Ayush Agarwal
[17114017]

Assignment 1

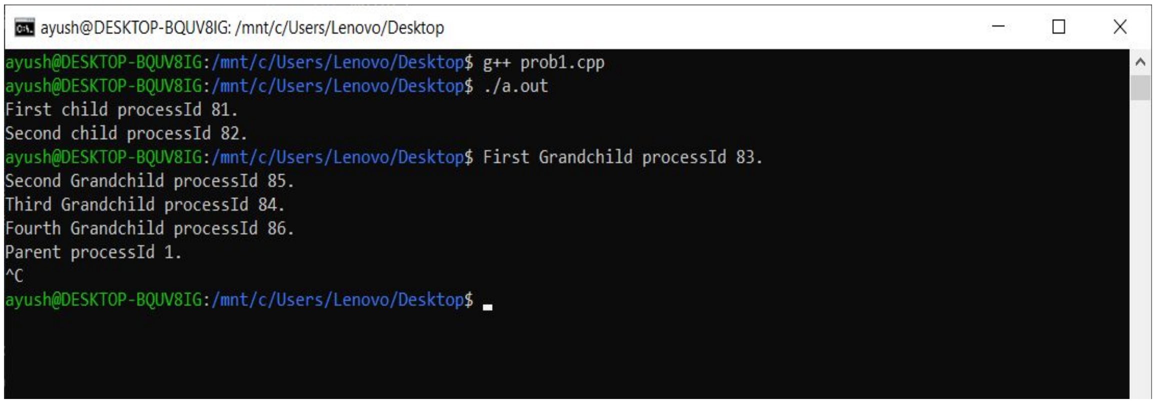
25th July, 2019

1. Write a C program in the UNIX system that creates two children and four grandchildren (two for each child). The program should then print the process-IDs of the two children, four grandchildren and the parent in this order.

   fork() is a system call function which can generate child process from parent main process. Using this command, children and their grandchildren are created.

```
int main()
{

 int processId = -1,
     processId1 = -1,
     processId2 = -1,
     processId11 = -1,
     processId12 = -1,
     processId21 = -1,
     processId22 = -1;

 processId1 = fork(); // first child
 processId2 = fork(); // second child

 if (processId1 > 0 and processId2 > 0) // parent
 {
   printf("Process Id of first child is %d.\n", processId1);
   printf("Process Id of second child is %d.\n", processId2);
 }

 else if (processId2 == 0 and processId1 != 0)
 {
   processId21 = fork();

   if (processId21 != 0) // second child
   {
     processId22 = fork();
     if (processId22 != 0) // second child
     {
       printf("Process Id of third grandchild is %d.\n", processId21);
```

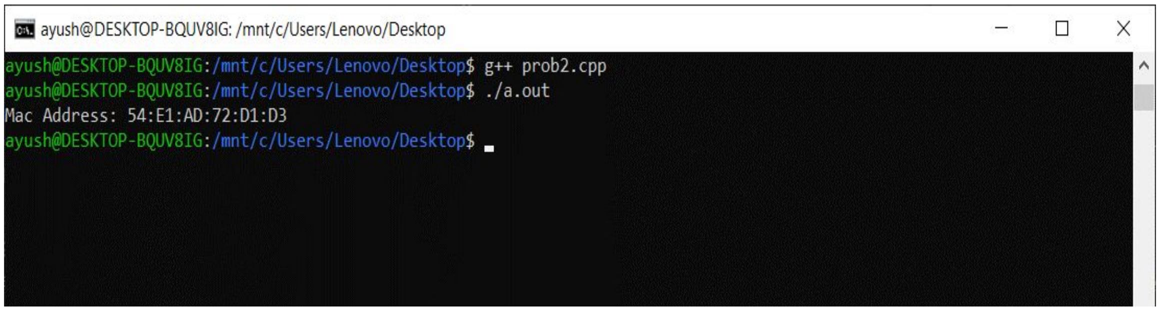
```
        printf("Process Id of fourth grandchild is %d.\n", processId22);
        printf("Process Id of parent is %d.\n", getppid());
      }
    }
  }

  else if (processId1 == 0 and processId2 > 0) // first child
  {
    processId11 = processId2;
    processId12 = fork();
    if (processId12 != 0) // first child
    {
      printf("Process Id of first grandchild is %d.\n", processId11);
      printf("Process Id of second grandchild is %d.\n", processId12);
    }
  }
}
```



```
ayush@DESKTOP-BQUV8IG: /mnt/c/Users/Lenovo/Desktop
ayush@DESKTOP-BQUV8IG:/mnt/c/Users/Lenovo/Desktop$ g++ prob1.cpp
ayush@DESKTOP-BQUV8IG:/mnt/c/Users/Lenovo/Desktop$ ./a.out
First child processId 81.
Second child processId 82.
ayush@DESKTOP-BQUV8IG:/mnt/c/Users/Lenovo/Desktop$ First Grandchild processId 83.
Second Grandchild processId 85.
Third Grandchild processId 84.
Fourth Grandchild processId 86.
Parent processId 1.
^C
ayush@DESKTOP-BQUV8IG:/mnt/c/Users/Lenovo/Desktop$
```

