1. Demonstrate the working of ls, cp, mv and rm commands using C.

List.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>
#include <pwd.h>
#include <grp.h>
#include <time.h>

int main(int argc, char* argv[])
{
      DIR *dir;
      struct dirent *file;
      struct stat stats;
      struct passwd *tf;
      struct group *gf;

      if (argc != 2)
      {
            printf("Usage: list <directory>\n");
            exit(0);
      }

      dir = opendir(argv[1]);

      while(file = readdir(dir))
      {
            stat(file->d_name, &stats);

            switch (stats.st_mode & S_IFMT) {
                  case S_IFDIR: printf("d"); break;  //Sub-directory
                  case S_IFCHR: printf("c"); break;  //Char-oriented file
                  case S_IFBLK: printf("b"); break;  //Block-oriented file
                  case S_IFREG: printf("r"); break;  //Regular file
                  case S_IFLNK: printf("l"); break;  //Sym link
                  case S_IFSOCK: printf("s"); break; //Socket
                  case S_IFIFO: printf("p"); break;  //Pipe
                  default: printf("-"); break;  //Filetype isn't identified
            }

            printf( (stats.st_mode & S_IRUSR) ? "r" : "-" );
            printf( (stats.st_mode & S_IWUSR) ? "w" : "-" );
            printf( (stats.st_mode & S_IXUSR) ? "x" : "-" );
            printf( (stats.st_mode & S_IRGRP) ? "r" : "-" );
            printf( (stats.st_mode & S_IWGRP) ? "w" : "-" );
            printf( (stats.st_mode & S_IXGRP) ? "x" : "-" );
            printf( (stats.st_mode & S_IROTH) ? "r" : "-" );
            printf( (stats.st_mode & S_IWOTH) ? "w" : "-" );
            printf( (stats.st_mode & S_IXOTH) ? "x" : "-" );

            printf("\t%ld", stats.st_nlink);

            tf = getpwuid(stats.st_uid);
            printf("\t%s", tf->pw_name);

            gf = getgrgid(stats.st_gid);
            printf("\t%s", gf->gr_name);

            printf(" %zu", stats.st_size);
            printf(" %s", file->d_name);
            printf(" %s", ctime(&stats.st_mtime));
      }

      closedir(dir);
}
```

```
sourav@ubuntu-VirtualBox:~/Documents/OS/os1$ gcc list.c
sourav@ubuntu-VirtualBox:~/Documents/OS/os1$ ./a.out .
rrw-rw-r--  1     sourav      sourav 297 remove.c Thu Nov  5 14:26:58 2020
rrw-rw-r--  1     sourav      sourav 462 move.c Mon Mar 15 21:52:22 2021
rrw-rw-r--  1     sourav      sourav 446 copy.c Mon Mar 15 21:52:34 2021
rrw-rw-r--  1     sourav      sourav 1703 list.c Mon Mar 15 22:46:09 2021
drwxrwxr-x  11    sourav      sourav 4096 .. Sun Mar  7 18:21:23 2021
rrwxrwxr-x  1     sourav      sourav 17248 a.out Mon Mar 15 22:46:59 2021
drwxrwxr-x  2     sourav      sourav 4096 . Mon Mar 15 22:46:59 2021
```

Copy.c

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<fcntl.h>

int main(int argc, char *argv[])
{
      int fd_src, fd_dest, n;
      char buffer[20];

      if (argc != 3)
      {
            printf("Usage: copy <src_file> <dest_file>\n");
            exit(0);
      }

      fd_src = open(argv[1], O_RDONLY);
      fd_dest = open(argv[2], O_WRONLY|O_CREAT|O_TRUNC|O_EXCL, 0600);

      while(n = read(fd_src, buffer, 20))
            write(fd_dest, buffer, n);

      close(fd_src);
      close(fd_dest);
}
```

```
sourav@ubuntu-VirtualBox:~/Documents/OS/os1$ cat text.txt
Text.txt
sourav@ubuntu-VirtualBox:~/Documents/OS/os1$ cat text_cp.txt
cat: text_cp.txt: No such file or directory
sourav@ubuntu-VirtualBox:~/Documents/OS/os1$ gcc copy.c
sourav@ubuntu-VirtualBox:~/Documents/OS/os1$ ./a.out text.txt text_cp.txt
sourav@ubuntu-VirtualBox:~/Documents/OS/os1$ cat text.txt
Text.txt
sourav@ubuntu-VirtualBox:~/Documents/OS/os1$ cat text_cp.txt
Text.txt
```

