CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY

CHANDUBHAI S. PATEL INSTITUTE OF TECHNOLOGY

U. & P. U. Patel Department of Computer Engineering

Subject Code: CE347 **Semester: 6th CE**

Subject Name: Internals of Operating System **AY: 2019-2020**

Name: Varun Kamal Ladha **ID: 17CE049**

INDEX

	[C]		Write a C program using execvp() system call which will count the characters from file 'wc', using program 'p.c'.	16		
4.			Write a program uses dup() and dup2() system call that will prove the following sentence: "*dup() always uses the smallest available (unused) file descriptor whereas dup2() uses new file descriptor.*"	20		
5.			Write the following programs using inter process communication – shared memory. The program 'writer.c' will print 1 to 100 in shared memory region. Another program 'reader.c' that will read all the numbers from shared memory to make addition of it and display it.	24		
6.			Consider a process executing on a CPU. Give an example scenario that can cause the process to undergo: (a) A voluntary context switch. (b) An involuntary context switch Write the program for both the cases.	28		
7.			Write a program to demonstrate the handling of the signals: SIGINT,SIGALRM & SIGQUIT.	31		
8.	[A]		Create 1GB swap area in your linux partition and free it. Check the allocation of swap space. Execute following commands to monitor swap space in linux. a. Swapon b. use of /proc/swaps c. free d. top e. atop f. htop g. glances h. vmstat	34		
	[B]		Write the simulation Paging Algorithms program for demand paging and show the page scheduling and total number of page faults according to FIFO, LRU, and optimal page replacement algorithm. Assume the memory of 'n' frames.	39		

9.			Assume that processes communicate by send/receive messages, which are unreliable, e.g. messages may be lost during send/recv. After sending a message, a process expects a reply. If it does not receive a reply within ‘t’ seconds, it re-sends the same message again. If it receives a reply within ‘t’ seconds, it must not send the same message again. Design an algorithm for the sending process and implement it.	44		
10.			Implementation of a device driver to find reverse string in kernel mode.	46		

Practical-1

Aim: Collect the following basic information about your machine using proc.

a. How many CPU cores does the machine have?

CPU cores parameter in cat cpuinfo displays number of cores.

```
user@user-VirtualBox:/proc$ cat cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 94
model name      : Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz
stepping        : 3
cpu MHz         : 3191.998
cache size      : 6144 KB
physical id     : 0
siblings        : 1
core id         : 0
cpu cores       : 1
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 22
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxs
r sse sse2 ht syscall nx rdtscp lm constant_tsc rep_good nopl xtopology nonstop_tsc cpuid pni pclmulqdq m
onitor ssse3 cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt aes xsave avx rdrand hypervisor lahf_lm abm 3dno
wprefetch invpcid_single pti fsgsbase avx2 invpcid rdseed clflushopt flush_l1d
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf
bogomips        : 6383.99
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:

user@user-VirtualBox:/proc$ cat /proc/cpuinfo | grep processor | wc -l
1
user@user-VirtualBox:/proc$
```

b. How much memory, and what fraction of it is free?

Memtotal and Memfree parameters in cat meminfo displays total and available free memory.

```
user@user-VirtualBox:/proc$ cat meminfo
MemTotal:        4973600 kB
MemFree:         2557496 kB
MemAvailable:    3533624 kB
Buffers:           68808 kB
Cached:          1113220 kB
SwapCached:            0 kB
Active:          1487648 kB
Inactive:         740284 kB
Active(anon):    1047188 kB
Inactive(anon):    34532 kB
Active(file):     440460 kB
Inactive(file):   705752 kB
Unevictable:          32 kB
Mlocked:              32 kB
SwapTotal:        999420 kB
SwapFree:         999420 kB
Dirty:             24924 kB
Writeback:             0 kB
AnonPages:       1045948 kB
Mapped:           364232 kB
Shmem:             35804 kB
Slab:              99588 kB
SReclaimable:      73500 kB
SUnreclaim:        26088 kB
KernelStack:        7696 kB
PageTables:        32784 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:     3486220 kB
Committed_AS:    4189744 kB
VmallocTotal:   34359738367 kB
VmallocUsed:           0 kB
```

c. How many context switches has the system performed since bootup?

Ctxt parameter in cat stat displays number of context switches.

```
user@user-VirtualBox:/proc$ cat stat
cpu  3856 19980 2174 2299 2778 0 92 0 0 0
cpu0 3856 19980 2174 2299 2778 0 92 0 0 0
intr 170676 31 950 0 0 0 0 0 0 0 0 0 220 0 0 406 0 0 0 10548 1521 22001 32 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
ctxt 654769
btime 1575359901
processes 2465
procs_running 3
procs_blocked 0
softirq 154941 3 71065 64 10984 20906 0 444 0 0 51475
user@user-VirtualBox:/proc$ man 5 proc
user@user-VirtualBox:/proc$
user@user-VirtualBox:/proc$ vmstat -f
         2483 forks
user@user-VirtualBox:/proc$
```

d. How many processes has it forked since bootup?

Vmstat -f displays number of processes forked since bootup.

```
user@user-VirtualBox:/proc$ cat stat
cpu  3856 19980 2174 2299 2778 0 92 0 0 0
cpu0 3856 19980 2174 2299 2778 0 92 0 0 0
intr 170676 31 950 0 0 0 0 0 0 0 0 0 0 220 0 0 406 0 0 0 10548 1521 22001 32 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
ctxt 654769
btime 1575359901
processes 2465
procs_running 3
procs_blocked 0
softirq 154941 3 71065 64 10984 20906 0 444 0 0 51475
user@user-VirtualBox:/proc$ man 5 proc
user@user-VirtualBox:/proc$
user@user-VirtualBox:/proc$ vmstat -f
         2483 forks
user@user-VirtualBox:/proc$
```

e. How many processors does your machine have?

Processor parameter in cat /proc/cpuinfo | grep processor | wc -l displays number of processors.

```
user@user-VirtualBox:/proc$ cat cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 94
model name      : Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz
stepping        : 3
cpu MHz         : 3191.998
cache size      : 6144 KB
physical id     : 0
siblings        : 1
core id         : 0
cpu cores       : 1
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 22
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxs
r sse sse2 ht syscall nx rdtscp lm constant_tsc rep_good nopl xtopology nonstop_tsc cpuid pni pclmulqdq m
onitor ssse3 cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt aes xsave avx rdrand hypervisor lahf_lm abm 3dno
wprefetch invpcid_single pti fsgsbase avx2 invpcid rdseed clflushopt flush_l1d
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf
bogomips        : 6383.99
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:

user@user-VirtualBox:/proc$ cat /proc/cpuinfo | grep processor | wc -l
1
user@user-VirtualBox:/proc$
```

f. What is the frequency of each processor?

Cpu Mhz parameter in cat cpuinfo displays processor frequency.

```
user@user-VirtualBox:/proc$ cat cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 94
model name      : Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz
stepping        : 3
cpu MHz         : 3191.998
cache size      : 6144 KB
physical id     : 0
siblings        : 1
core id         : 0
cpu cores       : 1
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 22
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxs
r sse sse2 ht syscall nx rdtscp lm constant_tsc rep_good nopl xtopology nonstop_tsc cpuid pni pclmulqdq m
onitor ssse3 cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt aes xsave avx rdrand hypervisor lahf_lm abm 3dno
wprefetch invpcid_single pti fsgsbase avx2 invpcid rdseed clflushopt flush_l1d
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf
bogomips        : 6383.99
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:

user@user-VirtualBox:/proc$ cat /proc/cpuinfo | grep processor | wc -l
1
user@user-VirtualBox:/proc$
```

g. Find out various states of process at time of observation.

column in top displays states of process.

```
user@user-VirtualBox:/proc$ top

top - 13:54:12 up 25 min,  1 user,  load average: 0.47, 0.41, 0.39
Tasks: 157 total,   2 running, 121 sleeping,   3 stopped,   1 zombie
%Cpu(s): 36.3 us,  3.4 sy,  0.0 ni, 60.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  4973600 total,  2522224 free,  1074796 used,  1376580 buff/cache
KiB Swap:   999420 total,   999420 free,        0 used.  3592184 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
 2150 user      20   0 1725776 253500 115232 R 26.9  5.1   3:36.17 Web Content
 1431 user      20   0 1267244 198124  81164 S  8.0  4.0   0:56.25 compiz
  816 root      20   0  407428  93176  39480 S  4.7  1.9   0:26.98 Xorg
 1995 user      20   0  670172  36144  28712 S  0.7  0.7   0:04.17 gnome-terminal-
    1 root      20   0  119640   5668   3896 S  0.0  0.1   0:01.34 systemd
    2 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kthreadd
    4 root       0 -20       0      0      0 I  0.0  0.0   0:00.00 kworker/0:0H
    5 root      20   0       0      0      0 I  0.0  0.0   0:00.19 kworker/u2:0
    6 root       0 -20       0      0      0 I  0.0  0.0   0:00.00 mm_percpu_wq
    7 root      20   0       0      0      0 S  0.0  0.0   0:00.35 ksoftirqd/0
    8 root      20   0       0      0      0 I  0.0  0.0   0:00.28 rcu_sched
    9 root      20   0       0      0      0 I  0.0  0.0   0:00.00 rcu_bh
   10 root      rt   0       0      0      0 S  0.0  0.0   0:00.00 migration/0
   11 root      rt   0       0      0      0 S  0.0  0.0   0:00.00 watchdog/0
   12 root      20   0       0      0      0 S  0.0  0.0   0:00.00 cpuhp/0
   13 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kdevtmpfs
   14 root       0 -20       0      0      0 I  0.0  0.0   0:00.00 netns
   15 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcu_tasks_kthre
   16 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kauditd
   17 root      20   0       0      0      0 S  0.0  0.0   0:00.00 khungtaskd
   18 root      20   0       0      0      0 S  0.0  0.0   0:00.00 oom_reaper
   19 root       0 -20       0      0      0 I  0.0  0.0   0:00.00 writeback
```

Learning from the practical: By implementing this practical, we learned how to get basic information about our machine or system using the proc directory. I.e. no. of cores the machine has, frequency of each core of the processor, memory, total processes, its state, etc.

