Charotar University of Science and Technology
Faculty of Technology and Engineering

Devang Patel Institute of Advance Technology & Research
Computer Engineering Department

Subject Name & Code : INTERNALS OF OPERATING SYSTEM (CE347)	Semester: 6

Index Page

Sr. No.	Name of Practical	Page Number	Date of Practical completion
1	Write a C program to read 4 int numbers from user and find the sum product and average of those numbers.	**5**	**05/02/2021**
2	Write programs using the following system calls of UNIX operating system: fork, exec, getpid, exit, wait, stat, readdir, opendir.	**6**	**12/02/2021**
	Task-01 Write a program to execute fork() and find out the process id by getpid() system call.		
	Task-02 Write a program to execute following system call fork(), execl(), getpid(), exit(), wait() for a process.		
	Task-03 Write a program to find out status of named file (program of working stat() system call).		
	Task-04 Write a program for “ls” command implementation using opendir() & readdir() system call.		
3	**IMPLEMENTATION OF I/O SYSTEM CALL**	**13**	**19/02/2021**
	Task-01 Write a program to implement open, close, read and write System Call.		
	Task-02 Write a program to opens a particular file twice, write once and read once via two different file descriptors.		
	Task-03 Using system calls write line of texts in a file.		
	Task-04 Write a program to open, read and write files and perform file copy operation.		
4	**IMPLEMENTATION OF STRUCTURE OF BUFFER STRUCTURE**	**18**	**26/02/2021**
	Task-01 Write a program to create a doubly link list of n buffer.		
	Task-02 Write a program to implement LRU algorithm.		

	Task-03 Write a program to divide doubly link list of n nodes into n/4 sub list of hash queue. (value of n must be even and greater than 15).		
5	**IMPLEMENTATE SYSTEM CALL FOR FILE SYSTEM**	**24**	**05/03/2021**
	Task-01 Write a program to implement lseek() system call. A process seeks to 1024 bytes beyond the end of the file mapped by fd (file descriptor).		
	Task-02 Write a program in which, the child process waits for the user input and once an input is entered, it writes into the pipe. And the parent process reads from the pipe.		
	Task-03 Write a program in which, open a file’s once and read it twice. Use duplicate system call.		
6	**INTERPROCESS COMMUNICATION**	**28**	**12/03/2021**
	Task-01 Inter process communication (POSIX-IPC) using shared memory Study system calls: mmap(), shm_open(), shm_unlink()		
	Task-02 Inter process communication (POSIX-IPC) using pipe Study system call: pipe()		
7	**NETWORK COMMUNICATION**	**33**	**19/03/2021**
	Task-01 Write a program to perform client server communication using TCP.		
	Task-02 Write a program to perform client server communication using UDP.		
8	Case study: RTOS(Real Time Operating System)	**38**	**26/03/2021**
9	Case study: Network Operating System	**40**	**02/04/2021**
10	Case study: a Distributed File System for very large files	**42**	**09/04/2021**

Course Outcomes:

Upon successful completion of this subject, students should be.

CO1	Describe an overview of Kernel architecture and outlining basic concepts of file subsystem and process, Discover various functionalities of Kernel and apply basic understanding of kernel in subsequent modules.
CO2	Compare and Contrast Various Kernel memory allocators based on evaluation criteria and Discuss memory management policies and test the performance on various paging algorithms.
CO3	Examine internal structure of file and develop lower level file system algorithms.
CO4	Interpret and formulate context of a process theoretically and programmatically, examine how one process traces and controls the execution of other process and implement system calls that control the process context.
CO5	Review the traditional methods by which process communicate with processes on other machines over a network and to Summarize the characteristics of Distributed File system to conclude strength and weakness of network file systems.
CO6	Describe basics of system administration in operating system and Apply various operating system concepts to understand case study of RTOS and Network Operating system.

Sr. No.	**Name of Practical**	**Course Outcome**					
		CO1	**CO2**	**CO3**	**CO4**	**CO5**	**CO6**
1	Write a C program to read 4 int numbers from user and find the sum product and average of those numbers.	✓					
2	Write programs using the following system calls of UNIX operating system: fork, exec, getpid, exit, wait, stat, readdir, opendir.	✓			✓		
	Task-01 Write a program to execute fork() and find out the process id by getpid() system call.						
	Task-02 Write a program to execute following system call fork(), execl(), getpid(), exit(), wait() for a process.						
	Task-03 Write a program to find out status of named file (program of working stat() system call).						
	Task-04 Write a program for “ls” command implementation using opendir() & readdir() system call.						
3	**IMPLEMENTATION OF I/O SYSTEM CALL**	✓			✓		
	Task-01 Write a program to implement open, close, read and write System Call.						
	Task-02 Write a program to opens a particular file twice, write once and read once via two different file descriptors.						
	Task-03 Using system calls write line of texts in a file.						

	Task-04 Write a program to open, read and write files and perform file copy operation.						
4	**IMPLEMENTATION OF STRUCTURE OF BUFFER STRUCTURE**		✓				
	Task-01 Write a program to create a doubly link list of n buffer.						
	Task-02 Write a program to implement LRU algorithm.						
	Task-03 Write a program to divide doubly link list of n nodes into n/4 sub list of hash queue. (value of n must be even and greater than 15).						
5	**IMPLEMENTATE SYSTEM CALL FOR FILE SYSTEM**			✓	✓		
	Task-01 Write a program to implement lseek() system call. A process seeks to 1024 bytes beyond the end of the file mapped by fd (file descriptor).						
	Task-02 Write a program in which, the child process waits for the user input and once an input is entered, it writes into the pipe. And the parent process reads from the pipe.						
	Task-03 Write a program in which, open a file's once and read it twice. Use duplicate system call.						
6	**INTERPROCESS COMMUNICATION**					✓	
	Task-01 Inter process communication (POSIX-IPC) using shared memory Study system calls: mmap(), shm_open(), shm_unlink()						
	Task-02 Inter process communication (POSIX-IPC) using pipe Study system call: pipe()						
7	**NETWORK COMMUNICATION**					✓	
	Task-01 Write a program to perform client server communication using TCP.						
	Task-02 Write a program to perform client server communication using UDP.						
8	Case study: RTOS(Real Time Operating System)						✓
9	Case study: Network Operating System						✓
10	Case study: a Distributed File System for very large files					✓	

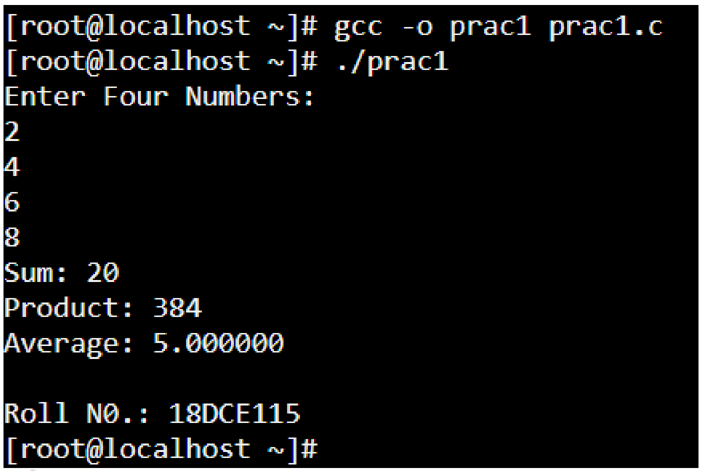
PRACTICAL - 1

Aim: Write a C program to read 4 int numbers from user and find the sum product and average of those numbers.

Input:

```
#include<stdlib.h>
Int main(){
        Int n1,n2,n3,n4;
        printf("Enter Four Numbers:\n");
        scanf("%d %d %d %d",&n1, &n2, &n3, &n4);
        printf("Sum: %d\n", n1+n2+n3+n4);
        printf("Product: %d\n", n1*n2*n3*n4);
        printf("Average: %f\n", (float)(n1+n2+n3+n4)/4);
        printf("\nRoll NO.: 18DCE115\n");
return 0;
}
```

Output:



```
[root@localhost ~]# gcc -o prac1 prac1.c
[root@localhost ~]# ./prac1
Enter Four Numbers:
2
4
6
8
Sum: 20
Product: 384
Average: 5.000000

Roll N0.: 18DCE115
[root@localhost ~]#
```

PRACTICAL - 2

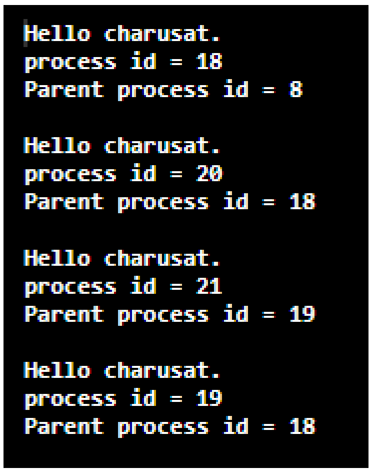
Aim: Write programs using the following system calls of UNIX operating system: fork, exec, getpid, exit, wait, stat, readdir, opendir.

2.1 - Write a program to execute fork() and find out the process id by getpid() system call.

