DOCUMENTATION

Submitted by,

Aaditya Umashankar- CB.EN.U4CSE18402

Sahasra Bommineni – CB.EN.U4CSE18414

Phanindra Reddy - CB.EN.U4CSE18433

Shri Hari Nithin K M - CB.EN.U4CSE18456

Swaran Karthikeyan - CB.EN.U4CSE18461

Vijay Swaminathan - CB.EN.U4CSE18467

Index

Introduction

DragonFly BSD is a free and open-source Unix-like operating system forked from FreeBSD 4.8. Matthew Dillon, an Amiga developer in the late 1980s and early 1990s and FreeBSD developer between 1994 and 2003, began working on DragonFly BSD in June 2003 and announced it on the FreeBSD mailing lists on 16 July 2003.

Dillon started DragonFly in the belief that the techniques adopted for threading and symmetric multiprocessing in FreeBSD 5. would lead to poor performance and maintenance problems. He sought to correct these anticipated problems within the FreeBSD project. Due to conflicts with other FreeBSD developers over the implementation of his ideas, his ability to directly change the codebase was eventually revoked. Despite this, the DragonFly BSD and FreeBSD projects still work together, sharing bug fixes, driver updates, and other improvements.

Intended as the logical continuation of the FreeBSD 4.x series, DragonFly has diverged significantly from FreeBSD, implementing lightweight kernel threads (LWKT), an in-kernel message passing system, and the HAMMER file system. Many design concepts were influenced by AmigaOS.

Features

- ***Education:*** Are you a student of computer science or a related engineering field? There is no better way of learning about operating systems, computer architecture, and networking than the hands-on, under-the-hood experience that DragonFly can provide. A number of freely available CAD, mathematical, and graphic design packages also make it highly useful to those whose primary interest in a computer is to get other work done!

- ***Research:*** With source code for the entire system available, DragonFly is an excellent platform for research in operating systems as well as other branches of computer science. DragonFly's freely available nature also makes it possible for remote groups to collaborate on ideas or shared development without having to worry about special licensing agreements or limitations on what may be discussed in open forums.

- ***Networking:*** Need a new router? A name server (DNS)? A firewall to keep people out of your internal network? DragonFly can easily turn that unused older PC sitting in the corner into an advanced router with sophisticated packet-filtering capabilities.

- ***X Window workstation:*** Unlike an X terminal, DragonFly allows many applications to be run locally if desired, thus relieving the burden on a central server. DragonFly can even boot diskless, making individual workstations even cheaper and easier to administer.

- ***Software Development:*** The basic DragonFly system comes with a full complement of development tools including the renowned GNU C/C++ compiler and debugger.

Security

The different security features built into DragonFly BSD are:

- Secure Connection Initialization
- Insecure Connection Initialization
- Generating a Single One-time Password
- Generating Multiple One-time Passwords
- Restricting Use of UNIX Passwords

Comparison with Linux

Feature	DragonFly BSD	Linux
compiler	gcc 5.4.1	gcc 4.7.2 (Debian 7), gcc 4.4.7 (Redhat 6), gcc 4.1.2 (Redhat 5), gcc 7.3.0 (Void)
Firewall	default: pf; other: ipfw2, ipf	default: iptables; other: pf
Update methods	git, cvsup, rsync, pkg	up2date, yum, apt-get, pacman, emerge, etc.
Release schedule	about twice a year (developer-driven)	Redhat: 18 month; Debian: feature-driven; Ubuntu: 6 month
Processor Architectures	AMD64	x86, AMD64, Sparc, PowerPC, etc
Update methods	git, cvsup, rsync, pkg	up2date, yum, apt-get, pacman, emerge, etc.
Release schedule	about twice a year (developer-driven)	Redhat: 18 month; Debian: feature-driven; Ubuntu: 6 month