Practical-2

Aim:

[A] Implement copy command using open, create, read, write, access and close system call. Be sure to include all necessary error checking including, ensuring the source file exists. Test your program with following specifications. a. File extension with .txt , .c , .zip , .exe , .tar b. Copy whole directory.

Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
char buffer[2048];
int version=1;
int main(int argc, char *argv[])
{
	int fdold,fdnew;
	if(argc!=3)
	{
		printf("Need 2 args");
		exit(1);
	}
	fdold=open(argv[1],O_RDONLY);
	if(fdold==-1)
	{
		printf("\nCannot open file");
		exit(1);
	}
	fdnew=open(argv[2],O_WRONLY);
	if(fdnew==-1)
	{
		printf("\nFile not exists creating new..");
		fdnew=creat(argv[2],0666);

	}
	else
		printf("\nFile exists");

	if(fdnew==-1)
```

```
        {
                printf("Cannot open file");
                exit(1);
        }
        printf("FDOLD: %d",fdold);
        copy(fdold,fdnew);
        return 0;
}
void copy(int fdold, int fdnew)
{
        int count;
        while((count=read(fdold,buffer,sizeof(buffer)))>0)
                write(fdnew,buffer,count);
}
```

Output:

```
user@ubuntu:~$ touch f1.txt
user@ubuntu:~$ echo "This is File1" > f1.txt
user@ubuntu:~$ touch f2.txt
user@ubuntu:~$ ./cpy f1.txt f2.txt

File existsFDOLD: 3user@ubuntu:~$
user@ubuntu:~$ cat f2.txt
This is File1
user@ubuntu:~$ ./cpy f1.zip f2.zip

File not exists creating new..FDOLD: 3user@ubuntu:~$
user@ubuntu:~$ ls
cpy       Desktop    examples.desktop  f2.txt  Pictures   Videos
cpy.c     Documents  f1.txt            f2.zip  Public
cpydir.c  Downloads  f1.zip            Music   Templates
user@ubuntu:~$
```

[B] Write a program for 'ls' command using 'opendir()' and 'readdir()' system call.

Code:

```
#include    <stdio.h>
#include    <sys/types.h>
#include    <dirent.h>
#include    <sys/stat.h>
#include    <pwd.h>
#include    <grp.h>
#include   <string.h>
#include    <unistd.h>
#include    <stdlib.h>
#include    <fcntl.h>
#include    <string.h>
#define BUFFERSIZE      4096
#define COPYMODE        0644
int c=0;
void open_dir(char[], char[]);
void copy_file(char *, char[]);
void oops(char *, char *);
void main(int ac, char *av[]){
  if ( ac == 2 )
            open_dir( ".", av[1]);
  if ( ac == 3)
            open_dir( av[1], av[2]);
}
 void open_dir( char dirname[], char tardir[] ){
  DIR               *dir_ptr;
  struct dirent     *direntp;
  if ( ( dir_ptr = opendir( dirname ) ) == NULL )
            fprintf(stderr,"ls1: cannot open %s\n", dirname);
  else {
            while ( ( direntp = readdir( dir_ptr ) ) != NULL )
                      printf("%s",direntp->d_name);
                      copy_file( direntp->d_name, tardir);
            closedir(dir_ptr);
  }
}
void copy_file( char fname[], char dirname[]){
            int in_fd, out_fd;
            char destfile[] = "/home/user/";
            DIR *dir_ptr;
            char filename[] = "";
  printf("%d",in_fd);
  strcat(filename,fname[c]);
  c++;
```

```
    if((in_fd=open(filename,O_RDONLY)) == -1)
    {
            printf("\nCannot open file");
            return;
    }
    if( (dir_ptr = opendir(dirname)) == NULL)
            printf("Cannot open %s\n",dirname);
    else{
            strcat(dirname,"/");
            strcat(destfile,dirname);
            strcat(destfile,filename);
            printf("%s",destfile);
            out_fd=creat(destfile,0666);
            if(out_fd==-1)                  {
                    printf("\nFile not exists creating new..");
                    out_fd=creat(destfile,0666);

            }
            else{
                    printf("\nFile exists");

            }
    }
    int count;
    char buffer[BUFFERSIZE];
    while((count=read(in_fd,buffer,sizeof(buffer)))>0)
            write(out_fd,buffer,count);
}
void oops(char *s1, char *s2){
    fprintf(stderr,"Error: %s ", s1);
    perror(s2);
    exit(1);
}
```

Output:

```
user@ubuntu:~$ time ./cpydir source/ dest/

real    0m0.004s
user    0m0.002s
sys     0m0.000s
user@ubuntu:~$ cd dest
user@ubuntu:~/dest$ ls
f1.txt
user@ubuntu:~/dest$ cat f1.txt
Hello World
user@ubuntu:~/dest$
```

Assignment:

a. Find the real time, processor time, user space time and kernel space time for copy command implementation

```
user@ubuntu:~$ time ./cpydir source/ dest/

real    0m0.004s
user    0m0.002s
sys     0m0.000s
```

```
top - 17:44:55 up 34 min,  1 user,  load average: 0.25, 0.37, 0.47
Tasks: 158 total,   1 running, 127 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1.3 us,  1.0 sy,  0.0 ni, 97.7 id,  0.0 wa,  0.0 hi,  0.0 s
KiB Mem :  4037360 total,   105396 free,  1170356 used,  2761608 buff
KiB Swap:        0 total,        0 free,        0 used.  1945792 avai

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ CO
 5390 ubuntu    20   0 3023220 400536 162032 S  1.0  9.9   0:43.61 We
 4018 ubuntu    20   0 2985028 409812 106124 S  0.7 10.2   1:19.90 gn
 9455 ubuntu    20   0   51180   4000   3388 R  0.7  0.1   0:00.10 to
 3888 root      20   0  529028  92364  54400 S  0.3  2.3   0:20.11 Xo
 4096 ubuntu    20   0  658676  21924  17056 S  0.3  0.5   0:00.50 gs
    1 root      20   0  160040   9660   7008 S  0.0  0.2   0:13.09 sy
    2 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kt
    3 root       0 -20       0      0      0 I  0.0  0.0   0:00.00 rc
    4 root       0 -20       0      0      0 I  0.0  0.0   0:00.00 rc
    6 root       0 -20       0      0      0 I  0.0  0.0   0:00.00 kw
    8 root       0 -20       0      0      0 I  0.0  0.0   0:00.00 mm
    9 root      20   0       0      0      0 S  0.0  0.0   0:01.84 ks
   10 root      20   0       0      0      0 I  0.0  0.0   0:02.01 rc
   11 root      rt   0       0      0      0 S  0.0  0.0   0:00.02 mi
   12 root     -51   0       0      0      0 S  0.0  0.0   0:00.00 id
   14 root      20   0       0      0      0 S  0.0  0.0   0:00.00 cp
   15 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kd
```

b. Copy the source file in two different files parallely. Find out does it take same time to finish. (Create two threads) (Files has to be atleast 100 MB)

Code:

```
#include <stdio.h>
#include <pthread.h>
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<fcntl.h>
#include<errno.h>
#include<dirent.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<string.h>
#define BUFF_SIZE 1024
struct arg_struct {
   char *source;
   char *destination;
};
int is_file(const char* path) {
struct stat buf;
stat(path, &buf);
return S_ISREG(buf.st_mode);
}
void *copy_file(void *arguments) {
struct arg_struct *args = arguments;
   char *source_file = (args -> source);
   char *desti_file = (args -> destination);
   printf("%s\n", source_file);
   printf("%s\n", desti_file);
int srcFD, dstFD, nbread, nbwrite;
char *buff[BUFF_SIZE];
srcFD = open(source_file, O_RDONLY);
if(srcFD == -1) {
printf("\nError opening file %s errno = %d\n",source_file,errno);
exit(EXIT_FAILURE);
}
dstFD = open(desti_file, O_RDWR);

if(dstFD == -1) {
```

```
dstFD = creat(desti_file, S_IRWXU|S_IWUSR|S_IRGRP|S_IROTH);
}
dstFD = open(desti_file, O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR
| S_IWGRP | S_IROTH | S_IWOTH);
if(dstFD == -1) {
printf("\nError opening file %s errno = %d\n",desti_file,errno);
exit(EXIT_FAILURE);
}
while((nbread = read(srcFD, buff, BUFF_SIZE)) > 0)
if(write(dstFD, buff, nbread) != nbread)
printf("\n Error in writing data to %s\n",desti_file);
if(nbread == -1)
printf("\nError in reading data from file %s\n",source_file);
if(close(srcFD) == -1)
printf("\nError in closing file %s\n",source_file);
if(close(dstFD) == -1)
printf("\nError in closing file %s\n",desti_file);
    pthread_exit(NULL);
    return NULL;
}
int main(int argc, char* argv[]) {
pthread_t some_thread;
    struct arg_struct args1, args2;
int chkfl1,chkfl2,chkdir1,chkdir2,chkfl3;
chkfl1 = is_file(argv[1]);
chkfl2 = is_file(argv[2]);
chkfl3 = is_file(argv[3]);
if((chkfl1 == 1)&&(chkfl2 == 1)&&(chkfl3 == 1)){
args1.source = argv[1];
args1.destination = argv[2];
if (pthread_create(&some_thread, NULL, &copy_file, (void *)&args1) != 0) {
  printf("ERROR creating thread!\n");
  return -1;
}
args2.source = argv[1];
args2.destination = argv[3];
if (pthread_create(&some_thread, NULL, &copy_file, (void *)&args2) != 0) {
  printf("ERROR creating thread!\n");
  return -1;
}
```

```
pthread_join(some_thread, NULL);
}
else
printf("Please, Correct your arguments.");
return 0;
}
```

Output:

```
user@user-VirtualBox:~/Documents$ time -p ./temp source_directory/ destination_directory/
real 0.63
user 0.02
sys 0.25
user@user-VirtualBox:~/Documents$ time -p ./temp source_directory/video.mp4  source_directory/video.mp4

Error opening file source_directory/video.mp4 errno = 0
real 0.09
user 0.00
sys 0.00
user@user-VirtualBox:~/Documents$ time -p ./temp source_directory/video.mp4  source_directory/video1.mp4  source_directory/video2.mp4
real 0.39
user 0.02
sys 0.21
```

c. Once you have correctly designed and tested the program, run the program using a utility (ptrace) that traces system calls. (Find out which system call/calls program has made internally)

Output:

```
strace: Process 5058 attached
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
100.00    0.728709      728709         1           futex
------ ----------- ----------- --------- --------- ----------------
100.00    0.728709                     1           total
```

Learning from the practical: By implementing this practical, we learned how to implement copy commands using native system calls. We also learned how to copy whole directories, as well as how to copy different types of files. We also found the statistics behind these processes. Like timing, and internally executed commands. We implemented ‘ls’ command using merely ‘opendir()’ and ‘readdir()’ system calls.

Practical-3

Aim:

[A] Write a C program that will print parent process id and child process id. Mention error checking if child process is not created.

Code:

```
#include <stdio.h>
#include <unistd.h>
int main(){
  pid_t pid, ppid;
  //get the process'es pid
  pid = getpid();
  //get the parent of this process' pid
  ppid = getppid();
  sleep(5);
  printf("My pid is: %d\n",pid);
  printf("My parent's pid is %d\n", ppid);
  return 0;
}
```

Output:

```
user@ubuntu:~$ gcc new.c -o new
user@ubuntu:~$ ./new
My pid is: 65777
My parent's pid is 2742
```

[B] In continuation of part (a), write a C program where parent process wait for child process to terminate.

Code:

```
#include  <stdio.h>
#include  <string.h>
#include  <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
int main(){
  pid_t c_pid1,c_pid2,c_pid3, pid;
  int status;
  c_pid1=fork();
  c_pid2=fork();
  c_pid3=fork();
  if( c_pid1 == 0 || c_pid2 == 0 || c_pid3 == 0){
    //child
```

```
    pid = getpid();
    printf("Child: %d: I'm the child %d %d %d\n", pid, c_pid1, c_pid2, c_pid3);
    printf("Child: sleeping for 10-seconds, then exiting with status 12\n");
    sleep(10);
  }
  else if (c_pid1 > 0 || c_pid2 > 0 || c_pid3 >0){
    //parent
    //waiting for child to terminate
    pid = wait(&status);
    if ( WIFEXITED(status) ){
      printf("Parent: Child exited with status: %d, %d %d %d\n",
  WEXITSTATUS(status),c_pid1, c_pid2, c_pid3);
    }
   }
   else{
    //error: The return of fork() is negative
    perror("Error");
    //exit(2);
   }
   return 0; //success
  }
```

Output:

```
user@ubuntu:~$ gcc new1.c -o new1
user@ubuntu:~$ ./new1
Child: 66049: I'm the child 66048 0 66052
Child: 66050: I'm the child 66048 66049 0
Child: sleeping for 10-seconds, then exiting with status 12
Child: sleeping for 10-seconds, then exiting with status 12
Child: 66052: I'm the child 66048 0 0
Child: 66048: I'm the child 0 66051 66053
Child: sleeping for 10-seconds, then exiting with status 12
Child: sleeping for 10-seconds, then exiting with status 12
Child: 66053: I'm the child 0 66051 0
Child: sleeping for 10-seconds, then exiting with status 12
Child: 66051: I'm the child 0 0 66054
Child: sleeping for 10-seconds, then exiting with status 12
Child: 66054: I'm the child 0 0 0
Child: sleeping for 10-seconds, then exiting with status 12

Parent: Child exited with status: 0, 66048 66049 66050
```

[C] Write a C program using execvp() system call which will count the characters from file 'wc', using program 'p.c'.

Code:

```
#include <stdio.h>  // perror()
#include <stdlib.h> // EXIT_SUCCESS, EXIT_FAILURE
int main(void) {
  char *const cmd[] = {"wc", "-c","newtxt.txt", NULL};
  execvp(cmd[0], cmd);
  perror("Return from execvp() not expected");
  exit(EXIT_FAILURE);
}
```

Output:

```
user@ubuntu:~$ echo "Hello World" > newtxt.txt
user@ubuntu:~$ ./new2
12 newtxt.txt
user@ubuntu:~$ cat newtxt.txt
Hello World
user@ubuntu:~$
```

Assignment:

- **Write a C program that will sleep the process for 5000 seconds. Run the same program for three times sequentially. Note down your observations for following questions:**

Code:

```
#include<stdio.h>
#include<unistd.h>
int main(){
        sleep(5000);
        return 0;
}
```

a. **Every time you execute the program, does that process have the same process id?**
   Answer:
   No, when we execute the same program the second time process gets a different process id each time when it executes.
b. **Note down the following characteristics of the process control block using proc and top command.**
   - **Process state**
   - **Process number**
   - **Program counter**
   - **Memory limits**
   - **Open files list**
   - **Number of voluntary context**

Answer:

```
user@ubuntu:/$ cd /proc/2354
user@ubuntu:/proc/2354$ cat status
Name:   goa-daemon
Umask:  0022
State:  S (sleeping)
Tgid:   2354
Ngid:   0
Pid:    2354
PPid:   2118
TracerPid:      0
Uid:    1001    1001    1001    1001
Gid:    1001    1001    1001    1001
FDSize: 64
Groups: 27 1001
NStgid: 2354
NSpid:  2354
NSpgid: 2146
NSsid:  2146
VmPeak: 134595148 kB
VmSize: 101436928 kB
VmLck:         0 kB
VmPin:         0 kB
VmHWM:     32136 kB
VmRSS:      1700 kB
RssAnon:             544 kB
RssFile:            1156 kB
RssShmem:              0 kB
VmData: 67161144 kB
VmStk:       132 kB
VmExe:        40 kB
VmLib:    130676 kB
VmPTE:       696 kB
VmSwap:     5172 kB
HugetlbPages:          0 kB
CoreDumping:    0
Threads:        5
SigQ:   0/11680
SigPnd: 0000000000000000
ShdPnd: 0000000000000000
```

```
user@ubuntu:/proc/2354$ cat wchan
poll_schedule_timeoutuser@ubuntu:/proc/2354$ cat limits
Limit                     Soft Limit           Hard Limit           Units
Max cpu time              unlimited            unlimited            seconds
Max file size             unlimited            unlimited            bytes
Max data size             unlimited            unlimited            bytes
Max stack size            8388608              unlimited            bytes
Max core file size        0                    unlimited            bytes
Max resident set          unlimited            unlimited            bytes
Max processes             11680                11680                processes
Max open files            1024                 4096                 files
Max locked memory         16777216             16777216             bytes
Max address space         unlimited            unlimited            bytes
Max file locks            unlimited            unlimited            locks
Max pending signals       11680                11680                signals
Max msgqueue size         819200               819200               bytes
Max nice priority         0                    0
Max realtime priority     0                    0
Max realtime timeout      unlimited            unlimited            us
user@ubuntu:/proc/2354$
```

```
user@ubuntu:/proc/2354$ cat uid_map
         0          0 4294967295
user@ubuntu:/proc/2354$
```