Move.c

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

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

int main(int argc, char *argv[])
{
	int fd_src, fd_dest, n;
	char buffer[20];

	if (argc != 3)
	{
		printf("Usage: copy <src_file> <dest_file>\n");
		exit(0);
	}

	fd_src = open(argv[1], O_RDONLY);
	fd_dest = open(argv[2], O_WRONLY|O_CREAT|O_TRUNC|O_EXCL, 0600);

	while(n = read(fd_src, buffer, 20))
		write(fd_dest, buffer, n);

	close(fd_src);
	close(fd_dest);
	remove(argv[1]);
}
```

```
sourav@ubuntu-VirtualBox:~/Documents/OS/os1$ cat text2.txt
Text.txt
sourav@ubuntu-VirtualBox:~/Documents/OS/os1$ cat text_mv.txt
cat: text_mv.txt: No such file or directory
sourav@ubuntu-VirtualBox:~/Documents/OS/os1$ gcc move.c
sourav@ubuntu-VirtualBox:~/Documents/OS/os1$ ./a.out text2.txt text_mv.txt
sourav@ubuntu-VirtualBox:~/Documents/OS/os1$ cat text2.txt
cat: text2.txt: No such file or directory
sourav@ubuntu-VirtualBox:~/Documents/OS/os1$ cat text_mv.txt
Text.txt
```

Remove.c

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

```c
int main(int argc, char *argv[])
{
      if (argc != 2)
      {
            printf("Usage: remove <file>\n");
            exit(0);
      }

      remove(argv[1]);
}
```

```
sourav@ubuntu-VirtualBox:~/Documents/OS/os1$ cat text.txt
Text.txt
sourav@ubuntu-VirtualBox:~/Documents/OS/os1$ gcc remove.c
sourav@ubuntu-VirtualBox:~/Documents/OS/os1$ ./a.out text.txt
sourav@ubuntu-VirtualBox:~/Documents/OS/os1$ cat text.txt
cat: text.txt: No such file or directory
```

2. Demonstrate the working of fork(), wait() and execl() system calls in C.

Fork.c

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

int main()
{
      pid_t pid = fork();

      if(pid < 0)
      {
            printf("Fork failed\n");
            exit(0);
      }
      else if(pid == 0)
      {
            printf("Child: pid value = %d\n", pid);
            printf("Child: Process id = %d\n", getpid());
            printf("Child: Parent-Process id = %d\n", getppid());
            printf("Child: Exiting from child\n");
            exit(0);
      }
      else
      {
            printf("Parent: Waiting for child\n");
            wait(0);
            printf("Parent: pid value = %d\n", pid);
            printf("Parent: Process id = %d\n", getpid());
            printf("Parent: Child-Process id = %d\n", pid);
      }
      return 0;
}
```

```
sourav@ubuntu-VirtualBox:~/Documents/OS/os2$ gcc fork.c
sourav@ubuntu-VirtualBox:~/Documents/OS/os2$ ./a.out
Parent: Waiting for child
```

```
Child: pid value = 0
Child: Process id = 3792
Child: Parent-Process id = 3791
Child: Exiting from child
Parent: pid value = 3792
Parent: Process id = 3791
Parent: Child-Process id = 3792
```