Input:

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

void main()
{
        fork();
        fork();
        printf("Hello charusat. \nprocess id = %d \nParent process id = 
%d\n\n",getpid(),getppid());
}
```

Output:



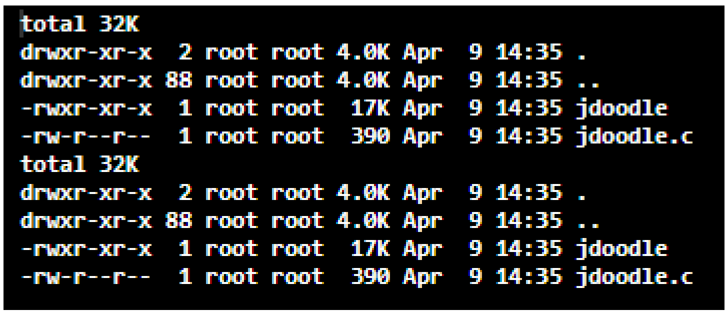
```
Hello charusat.
process id = 18
Parent process id = 8

Hello charusat.
process id = 20
Parent process id = 18

Hello charusat.
process id = 21
Parent process id = 19

Hello charusat.
process id = 19
Parent process id = 18
```

2.2 - Write a program to execute following system call fork(), execl(), getpid(), exit(), wait() for a process.

Input:

```
#include<unistd.h>#include<stdio.h>
#include<stdlib.h>#include<sys/types.h>
#include<sys/wait.h>
int main(int argc, char *argv[]){
        int p=fork();
        char *args[]={"hello","charusat",NULL};
        if(p==0)
                execv("./hello",args);
        wait(NULL);   printf("\n--------------------------------------------------------\n");
        execl("/bin/ls","/bin/ls", "-alh", (char *) NULL);
        return(0);
}
```

Output:



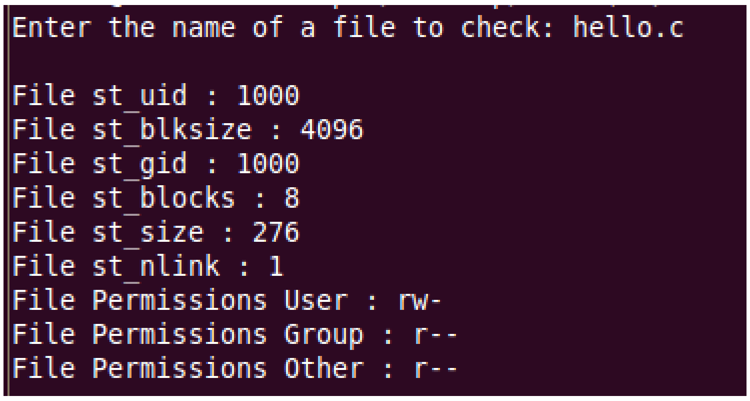
```
total 32K
drwxr-xr-x  2 root root 4.0K Apr  9 14:35 .
drwxr-xr-x 88 root root 4.0K Apr  9 14:35 ..
-rwxr-xr-x  1 root root  17K Apr  9 14:35 jdoodle
-rw-r--r--  1 root root  390 Apr  9 14:35 jdoodle.c
total 32K
drwxr-xr-x  2 root root 4.0K Apr  9 14:35 .
drwxr-xr-x 88 root root 4.0K Apr  9 14:35 ..
-rwxr-xr-x  1 root root  17K Apr  9 14:35 jdoodle
-rw-r--r--  1 root root  390 Apr  9 14:35 jdoodle.c
```

2.3 - Write a program to find out status of named file (program of working stat() system call).

Input:

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

void sfile(char const filename[]);

int main(){
  ssize_t read;
  char* buffer = 0;
  size_t buf_size = 0;

  printf("Enter the name of a file to check: ");
  read = getline(&buffer, &buf_size, stdin);

  if (read <=0 ){
    printf("getline failed\n");
    exit(1);
  }

  if (buffer[read-1] == '\n'){
    buffer[read-1] = 0;
  }

  int s=open(buffer,O_RDONLY);
  if(s==-1){
    printf("File doesn't exist\n");
    exit(1);
  }
  else{
    sfile(buffer);
  }
  free(buffer);
  return 0;
}

void sfile(char const filename[]){

  struct stat sfile;

  if(stat(filename,&sfile)==-1){
    printf("Error Occurred\n");
  }

  printf("\nFile st_uid : %d ",sfile.st_uid);
  printf("\nFile st_blksize : %ld",sfile.st_blksize);
  printf("\nFile st_gid : %d",sfile.st_gid);
  printf("\nFile st_blocks : %ld ",sfile.st_blocks);
  printf("\nFile st_size : %ld ",sfile.st_size);
  printf("\nFile st_nlink : %u ",(unsigned int)sfile.st_nlink);
  printf("\nFile Permissions User : ");
  printf((sfile.st_mode & S_IRUSR)? "r":"-");
  printf((sfile.st_mode & S_IWUSR)? "w":"-");
  printf((sfile.st_mode & S_IXUSR)? "x":"-");
  printf("\nFile Permissions Group : ");
  printf((sfile.st_mode & S_IRGRP)? "r":"-");
  printf((sfile.st_mode & S_IWGRP)? "w":"-");
  printf((sfile.st_mode & S_IXGRP)? "x":"-");
  printf("\nFile Permissions Other : ");
  printf((sfile.st_mode & S_IROTH)? "r":"-");
  printf((sfile.st_mode & S_IWOTH)? "w":"-");
  printf((sfile.st_mode & S_IXOTH)? "x":"-");
  printf("\n\n");
}
```

Output:



```
Enter the name of a file to check: hello.c

File st_uid : 1000
File st_blksize : 4096
File st_gid : 1000
File st_blocks : 8
File st_size : 276
File st_nlink : 1
File Permissions User : rw-
File Permissions Group : r--
File Permissions Other : r--
```

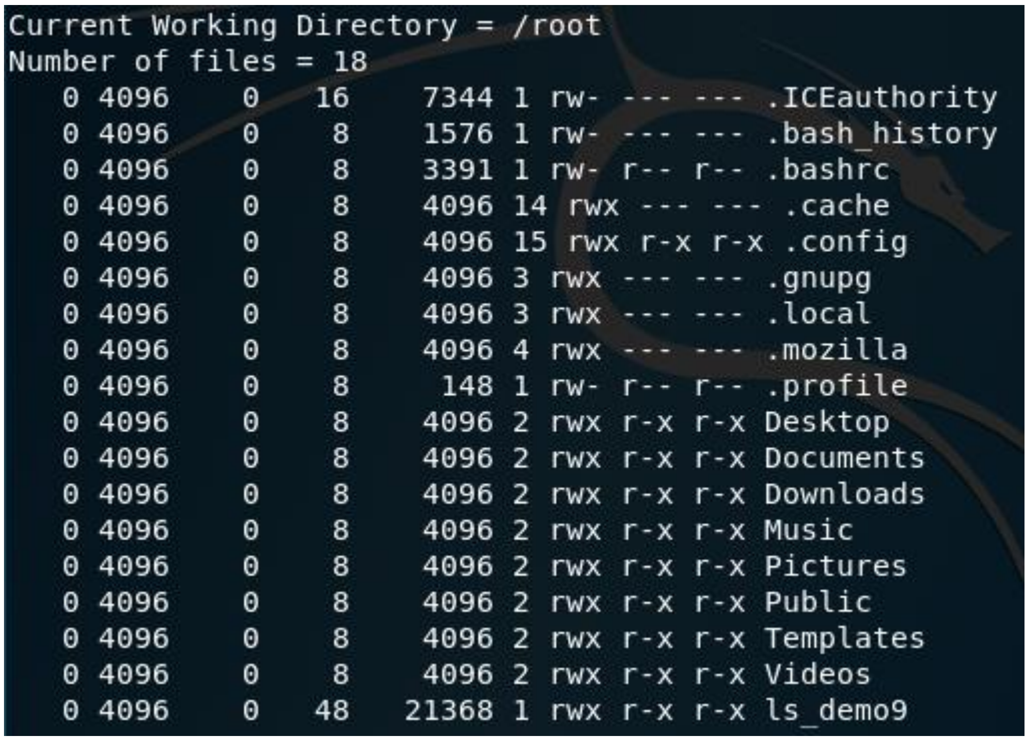
2.4 - Write a program for "ls" command implementation using opendir() & readdir() system call.