```
user@ubuntu:/proc/2354$ cat stat
2354 (goa-daemon) S 2118 2146 2146 0 -1 4194304 2348 0 160 0 4 39 0 0 20 0 5 0 19478 103871414272 425 18446744073709551615 94526202331136 94526202372064 140732178316800 0 0 0 0 4096
 2 0 0 0 17 1 0 0 17 0 0 94526204471128 94526204473520 94526219001856 140732178319864 140732178319906 140732178319906 140732178321358 0
user@ubuntu:/proc/2354$
```

```
user@ubuntu:/proc/2354$ cat sched
goa-daemon (2354, #threads: 5)
-------------------------------------------------------------------
se.exec_start                                :       3506147.416453
se.vruntime                                  :          4596.648627
se.sum_exec_runtime                          :           365.553092
se.nr_migrations                             :                    6
nr_switches                                  :                  246
nr_voluntary_switches                        :                  207
nr_involuntary_switches                      :                   39
se.load.weight                               :              1048576
se.runnable_weight                           :              1048576
se.avg.load_sum                              :                  447
se.avg.runnable_load_sum                     :                  447
se.avg.util_sum                              :               123303
se.avg.load_avg                              :                    8
se.avg.runnable_load_avg                     :                    8
se.avg.util_avg                              :                    1
se.avg.last_update_time                      :       3506147416064
policy                                       :                    0
prio                                         :                  120
clock-delta                                  :                   33
mm->numa_scan_seq                            :                    0
numa_pages_migrated                          :                    0
numa_preferred_nid                           :                   -1
total_numa_faults                            :                    0
current_node=0, numa_group_id=0
numa_faults node=0 task_private=0 task_shared=0 group_private=0 group_shared=0
user@ubuntu:/proc/2354$
```

Learning from the practical: By implementing this practical, we implemented a c program which can print the details of the parent and child process. We also learned how to program in such a way that the parent process can wait for its child process to terminate before its own termination. We also learned how execvp() system call works and how to use it. We found different characteristics of a process and process control block.

Practical-4

Aim: Write a program uses dup() and dup2() system call that will prove the following sentence: "dup() always uses the smallest available (unused) file descriptor whereas dup2() uses new file descriptor."

Theory:

dup():

- The dup() system call creates a copy of file descriptor
- It uses the lowest-numbered unused descriptor for the new descriptor
- If the copy is successfully created, then the original and copy file descriptors may be used interchangeably
- They both refer to the same open file description and thus share file offset and file status flags

dup2():

The dup2() system call is similar to dup() but the basic difference between them is that instead of using the lowest-numbered unused file descriptor, it uses the descriptor number specified by the user.

dup2() makes newfd be the copy of oldfd, closing newfd first if necessary, but note the following:

- If oldfd is not a valid file descriptor, then the call fails, and newfd is not closed.
- If oldfd is a valid file descriptor, and newfd has the same value as oldfd, then dup2() does nothing, and returns newfd.

Syntax: dup2(newfd,oldfd/number)

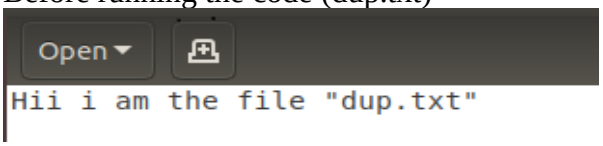
Code: (dup())

```
#include<stdio.h>
#include <unistd.h>
#include <fcntl.h>
int main()
{
	// open() returns a file descriptor file_desc to a
	// the file "dup.txt" here"

	int file_desc = open("dup.txt", O_WRONLY | O_APPEND);

	if(file_desc < 0)
		printf("Error opening the file\n");

	// dup() will create the copy of file_desc as the copy_desc
	// then both can be used interchangeably.

	int copy_desc = dup(file_desc);

	// write() will write the given string into the file
	// referred by the file descriptors

	write(copy_desc,"This will be output to the file named dup.txt\n", 46);

	write(file_desc,"This will also be output to the file named dup.txt\n", 51);

	printf("Old file descriptor value: %d\n",file_desc);
	printf("New file descriptor value: %d\n",copy_desc);
	return 0;
}
```

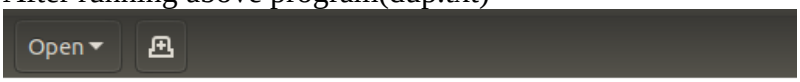
Explanation: The open() returns a file descriptor file_desc to the file named “dup.txt”. file_desc can be used to do some file operation with file “dup.txt”. After using the dup() system call, a copy of file_desc is created copy_desc. This copy can also be used to do some file operation with the same file “dup.txt”. After two write operations one with file_desc and another with copy_desc, same file is edited i.e. “dup.txt”.

Output:

Before running the code (dup.txt)



After running above program(dup.txt)



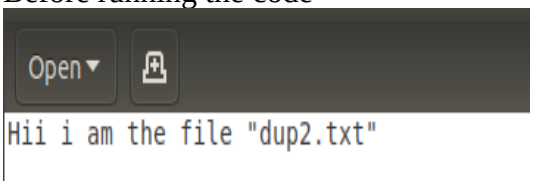

```
Hii i am the file "dup.txt"
This will be output to the file named dup.txt
This will also be output to the file named dup.txt
```

Code: (dup2())

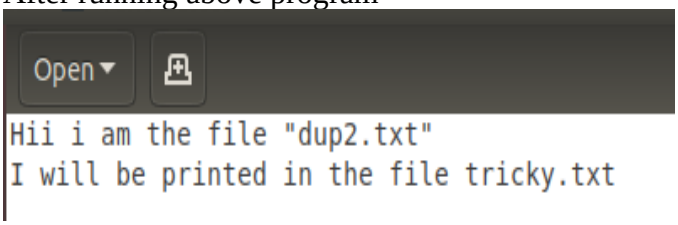
```
#include<stdlib.h>
#include<unistd.h>
#include<stdio.h>
#include<fcntl.h>
int main()
{
        int file_desc = open("dup2.txt",O_WRONLY | O_APPEND);
        // here the newfd is the file descriptor of stdout (i.e. 1)
        printf("Old file descriptor value: %d\n",file_desc);
        dup2(file_desc, 1) ;

        // All the printf statements will be written in the file
        // "tricky.txt"
        printf("I will be printed in the file tricky.txt\n");
        printf("New file descriptor value: %d\n",file_desc);
        return 0;
}
```

Output:

Before running the code



After running above program



Learning from the practical: We learned how to use dup() and dup2() calls to assign file descriptor to a file. We also saw that dup() always uses the smallest available (unused) file descriptor whereas dup2() uses a new file descriptor.

Practical-5

Aim: Write the following programs using inter-process communication – shared memory. The program 'writer.c' will print 1 to 100 in shared memory region. Another program 'reader.c' that will read all the numbers from shared memory to make addition of it and display it.

Code:

Writer.c:

```
#include<sys/shm.h>
#include<sys/ipc.h>
#include<sys/types.h>
#include<stdio.h>
#define shmsize 20
void main(){
        key_t keyid;
        keyid = ftok("/etc/passwd",'b');
        printf("Key: %d\n",keyid);
        int shmid;
        shmid = shmget(keyid,shmsize,IPC_CREAT|0666);
        printf("ShmID: %d\n",shmid);
        char *s;
        s = shmat(shmid,NULL,0);
        int i;
        int *shat = s;
        for(i=1;i<=100;i++){
                *shat++ = (char)i;
        }
}
```

Reader.c

```
#include<sys/shm.h>
#include<sys/ipc.h>
#include<sys/types.h>
#include<stdio.h>
#define shmsize 20
void main(){
        key_t keyid;
        keyid = ftok("/etc/passwd",'b');
        printf("Key: %d\n",keyid);
        int shmid;
        shmid = shmget(keyid,shmsize,IPC_CREAT|0666);
        printf("ShmID: %d\n",shmid);
        int *s;
        s = shmat(shmid,NULL,0);
        int i;
        int *shat;
        int sum = 0;
        for(shat=s;*shat!=NULL;shat++){
                printf("%d ",*shat);
                sum = sum + *shat;
        }
        printf("\nSum:%d\n",sum);
}
```

Output:

```
student@ubuntu:~$ gcc writer.c -o writer
writer.c: In function ‘main’:
writer.c:31:13: warning: initialization from incompatible pointer type [-Wincom
patible-pointer-types]
 int *shat = s;
             ^
student@ubuntu:~$ ./writer
Key: 1644238964
ShmID: 819206
student@ubuntu:~$
```

```
student@ubuntu:~$ gedit reader.c
student@ubuntu:~$ gcc reader.c -o reader
reader.c: In function ‘main’:
reader.c:35:17: warning: comparison between pointer and integer
 for(shat=s;*shat!=NULL;shat++){
                 ^~
student@ubuntu:~$ ./reader
Key: 1644238964
ShmID: 819206
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 3
0 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56
 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82
83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
Sum:5050
student@ubuntu:~$
```

Assignment:

Aim: Solve above issue using pipe().

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#define MAX 100
int main() {
int fd[2], i = 0;
pipe(fd);
pid_t pid = fork();
if (pid > 0) {
wait(NULL);
close(0);
close(fd[1]);
dup(fd[0]);
int arr[MAX];
int n = read(fd[0], arr, sizeof(arr));
for (i = 0; i < n / 4; i++) printf("%d ", arr[i]);
} else if (pid == 0) {
int arr[100];
for (int i = 1; i < 101; i++) {
arr[i - 1] = i;
}
close(fd[0]);
close(1);
dup(fd[1]);
write(1, arr, sizeof(arr));
```

```
} else {
perror("[-] Error\n");
}
printf("\n");
}
```

Output:

```
student@ubuntu:~$ gedit assignment.c
student@ubuntu:~$ gcc assignment.c -o assignment
student@ubuntu:~$ ./assignment
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 3
0 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56
 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82
83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
student@ubuntu:~$
```

Learning from the practical: In this practical, we learned one of the methods of inter-process communication: using the swap area. We learned how processes can communicate using swap area and share data also. In this practical, we made a swap are and in that shared memory region, a process can use the data from this region weather it is created by that process or any other process. It can also add its own data.