Execl.c

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<wait.h>

int main(int argc, char* argv[])
{
      if (argc != 3)
      {
            printf("Usage: execl <int1> <int2>\n");
            exit(0);
      }

      pid_t pid = fork();

      if(pid < 0)
      {
            printf("Fork failed\n");
            exit(0);
      }
      else if(pid == 0)
      {
            printf("Child:\n");
            execl("sum_diff", "sum_diff", argv[1], argv[2], NULL);
            exit(0);
      }
      else
      {
            wait(0);
            printf("Parent:\n");
      }
}
```

Sum_diff.c

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

int main(int argc, char *argv[])
{
      int a = atoi(argv[1]);
      int b = atoi(argv[2]);
      printf("Sum of %d and %d is %d\n", a, b, a+b);
      printf("Difference of %d and %d is %d\n", a, b, a-b);
}
```

```
sourav@ubuntu-VirtualBox:~/Documents/OS/os2$ gcc sum_diff.c -o sum_diff
sourav@ubuntu-VirtualBox:~/Documents/OS/os2$ gcc execl.c
sourav@ubuntu-VirtualBox:~/Documents/OS/os2$ ./a.out 5 5
Child:
Sum of 5 and 5 is 10
Difference of 5 and 5 is 0
Parent:
```

3. Demonstrate the working of POSIX threads API in C

Thread.c

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

int sum = 0;

void *func(void *arg)
{
      int n = atoi(arg);
      for(int i = 1; i <= n; i++)
            sum += i;
      pthread_exit(0);
}

int main(int argc, char *argv[])
{
      if (argc != 2)
      {
            printf("Usage: thread <natural_num>\n");
            exit(0);
      }

      pthread_t tid;
      pthread_attr_t attr;

      pthread_attr_init(&attr);
      pthread_create(&tid, &attr, func, argv[1]);
      pthread_join(tid, NULL);

      printf("Sum of %d natural numbers is %d\n", atoi(argv[1]), sum);
}
```

```
sourav@ubuntu-VirtualBox:~/Documents/OS/os3$ gcc thread.c -l pthread
sourav@ubuntu-VirtualBox:~/Documents/OS/os3$ ./a.out 10
Sum of 10 natural numbers is 55
```

5. Write C programs for illustrating producer-consumer, reader-writer and dining-philosopher synchronisation problems.

Producer_consumer.c

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
#include<semaphore.h>

#define MAX 5

sem_t in_sem;
sem_t rm_sem;

void* producer()
{
      int count = 0;
      while(1)
      {
            sem_wait(&in_sem);
            sleep(rand()%3);
            printf("Producer inserted at %d\n", count % MAX + 1);
            count++;
            sem_post(&rm_sem);
      }
}

void* consumer()
{
      int count = 0;
      while(1)
      {
            sem_wait(&rm_sem);
            printf("Consumer removed at %d\n", count % MAX + 1);
            sleep(rand()%3);
            count++;
            sem_post(&in_sem);
      }
}
```

```
int main()
{
        pthread_t threads[2];
        sem_init(&in_sem, 0, MAX);
        sem_init(&rm_sem, 0, 0);

        srand(time(NULL));

        pthread_create(&threads[0], NULL, producer, NULL);
        pthread_create(&threads[1], NULL, consumer, NULL);

        pthread_join(threads[0], NULL);
        pthread_join(threads[1], NULL);

        sem_destroy(&in_sem);
        sem_destroy(&rm_sem);
}
```

```
sourav@ubuntu-VirtualBox:~/Documents/OS/os5$ gcc producer_consumer.c -l
pthread
sourav@ubuntu-VirtualBox:~/Documents/OS/os5$ ./a.out
Producer inserted at 1
Consumer removed at 1
Producer inserted at 2
Producer inserted at 3
Consumer removed at 2
Consumer removed at 3
Producer inserted at 4
Consumer removed at 4
Producer inserted at 5
Producer inserted at 1
Consumer removed at 5
Producer inserted at 2
Producer inserted at 3
Consumer removed at 1
Producer inserted at 4
Producer inserted at 5
Consumer removed at 2
Consumer removed at 3
Consumer removed at 4
Consumer removed at 5
Producer inserted at 1
```

```
Consumer removed at 1
Producer inserted at 2
Consumer removed at 2
Producer inserted at 3
Producer inserted at 4
Producer inserted at 5
Consumer removed at 3
Consumer removed at 4
Producer inserted at 1
```