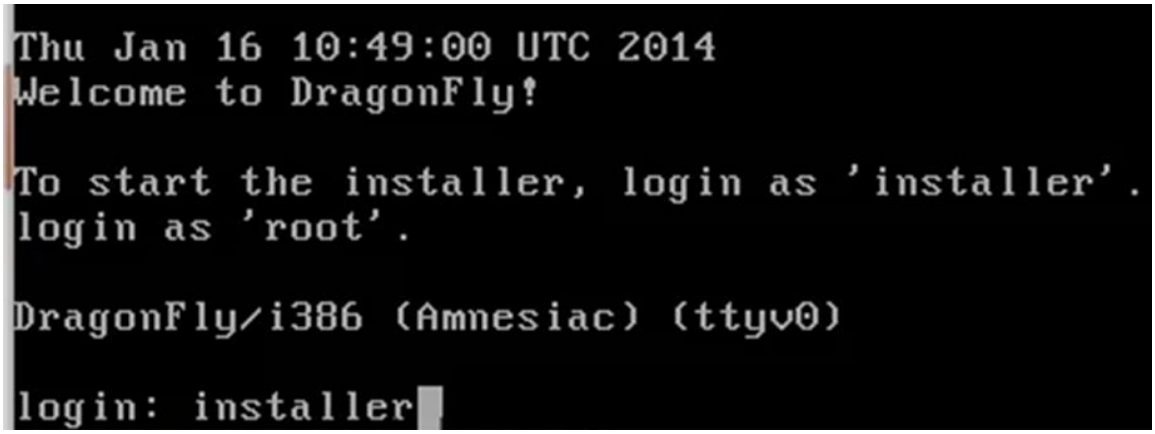
Hardware Requirements and Supported Hardware

The hardware requirements to install DragonFly BSD vary by architecture. Hardware architectures and devices supported by a DragonFly BSD release are listed on the DragonFly BSD Release Information page. The DragonFly BSD download page also has recommendations for choosing the correct image for different architectures.

A DragonFly BSD installation requires a minimum of 96 MB of RAM and 1.5 GB of free hard drive space. However, such small amounts of memory and disk space are really only suitable for custom applications like embedded appliances. General-purpose desktop systems need more resources. 2-4 GB RAM and at least 8 GB hard drive space is a good starting point.

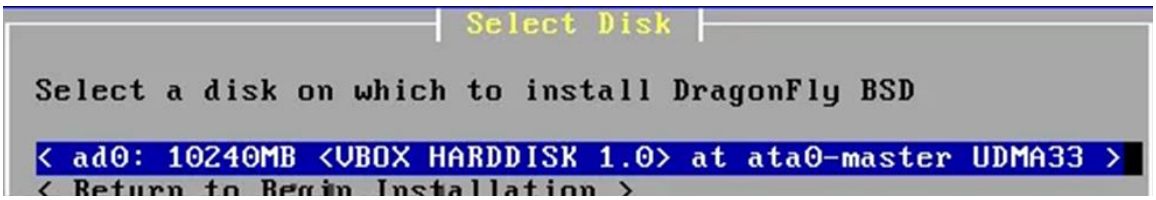
Installation Process

- Create a virtual machine in VirtualBox using the FreeBSD (64 bit) template with 4 GB memory, 128 GB dynamically allocated VDI virtual disk image
- Download and extract the x86_64 CD installation media and then add it to the virtual machine, but remember to remove it before taking VirtualBox snapshots.
- Start the virtual machine. Before it finishes booting, the will be greeted with the following boot menu.
- After the computer finishes booting, you will be greeted with the following welcome message and login prompt.
- Type 'installer' without quotes and press enter.

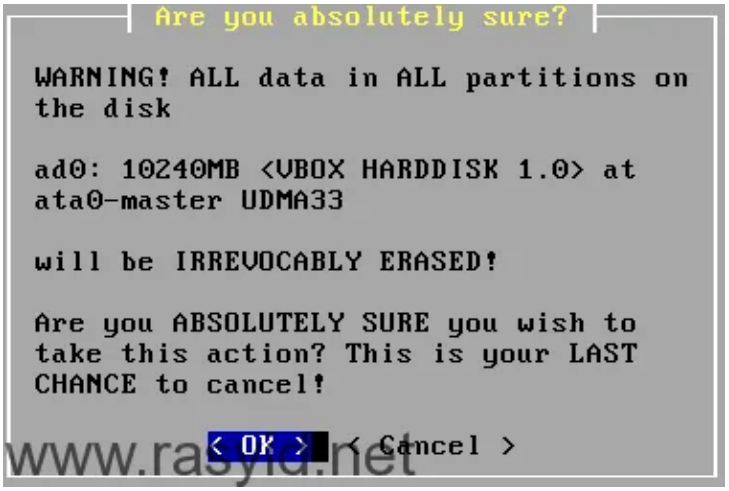


- Highlight the option < Install DragonFly BSD > and press enter.
- The following menu, specific to your hardware, will be displayed

- Highlight the disk on which to install DragonFly BSD, in this case ad0, and press Enter.