Practical-6

Aim: Consider a process executing on a CPU. Give an example scenario that can cause the process to undergo:
(a) A voluntary context switch.
(b) An involuntary context switch
Write the program for both the cases.

Code: (First.c)

```
#include<stdio.h>
int main(){
        int p;
        int i;
        for(i=0;i<50;i++){
                printf("process sleep for %d times",i);
                sleep(i);
        }
}
```

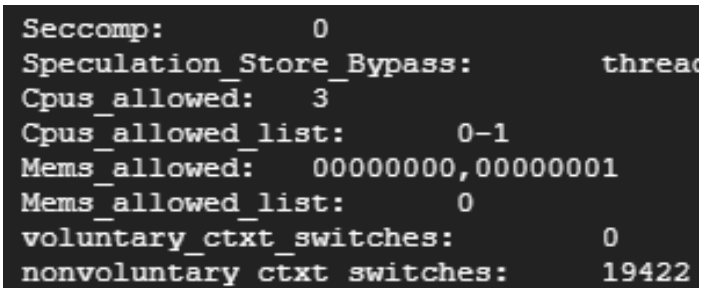
Output:

```
Seccomp:        0
Speculation_Store_Bypass:       thread vulnerable
Cpus_allowed:   3
Cpus_allowed_list:      0-1
Mems_allowed:   00000000,00000001
Mems_allowed_list:      0
voluntary_ctxt_switches:        15
nonvoluntary_ctxt_switches:     1
```

Code: (cpu.c)

```
#include <unistd.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
        unsigned int i,j;
        while(1)
        {
                j = 1;
                for(i = 1; i <= 10; i++){
                        j = j*i;
                }
        }
}
```

Output:



Code: (disk.c)

```
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <string.h>
#include <errno.h>
#define FNAME_SIZE 100
#define MAX_FILE_NO 300
#define BLOCK_SIZE 1024
int main(int argc, char *argv[]){
        int n, file_no;
        FILE *fp;
        char dest_file_name[FNAME_SIZE];
        char buf[BLOCK_SIZE];
        while(1)
        {
                file_no = rand() % MAX_FILE_NO;
                bzero(dest_file_name, FNAME_SIZE);
                sprintf(dest_file_name, "files/foo%d.txt", file_no);
                fp = fopen(dest_file_name, "rb");
                if (fp == NULL) {
                     perror("Can't open dest file");
                exit(1);
                }
                bzero(buf,BLOCK_SIZE);
                while ( (n = (int)fread( buf, 1, BLOCK_SIZE, fp )) > 0)
                {
                        //do nothing with the read data;
                        bzero(buf,BLOCK_SIZE);
                }
                fclose(fp);
        }
}
```

Output:

```
student@student-VirtualBox:~/files$ head -c 1048576 </dev/urandom >foo0.txt
student@student-VirtualBox:~/files$ ls -ltrh foo0.txt
-rw-r--r-- 1 student student 1.0M Mar  6 13:20 foo0.txt
```

```
Every 0.5s: grep ctxt /proc/184...  student-VirtualBox: Sat Mar  7 02:35:37 2020

voluntary_ctxt_switches:        0
nonvoluntary_ctxt_switches:     4553
```

Learning from the practical: We learned about context switch and its types of processes: voluntary context switch and non-voluntary context switch. We learned how to get them and how to manage those context switches.

Practical-7

Aim: Write a program to demonstrate the handling of the signals: SIGINT, SIGALRM & SIGQUIT.

- **SIGINT**

Theory:

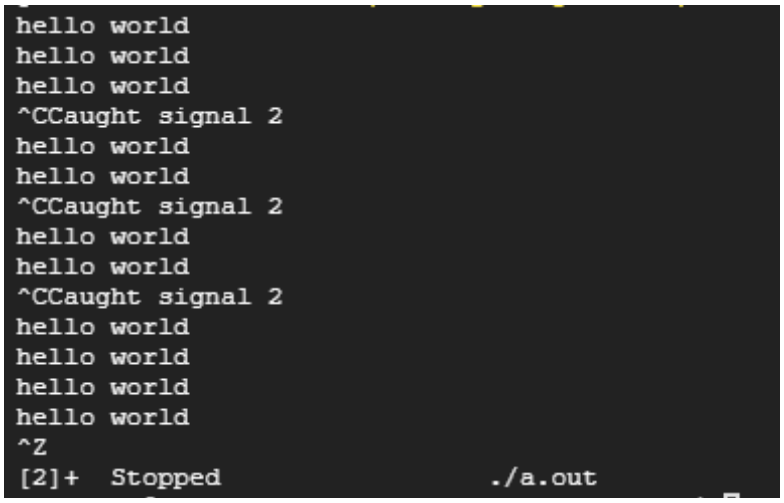
SIGINT is the interrupt signal. The terminal sends it to the foreground process when the user presses ctrl-c. The default behavior is to terminate the process, but it can be caught or ignored.

Code:

To handle the signal SIGINT

```
#include<stdio.h>
#include<signal.h>
void handle_sigint(int sig){
   printf("Caught signal %d\n", sig);
}
int main(){
   signal(SIGINT, handle_sigint);
   while (1) {
      printf("hello world\n");
      sleep(1);
   }
   return 0;
}
```

Output:



- **SIGALRM**

Theory:

Search Results

SIGALRM is an asynchronous signal. The SIGALRM signal is raised when a time interval specified in a call to the alarm or alarmd function expires. Because SIGALRM is an asynchronous signal, the SAS/C library discovers the signal only when you call a function, when a function returns, or when you issue a call to sigchk

Code:

```
#include <signal.h>
#include <stdio.h>
#include <stdbool.h>
#include <unistd.h>
volatile sig_atomic_t print_flag = false;
void handle_alarm( int sig ) {
   print_flag = true;
}
int main() {
   signal( SIGALRM, handle_alarm ); // Install handler first,
   alarm( 1 ); // before scheduling it to be called.
   for (;;) {
      if ( print_flag ) {
         printf( "Hello\n" );
         print_flag = false;
         alarm( 1 );
      }
   }
}
```

Output:

SIGQUIT

Theory:

SIGQUIT is the dump core signal. The terminal sends it to the foreground process when the user presses ctrl-\. The default behavior is to terminate the process and dump core, but it can be caught or ignored. The intention is to provide a mechanism for the user to abort the process.

Code:

```
#include <signal.h>
static void
sigHandler(int sig)
{
        static int count = 0;
    if (sig == SIGINT) {
        count++;
        printf("Caught SIGINT (%d)\n", count);
        return;
    }
    printf("Caught SIGQUIT - that's all!\n");
}
int main(int argc, char *argv[]){

    if (signal(SIGINT, sigHandler) == SIG_ERR)
        printf("signal error");
    if (signal(SIGQUIT, sigHandler) == SIG_ERR)
        printf("signal error");

    for (;;)
        pause();
}
```

Output:

```
^CCaught SIGINT (1)
^CCaught SIGINT (2)
^CCaught SIGINT (3)
^Y

^\Caught SIGQUIT - that's all folks!
```

Learning from the practical: We learned how to handle the following signals: SIGINT whose default behavior is to terminate a process, SIGALRM which is raised when a time interval specified in a call to the alarm or alarmd function expires and SIGQUIT which is used to provide a mechanism to the user to abort a process.

Practical-8

Aim:

[A] Create a 1GB swap area in your linux partition and free it. Check the allocation of swap space. Execute following commands to monitor swap space in linux.