Reader_writer.c

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
#include<semaphore.h>

#define N 10

int process[N] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
int r_count = 0;
sem_t rw_sem;
sem_t rc_sem;

void* reader(void* num)
{
      int n = *(int*)num;

      sem_wait(&rc_sem);
      if(!r_count)
      {
            printf("Waiting for reading\n");
            sem_wait(&rw_sem);
      }
      r_count++;
      sem_post(&rc_sem);

      printf("Process %d is reading\n", n+1);
      sleep(2);
```

```
    sem_wait(&rc_sem);
    r_count--;
    if(!r_count)
    {
        printf("Finished reading\n");
        sem_post(&rw_sem);
    }
    sem_post(&rc_sem);
}

void* writer(void* num)
{
    int n = *(int*)num;

    printf("Waiting for writing\n");
    sem_wait(&rw_sem);

    printf("Process %d is writing\n", n+1);
    sleep(4);

    printf("Finished writing\n");
    sem_post(&rw_sem);
}

int main()
{
    pthread_t threads[N];
    sem_init(&rw_sem, 0, 1);
    sem_init(&rc_sem, 0, 1);

    srand(time(NULL));
    int r = rand()%N;

    for(int i = 0; i < N; i++)
    {
        if(r == i)
        {
            pthread_create(&threads[i], NULL, writer, &process[i]);
            r = rand()%(N-i) + i;
            continue;
        }
        pthread_create(&threads[i], NULL, reader, &process[i]);
    }
    for(int i = 0; i < N; i++)
    {
        pthread_join(threads[i], NULL);
    }