2. Write a C++ program to print the MAC address of your computer.

```
int main()
{
 struct ifreq s;
 int fd = socket(PF_INET, SOCK_DGRAM, IPPROTO_IP);

 strcpy(s.ifr_name, "eth0");
 if (0 == ioctl(fd, SIOCGIFHWADDR, &s))
 {
   int i;
   unsigned char *hwaddr = (unsigned char *)s.ifr_addr.sa_data;
   printf("Mac Address: %02X:%02X:%02X:%02X:%02X:%02X\n",
          hwaddr[0], hwaddr[1], hwaddr[2],
          hwaddr[3], hwaddr[4], hwaddr[5]);
   return 0;
 }

 return 1;
}
```

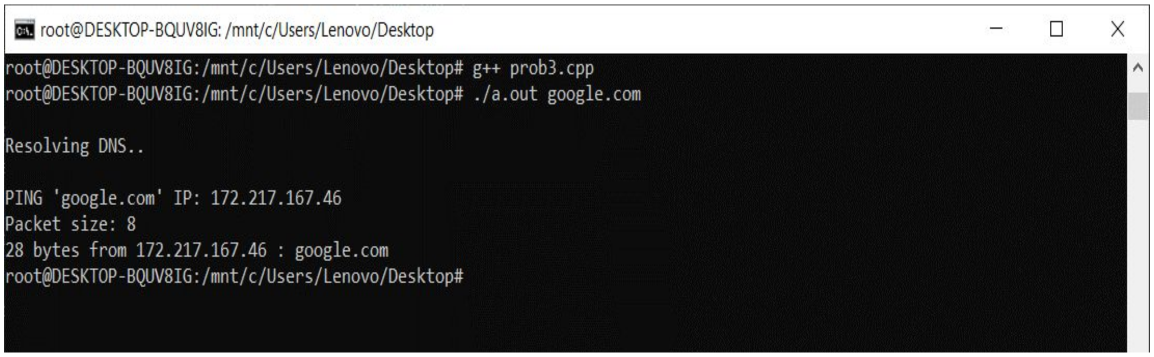


```
ayush@DESKTOP-BQUV8IG:/mnt/c/Users/Lenovo/Desktop$ g++ prob2.cpp
ayush@DESKTOP-BQUV8IG:/mnt/c/Users/Lenovo/Desktop$ ./a.out
Mac Address: 54:E1:AD:72:D1:D3
ayush@DESKTOP-BQUV8IG:/mnt/c/Users/Lenovo/Desktop$
```