- **a. Swapon**
- **b. use of /proc/swaps**
- **c. free**
- **d. top**
- **e. atop**
- **f. htop**
- **g. glances**
- **h. vmstat**

Setting up:

1. Create swap file

```
ay@ub:~/tmp$
ay@ub:~/tmp$ ls
ay@ub:~/tmp$
ay@ub:~/tmp$ fallocate -l 1G swapfile
ay@ub:~/tmp$
ay@ub:~/tmp$ ls -lh
total 1.1G
-rw-r--r-- 1 ay ay 1.0G Apr  7 21:23 swapfile
ay@ub:~/tmp$
ay@ub:~/tmp$
```

2. Changing permission

```
ay@ub:~/tmp$
ay@ub:~/tmp$ chmod 600 swapfile
ay@ub:~/tmp$
ay@ub:~/tmp$ ll
total 1048588
drwxr-xr-x  2 ay ay       4096 Apr  7 21:23 ./
drwxr-xr-x 29 ay ay       4096 Apr  7 21:07 ../
-rw-------  1 ay ay 1073741824 Apr  7 21:23 swapfile
ay@ub:~/tmp$
ay@ub:~/tmp$
```

3. Enabling swap file

```
ay@ub:~/tmp$ sudo mkswap swapfile
mkswap: swapfile: insecure file owner 1000, 0 (root) suggested.
Setting up swapspace version 1, size = 1024 MiB (1073737728 bytes)
no label, UUID=6082e6f8-3847-412c-96f8-6de653aa9434
ay@ub:~/tmp$
ay@ub:~/tmp$
ay@ub:~/tmp$ sudo swapon swapfile
swapon: /home/ay/tmp/swapfile: insecure file owner 1000, 0 (root) suggested.
ay@ub:~/tmp$
```

4. Verifying swap file

```
ay@ub:~/tmp$ sudo swapon --show
NAME                 TYPE       SIZE USED PRIO
/dev/sda11           partition  3.8G   0B   -2
/home/ay/tmp/swapfile file      1024M   0B   -3
ay@ub:~/tmp$
```

5. Remove swap file

```
ay@ub:~/tmp$
ay@ub:~/tmp$ sudo swapoff -v swapfile
swapoff swapfile
ay@ub:~/tmp$
ay@ub:~/tmp$ ls
swapfile
ay@ub:~/tmp$
ay@ub:~/tmp$ sudo rm swapfile
ay@ub:~/tmp$
```

Executing the commands to monitor swap space in linux:

1. Swapon

This command helps you to specify the devices on which paging and swapping will be done.

If you want to view a summary of swap space usage by device:

```
ay@ub:~/tmp$ swapon --summary
Filename                                Type            Size    Used    Priority
/dev/sda11                              partition       3999740 0       -2
ay@ub:~/tmp$
```

2. /proc/swaps

Equivalent as swapon

```
ay@ub:~/tmp$
ay@ub:~/tmp$ cat /proc/swaps
Filename                                Type            Size    Used    Priority
/dev/sda11                              partition       3999740 0       -2
ay@ub:~/tmp$
```

3. Free

The free command is used to display the amount of free and used system memory.

```
ay@ub:~/tmp$ free
              total        used        free      shared  buff/cache   available
Mem:        7946232     1954852     3462596      356812     2528784     5362912
Swap:       3999740           0     3999740
ay@ub:~/tmp$
```

4. Top

The top command displays processor activity of your Linux system, tasks managed by kernel in real-time.

```
top - 21:36:43 up 42 min,  1 user,  load average: 0.88, 0.90, 1.08
Tasks: 266 total,   1 running, 210 sleeping,   2 stopped,   0 zombie
%Cpu(s):  6.6 us,  1.4 sy,  0.0 ni, 91.3 id,  0.0 wa,  0.0 hi,  0.7 si,  0.0 st
KiB Mem :  7946232 total,  3485944 free,  1949104 used,  2511184 buff/cache
KiB Swap:  3999740 total,  3999740 free,        0 used.  5386644 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 1946 ay        20   0 3872768 279364  79240 S  23.4  3.5   4:51.41 gnome-shell
 1813 ay        20   0  701708  94752  70504 S   5.6  1.2   2:46.60 Xorg
  388 root     -51   0       0      0      0 D   1.3  0.0   0:08.25 irq/87-SYN+
 7544 ay        20   0   51320   3968   3320 R   1.0  0.0   0:00.15 top
  517 root     -51   0       0      0      0 S   0.7  0.0   0:04.02 irq/132-i2+
 3531 ay        20   0  877672 184716 114488 S   0.7  2.3   1:58.65 chrome
    1 root      20   0  225928   9756   6792 S   0.3  0.1   2:29.88 systemd
 7022 ay        20   0  880900 217176  93468 S   0.3  2.7   0:19.86 chrome
```

5. Atop

The atop command is a system monitor that reports about activities of various processes. But importantly it also shows information about free and used memory space.

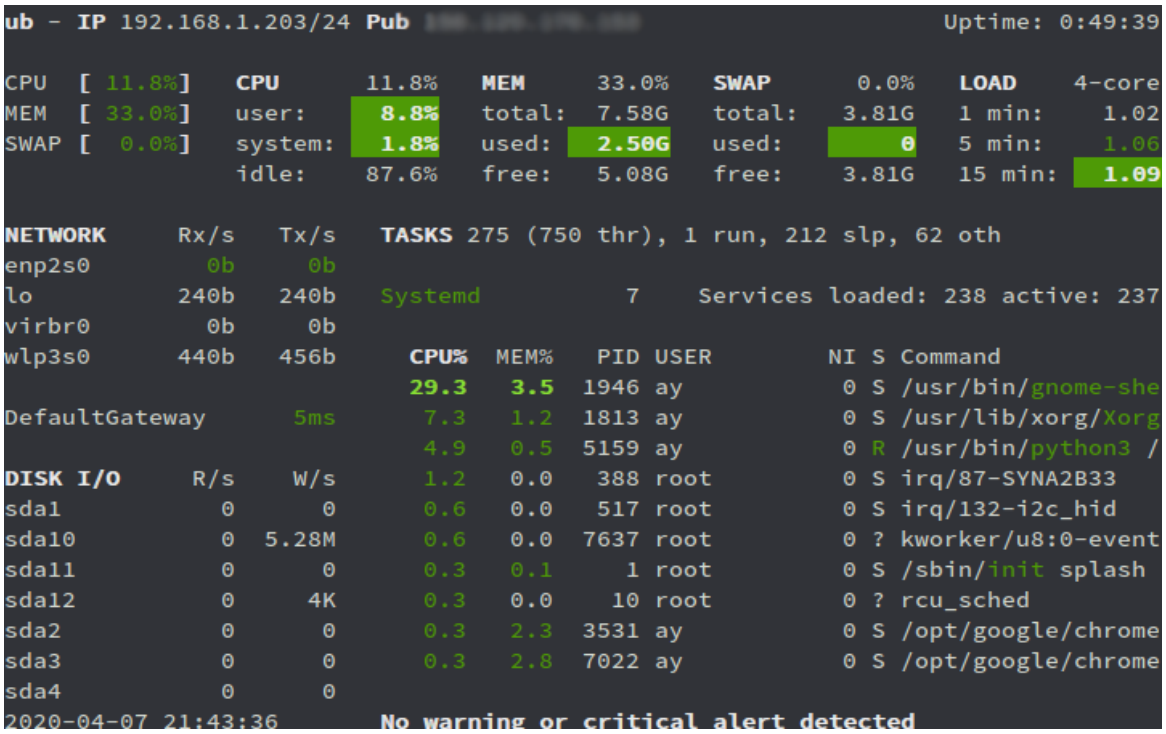
```
ATOP - ub          2020/04/07  21:39:46         --------------          10s elapsed
PRC | sys    0.32s | user   0.54s | #proc    289 | #zombie    0 | #exit      0 |
CPU | sys      4% | user     6% | irq       0% | idle    390% | wait       1% |
cpu | sys      1% | user     1% | irq       0% | idle     98% | cpu001 w   0% |
cpu | sys      1% | user     2% | irq       0% | idle     96% | cpu003 w   0% |
cpu | sys      1% | user     1% | irq       0% | idle     97% | cpu002 w   1% |
cpu | sys      1% | user     1% | irq       0% | idle     99% | cpu000 w   0% |
CPL | avg1    1.00 | avg5    0.99 | avg15   1.09 | csw    11762 | intr   44264 |
MEM | tot     7.6G | free    3.2G | cache   2.2G | buff  218.1M | slab  169.4M |
SWP | tot     3.8G | free    3.8G |              | vmcom   7.2G | vmlim   7.6G |
DSK |          sda | busy      1% | read      0 | write     6 | avio 15.3 ms |
NET | transport    | tcpi     10 | tcpo      9 | udpi      0 | udpo       0 |
NET | network      | ipi      10 | ipo       9 | ipfrw     0 | deliv     10 |
NET | wlp3s0    0% | pcki     10 | pcko      9 | si    0 Kbps | so    1 Kbps |

  PID SYSCPU  USRCPU  VGROW   RGROW   ST EXC   THR  S CPUNR   CPU  CMD        1/5
 1946  0.05s   0.25s    36K    168K   --   -    14  S     2    3%  gnome-shell
 1813  0.07s   0.15s     0K    272K   --   -     5  S     1    2%  Xorg
  388  0.07s   0.00s     0K      0K   --   -     1  S     2    1%  irq/87-SYNA2B3
 7263  0.01s   0.04s     0K      0K   --   -     4  S     2    1%  gnome-terminal
 3531  0.01s   0.03s     0K      0K   --   -    26  S     0    0%  chrome
  517  0.04s   0.00s     0K      0K   --   -     1  S     1    0%  irq/132-i2c_hi
 3572  0.01s   0.02s     0K      0K   --   -     8  S     2    0%  chrome
 1984  0.01s   0.01s     0K      0K   --   -     3  S     1    0%  ibus-daemon
```