    sem_destroy(&rw_sem);
    sem_destroy(&rc_sem);
}
```

```
sourav@ubuntu-VirtualBox:~/Documents/OS/os5$ gcc reader_writer.c -l pthread
sourav@ubuntu-VirtualBox:~/Documents/OS/os5$ ./a.out
Waiting for reading
Process 3 is reading
Waiting for writing
Process 5 is reading
Process 6 is reading
Process 7 is reading
Process 8 is reading
Process 9 is reading
Waiting for writing
Process 2 is reading
Process 1 is reading
Finished reading
Process 4 is writing
Finished writing
Process 10 is writing
Finished writing
```

Dining_philosopher.c

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
#include<semaphore.h>

#define N 5
#define THINKING 0
#define HUNGRY 1
#define EATING 2
```

```
#define LEFT(i) (((i)+4)%N)
#define RIGHT(i) (((i)+1)%N)

int phil[N] = { 0, 1, 2, 3, 4 };
int p_state[N] = {};
sem_t eat_sem[N];
sem_t st_sem;

void test(int n)
{
      if(p_state[n] == HUNGRY && p_state[LEFT(n)] != EATING &&
p_state[RIGHT(n)] != EATING)
      {
            p_state[n] = EATING;
            printf("Philosopher %d is taking chopsticks %d and %d\n", n+1,
n+1, LEFT(n)+1);
            printf("Philosopher %d is eating\n", n+1);
            sem_post(&eat_sem[n]);
      }
}

void take_fork(int n)
{
      sem_wait(&st_sem);
      p_state[n] = HUNGRY;
      printf("Philosopher %d is hungry\n", n+1);
      test(n);
      sem_post(&st_sem);
}

void keep_fork(int n)
{
      sem_wait(&st_sem);
      p_state[n] = THINKING;
      printf("Philosopher %d is keeping chopsticks %d and %d\n", n+1, n+1,
LEFT(n)+1);
      printf("Philosopher %d is thinking\n", n+1);
      test(LEFT(n));
      test(RIGHT(n));
      sem_post(&st_sem);
}

void* philosopher(void* num)
```

```
{
    int n = *(int*)num;

    printf("Philosopher %d is thinking\n", n+1);
    while(1)
    {
        sleep(5);
        take_fork(n);
        sleep(1);
        sem_wait(&eat_sem[n]);
        keep_fork(n);
    }
}

int main() {
    pthread_t threads[N];
    sem_init(&st_sem, 0, 1);
    for(int i = 0; i < N; i++)
        sem_init(&eat_sem[i], 0, 0);

    for(int i = 0; i < N; i++)
        pthread_create(&threads[i], NULL, philosopher, &phil[i]);

    for(int i = 0; i < N; i++)
        pthread_join(threads[i], NULL);

    sem_destroy(&st_sem);
    for(int i = 0; i < N; i++)
        sem_destroy(&eat_sem[i]);
}
```

```
sourav@ubuntu-VirtualBox:~/Documents/OS/os5$ gcc dining_philosopher.c -l
pthread
sourav@ubuntu-VirtualBox:~/Documents/OS/os5$ ./a.out
Philosopher 3 is thinking
Philosopher 2 is thinking
Philosopher 4 is thinking
Philosopher 1 is thinking
Philosopher 5 is thinking
Philosopher 3 is hungry
Philosopher 3 is taking chopsticks 3 and 2
Philosopher 3 is eating
Philosopher 2 is hungry
Philosopher 4 is hungry
Philosopher 1 is hungry
Philosopher 1 is taking chopsticks 1 and 5
Philosopher 1 is eating
Philosopher 5 is hungry
Philosopher 3 is keeping chopsticks 3 and 2
Philosopher 3 is thinking
Philosopher 4 is taking chopsticks 4 and 3
Philosopher 4 is eating
Philosopher 4 is keeping chopsticks 4 and 3
Philosopher 4 is thinking
Philosopher 1 is keeping chopsticks 1 and 5
Philosopher 1 is thinking
Philosopher 5 is taking chopsticks 5 and 4
Philosopher 5 is eating
Philosopher 2 is taking chopsticks 2 and 1
Philosopher 2 is eating
Philosopher 5 is keeping chopsticks 5 and 4
Philosopher 5 is thinking
Philosopher 2 is keeping chopsticks 2 and 1
Philosopher 2 is thinking
```

6. Demonstrate the working of file locking using C.

Lock.c

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

int main(int argc, char *argv[])
{
      int fd;
      struct flock lock;

      if(argc != 3)
      {
            printf("Usage: lock <file> <lock_type (RD:0, WR:!0)>\n");
            exit(0);
      }

      fd = open(argv[1], O_RDWR);
      if(fd == -1)
      {
            printf("Error while opening the file\n");
            exit(1);
      }

      lock.l_type = atoi(argv[2]) ? F_WRLCK : F_RDLCK ;
      lock.l_whence = SEEK_SET;
      lock.l_start = 0;
      lock.l_len = 100;

      if(fcntl(fd, F_SETLK, &lock) == -1)
      {
            printf("Unable lock the file\n");
            fcntl(fd, F_GETLK, &lock);
            printf("File already locked by process (pid):%d\n",
lock.l_pid);
            return -1;
      }
      else
      {
            printf("File %s locked\n", atoi(argv[2]) ? "write" : "read" );
            printf("Press enter to release lock\n");
            getchar();

            lock.l_type = F_UNLCK;
            if(fcntl(fd, F_SETLK, &lock) == -1)
            {
                  printf("Unable unlock the file\n");
                  exit(0);
            }
            printf("File unlocked\n");
      }
      close(fd);
}
```