3. Write your own version of ping program in C language.

```
char *dns_lookup(char *addr_host, struct sockaddr_in *addr_con)
{
   printf("\nResolving DNS..\n");
   struct hostent *host_entity;
   char *ip = (char *)malloc(NI_MAXHOST * sizeof(char));
   int i;

   if ((host_entity = gethostbyname(addr_host)) == NULL)
   {
       // No ip found for hostname
       return NULL;
   }

   //filling up address structure
   strcpy(ip, inet_ntoa(*(struct in_addr *)
                          host_entity->h_addr));

   (*addr_con).sin_family = host_entity->h_addrtype;
   (*addr_con).sin_port = htons(0);
   (*addr_con).sin_addr.s_addr = *(long *)host_entity->h_addr;

   return ip;
}

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

   int sockfd;
   char *ip_addr, *reverse_hostname;
   struct sockaddr_in addr_con;

   if (argc != 2)
   {
       printf("\nIncorrect Format %s <address>\n", argv[0]);
       return 0;
   }
```

```
ip_addr = dns_lookup(argv[1], &addr_con);
// PING google.com (172.217.167.46) 56(84) bytes of data.

printf("\nPING '%s' IP: %s\n", argv[1], ip_addr);

// 1. Creating Socket
int s = socket(PF_INET, SOCK_RAW, 1);

//  -> Exit the app if the socket failed to be created
if (s <= 0)
{
    perror("Socket Error");
    exit(0);
}

// 2. Create the ICMP Struct Header
typedef struct
{
    uint8_t type;
    uint8_t code;
    uint16_t chksum;
    uint32_t data;
} icmp_hdr_t;

//  3. Use the newly created struct to make a variable.
icmp_hdr_t pckt;

//  4. Set the appropriate values to our struct, which is our ICMP header
pckt.type = 8;          // The echo request is 8
pckt.code = 0;          // No need
pckt.chksum = 0xfff7; // Fixed checksum since the data is not changing
pckt.data = 0;          // We don't send anything.

//  5. Creating an IP Header from a struct that exists in another library
printf("Packet size: %ld\n", sizeof(pckt));

struct sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_port = 0;
addr.sin_addr.s_addr = inet_addr("ip_address");

//  6. Send our PING
int actionSendResult = sendto(s, &pckt, sizeof(pckt),
                              0, (struct sockaddr *)&addr, sizeof(addr));

//  -> Exit the app if the option failed to be set
if (actionSendResult < 0)
{
    perror("Ping Error");
    exit(0);
    }

    // 7. Prepare all the necessary variable to handle the response
    unsigned int resAddressSize;
    unsigned char res[30] = "";
    struct sockaddr resAddress;

    //  8. Read the response from the remote host
    int response = recvfrom(s, res, sizeof(res), 0, &resAddress,
                                 &resAddressSize);

    //  -> Display the response in its raw form (hex)
    if (response > 0)
    {
        // 64 bytes from del03s16-in-f14.1e100.net (172.217.167.46): icmp_seq=3
 ttl=50 time=46.1 ms
        printf("%d bytes from %s : %s\n", response, ip_addr, argv[1]);

        exit(0);
    }
    else
    {
        perror("Response Error");
        exit(0);
    }

    return 0;
}
```



```
root@DESKTOP-BQUV8IG: /mnt/c/Users/Lenovo/Desktop
root@DESKTOP-BQUV8IG:/mnt/c/Users/Lenovo/Desktop# g++ prob3.cpp
root@DESKTOP-BQUV8IG:/mnt/c/Users/Lenovo/Desktop# ./a.out google.com

Resolving DNS..

PING 'google.com' IP: 172.217.167.46
Packet size: 8
28 bytes from 172.217.167.46 : google.com
root@DESKTOP-BQUV8IG:/mnt/c/Users/Lenovo/Desktop#
```

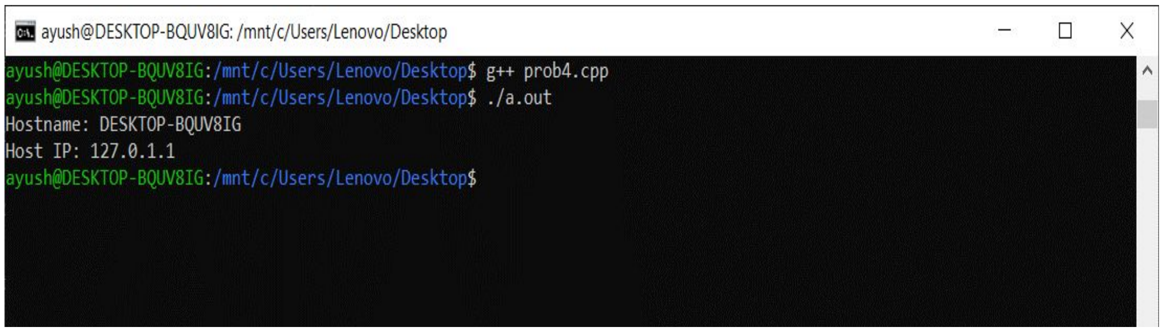
4. Write a C program to find the host name and the IP address of your computer.

   gethostname() : The gethostname function retrieves the standard host name for the local computer.

   gethostbyname() : The gethostbyname function retrieves host information corresponding to a host name from a host database.

inet_ntoa() : The inet_ntoa function converts an (Ipv4) Internet network address into an ASCII string in Internet standard dotted-decimal format.

```
int main()
{
   char hostbuffer[256];
   char *IPbuffer;
   struct hostent *host_entry;
   int hostname;

   // To retrieve hostname
   hostname = gethostname(hostbuffer, sizeof(hostbuffer));

   // To retrieve host information
   host_entry = gethostbyname(hostbuffer);

   // To convert an Internet network
   // address into ASCII string
   IPbuffer = inet_ntoa(*((struct in_addr *)
                           host_entry->h_addr_list[0]));

   printf("Hostname: %s\n", hostbuffer);
   printf("Host IP: %s\n", IPbuffer);

   return 0;
}
```


ayush@DESKTOP-BQUV8IG: /mnt/c/Users/Lenovo/Desktop

```
ayush@DESKTOP-BQUV8IG:/mnt/c/Users/Lenovo/Desktop$ g++ prob4.cpp
ayush@DESKTOP-BQUV8IG:/mnt/c/Users/Lenovo/Desktop$ ./a.out
Hostname: DESKTOP-BQUV8IG
Host IP: 127.0.1.1
ayush@DESKTOP-BQUV8IG:/mnt/c/Users/Lenovo/Desktop$
```