6. Htop

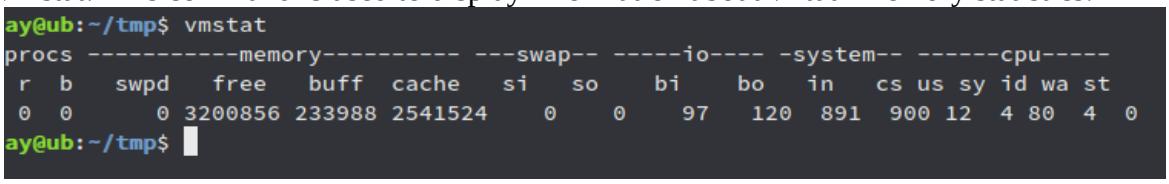
The htop command is used to view processes in an interactive mode and also displays information about memory usage.

```
  1  [||||                      11.5%]   Tasks: 166, 476 thr; 1 running
  2  [||                         6.4%]   Load average: 0.91 1.00 1.08
  3  [||                         4.0%]   Uptime: 00:46:57
  4  [|||                        7.8%]
  Mem[||||||||||||||||||| 2.20G/7.58G]
  Swp[                      0K/3.81G]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 1946 ay         20   0 3779M  272M 79176 S 19.9  3.5  4:57.47 /usr/bin/gnome-sh
 1813 ay         20   0  683M 93412 69164 S  5.3  1.2  2:51.78 /usr/lib/xorg/Xor
 3757 ay         20   0 40696  4812  3888 R  2.7  0.1  0:00.32 htop
 1819 ay         20   0  683M 93412 69164 S  1.3  1.2  0:12.31 /usr/lib/xorg/Xor
    1 root       20   0  220M  9864  6792 S  0.7  0.1  0:06.31 /sbin/init splash
 3531 ay         20   0  861M  180M  111M S  0.0  2.3  2:03.33 /opt/google/chrom
 3549 ay         20   0  861M  180M  111M S  0.0  2.3  0:37.74 /opt/google/chrom
 1337 kernoops   20   0 56940   424     0 S  0.0  0.0  0:00.08 /usr/sbin/kernelo
 1977 ay         20   0 3779M  272M 79176 S  0.0  3.5  0:00.39 /usr/bin/gnome-sh
 1979 ay         20   0 3779M  272M 79176 S  0.0  3.5  0:00.45 /usr/bin/gnome-sh
 7022 ay         20   0  861M  213M 93468 S  0.0  2.8  0:20.41 /opt/google/chrom
 3608 ay         20   0  557M  116M 57372 S  0.0  1.5  0:18.34 /opt/google/chrom
 7263 ay         20   0  782M 39976 28272 S  0.0  0.5  0:05.97 /usr/lib/gnome-te
 7198 ay         20   0  854M  230M 83680 S  0.0  3.0  1:40.03 /opt/google/chrom
```

7. Glances: This is a cross-platform system monitoring tool that displays information about running processes, cpu load, storage space usage, memory usage, swap space usage and many more.



8. Vmstat: This command is used to display information about virtual memory statistics.



Learning from the practical:: In this practical, we have created 1GB swap area in our Linux partition and freed it using some swapping commands.

[B] Write the simulation Paging Algorithms program for demand paging and show the page scheduling and total number of page faults according to FIFO, LRU, and optimal page replacement algorithm. Assume the memory of 'n' frames.

Code:

FIFO:

```
#include<bits/stdc++.h>
using namespace std;
int pageFaults(int pages[], int n, int capacity) {
	unordered_set<int> s;
	queue<int> indexes;
	int page_faults = 0;
	for (int i=0; i<n; i++){
		if (s.size() < capacity) {
			if (s.find(pages[i])==s.end()){
				s.insert(pages[i]);
				page_faults++;
				indexes.push(pages[i]);
			}
		}
		else {
			if (s.find(pages[i]) == s.end()) {
				int val = indexes.front();
				indexes.pop();
				s.erase(val);
				s.insert(pages[i]);
				indexes.push(pages[i]);
				page_faults++;
			}
		}
	}
	return page_faults;
}
int main() {
	int n;
	cout << "\n[+] Enter number of pages: ";
	cin >> n;
	int *pages = new int[n];
	cout << "[+] Enter pages: ";
	for(int i = 0; i < n; i++){
		cin >> pages[i];
	}
	int c;
	cout << "[+] Enter capacity: ";
	cin >> c;
```

```
	cout << "\n[+] PageFaults: ";
	cout << pageFaults(pages, n, c) << endl << endl;
	return 0;
}
```

Output:

```
[+] Enter number of pages: 13
[+] Enter pages: 7 0 1 2 0 3 0 4 2 3 0 3 2
[+] Enter capacity: 4

[+] PageFaults: 7
```

LRU:

Code:

```
#include<bits/stdc++.h>
using namespace std;
int pageFaults(int pages[], int n, int capacity)
{
	unordered_set<int> s;
	unordered_map<int, int> indexes;
	int page_faults = 0;
	for (int i=0; i<n; i++)
	{
		if (s.size() < capacity)
		{
			if (s.find(pages[i])==s.end())
			{
				s.insert(pages[i]);
				page_faults++;
			}
			indexes[pages[i]] = i;
		}
		else
		{
			if (s.find(pages[i]) == s.end())
			{
				int lru = INT_MAX, val;
				for (auto it=s.begin(); it!=s.end(); it++)
				{
```

```
                                        if (indexes[*it] < lru)
                                        {
                                                lru = indexes[*it];
                                                val = *it;
                                        }
                                }
                                s.erase(val);
                                s.insert(pages[i]);
                                page_faults++;
                        }
                        indexes[pages[i]] = i;
                }
        }
        return page_faults;
}
int main()
{
        int n;
        cout << "\n[+] Enter number of pages: ";
        cin >> n;
        int *pages = new int[n];
        cout << "[+] Enter pages: ";
        for(int i = 0; i < n; i++){
                cin >> pages[i];
        }
        int c;
        cout << "[+] Enter capacity: ";
        cin >> c;
        cout << "\n[+] PageFaults: ";
        cout << pageFaults(pages, n, c);
        return 0;
}
```

Output:

```
[+] Enter number of pages: 13
[+] Enter pages: 7 0 1 2 0 3 0 4 2 3 0 3 2
[+] Enter capacity: 4

[+] PageFaults: 6
```

Optimal Page Replacement:

Code:

```
#include <bits/stdc++.h>
using namespace std;
bool search(int key, vector<int>& fr)
{
	for (int i = 0; i < fr.size(); i++)
		if (fr[i] == key)
			return true;
	return false;
}
int predict(int pg[], vector<int>& fr, int pn, int index)
{
	int res = -1, farthest = index;
	for (int i = 0; i < fr.size(); i++) {
		int j;
		for (j = index; j < pn; j++) {
			if (fr[i] == pg[j]) {
				if (j > farthest) {
					farthest = j;
					res = i;
				}
				break;
			}
		}
		if (j == pn)
			return i;
	}
	return (res == -1) ? 0 : res;
}
void optimalPage(int pg[], int pn, int fn)
{
	vector<int> fr;
	int hit = 0;
	for (int i = 0; i < pn; i++) {
		if (search(pg[i], fr)) {
			hit++;
			continue;
		}
		if (fr.size() < fn)
			fr.push_back(pg[i]);
		else {
			int j = predict(pg, fr, pn, i + 1);
			fr[j] = pg[i];
```

```
                }
        }
        cout << "No. of hits = " << hit << endl;
        cout << "No. of misses = " << pn - hit << endl;
}
int main()
{
        int n;
        cout << "\n[+] Enter number of pages: ";
        cin >> n;
        int *pages = new int[n];
        cout << "[+] Enter pages: ";
        for(int i = 0; i < n; i++){
                cin >> pages[i];
        }
        int fn;
        cout << "[+] Enter number of frames: ";
        cin >> fn;
        optimalPage(pages, n, fn);
        return 0;
}
```

Output:

```
[+] Enter number of pages: 13
[+] Enter pages: 7 0 1 2 0 3 0 4 2 3 0 3 2
[+] Enter number of frames: 4
No. of hits = 7
No. of misses = 6
```

Learning from the practical:: In this practical we have implemented the simulation Paging Algorithms program for demand paging and show the page scheduling and total number of page faults according to FIFO, LRU, and optimal page replacement algorithm.

Practical-9

Aim: Assume that processes communicate by send/receive messages, which are unreliable, e.g. messages may be lost during send/recv. After sending a message, a process expects a reply. If it does not receive a reply within 't' seconds, it re-sends the same message again. If it receives a reply within 't' seconds, it must not send the same message again. Design an algorithm for the sending process and implement it.