```
sourav@ubuntu-VirtualBox:~/Documents/OS/os6$ gcc lock.c
sourav@ubuntu-VirtualBox:~/Documents/OS/os6$ ./a.out lock.txt 0
File read locked
Press enter to release lock
;<1>

File unlocked
sourav@ubuntu-VirtualBox:~/Documents/OS/os6$ ./a.out lock.txt 1
File write locked
Press enter to release lock
;<2>

File unlocked
```

```
;<1>
sourav@ubuntu-VirtualBox:~/Documents/OS/os6$ ./a.out lock.txt 0
File read locked
Press enter to release lock

File unlocked
sourav@ubuntu-VirtualBox:~/Documents/OS/os6$ ./a.out lock.txt 1
Unable lock the file
File already locked by process (pid):6149
```

```
;<2>
```

```
sourav@ubuntu-VirtualBox:~/Documents/OS/os6$ ./a.out lock.txt 0
Unable lock the file
File already locked by process (pid):6188
sourav@ubuntu-VirtualBox:~/Documents/OS/os6$ ./a.out lock.txt 1
Unable lock the file
File already locked by process (pid):6188
```

7. Write a program to detect deadlocks using Banker’s algorithm in C.

Banker.c

```
#include<stdio.h>

int main()
{
      int processes, resources;
      int i, j;

      printf("Enter the number of processes:\n");
      scanf("%d", &processes);

      printf("Enter the number of resources:\n");
      scanf("%d", &resources);

      int max[processes][resources];
      int allocated[processes][resources];
      int need[processes][resources];
      int total[resources];
      int available[resources];
      int completed[processes];

      printf("Enter total resource instances:\n");
      for(j = 0; j < resources; j++)
      {
            scanf("%d", &total[j]);
            available[j] = total[j];
      }


      printf("Enter max resource instances required for every process:\n");
      for(i = 0; i < processes; i++)
      {
            printf("For P[%d] ", i+1);
            for(j = 0; j < resources; j++)
            {
                  scanf("%d", &max[i][j]);
                  completed[i] = 0;
            }
      }
```

```
printf("Enter allocated resource instances for every process:\n");
for(i = 0; i < processes; i++)
{
      printf("For P[%d] ", i+1);
      for(j = 0; j < resources; j++)
      {
            scanf("%d", &allocated[i][j]);
            available[j] -= allocated[i][j];
            need[i][j] = max[i][j] - allocated[i][j];
      }
}

printf("Execution Sequence:\n");
int finished = 0;
int last_fin = 0;
i = 0;
do
{
      if(completed[i])
      {
            i = (i+1) % processes;
            continue;
      }

      int satisfied = 0;
      for(j = 0; j < resources; j++)
      {
            if(need[i][j] <= available[j])
                  satisfied++;
      }

      if(satisfied == resources)
      {
            completed[i] = 1;
            for(j = 0; j < resources; j++)
                  available[j] += allocated[i][j];
            printf("P[%d] ", i+1);
            finished++;
            last_fin = i;
      }

      if(finished == processes)
```

```
                  break;

            i = (i+1) % processes;
      } while(i != last_fin);

      if(finished == processes)
            printf("\nSafe state : No deadlock\n");
      else
            printf("\nNot safe state : Deadlock\n");
}
```