Input:

```
#include <sys/types.h>
#include <sys/dir.h>
#include <sys/param.h>
#include <stdio.h>
#include <stdlib.h>
#include<sys/stat.h>
#include<fcntl.h>

#define FALSE 0
#define TRUE !FALSE

extern  int alphasort(); //Inbuilt sorting function

char pathname[MAXPATHLEN]="./";

void die(char *msg)
{
  perror(msg);
  exit(0);
}

int file_select(struct direct *entry)
{
    if ((strcmp(entry->d_name, ".") == 0) || (strcmp(entry->d_name, "..") == 0))
        return (FALSE);
    else
        return (TRUE);
}

int main()
{
    int count,i;
    struct direct **files;

    if(!getcwd(pathname, sizeof(pathname)))
        die("\nError getting pathname\n");

    printf("\nCurrent Working Directory = %s\n",pathname);
    count = scandir(pathname, &files, file_select, alphasort);

    /* If no files found, make a non-selectable menu item */
    if(count <= 0)
```

```
      die("No files in this directory\n");

    printf("Number of files = %d\n",count);
    for (i=1; i<count+1; ++i)
        sfile(files[i-1]->d_name);
    printf("\n"); /* flush buffer */
}

void sfile(char const filename[]){

  struct stat sfile;

  if(stat(filename,&sfile)==-1){
    printf("Error Occurred\n");
  }

  printf("%4d ",sfile.st_uid);
  printf("%4ld ",sfile.st_blksize);
  printf("%4d ",sfile.st_gid);
  printf("%4ld ",sfile.st_blocks);
  printf("%7ld ",sfile.st_size);
  printf("%u ",(unsigned int)sfile.st_nlink);
  printf((sfile.st_mode & S_IRUSR)? "r":"-");
  printf((sfile.st_mode & S_IWUSR)? "w":"-");
  printf((sfile.st_mode & S_IXUSR)? "x":"-");
  printf(" ");
  printf((sfile.st_mode & S_IRGRP)? "r":"-");
  printf((sfile.st_mode & S_IWGRP)? "w":"-");
  printf((sfile.st_mode & S_IXGRP)? "x":"-");
  printf(" ");
  printf((sfile.st_mode & S_IROTH)? "r":"-");
  printf((sfile.st_mode & S_IWOTH)? "w":"-");
  printf((sfile.st_mode & S_IXOTH)? "x":"-");
  printf(" %s ",filename);
  printf("\n");
}
```

Output:



```
Current Working Directory = /root
Number of files = 18
   0 4096    0   16    7344 1 rw- --- --- .ICEauthority
   0 4096    0    8    1576 1 rw- --- --- .bash_history
   0 4096    0    8    3391 1 rw- r-- r-- .bashrc
   0 4096    0    8    4096 14 rwx --- --- .cache
   0 4096    0    8    4096 15 rwx r-x r-x .config
   0 4096    0    8    4096 3 rwx --- --- .gnupg
   0 4096    0    8    4096 3 rwx --- --- .local
   0 4096    0    8    4096 4 rwx --- --- .mozilla
   0 4096    0    8     148 1 rw- r-- r-- .profile
   0 4096    0    8    4096 2 rwx r-x r-x Desktop
   0 4096    0    8    4096 2 rwx r-x r-x Documents
   0 4096    0    8    4096 2 rwx r-x r-x Downloads
   0 4096    0    8    4096 2 rwx r-x r-x Music
   0 4096    0    8    4096 2 rwx r-x r-x Pictures
   0 4096    0    8    4096 2 rwx r-x r-x Public
   0 4096    0    8    4096 2 rwx r-x r-x Templates
   0 4096    0    8    4096 2 rwx r-x r-x Videos
   0 4096    0   48   21368 1 rwx r-x r-x ls_demo9
```

PRACTICAL - 3

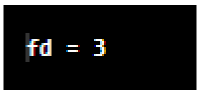
Aim: Implementation of I/O System Call

3.1 - Write a program to implement open, close, read and write System Call.

Open system call

Input:
```
#include<stdio.h>
#include<fcntl.h>
#include<errno.h>
extern int errno;
int main()
{
   int fd = open("foo.txt", O_RDONLY | O_CREAT);

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

   if (fd ==-1)
   {
      printf("Error Number % d\n", errno);
      perror("Program");
   }
   return 0;
}
```

Output:

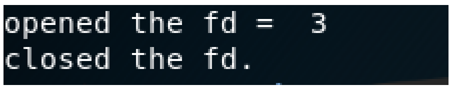

```
fd = 3
```

Close system call

Input:
```
#include<stdio.h>
#include <fcntl.h>
int main()
{
   int fd1 = open("foo.txt", O_RDONLY);
   if (fd1 < 0)
   {
      perror("c1");
      exit(1);
   }
   printf("opened the fd = % d\n", fd1);
   if (close(fd1) < 0)
```

```
    {
        perror("c1");
        exit(1);
    }
    printf("closed the fd.\n");
}
```

Output:



```
opened the fd =  3
closed the fd.
```

Read system call

Input:

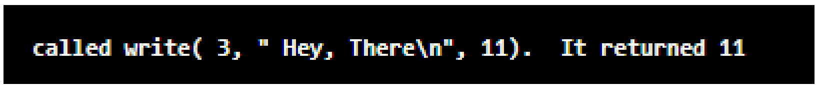
```
#include<stdio.h>
#include <fcntl.h>
int main()
{
  int fd, sz;
  char *c = (char *) calloc(100, sizeof(char));
  fd = open("foo.txt", O_RDONLY);
  if (fd < 0) { perror("r1"); exit(1); }
            sz = read(fd, c, 20);
  printf("called read(% d, c, 20).  returned that %d bytes  were read.\n", fd, sz);
  c[sz] = '\0';
  printf("Those bytes are as follows: % s\n", c);
}
```

Write system call

Input:

```
#include<stdio.h>
#include <fcntl.h>
main()
{
  int sz;
  int fd = open("foo.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
  if (fd < 0)
  {
     perror("r1");
     exit(1);
  }

  sz = write(fd, "Hey, There.\n", strlen("Hey, There\n"));
  printf("called write(% d, \"Hey, There\\n\", %d It returned %d\n", fd, strlen("Hey, There\n"),
sz);

  close(fd);
}
```

Output:



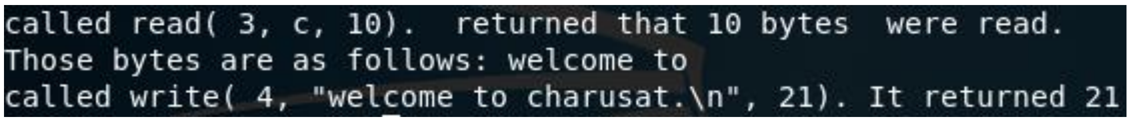
```
called write( 3, " Hey, There\n", 11).  It returned 11
```

3.2 - Write a program to opens a particular file twice, write once and read once via two different file descriptors.

Input:

```
#include<stdio.h>
#include <fcntl.h>
int main()
{
  int fd, sz,sz1;
  char *c = (char *) calloc(100, sizeof(char));

  fd = open("foo.txt", O_RDONLY);
  if (fd < 0)
  {
    perror("r1");
    exit(1);
  }
  sz = read(fd, c, 10);
  printf("called read(% d, c, 10).  returned that %d bytes  were read.\n", fd, sz);
  c[sz] = '\0';
  printf("Those bytes are as follows: % s\n", c);
  int fd1 = open("foo.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
  if (fd1 < 0)
  {
     perror("r1");
     exit(1);
  }
  sz1 = write(fd1, "welcome to charusat.\n", strlen("welcome to charusat.\n"));
  printf("called write(% d, \"welcome to charusat.\\n\", %d). It returned %d\n", fd1,
strlen("welcome to charusat.\n"), sz1);
  close(fd);
  close(fd1);
}
```

Output:



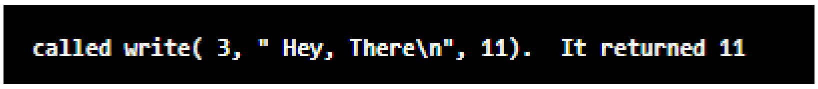
```
called read( 3, c, 10).  returned that 10 bytes  were read.
Those bytes are as follows: welcome to
called write( 4, "welcome to charusat.\n", 21). It returned 21
```

3.3 Using system calls write line of texts in a file.

Input:

```
#include<stdio.h>
#include <fcntl.h>
main()
{
  int sz;
  int fd = open("foo.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
  if (fd < 0)
  {
     perror("r1");
     exit(1);
  }

  sz = write(fd, "Hey, There\n", strlen("Hey, There\n"));

  printf("called write(% d, \"Hey, There\\n\", %d It returned %d\n", fd, strlen("Hey, There\n"),
sz);

  close(fd);
}
```

Output:



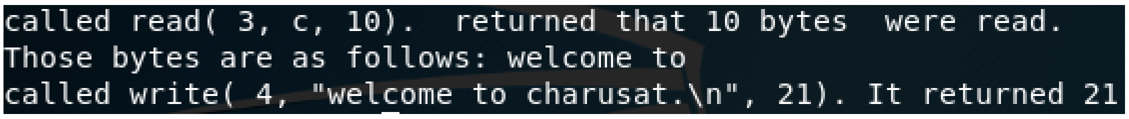
```
called write( 3, " Hey, There\n", 11).  It returned 11
```

3.4 Write a program to open, read and write files and perform file copy operation.
Input:

```
#include<stdio.h>
#include <fcntl.h>
int main()
{
  int fd, sz,sz1;
  char *c = (char *) calloc(100, sizeof(char));

  fd = open("foo.txt", O_RDONLY);
```

```
 if (fd < 0)
 {
   perror("r1");
   exit(1);
 }
 sz = read(fd, c, 10);
 printf("called read(% d, c, 10).  returned that %d bytes  were read.\n", fd, sz);
 c[sz] = '\0';
 printf("Those bytes are as follows: % s\n", c);
 int fd1 = open("foo1.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
 if (fd1 < 0)
 {
    perror("r1");
    exit(1);
 }
 sz1 = write(fd1, c, strlen(c));
 printf("called write(% d,\"c\", %d). It returned %d\n", fd1, strlen(c), sz1);
 close(fd);
 close(fd1);
}
```

Output:



```
called read( 3, c, 10).  returned that 10 bytes  were read.
Those bytes are as follows: welcome to
called write( 4, "welcome to charusat.\n", 21). It returned 21
```

PRACTICAL - 4

Aim: Implementation of structure of buffer structure

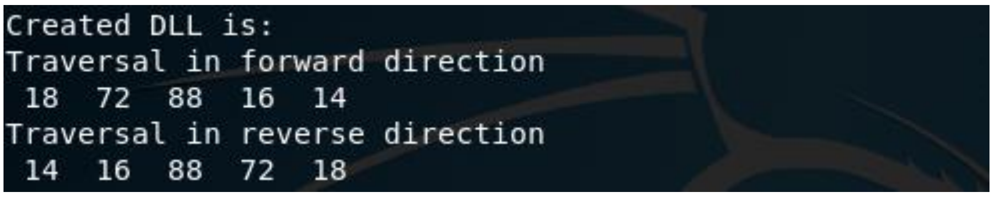
4.1 Write a program to create a doubly link list of n buffer.