Code:

```
#include<stdio.h>
#include "mpi.h"
#include<time.h>
void main(int argc, char* argv[]){
        int my_rank, numbertoreceive, numbertosend0=36,
        numbertosend1=1, flag = 1; clock_t t;
        MPI_Status
        status;
        MPI_Init(&a
        rgc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD,
        &my_rank); if (my_rank==0){
                while(1){ t = clock();
                        MPI_Send( &numbertosend0, 1, MPI_INT, 1,        10,
                        MPI_COMM_WORLD); if((clock()-t)>100){
                                t = clock();
                                MPI_Send( &numbertosend0, 1, MPI_INT, 1,  10,
                                MPI_COMM_WORLD);
                        }
                        MPI_Recv(&numbertoreceive, 1, MPI_INT,1, 20,
                        MPI_COMM_WORLD,&status);
                        if(numbertoreceive == 1){
                                printf("0 received : %d\n",
                                numbertoreceive); break;
                        }}
        }else if(my_rank== 1){
                numbertoreceive = 0;
                MPI_Recv(&numbertoreceive, 1, MPI_INT, 0, 10, MPI_COMM_WORLD,
                &status); if(numbertoreceive){
                        MPI_Send(&numbertosend1, 1, MPI_INT, 0, 20,
                        MPI_COMM_WORLD); printf("1 received : %d\n",
                        numbertoreceive);
                }}
        MPI_Finalize();
}
```

Output:

Learning from the practical: In this practical, we have made a chat application using Java.We have observed that the client sends the message and waits for the sender reply and if waiting for a finite amount of time if the response is not received the client again sends the same message and repeats the same procedure till the sender sends the response.

Practical-10

Aim: Implementation of a device driver to find reverse string in kernel mode.

Theory:

There are 2 main types of device files, a character device file and a block device file. The differences are, a block device is buffered (meaning it doesn't offer direct access to the device and ultimately means that you don't know how long it will take before a write is pushed to the actual device) and a block device allows reads or writes of any size, character device reads and writes are aligned to block boundaries.

We will be using a character device because they are simpler to understand (as we will use the device file in exactly the same way that we would use a regular file), we have no need for random access to the device and it provides direct access to the device.

The beauty of UNIX is that devices are represented as files. Both character devices and block devices are represented by respective files in the /dev directory. This means that you can read and write into the device by manipulating those file using standard system calls like open, read, write, close etc.

```
student@ubuntu:~$ ls -l /dev/console
crw------- 1 root root 5, 1 Apr  5 04:23 /dev/console
student@ubuntu:~$ stat /dev/console
  File: /dev/console
  Size: 0               Blocks: 0          IO Block: 4096   character special f
ile
Device: 6h/6d   Inode: 14          Links: 1     Device type: 5,1
Access: (0600/crw-------)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2020-04-05 04:23:38.812124531 -0700
Modify: 2020-04-05 04:23:38.812124531 -0700
Change: 2020-04-05 04:23:38.812124531 -0700
 Birth: -
student@ubuntu:~$
```

Code:

Simple reverse.c file:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char data[513] = "No data";

void insert_word(char *word, unsigned int n)
{
   int i, c;
   char tmpword[512+1];
   for (i = strlen(word)-1, c = 0; i >= 0; i--, c++) {
      tmpword[c] = word[i];
   }
   tmpword[strlen(word)] = '\0';
   if (n == 0) {
```

```
        memset(data, 0, sizeof data);
        strcpy(data, tmpword);
    } else {
        data[strlen(data)] = ' ';
        data[strlen(data)+1] = '\0';
        strcat(data, tmpword);
    }
}

void reverse(char *tmpdata)
{
    int i, c;
    unsigned int n = 0;
    char word[512+1];
    for (i = strlen(tmpdata)-1, c = 0; i >= 0; i--, c++) {
        if (tmpdata[i] == ' ') {
            word[c] = '\0';
            insert_word(word, n);
            n += 1;
            c = -1;
        } else
            word[c] = tmpdata[i];

    }
    word[c] = '\0';
    insert_word(word, n);
    data[strlen(tmpdata)] = '\0';
}

int main(int argc, char **argv)
{
    if (argc < 2) {
        printf("Usage: %s <string>\n", argv[0]);
        exit(1);
    }

    printf("Before: %s\n", data);
    reverse(argv[1]);
    printf("After: %s\n", data);
}
```

Output:

```
student@ubuntu:~$ gedit rev.c
student@ubuntu:~$ gcc rev.c -o rev
student@ubuntu:~$ ./rev "reverse application test"
Before: No data
After: test application reverse
student@ubuntu:~$ ./rev "test application reverse"
Before: No data
After: reverse application test
student@ubuntu:~$
```

Our character device:

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/miscdevice.h>
#include <linux/fs.h>
#include <linux/uaccess.h>

MODULE_AUTHOR("0xe7, 0x1e");
MODULE_DESCRIPTION("A simple character device which reverses the words in a
string");
MODULE_LICENSE("GPL");

#define DEVICE_SIZE 512

char data[DEVICE_SIZE+1]="no data has been written yet";

void insert_word(char *word, unsigned int n)
{
   int i, c;
   char tmpword[DEVICE_SIZE+1];
   for (i = strlen(word)-1, c = 0; i >= 0; i--, c++) {
      tmpword[c] = word[i];
   }
   tmpword[strlen(word)] = '\0';
   if (n == 0) {
      memset(data, 0, sizeof data);
      strcpy(data, tmpword);
   } else {
      data[strlen(data)] = ' ';
      data[strlen(data)+1] = '\0';
      strcat(data, tmpword);
   }
}

void reverse(char *tmpdata)
{
   int i, c;
```

```
    unsigned int n = 0;
    char word[DEVICE_SIZE+1];
    for (i = strlen(tmpdata)-1, c = 0; i >= 0; i--, c++) {
        if (tmpdata[i] == ' ') {
            word[c] = '\0';
            insert_word(word, n);
            n += 1;
            c = -1;
        } else
            word[c] = tmpdata[i];

    }
    word[c] = '\0';
    insert_word(word, n);
    data[strlen(tmpdata)] = '\0';
}

ssize_t reverse_read(struct file *filep,char *buff,size_t count,loff_t *offp )
{
    if ( copy_to_user(buff,data,strlen(data)) != 0 ) {
        printk( "Kernel -> userspace copy failed!\n" );
        return -1;
    }
    return strlen(data);
}

ssize_t reverse_write(struct file *filep,const char *buff,size_t count,loff_t *offp )
{
    char tmpdata[DEVICE_SIZE+1];
    if ( copy_from_user(tmpdata,buff,count) != 0 ) {
        printk( "Userspace -> kernel copy failed!\n" );
        return -1;
    }
    reverse(tmpdata);
    return 0;
}

struct file_operations reverse_fops = {
    read: reverse_read,
    write: reverse_write
};

static struct miscdevice reverse_misc_device = {
    .minor = MISC_DYNAMIC_MINOR,
    .name = "reverse",
    .fops = &reverse_fops
```

```
};

static int __init reverse_init(void)
{
   misc_register(&reverse_misc_device);

      return 0;
}

static void __exit reverse_exit(void)
{
   misc_deregister(&reverse_misc_device);
}

module_init(reverse_init);
module_exit(reverse_exit);
```

Compile the file:
Makefile
obj-m += reverse.o

Output:

```
student@ubuntu:~$ gedit reverse.c
student@ubuntu:~$ gedit Makefile
student@ubuntu:~$ make -C /lib/modules/$(uname -r)/build M=$PWD modules
make: Entering directory '/usr/src/linux-headers-4.15.0-91-generic'
  CC [M]  /home/student/reverse.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/student/reverse.mod.o
  LD [M]  /home/student/reverse.ko
make: Leaving directory '/usr/src/linux-headers-4.15.0-91-generic'
student@ubuntu:~$
```

Reverse application:

```
#include <stdio.h>
#include <paths.h>
#include <string.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>

#define CDEV_DEVICE "reverse"
static char buf[512+1];

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

   if (argc != 2) {
      printf("Usage: %s <string>\n", argv[0]);
      exit(0);
   }

   if ((len = strlen(argv[1]) + 1) > 512) {
      printf("ERROR: String too long\n");
      exit(0);
   }

   if ((fd = open("/dev/" CDEV_DEVICE, O_RDWR)) == -1) {
      perror("/dev/" CDEV_DEVICE);
      exit(1);
   }

   printf("fd :%d\n",fd);

   if (read(fd, buf, len) == -1)
      perror("read()");
   else
      printf("Before: \"%s\".\n", buf);

   if (write(fd, argv[1], len) == -1)
      perror("write()");
   else
      printf("Wrote: \"%s\".\n", argv[1]);

   if (read(fd, buf, len) == -1)
      perror("read()");
   else
      printf("After: \"%s\".\n", buf);
```

```
    if ((close(fd)) == -1) {
        perror("close()");
        exit(1);
    }

    exit(0);
}
```

Output:

```
student@ubuntu:~$ sudo insmod ./reverse.ko
[sudo] password for student:
```

```
student@ubuntu:~$ lsmod | grep reverse
reverse                16384  0
```

```
student@ubuntu:~$ sudo ./test_reverse "this is testing of reverse string block"

fd :3
Before: "no data has been written yet".
Wrote: "this is testing of reverse string block".
After: "block string reverse of testing is this".
student@ubuntu:~$ sudo ./test_reverse "block string reverse of testing is this"

fd :3
Before: "block string reverse of testing is this".
Wrote: "block string reverse of testing is this".
After: "this is testing of reverse string block".
student@ubuntu:~$
```

Learning from the practical: Character devices can be very useful for userland/kernelland communication, this can be done with system calls to a degree but its a lot more difficult to implement a system call in an LKM.

When doing any kernel development, the kernel source is a necessity, you can download it from https://www.kernel.org/, see what version of the kernel you have, using uname -r, and download the correct source. Getting used to the kernel source will make you a much better kernel developer and ultimately a better rootkit developer.