```
sourav@ubuntu-VirtualBox:~/Documents/OS/os7$ gcc banker.c
sourav@ubuntu-VirtualBox:~/Documents/OS/os7$ ./a.out
Enter the number of processes:
5
Enter the number of resources:
3
Enter total resource instances:
7 4 2
Enter max resource instances required for every process:
For P[1] 3 3 2
For P[2] 2 1 2
For P[3] 3 0 1
For P[4] 2 0 0
For P[5] 0 0 1
Enter allocated resource instances for every process:
For P[1] 0 2 0
For P[2] 2 1 1
For P[3] 3 0 1
For P[4] 1 0 0
For P[5] 1 0 0
Execution Sequence:
P[3] P[4] P[5] P[2] P[1]
Safe state : No deadlock
```

8. Build a sample Buddy system based on Buddy memory allocation algorithm.

Buddy.c

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

struct {
      char size;
      char* name;
} tree[32] = {};

int max;

int can_alloc(int node)
{
      while(node != 1)
      {
            node /= 2;
            if(tree[node].size == 0)
                  continue;
            if(tree[node].size == -1)
                  return 1;
            return 0;
      }
      return 1;
}

void mem_alloc(char name[], int req)
{
      if(req > max)
      {
            printf("Requirement exceeds capacity!!!\n");
            return;
      }

      int level = 0;
      int s = max;
      while(1)
      {
            if(req < s && req > s/2)
                  break;
            s /= 2;
            level++;
      }

      for(int i = 1 << level; i < 1 << (level+1); i++)
            if(tree[i].size == 0 && can_alloc(i))
            {
                  tree[i].size = req;
                  tree[i].name = strdup(name);
                  for(i /= 2; tree[i].size == 0; tree[i].size = -1, i /=
2);
                  printf("Successful allocation\n");

                  if(level > tree[0].size)
                        tree[0].size = level;
                  return;
            }

      printf("The system don't have enough free memory\n");
}

void mem_dealloc(char name[])
{
      int node = 1;
      int last = 1 << (tree[0].size+1);
      for(; node < last; node++)
      {
            if(tree[node].size > 0 && strcmp(name, tree[node].name) == 0)
                  break;
      }

      if(node == last)
      {
            printf("Process %s is not present in memory\n", name);
            return;
      }

      tree[node].size = 0;
      free(tree[node].name);
      tree[node].name = NULL;
      printf("Successful deallocation\n");
      while(node != 1)
      {
            int sib = node%2 == 0 ? node + 1 : node - 1;
            if(tree[sib].size == 0 && tree[node].size == 0)
            {
                  node /= 2;
                  tree[node].size = 0;
            }
            else break;
      }
}

void mem_print(int node, int level)
{
      for(int i = 0; i < level; i++)
            printf("|   ");

      int size = max/(1<<level);
      if(tree[node].size == 0)
            printf("%d   ---> FREE\n", size);

      else if(tree[node].size == -1)
      {
            printf("%d   ---> DIVIDED\n", size);
            mem_print(node*2, level+1);
            mem_print(node*2+1, level+1);
      }

      else
            printf("%d: %s - %d\n", size, tree[node].name,
tree[node].size);
}

int main()
{
      int c, req;
      char name[10];

      printf("B U D D Y   S Y S T E M   R E Q U I R E M E N T S\n");

      printf("Enter the total size of the memory:\n");
      scanf("%d", &max);
```

```
    while(1)
    {
        printf("\nB U D D Y   S Y S T E M\n");
        printf("1. Locate the process into memory\n");
        printf("2. Remove the process from memory\n");
        printf("3. Tree structure for memory allocation map\n");
        printf("<other> Exit\n");
        printf("Enter your choice:\n");
        scanf("%d", &c);

        switch(c)
        {
            case 1:
                printf("\nM E M O R Y   A L L O C A T I O N\n");
                printf("Enter the process name:\n");
                scanf("%9s", name);
                printf("Enter the process size:\n");
                scanf("%d", &req);
                mem_alloc(name, req);
                break;

            case 2:
                printf("\nM E M O R Y   D E A L L O C A T I O
N\n");

                printf("Enter the process name:\n");
                scanf("%9s", name);
                mem_dealloc(name);
                break;

            case 3:
                printf("\nM E M O R Y   A L L O C A T I O N   M A
P\n");

                mem_print(1, 0);
                break;

            default:
                return 0;
        }
    }
}
```

```
sourav@ubuntu-VirtualBox:~/Documents/OS/os8$ gcc buddy.c
sourav@ubuntu-VirtualBox:~/Documents/OS/os8$ ./a.out
B U D D Y   S Y S T E M   R E Q U I R E M E N T S
Enter the total size of the memory:
1024

B U D D Y   S Y S T E M
1. Locate the process into memory
2. Remove the process from memory
3. Tree structure for memory allocation map
<other> Exit
Enter your choice:
1

M E M O R Y   A L L O C A T I O N
Enter the process name:
P1
Enter the process size:
99
Successful allocation

B U D D Y   S Y S T E M
1. Locate the process into memory
2. Remove the process from memory
3. Tree structure for memory allocation map
<other> Exit
Enter your choice:
1

M E M O R Y   A L L O C A T I O N
Enter the process name:
P2
Enter the process size:
279
Successful allocation

B U D D Y   S Y S T E M
1. Locate the process into memory
2. Remove the process from memory
3. Tree structure for memory allocation map
<other> Exit
Enter your choice:
3
```

```
M E M O R Y   A L L O C A T I O N   M A P
1024   ---> DIVIDED
|   512   ---> DIVIDED
|   |   256   ---> DIVIDED
|   |   |   128: P1 - 99
|   |   |   128   ---> FREE
|   |   256   ---> FREE
|   512: P2 - 23

B U D D Y   S Y S T E M
1. Locate the process into memory
2. Remove the process from memory
3. Tree structure for memory allocation map
<other> Exit
Enter your choice:
2

M E M O R Y   D E A L L O C A T I O N
Enter the process name:
P1
Successful deallocation

B U D D Y   S Y S T E M
1. Locate the process into memory
2. Remove the process from memory
3. Tree structure for memory allocation map
<other> Exit
Enter your choice:
3

M E M O R Y   A L L O C A T I O N   M A P
1024   ---> DIVIDED
|   512   ---> FREE
|   512: P2 - 23

B U D D Y   S Y S T E M
1. Locate the process into memory
2. Remove the process from memory
3. Tree structure for memory allocation map
<other> Exit
Enter your choice:
0
```