Input:

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

struct Node {
        int data;
        struct Node* next;
        struct Node* prev;
};

void push(struct Node** head_ref, int new_data)
{
        struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
        new_node->data = new_data;
        new_node->next = (*head_ref);
        new_node->prev = NULL;
        if ((*head_ref) != NULL)
                (*head_ref)->prev = new_node;
        (*head_ref) = new_node;
}

void insertAfter(struct Node* prev_node, int new_data)
{
        if (prev_node == NULL) {
                printf("the given previous node cannot be NULL");
                return;
        }
        struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
        new_node->data = new_data;
        new_node->next = prev_node->next;
        prev_node->next = new_node;
        new_node->prev = prev_node;
        if (new_node->next != NULL)
                new_node->next->prev = new_node;
}

void append(struct Node** head_ref, int new_data)
{
        struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
        struct Node* last = *head_ref; /* used in step 5*/
```

```
	new_node->data = new_data;
	new_node->next = NULL;
	if (*head_ref == NULL) {
		new_node->prev = NULL;
		*head_ref = new_node;
		return;
	}
	while (last->next != NULL)
		last = last->next;
	last->next = new_node;
	new_node->prev = last;
	return;
}

void printList(struct Node* node)
{
	struct Node* last;
	printf("\nTraversal in forward direction \n");
	while (node != NULL) {
		printf(" %d ", node->data);
		last = node;
		node = node->next;
	}

	printf("\nTraversal in reverse direction \n");
	while (last != NULL) {
		printf(" %d ", last->data);
		last = last->prev;
	}
}

int main()
{
	struct Node* head = NULL;
	append(&head, 16);
	push(&head, 72);
	push(&head, 18);
	append(&head, 14);
	insertAfter(head->next, 88);

	printf("Created DLL is: ");
	printList(head);

	getchar();
	return 0;
}
```

Output:



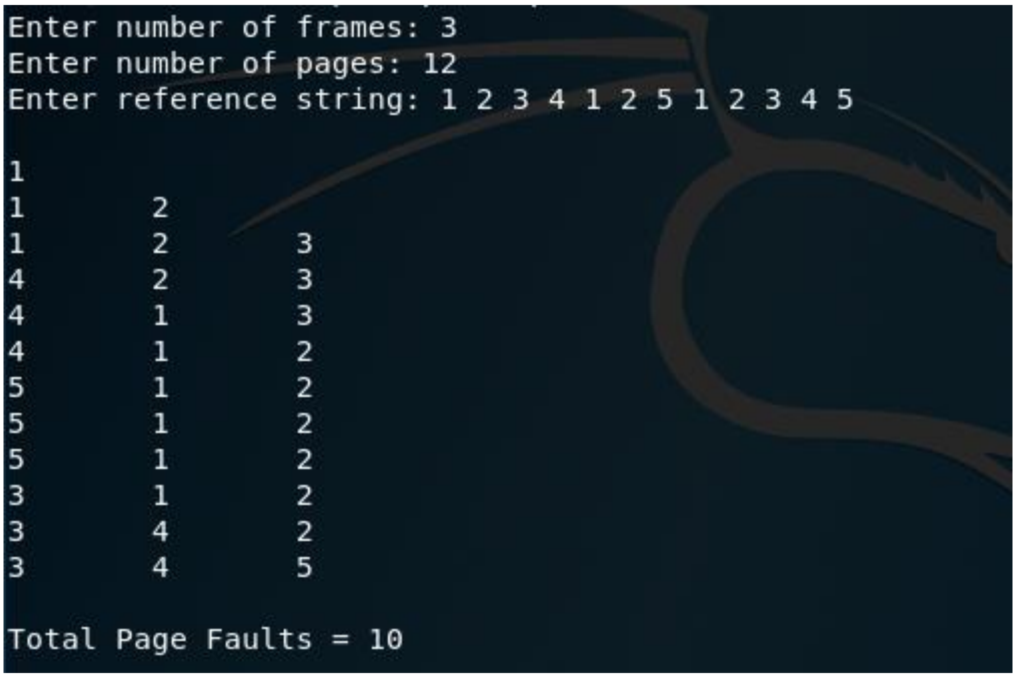
```
Created DLL is:
Traversal in forward direction
 18  72  88  16  14
Traversal in reverse direction
 14  16  88  72  18
```

4.2 Write a program to implement LRU algorithm.

Input:

```
#include<stdio.h>

int findLRU(int time[], int n){
	int i, minimum = time[0], pos = 0;

	for(i = 1; i < n; ++i){
		if(time[i] < minimum){
			minimum = time[i];
			pos = i;
		}
	}

	return pos;
}

int main()
{
    int no_of_frames, no_of_pages, frames[10], pages[30], counter = 0, time[10], flag1, flag2,
i, j, pos, faults = 0;
	printf("Enter number of frames: ");
	scanf("%d", &no_of_frames);

	printf("Enter number of pages: ");
	scanf("%d", &no_of_pages);

	printf("Enter reference string: ");

    for(i = 0; i < no_of_pages; ++i){
	scanf("%d", &pages[i]);
    }

	for(i = 0; i < no_of_frames; ++i){
	frames[i] = -1;
    }
   for(i = 0; i < no_of_pages; ++i){
        flag1 = flag2 = 0;

        for(j = 0; j < no_of_frames; ++j){
                if(frames[j] == pages[i]){
                        counter++;
                        time[j] = counter;
                                flag1 = flag2 = 1;
                                break;
                        }
        }

        if(flag1 == 0){
                        for(j = 0; j < no_of_frames; ++j){
                        if(frames[j] == -1){
                                counter++;
                                faults++;
                                frames[j] = pages[i];
                                time[j] = counter;
                                flag2 = 1;
                                break;
                        }
                }
        }

        if(flag2 == 0){
                pos = findLRU(time, no_of_frames);
                counter++;
                faults++;
                frames[pos] = pages[i];
                time[pos] = counter;
        }

        printf("\n");

        for(j = 0; j < no_of_frames; ++j){
                if(frames[j] != -1 )
                        printf("%d\t", frames[j]);
        }
        }

        printf("\n\nTotal Page Faults = %d\n\n", faults);

   return 0;
}
```

Output:



```
Enter number of frames: 3
Enter number of pages: 12
Enter reference string: 1 2 3 4 1 2 5 1 2 3 4 5

1
1       2
1       2       3
4       2       3
4       1       3
4       1       2
5       1       2
5       1       2
5       1       2
3       1       2
3       4       2
3       4       5

Total Page Faults = 10
```

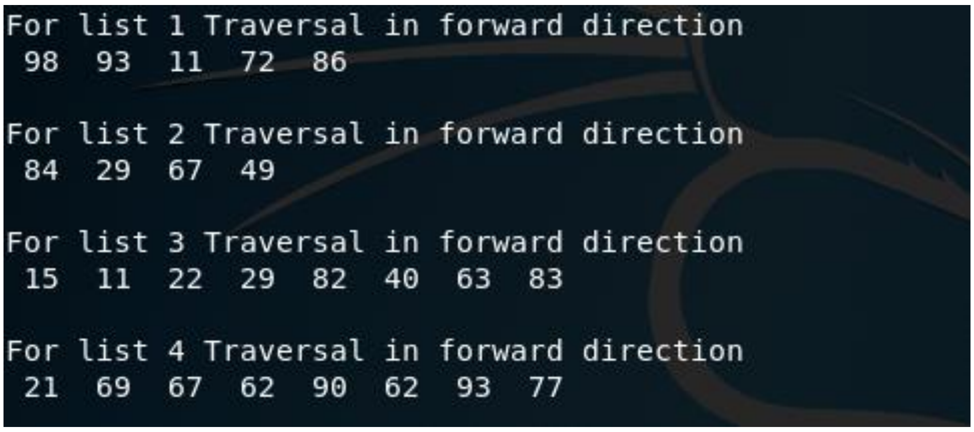
<u>4.3</u> Write a program to divide doubly link list of n nodes into n/4 sub list of hash queue. (Value of n must be even and greater than 15).

Input:

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

struct Node {
        int data;
        struct Node* next;
        struct Node* prev;
};

void push(struct Node** head_ref, int new_data)
{
        struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
        new_node->data = new_data;
        new_node->next = (*head_ref);
        new_node->prev = NULL;
        if ((*head_ref) != NULL)
                (*head_ref)->prev = new_node;
        (*head_ref) = new_node;
}
```

```
void printList(struct Node* node)
{
	struct Node* last;
	printf("Traversal in forward direction \n");
	while (node != NULL) {
		printf(" %d ", node->data);
		last = node;
		node = node->next;
	}
	printf("\n\n");
}


int main()
{
	struct Node* head[4] = {NULL,NULL,NULL,NULL};

	for(int i=0;i<25;i++)
		push(&head[rand()%4],rand()%100);

	for(int i=0;i<4;i++)
	{
		printf("For list %d ",(i+1));
		printList(head[i]);
	}

	return 0;
}
```