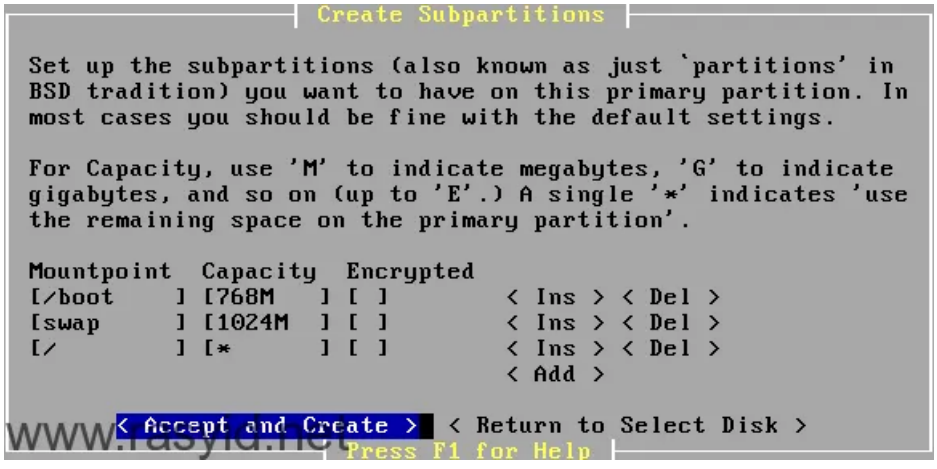
- If you would like to use your entire disk, then highlight < Use Entire Disk > and press enter.
- The following menu will be displayed.



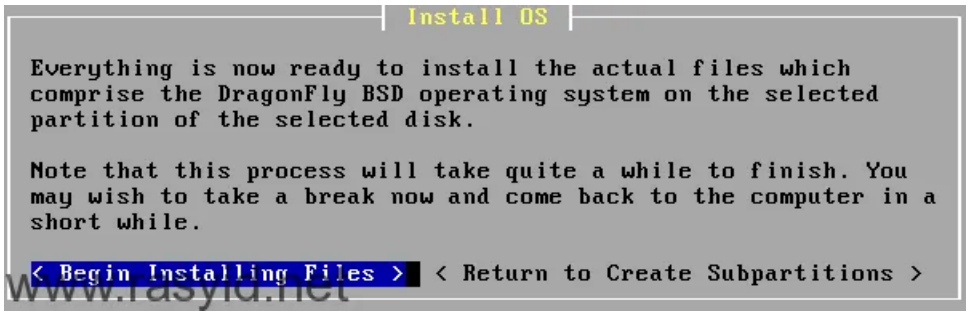
- If you are absolutely sure, highlight < OK > and press enter.
- The following menu will be displayed.

  - Press enter.
  - The following menu will be displayed

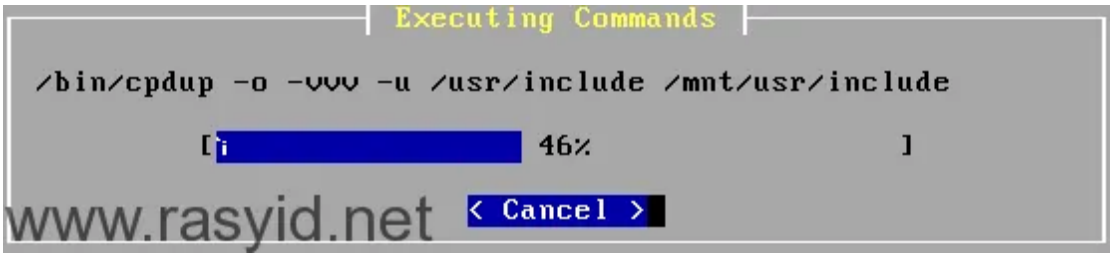
- If you want to use HAMMER, then highlight HAMMER and press enter.
- The following menu will be displayed



