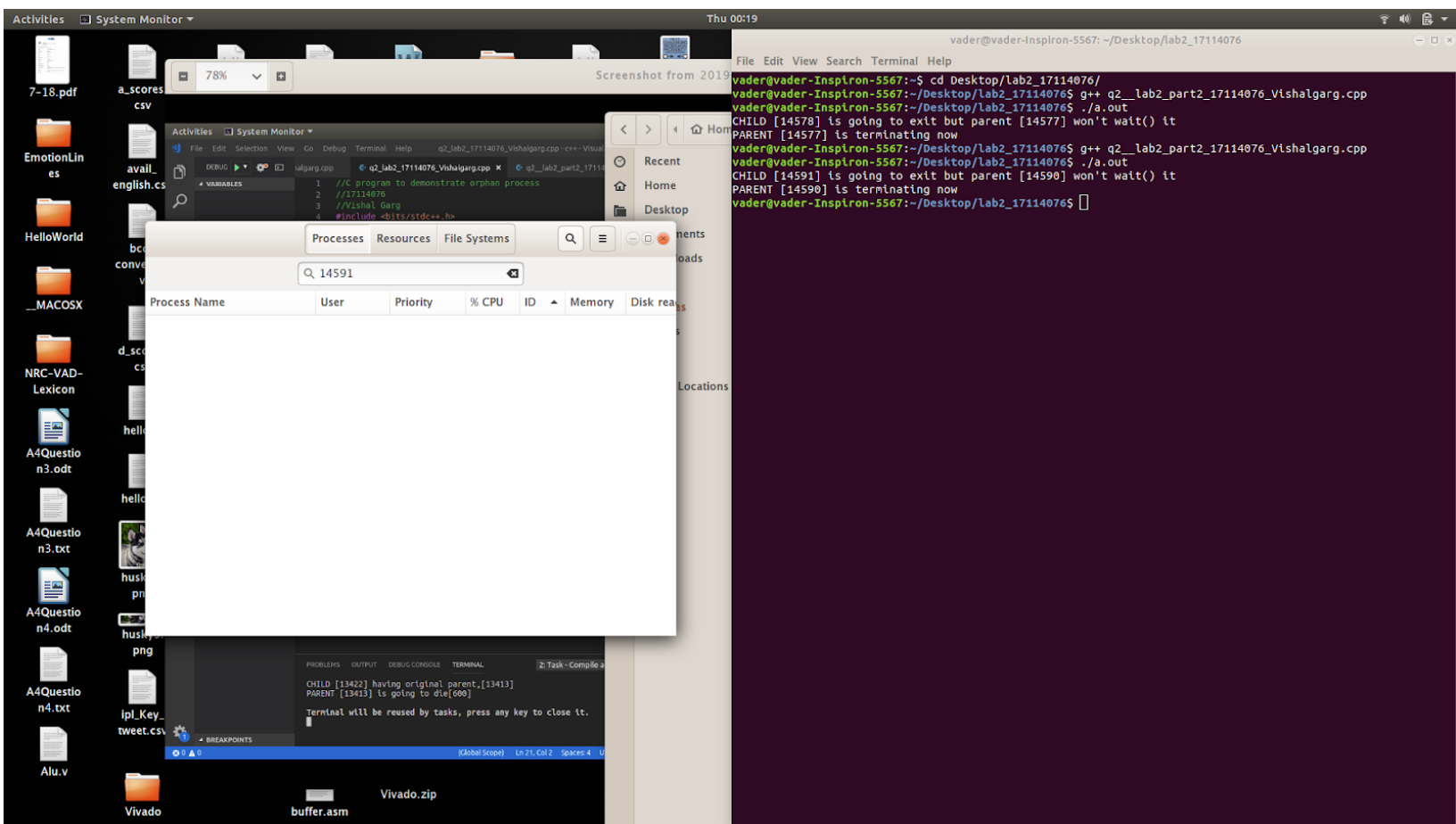
root@vader-Inspiron-5567:~# vi
root@vader-Inspiron-5567:~# cd /home/vader/Desktop/c++/
root@vader-Inspiron-5567:~/home/vader/Desktop/c++# ./a.out
CHILD [13422] having original parent,[13413]
PARENT [13413] is going to die[600]
Terminal will be reused by tasks, press any key to close it.

root@vader-Inspiron-5567:~/home/vader/Desktop/c++# CHILD [14072] having original
parent,[1809]
```

As you can see the child process has been adopted by systemd (init in case of linux-vader)



Child has completed, but is waiting on parent process to wait(), thus is present in process table as a zombie.



## Problem Statement 1:

Write a socket program in C to connect two nodes on a network to communicate with each other, where one socket listens on a particular port at an IP, while other socket reaches out to the other to form a connection.

Node1-acting as a server

```
//17114076
//Vishal Garg

#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>

#define MAX 80 //define msg/buffer size
#define PORT 8080 //just a porrt no. rest its your choice choose whatever you want to
#define SAi struct sockaddr_in
#define blog 5

int main(int argc, char const *argv[])
{
    int node1_fd, new_socket, msg;
    SAi address;
    int opt = 1;
    int addrlen=sizeof(address);
    char buffer[MAX]={0};
    char *hola="Hola by node1";

    //Creates socket file descriptor
    if((node1_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) //creates socket AF_NET signifies
    IPV4,STREAM refers to TCP, and ) for IP
```