Output:



```
For list 1 Traversal in forward direction
 98  93  11  72  86

For list 2 Traversal in forward direction
 84  29  67  49

For list 3 Traversal in forward direction
 15  11  22  29  82  40  63  83

For list 4 Traversal in forward direction
 21  69  67  62  90  62  93  77
```

PRACTICAL - 5

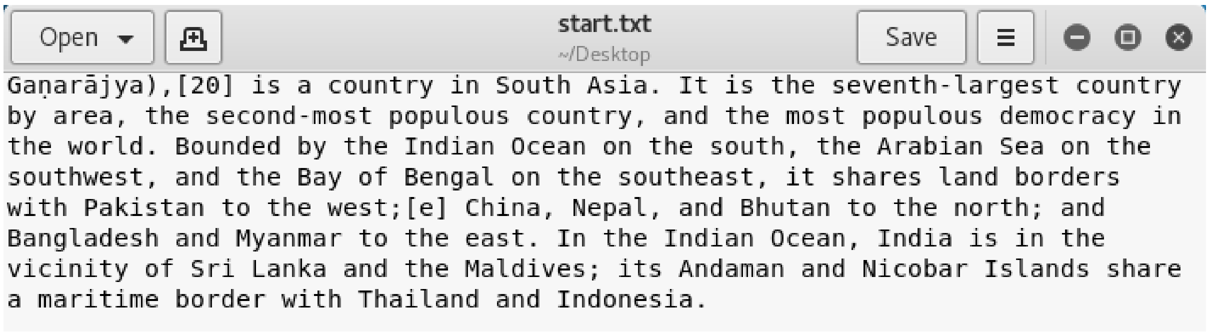
Aim: Implementate system call for file system

5.1 Write a program to implement lseek() system call. A process seeks to 1024 bytes beyond the end of the file mapped by fd (file descriptor).

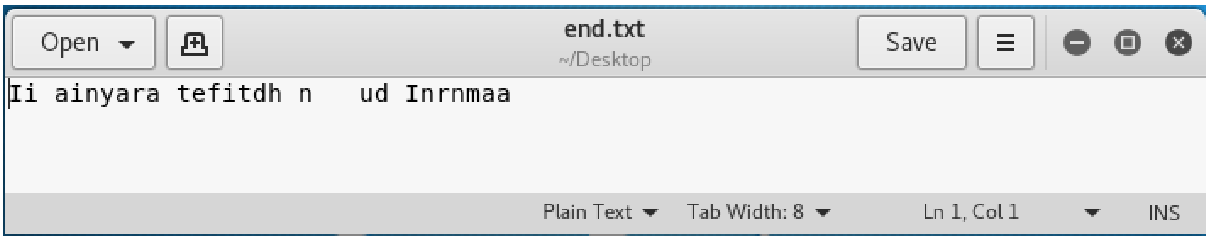
Input:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <fcntl.h>

int main()
{
	char arr[100];
	int n=1024;
	int f_write = open("start.txt", O_RDONLY);
	int f_read = open("end.txt", O_WRONLY);

	int count = 0;
	while (read(f_write, arr, 1))
	{

		if (count < n)
		{
			lseek (f_write, n, SEEK_CUR);
			write (f_read, arr, 1);
			count = n;
		}

		else
		{
			count = (2*n);
			lseek(f_write, count, SEEK_CUR);
			write(f_read, arr, 1);
		}
	}
	close(f_write);
	close(f_read);
	return 0;
}
```

Output:



```
Gaṇarājya),[20] is a country in South Asia. It is the seventh-largest country
by area, the second-most populous country, and the most populous democracy in
the world. Bounded by the Indian Ocean on the south, the Arabian Sea on the
southwest, and the Bay of Bengal on the southeast, it shares land borders
with Pakistan to the west;[e] China, Nepal, and Bhutan to the north; and
Bangladesh and Myanmar to the east. In the Indian Ocean, India is in the
vicinity of Sri Lanka and the Maldives; its Andaman and Nicobar Islands share
a maritime border with Thailand and Indonesia.
```



```
Ii ainyara tefitdh n   ud Inrnmaa
```

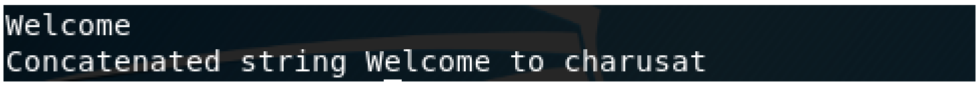
5.2 Write a program in which, the child process waits for the user input and once an input is entered, it writes into the pipe. And the parent process reads from the pipe.

Input:

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<string.h>
#include<sys/wait.h>
int main()
{
        int fd1[2];
        int fd2[2];
        char fixed_str[] = " to charusat";
        char input_str[100];
        pid_t p;
        if (pipe(fd1)==-1)
        {
                fprintf(stderr, "Pipe Failed" );
                return 1;
        }
        if (pipe(fd2)==-1)
        {
                fprintf(stderr, "Pipe Failed" );
                return 1;
        }
```

```
	scanf("%s", input_str);
	p = fork();
	if (p < 0)
	{
		fprintf(stderr, "fork Failed" );
		return 1;
	}
	else if (p > 0)
	{
		char concat_str[100];
		close(fd1[0]);
		write(fd1[1], input_str, strlen(input_str)+1);
		close(fd1[1]);
		wait(NULL);
		close(fd2[1]);
		read(fd2[0], concat_str, 100);
		printf("Concatenated string %s\n", concat_str);
		close(fd2[0]);
	}
	else
	{
		close(fd1[1]);
		char concat_str[100];
		read(fd1[0], concat_str, 100);
		int k = strlen(concat_str);
		int i;
		for (i=0; i<strlen(fixed_str); i++)
			concat_str[k++] = fixed_str[i];

		concat_str[k] = '\0';
		close(fd1[0]);
		close(fd2[0]);
		write(fd2[1], concat_str, strlen(concat_str)+1);
		close(fd2[1]);
		exit(0);
	}
}
```

Output:



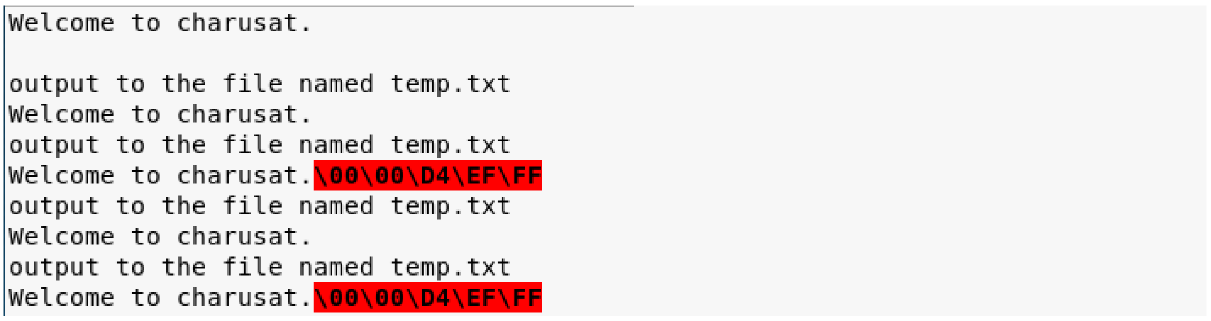
```
Welcome
Concatenated string Welcome to charusat
```

5.3 Write a program in which, open a file's once and read it twice. Use duplicate system call.

Input:

```
#include<stdio.h>
#include <unistd.h>
#include <fcntl.h>
int main()
{       int file_desc = open("temp.txt", O_WRONLY | O_APPEND);
        if(file_desc < 0)
                printf("Error opening the file\n");
        int copy_desc = dup(file_desc);
        write(copy_desc,"output to the file named temp.txt\n", 46);
        write(file_desc,"output to the file named temp.txt\n", 51);
        return 0;
}
```

Output:



```
Welcome to charusat.

output to the file named temp.txt
Welcome to charusat.
output to the file named temp.txt
Welcome to charusat.\00\00\D4\EF\FF
output to the file named temp.txt
Welcome to charusat.
output to the file named temp.txt
Welcome to charusat.\00\00\D4\EF\FF
```