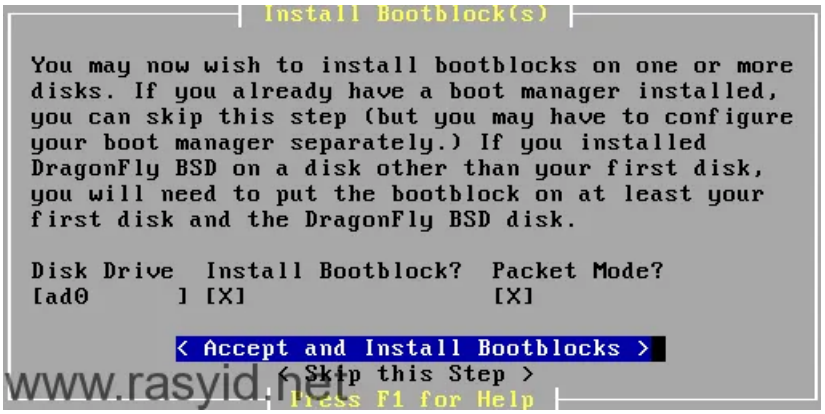
  - Highlight < Accept and Create > and press enter.
  - The following menu will be displayed



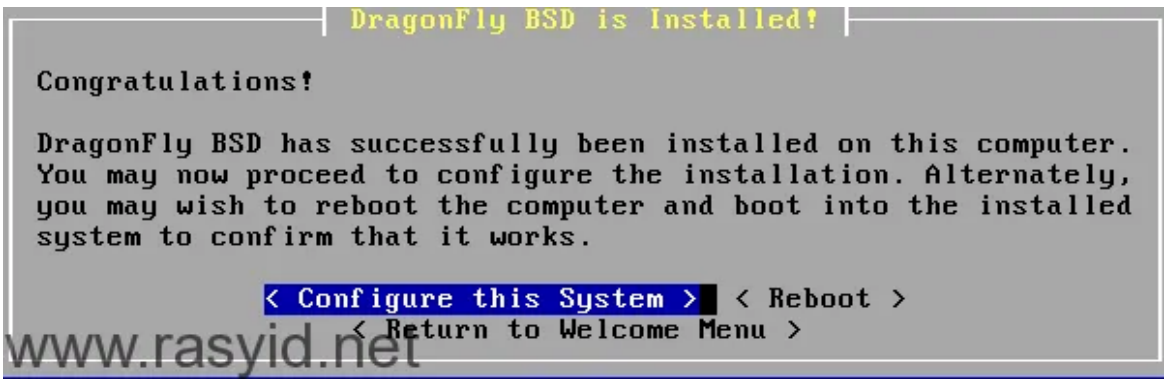
- Highlight < Begin Installing Files > and press enter.
- The following progress bar will be displayed



- When it finishes, the following menu will be displayed



- Highlight < Accept and Install Bootblocks > and press enter.
- The following dialog box will be displayed.



- The configuration settings can be selected according to the convenience of the user.

General Commands

Commands	Function
• newfs	construct a new UFS file system
• vinum	Logical Volume Manager control program
• camcontrol	CAM control program
• boot0cfg	boot manager installation/configuration utility
• ddb	interactive kernel debugger
• drm	Direct Rendering Manager (DRI kernel support)
• unset	to clear local environment variable
• gpt	GUID partition table maintenance utility
• hammer	HAMMER file system utility
• newfs_hammer	construct a new HAMMER file system

Some Outputs Of Commands

- boot0cfg

```
# boot0cfg -m 0x3 cd0
dscheck(cd0): b_bcount 512 is not on a sector boundary (ssize 2048)
```

- camcontrol

```
        camcontrol inquiry    [dev_id][generic args] [-D] [-S] [-R]
        camcontrol reportluns [dev_id][generic args] [-c] [-l] [-r report]
        camcontrol readcap    [dev_id][generic args] [-b] [-h] [-H] [-N]
                              [-q] [-s]
        camcontrol start      [dev_id][generic args]
        camcontrol stop       [dev_id][generic args]
        camcontrol load       [dev_id][generic args]
        camcontrol eject      [dev_id][generic args]
        camcontrol rescan     <all | bus[:target:lun]>
        camcontrol reset      <all | bus[:target:lun]>
        camcontrol defects    [dev_id][generic args] <-f format> [-P][-G]
        camcontrol modepage   [dev_id][generic args] <-m page | -l>
                              [-P pagectl][-e | -b][-d]
        camcontrol cmd        [dev_id][generic args] <-c cmd [args]>
                              [-i len fmt|-o len fmt [args]]
        camcontrol debug      [-I][-P][-T][-S][-X][-c]
                              <all|bus[:target[:lun]]|off>
        camcontrol tags       [dev_id][generic args] [-N tags] [-q] [-v]
        camcontrol negotiate  [dev_id][generic args] [-a][-c]
                              [-D <enable|disable>][-O offset][-q]
                              [-R syncrate][-v][-T <enable|disable>]
                              [-U][-W bus_width]
        camcontrol format     [dev_id][generic args][-q][-r][-w][-y]
        camcontrol help
#
```