```
{
    perror("unable to create socket");
    exit(EXIT_FAILURE);
}

address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons( PORT );

if (bind(node1_fd, (struct sockaddr *)&address, sizeof(address))<0) //binds to the address and
port number specified
{
    perror("binding failure");
    exit(EXIT_FAILURE);
}

if (listen(node1_fd, blog) < 0) //makes node1 to listen
{
    perror("listen");
    exit(EXIT_FAILURE);
}

if ((new_socket = accept(node1_fd, (struct sockaddr *)&address, //extracts first connection
req on the queue of pending connections and creates a new connected socket
                        (socklen_t*)&addrlen))<0)
{
    perror("accept");
    exit(EXIT_FAILURE);
}

msg = read( new_socket , buffer, MAX);

printf("%s\n",buffer );

send(new_socket , hola , strlen(hola) , 0 );
```



```
printf("Hello message sent\n");  
return 0;
```

```
}
```

```
//17114076  
//Vishal Garg  
#include <stdio.h>  
#include <sys/socket.h>  
#include <arpa/inet.h>  
#include <unistd.h>  
#include <string.h>  
#define MAX 80 //define msg/buffer size  
#define PORT 8080 //just a porrt no. rest its your choice choose whatever you want to  
#define SAi struct sockaddr_in  
  
int main(int argc, char const *argv[])  
{  
    int node2_fd, msg;  
    SAi s_address;  
    char buffer[MAX]={0};  
    char *hola="Hola by node2";  
  
    //Creates socket file descriptor  
    if((node2_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) //creates socket AF_INET signifies  
    IPV4, STREAM refers to TCP, and 0 for IP
```

```
{  
    perror("unable to create socket");  
    return -1;  
}  
  
s_address.sin_family=AF_INET;  
s_address.sin_port = htons(PORT);  
  
if(inet_pton(AF_INET, "127.0.0.1", &s_address.sin_addr)<=0)  
{  
    printf("\nInvalid address/ Address not supported \n");  
    return -1;  
}  
  
if (connect(node2_fd, (struct sockaddr *)&s_address, sizeof(s_address)) < 0)  
{  
    printf("\nConnection Failed \n");  
    return -1;  
}  
  
send(node2_fd , hola , strlen(hola) , 0 );  
printf("Hello message sent\n");  
msg = read( node2_fd , buffer, MAX);  
printf("%s\n",buffer );  
return 0;  
}
```

The image shows a Visual Studio Code editor with a C++ file named `node1.c` and a terminal window. The code defines a server and a client that communicate over a TCP socket. The terminal shows the server running, the client connecting, and the exchange of 'Hello' messages.

```

1 //17114076
2 //Vishal Garg
3 #include <unistd.h>
4 #include <stdio.h>
5 #include <sys/socket.h>
6 #include <stdlib.h>
7 #include <netinet/in.h>
8 #include <string.h>
9 #define MAX 80 //define msg/buffer size
10 #define PORT 8080 //just a port no. rest its your choice choose whatever you want
11 #define SAI struct sockaddr_in
12 #define blog 5
13 int main(int argc, char const *argv[])
14 {
15     int node1_fd, new_socket, msg;
16     SAI address;
17     int opt = 1;
18     int addrlen=sizeof(address);
19     char buffer[MAX]={0};
20     char *hola="Hola by node1";
21
22     //Creates socket file descriptor
23     if((node1_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) //creates socket
24     {
25         perror("unable to create socket");
26         exit(EXIT_FAILURE);
27     }
28     address.sin_family = AF_INET;
29     address.sin_addr.s_addr = INADDR_ANY;
30     address.sin_port = htons( PORT );
31     if (bind(node1_fd, (struct sockaddr *)&address, sizeof(address))<0) //bind
32     {
33         perror("binding failure");
34     }
35
36     listen(node1_fd, 5);
37
38     while(1)
39     {
40         msg = 0;
41         new_socket = accept(node1_fd, (SAI *)&address, (&addrlen));
42         if(new_socket == -1)
43         {
44             perror("accept failed");
45             continue;
46         }
47         read(new_socket, buffer, MAX);
48         printf("Received %s\n", buffer);
49         write(new_socket, hola, strlen(hola));
50         close(new_socket);
51     }
52
53     return 0;
54 }

```

The terminal output shows the server running and the client connecting. The client sends 'Hello message sent' and the server responds with 'Hola by node1'.

```

vader@vader-Inspiron-5567:~/Desktop/c++$ cd Desktop
vader@vader-Inspiron-5567:~/Desktop$ cd c++
vader@vader-Inspiron-5567:~/Desktop/c++$ ./server
binding failure: Address already in use
vader@vader-Inspiron-5567:~/Desktop/c++$ ./client
Hello message sent
Hola by node1
vader@vader-Inspiron-5567:~/Desktop/c++$ ./server
Connection Failed
vader@vader-Inspiron-5567:~/Desktop/c++$ ./server
Hola by node2
Hello message sent
vader@vader-Inspiron-5567:~/Desktop/c++$

```

## Data Structure And algorithm used:

Create TCP socket

using `bind()`, Bind the socket to server address

using `listen()`, put the server socket in a passive mode, where it waits for the client to approach the server to make a connection

using `accept()`, At this point, connection is established between client and server they are now ready to transfer data.