PRACTICAL - 6

Aim: InterProcess Communication

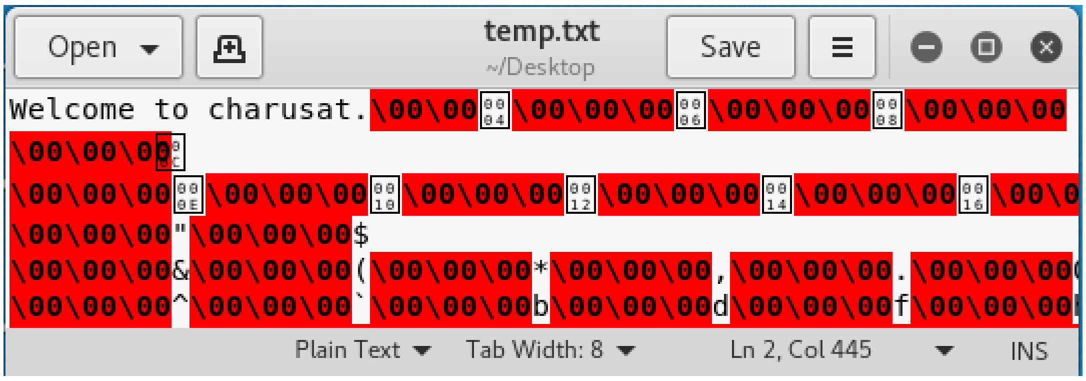
6.1 Inter process communication (POSIX-IPC) using shared memory Study system calls: mmap(), shm_open(), shm_unlink()

Mmap system call

Input:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>

#define FILEPATH "./temp.txt"
#define NUMINTS  (1000)
#define FILESIZE (NUMINTS * sizeof(int))

int main(int argc, char *argv[])
{
   int i,fd,result;
   int *map;

   fd = open(FILEPATH, O_RDWR | O_CREAT | O_TRUNC, (mode_t)0600);
   if (fd == -1) {
        perror("Error opening file for writing");
        exit(EXIT_FAILURE);
   }

   result = lseek(fd, FILESIZE-1, SEEK_SET);
   if (result == -1) {
        close(fd);
        perror("Error calling lseek() to 'stretch' the file");
        exit(EXIT_FAILURE);
   }

   result = write(fd, "", 1);
   if (result != 1) {
        close(fd);
        perror("Error writing last byte of the file");
        exit(EXIT_FAILURE);
   }
   map = mmap(0, FILESIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
   if (map == MAP_FAILED) {
           close(fd);
           perror("Error mmapping the file");
           exit(EXIT_FAILURE);
   }

   for (i = 1; i <=NUMINTS; ++i)
           map[i] = 2 * i;

   if (munmap(map, FILESIZE) == -1)
           perror("Error un-mmapping the file");

   close(fd);
   return 0;
}
```

Output:



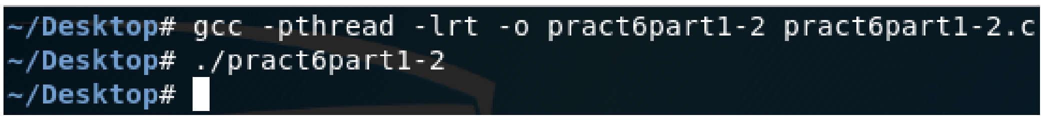
```
Welcome to charusat.\00\00 0004\00\00\00 0006\00\00\00 0008\00\00\00
\00\00\00 000C
\00\00\00 000E\00\00\00 0010\00\00\00 0012\00\00\00 0014\00\00\00 0016\00\0
\00\00\00"\00\00\00$
\00\00\00&\00\00\00(\00\00\00*\00\00\00,\00\00\00.\00\00\00
\00\00\00^\00\00\00`\00\00\00b\00\00\00d\00\00\00f\00\00\00
```

<u>**Shm_open system call**</u>

Input:

```
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>
#include <sys/file.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <errno.h>

int main()
{
        int fd;
        caddr_t pg_addr;
        int size = 5000;
        int mode =  S_IRWXO|S_IRWXG|S_IRWXU;

        fd = shm_open("memory", O_RDWR|O_CREAT, mode);
        if(fd < 0){
          perror("open error ");
          return 0;
        }

        if((ftruncate(fd, size)) == -1){
            perror("ftruncate failure");
            return 0;
        }

        pg_addr = (caddr_t) mmap(0, size, PROT_READ|PROT_WRITE|PROT_EXEC,
MAP_SHARED, fd, 0);

        if(pg_addr == (caddr_t) -1){
          perror("mmap failure");
          return 0;
        }

        if(mlock(pg_addr,size) != 0){
          perror("mlock failure");
          return 0;
        }

        if(munmap(pg_addr, size) < 0)
          perror("unmap error");
        close(fd);
        return 0;
}
```

Output:



```
~/Desktop# gcc -pthread -lrt -o pract6part1-2 pract6part1-2.c
~/Desktop# ./pract6part1-2
~/Desktop# 
```

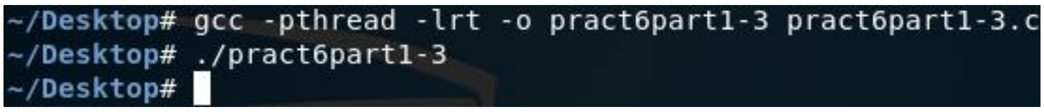
<u>**Shm_unlink system call**</u>

Input:

```
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>
#include <sys/file.h>
```

```
#include <sys/mman.h>
#include <sys/stat.h>
#include <errno.h>

int main()
{
        int fd;
        caddr_t pg_addr;

        int size = 5000;
        int mode =  S_IRWXO|S_IRWXG|S_IRWXU;

        fd = shm_open("memory", O_RDWR|O_CREAT, mode);
        if(fd < 0){
          perror("open error ");
          return 0;
        }

        if((ftruncate(fd, size)) == -1){
            perror("ftruncate failure");
            return 0;
        }

        pg_addr = (caddr_t) mmap(0, size, PROT_READ|PROT_WRITE|PROT_EXEC,
MAP_SHARED,
                        fd, 0);

        if(pg_addr == (caddr_t) -1){
          perror("mmap failure");
          return 0;
        }

        if(mlock(pg_addr,size) != 0){
          perror("mlock failure");
          return 0;
        }

        if(munmap(pg_addr, size) < 0)
          perror("unmap error");
        close(fd);
        shm_unlink("memory");
        return 0;
}
```

Output:



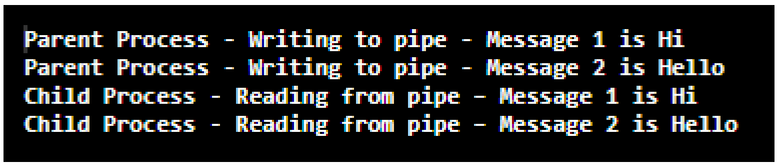
```
~/Desktop# gcc -pthread -lrt -o pract6part1-3 pract6part1-3.c
~/Desktop# ./pract6part1-3
~/Desktop# 
```

6.2 Inter process communication (POSIX-IPC) using pipe Study system call: pipe()

Input:

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

int main() {
   int pipefds[2];
   int returnstatus;
   int pid;
   char writemessages[2][20]={"Hi", "Hello"};
   char readmessage[20];
   returnstatus = pipe(pipefds);
   if (returnstatus == -1) {
      printf("Unable to create pipe\n");
      return 1;
   }
   pid = fork();

   if (pid == 0) {
      read(pipefds[0], readmessage, sizeof(readmessage));
      printf("Child Process - Reading from pipe – Message 1 is %s\n", readmessage);
      read(pipefds[0], readmessage, sizeof(readmessage));
      printf("Child Process - Reading from pipe – Message 2 is %s\n", readmessage);
   } else {
      printf("Parent Process - Writing to pipe - Message 1 is %s\n", writemessages[0]);
      write(pipefds[1], writemessages[0], sizeof(writemessages[0]));
      printf("Parent Process - Writing to pipe - Message 2 is %s\n", writemessages[1]);
      write(pipefds[1], writemessages[1], sizeof(writemessages[1]));
   }
   return 0;
}
```

Output:



```
Parent Process - Writing to pipe - Message 1 is Hi
Parent Process - Writing to pipe - Message 2 is Hello
Child Process - Reading from pipe – Message 1 is Hi
Child Process - Reading from pipe – Message 2 is Hello
```