9. Demonstrate the working of static and dynamic linking of libraries in C.

Prog.c

```c
#include<stdio.h>
#include<stdlib.h>
#include"sum.c"
#include"diff.c"

int main(int argc, char* argv[])
{
      if (argc != 3)
      {
            printf("Usage: prog <int1> <int2>\n");
            exit(0);
      }

      int a = atoi(argv[1]);
      int b = atoi(argv[2]);

      printf("Sum: %d\n", sum(a, b));
      printf("Diff: %d\n", diff(a, b));
}
```

Sum.c

```c
int sum(int a, int b)
{
      return a+b;
}
```

Diff.c

```c
int diff(int a, int b)
{
      return a-b;
}
```

```
sourav@ubuntu-VirtualBox:~/Documents/OS/os9$ gcc -c sum.c diff.c
sourav@ubuntu-VirtualBox:~/Documents/OS/os9$ ar cr libstest.a sum.o diff.o
sourav@ubuntu-VirtualBox:~/Documents/OS/os9$ gcc prog.c libstest.a
sourav@ubuntu-VirtualBox:~/Documents/OS/os9$ ./a.out 5 5
Sum: 10
Diff: 0
```

```
sourav@ubuntu-VirtualBox:~/Documents/OS/os9$ gcc -fPIC -c sum.c diff.c
sourav@ubuntu-VirtualBox:~/Documents/OS/os9$ gcc -shared -o libdtest.so
sum.o diff.o
sourav@ubuntu-VirtualBox:~/Documents/OS/os9$ gcc prog.c -L. -l dtest
sourav@ubuntu-VirtualBox:~/Documents/OS/os9$ ./a.out 5 5
Sum: 10
Diff: 0
```