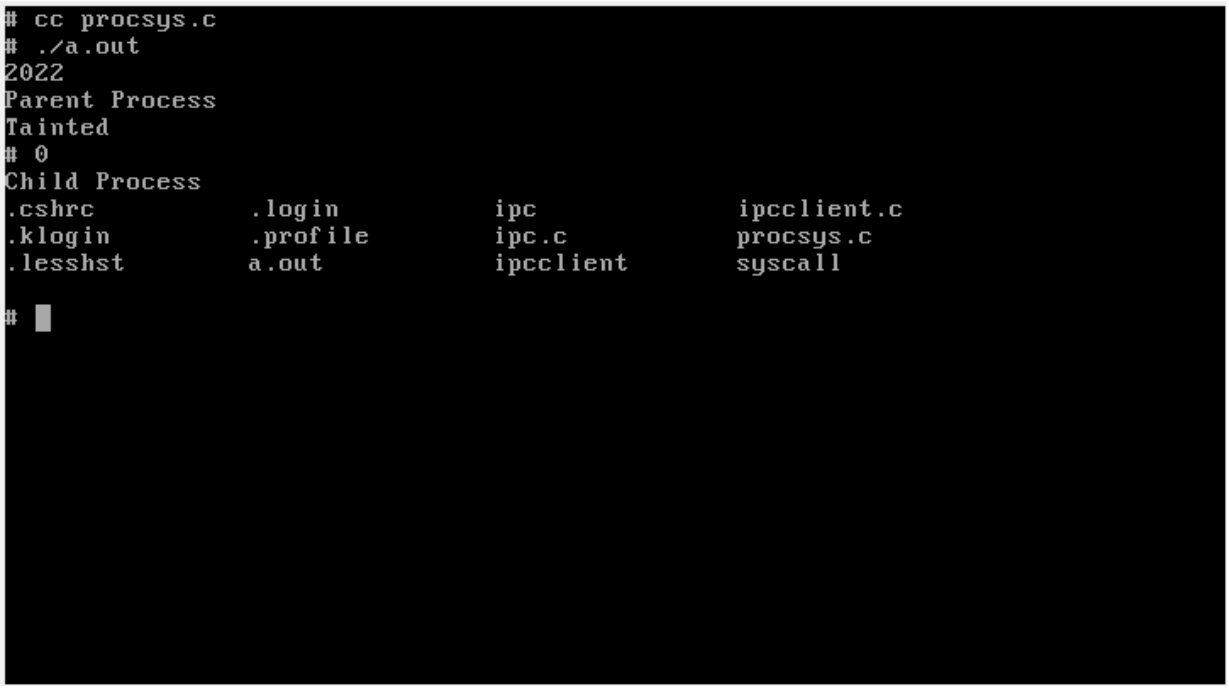
System Calls

Process system calls : rfork, execv, issetupugid

Program:

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
int main(int argc,char *argv[]){
        int pid=rfork(RFPROC);
        printf("%d",pid);
        if(pid==0){
                printf("\nParent Process\n");
                execv("/bin/ls",argv);
        }
        else
                printf("\n Parent Process");
        if(issetupugid()==0)
                printf("\nTainted\n");
        else
                printf("\nNot Tainted\n");
return 0;
}
```

Output:



Other System Calls Related To Process, File and Directory:

- procctl
- pread
- pwrite
- stat

Interprocess Communication Using Shared Memory

- Comparison Between Linux And DragonFlyBSD

Linux	Dragonfly BSD:
Shmget: Flag can have the following value: • IPC_CREAT |0666 • IPC_EXCL |0666	***Shmget***: Flag can have the following value: • IPC_CREAT |SHM_R |SHM_W|(SHM_R>>3) |(SHM_W>>3)|(SHM_R>>6)|(SHM_W>>6) • IPC_EXCL |SHM_R |SHM_W|(SHM_R>>3)|(SHM_W>>3)|(SHM_R>>6)|(SHM_W>>6)
Shmat: Flag can have the following value: • SHM_RND • SHM_EXEC • SHM_RDONLY • SHM_REMAP	***Shmat*****:** Flag can have only SHM_RND as value.

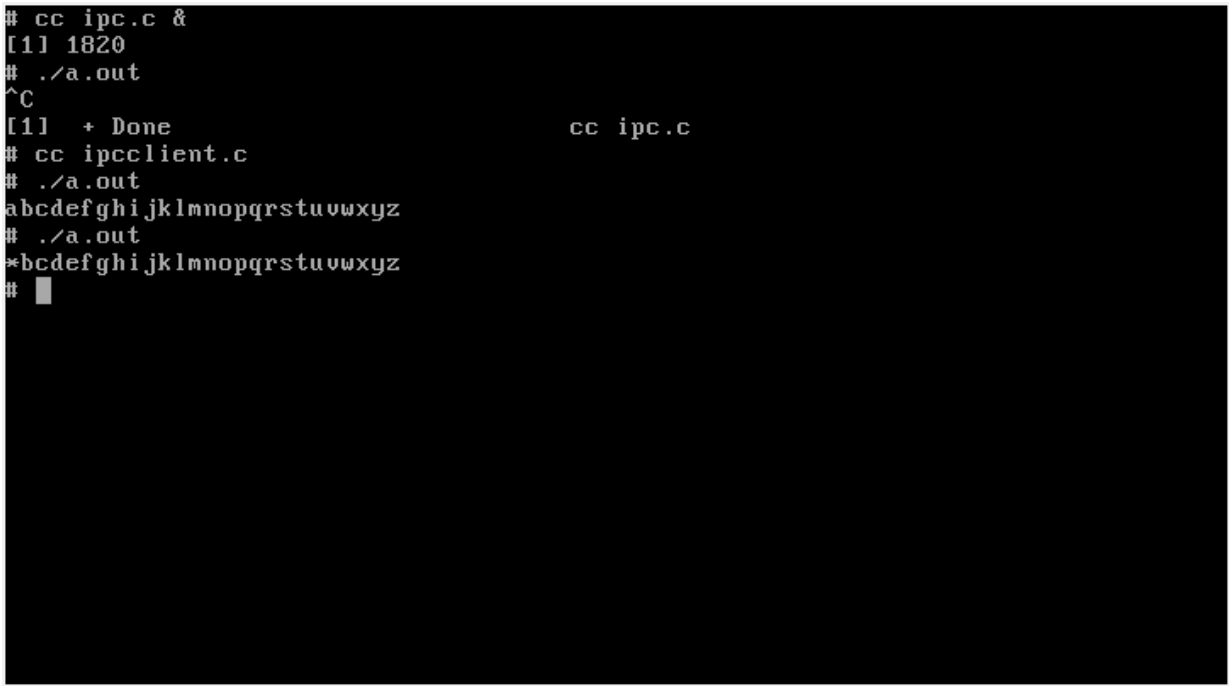
- Program for IPC using Shared Memory(Server)

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>
#define MAXSIZE     27
void die(char *s){
    perror(s);
    exit(1);
}
int main(){
    char c;
    int shmid;
    key_t key;
    char *shm, *s;
    key = 5678;
    if ((shmid = shmget(key, MAXSIZE, IPC_CREAT |SHM_R |SHM_W|(SHM_R>>3)
|(SHM_W>>3)|(SHM_R>>6)|(SHM_W>>6 )) < 0)
       die("shmget");
    if ((shm = shmat(shmid, NULL, 0)) == (char *) -1)
        die("shmat");
    s = shm;
    for (c = 'a'; c <= 'z'; c++)
        *s++ = c;
    while (*shm != '*')
        sleep(1);
    exit(0);
}
```

- Program for IPC using Shared Memory(Client)

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>
#define MAXSIZE     27
void die(char *s){
    perror(s);
    exit(1);
}
int main(){
    int shmid;
    key_t key;
    char *shm, *s;
    key = 5678;
    if((shmid = shmget(key, MAXSIZE,SHM_R |SHM_W|(SHM_R>>3)
|(SHM_W>>3)|(SHM_R>>6)|(SHM_W>>6 ) )) < 0)
       die("shmget");
if ((shm = shmat(shmid, NULL, 0)) == (char *) -1)
        die("shmat");
for (s = shm; *s != '\0'; s++)
        putchar(*s);
 putchar('\n');
    *shm = '*';
exit(0);
}
```

- Output:


cc ipc.c &
[1] 1820
./a.out
^C
[1] + Done   cc ipc.c
cc ipcclient.c
./a.out
abcdefghijklmnopqrstuvwxyz
./a.out
*bcdefghijklmnopqrstuvwxyz
#