PRACTICAL - 7

Aim: Network Communication

7.1 Write a program to perform client server communication using TCP.

Server.c

Input:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <time.h>
#include <unistd.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/in.h>

int main()
{
   time_t clock;
         char dataSending[1025];
         int clintListn = 0, clintConnt = 0;
         struct sockaddr_in ipOfServer;
         clintListn = socket(AF_INET, SOCK_STREAM, 0);
         memset(&ipOfServer, '0', sizeof(ipOfServer));
         memset(dataSending, '0', sizeof(dataSending));
         ipOfServer.sin_family = AF_INET;
         ipOfServer.sin_addr.s_addr = htonl(INADDR_ANY);
         ipOfServer.sin_port = htons(2017);
         bind(clintListn, (struct sockaddr*)&ipOfServer , sizeof(ipOfServer));
         listen(clintListn , 20);

         while(1)
         {
                  printf("\n\nHi,Iam running server.Some Client hit me\n");
                  clintConnt = accept(clintListn, (struct sockaddr*)NULL, NULL);

                  clock = time(NULL);
                  snprintf(dataSending, sizeof(dataSending), "%.24s\r\n", ctime(&clock));
                  write(clintConnt, dataSending, strlen(dataSending));

      close(clintConnt);
      sleep(1);
```

```
    }

    return 0;
}
```

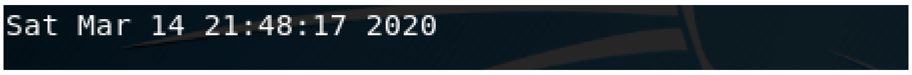
Client.c

Input:

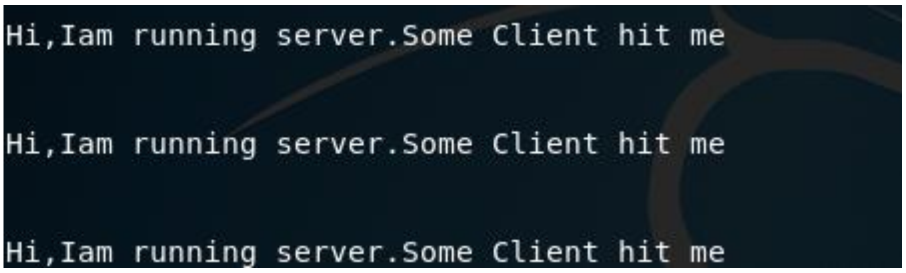
```
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <arpa/inet.h>

int main()
{
   int CreateSocket = 0,n = 0;
   char dataReceived[1024];
   struct sockaddr_in ipOfServer;

   memset(dataReceived, '0' ,sizeof(dataReceived));

   if((CreateSocket = socket(AF_INET, SOCK_STREAM, 0))< 0)
   {
      printf("Socket not created \n");
      return 1;
   }

   ipOfServer.sin_family = AF_INET;
   ipOfServer.sin_port = htons(2017);
   ipOfServer.sin_addr.s_addr = inet_addr("127.0.0.1");

   if(connect(CreateSocket, (struct sockaddr *)&ipOfServer, sizeof(ipOfServer))<0)
   {
      printf("Connection failed due to port and ip problems\n");
      return 1;
   }

   while((n = read(CreateSocket, dataReceived, sizeof(dataReceived)-1)) > 0)
   {
      dataReceived[n] = 0;
```

```
        if(fputs(dataReceived, stdout) == EOF)
            printf("\nStandard output error");
        printf("\n");
    }

    if( n < 0)
        printf("Standard input error \n");

    return 0;
}
```

Output:



```
Sat Mar 14 21:48:17 2020
```



```
Hi,Iam running server.Some Client hit me

Hi,Iam running server.Some Client hit me

Hi,Iam running server.Some Client hit me
```

7.2 Write a program to perform client server communication using UDP.

Server.c

Input:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>

#define PORT     8080
#define MAXLINE 1024

int main() {
    int sockfd;
    char buffer[MAXLINE];
    char *hello = "Hello from server";
    struct sockaddr_in servaddr, cliaddr;
    if ( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }

    memset(&servaddr, 0, sizeof(servaddr));
    memset(&cliaddr, 0, sizeof(cliaddr));

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

    if ( bind(sockfd, (const struct sockaddr *)&servaddr, sizeof(servaddr)) < 0 )
    {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }

    int len, n;

    len = sizeof(cliaddr);

    n = recvfrom(sockfd, (char *)buffer, MAXLINE,
                MSG_WAITALL, ( struct sockaddr *) &cliaddr, &len);
    buffer[n] = '\0';
    printf("Client : %s\n", buffer);
    sendto(sockfd, (const char *)hello, strlen(hello),
        MSG_CONFIRM, (const struct sockaddr *) &cliaddr, len);
    printf("Hello message sent.\n");

    return 0;
}
```

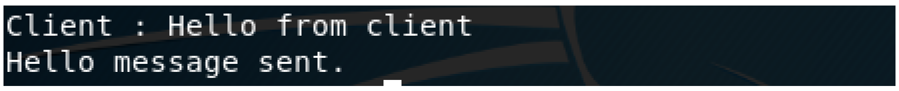
Client.c

Input:

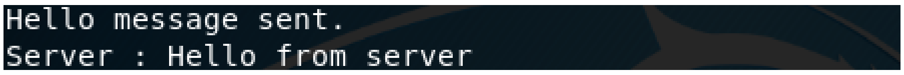
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
```

```
#define PORT     8080
#define MAXLINE 1024

int main() {
    int sockfd;
    char buffer[MAXLINE];
    char *hello = "Hello from client";
    struct sockaddr_in     servaddr;

    if ( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }

    memset(&servaddr, 0, sizeof(servaddr));

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

    int n, len;

    sendto(sockfd, (const char *)hello, strlen(hello),
        MSG_CONFIRM, (const struct sockaddr *) &servaddr,  sizeof(servaddr));
    printf("Hello message sent.\n");

    n = recvfrom(sockfd, (char *)buffer, MAXLINE,
                MSG_WAITALL, (struct sockaddr *) &servaddr,  &len);
    buffer[n] = '\0';
    printf("Server : %s\n", buffer);

    close(sockfd);
    return 0;
}
```

Output:



```
Client : Hello from client
Hello message sent.
```

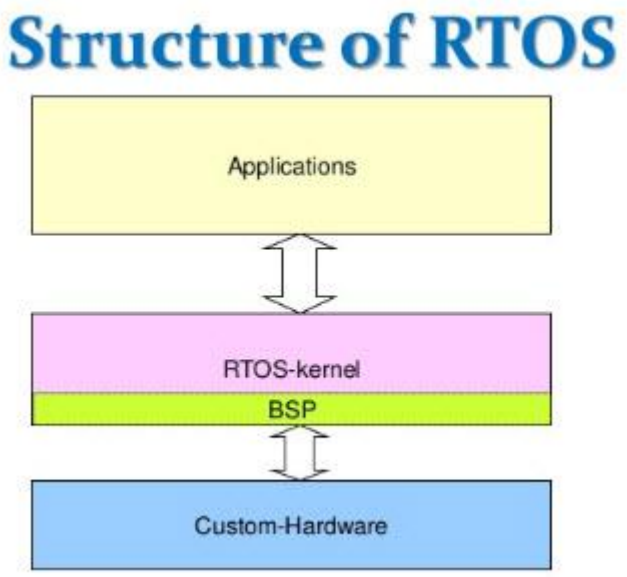


```
Hello message sent.
Server : Hello from server
```

PRACTICAL - 8

Aim: Case study: RTOS(Real Time Operating System)

- Most operating systems appear to allow multiple programs to execute at the same time.
- This is called multi-tasking. In reality, each processor core can only be running a single thread of execution at any given point in time.
- A part of the operating system called the scheduler is responsible for deciding which program to run when, and provides the illusion of simultaneous execution by rapidly switching between each program.
- The type of an operating system is defined by how the scheduler decides which program to run when. For example, the scheduler used in a multi user operating system (such as Unix) will ensure each user gets a fair amount of the processing time.
- As another example, the scheduler in a desk top operating system (such as Windows) will try and ensure the computer remains responsive to its user.
- The scheduler in a Real Time Operating System (RTOS) is designed to provide a predictable (normally described as deterministic) execution pattern. This is particularly of interest to embedded systems as embedded systems often have real time requirements.



- A real time requirements is one that specifies that the embedded system must respond to a certain event within a strictly defined time (the deadline).
- A guarantee to meet real time requirements can only be made if the behaviour of the operating system's scheduler can be predicted (and is therefore deterministic).
- Traditional real time schedulers, such as the scheduler used in FreeRTOS, achieve determinism by allowing the user to assign a priority to each thread of execution.
- The scheduler then uses the priority to know which thread of execution to run next.
- In FreeRTOS, a thread of execution is called a task.
- FreeRTOS is a class of RTOS that is designed to be small enough to run on a microcontroller – although its use is not limited to microcontroller applications.

- A microcontroller is a small and resource constrained processor that incorporates, on a single chip, the processor itself, read only memory (ROM or Flash) to hold the program to be executed, and the random access memory (RAM) needed by the programs it executes.
- Typically the program is executed directly from the read only memory.
- Microcontrollers are used in deeply embedded applications (those applications where you never actually see the processors themselves, or the software they are running) that normally have a very specific and dedicated job to do.
- The size constraints, and dedicated end application nature, rarely warrant the use of a full RTOS implementation – or indeed make the use of a full RTOS implementation possible.
- FreeRTOS therefore provides the core real time scheduling functionality, inter-task communication, timing and synchronisation primitives only.
- This means it is more accurately described as a real time kernel, or real time executive. Additional functionality, such as a command console interface, or networking stacks, can then be included with add-on components.

PRACTICAL - 9

Aim: Case study: Network Operating System

- A network operating system (NOS) is an operating system that manages network resources: essentially, an operating system that includes special functions for connecting computers and devices into a local area network (LAN).
- The NOS manages multiple requests (inputs) concurrently and provides the security necessary in a multiuser environment.
- It may be a completely self-contained operating system, such as NetWare, Unix, Windows 2000, or Mac OS X, or it may require an existing operating system in order to function (e.g., Windows 3.11 for Workgroups requires DOS; LAN Server requires OS/2; LANtastic requires DOS).
- In addition to file and print services, a NOS may also offer directory services and a messaging system (email), as well as network management and multiprotocol routing capabilities.

- An operating system (OS) is basically a collection of software that manages computer hardware resources and provides common services for computer programs.
- Operating system is a crucial component of the system software in a computer system.
- Network Operating System is one of the important type of operating system.
- Network Operating System runs on a server and gives the server the capability to manage data, users, groups, security, applications, and other networking functions.
- The basic purpose of the network operating system is to allow shared file and printer access among multiple computers in a network, typically a local area network (LAN), a private network or to other networks.
- Some examples of network operating systems include Microsoft Windows Server 2003, Microsoft Windows Server 2008, UNIX, Linux, Mac OS X, Novell NetWare, and BSD.
- A network operating system is an operating system designed for the sole purpose of supporting workstations, database sharing, application sharing and file and printer access sharing among multiple computers in a network.
- Certain standalone operating systems, such as Microsoft Windows NT and Digital’s OpenVMS, come with multipurpose capabilities and can also act as network operating systems.

- Some of the most well-known network operating systems include Microsoft Windows Server 2003, Microsoft Windows Server 2008, Linux and Mac OS X.

Advantages

- Centralized servers are highly stable.
- Security is server managed.
- Upgradation of new technologies and hardware can be easily integrated into the system.
- It is possible to remote access to servers from different locations and types of systems.

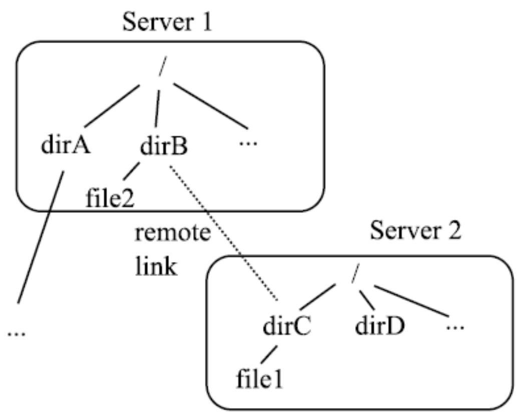
Disadvantages

- High cost of buying and running a server.
- Dependency on a central location for most operations.
- Regular maintenance and updates are required.

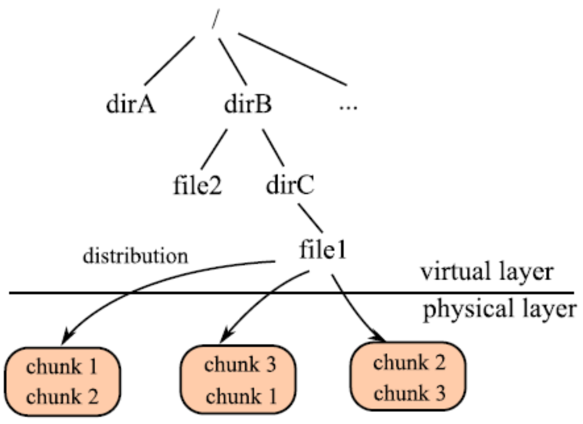
PRACTICAL - 10

Aim: Case study: a Distributed File System for very large files

- To conclude this introductory part, we study a simple distributed service: a file system that serves very large data files (hundreds of Gigabytes or Terabytes).
- The architecture presented here is a slightly simplified description of the Google File System and of several of its descendants, including the HADOOP Distributed File System (HDFS) available as an opensource project.
- The technical environment is that of a high speed local network connecting a cluster of servers. The file systems is designed to satisfy some specific requirements: (i) we need to handle very large collections of unstructured to semi-structured documents, (ii) data collections are written once and read many times, and (iii) the infrastructure that supports these components consists of thousands of connected machines, with high failure probability.
- These particularities make common distributed system tools only partially appropriate.



A traditional network file system



A large scale distributed file system

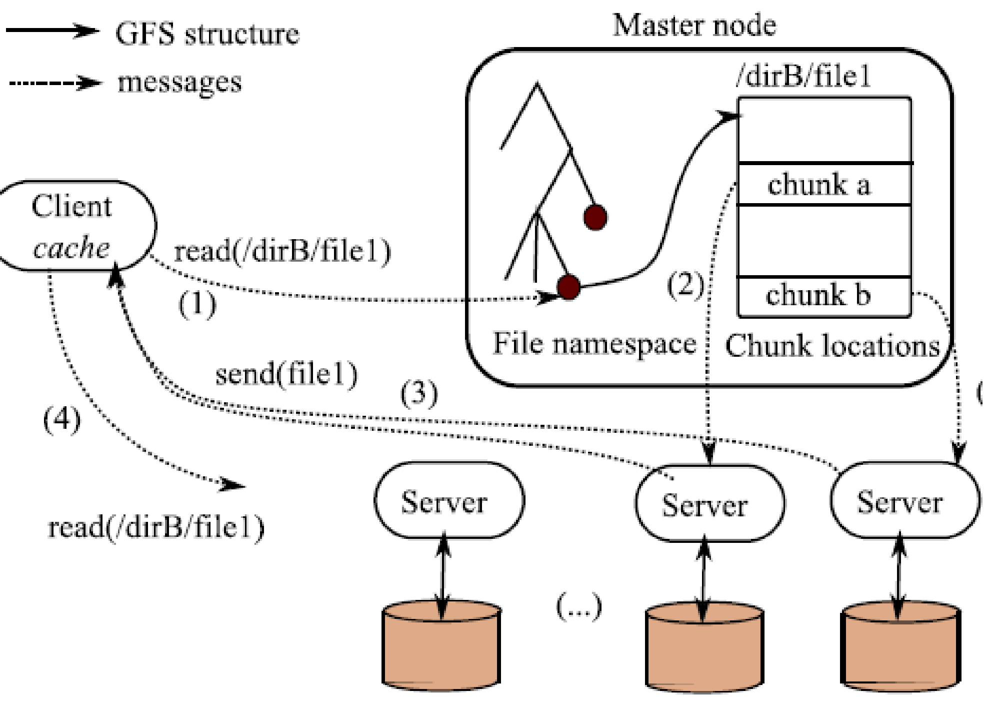
Architecture

We now turn to the architecture of GFS, summarized on Figure 14.10. The distributed system consists of a Master node and many server nodes. The Master plays the role of a coordinator: it receives Client connections, maintains the description of the global file system namespace, and the allocation of file chunks. The Master also monitors the state of the system with "heartbeat" messages in order to detect any failure as early as possible. The role of Servers is straight forward. They receive files chunks, and must take appropriate local measures to ensure the availability and reliability of their (local) storage.

A single-master architecture brings simplicity to the design of the system but gives rise to some concern for its scalability and reliability. The scalability concern is addressed by a Client cache, called Client image in the following. Let us examine in detail how the system handles a read() request, as illustrated on Figure 14.10 with dotted arrows:

1. The Client sends a first read(/dirB/file1) request; since it knows nothing about the file
distribution, the request is routed to the Master (1).

2. The Master inspects the namespace and finds that file1 is mapped to a list of chunks; their location is found in a local table (2).
3. Each server holding a chunk of file1 is required to transmit this chunk to the Client.
4. The Client keeps in its cache the addresses of the nodes that serve file1 (but not the fileitself); this knowledge can be used for subsequent accesses to file1 (4).



The approach is typical of distributed structures, and will be met in several other distributed services further on. The Client cache avoids a systematic access to the Master, a feature that would make the structure non scalable. By limiting the exchanges with the Master to messages that require metadata information, the coordination task is reduced and can be handled by a single computer.

From the Client point of view1, the distributed file system appears just like a directory hierarchy equipped with the usual Unix navigation (chddir, ls) and access (read, write) commands.

Observe again that the system works best for a relatively small number of very large files. GFS (and similar systems) expects typical files of several hundreds of MBs each, and sets accordingly the chunk size to 64 MBs. This can be compared to the traditional block-based organization of centralized storage systems (e.g., databases) where is block size is a small multiple of the disk physical block (typically, 4KB-8KB in a database block).

Failure handling

Failure is handled by standard replication and monitoring techniques. First, a chunk is not written on a single server but is replicated on at least 2 other servers. Having three copies of the same chunk is sufficient to face failures. The Master is aware of the existing replicas because each server that joins the clusters initially sends the chunk that it is ready to serve.

Second, the Master is in charge of sending background heartbeat messages to each server. If a server does not answer to a heartbeat messages, the Master initiates a server replacement by asking to one of the (at least 2) remaining servers to copy to a new server the chunks that fell under their replication factor.

The Master itself must be particularly protected because it holds the file namespace. A recovery mechanism is used for all the updates that affect the namespace structure, similar to that presented in Figure 14.5, page 289. We refer the reader to the original paper (see last section) for technical details on the management of aspects that fall beyond the scope of our limited presentation, in particular access rights and data consistency.