RTOS LAB PROGRAMS

1. **To create a new process using Fork system.**

```
# include<stdio.h>
#include<unistd.h>
main()
{
int pid;
pid = fork();
if(pid<0)
{
printf("\n child is not created \n");
return -1;
}
else if(pid==0)
{
printf("\n in child process");
printf("\n Child pid = %d, parent pid = %d\n", pid, getpid());
}
}
```

Output :

child pid =0, parent pid = 26553

2. **To create a new process and new file, write into file using child and parent process.**

```
#include<stdio.h>
#include<unistd.h>
#include<fcntl.h>
main()
{
int pid, fd;
char buf[10];
fd = open("myfile.txt", O_WRONLY | O_CREAT, 0744);
pid = fork();
if(pid==0)
{
printf(" \n The child is created \n");
printf("\n fd = %d", fd);
printf("\n Child process : Enter the data:");
scanf("%s", buf);
write(fd,buf,sizeof(buf));
```

```
}
else
{
wait()
printf("\n In parent process\n");
printf("\fd=%d",fd);
printf("\n parent process: enter the data:")
scanf("%s",buf);
write (fd, buf, sizeof(buf));
}
printf("Terminating main");
}
```

Output:

The child is created
fd = 3
child process : Enter the data : HELLO
In parent process
Fd = 3
Parent process: Enter the data: WORLD

3. Program for creating thread.

```
#include<stdio.h>
#include<pthread.h>
 int i=1;
void* fn()
{
printf("thread %d is executing \n", i++);
}
main()
{
int i;
pthreat_t fid;
for(i=0;i<3;i++)
{
Pthread_create (&fid, NULL, fn, NULL);
Pthread_join(fid,NULL);
}
}
```

Output

Thread 1 is executing
Thread 2 is executing
Thread 3 is executing

4. **Program to create unnamed pipe.**

```
#include<stdio.h>
#include<string.h>
#define READ 0
#define WRITE 1
char *phrase = “stuff in your pipe and smoke it”;
main()
{
int fd[2], byte_Read;
char message[100];
pipe(fd);
if(fork()==0)
{
close(fd[READ]);
write(fd[WRITE], phrase, strlen(phrase)+1);
close(fd[WRITE]);
printf(“child:wrote “%s” to pipe\n”, phrase);
}
else
{
close(fd[WRITE]);
byte_Read = read(fd[READ], message,100);
printf(“parent:read ‘%d’ bytes from pipe : %s \n”, byte_Read, message);
close(fd[READ]);
}
}
```

Output:

Child: wrote “stuff in your pipe and smoke it”

Parent : read 37 bytes from pipe : stuff in your pipe and smoke it

5. **Program for creation of named pipes**

   1) **Named pipe(write)**

```
#include<stdio.h>
#include<string.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcnt1.h>
char *phrase =  “stuff in your pipe and smoke it”;
int main()
{
```

```
int fd1;
fd1 = open(“mypipe”, O_RDWR);
write(fd1,phrase, strlen(phrase)+1);
Close(fd1);
}
```

2) Named pipe(Read)

```
#include<stdio.h>
#include<string.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcnt1.h>
int main()
{
int fd1, char buf[100];
fd1=open(“mypipe”, O_RDWR);
read(fd1,buf,100);
printf(“%s\n”,buf);
close(fd1);
}
```

Output:

Child: wrote “stuff in your pipe and smoke it”
Parent : read 37 bytes from pipe : stuff in your pipe and smoke it

6. Program using SEMAPHORE to lock and unlock critical code.

```
#include<stdio.h>
#include<sys/ipc.h>
#include<sys/sem.h>
main()
{
int key,semid;
key = ftok(“Semaphore.c”,’d’);
printf(“key = %d”,key);
struct sembuf buf = {0, -1, 0};
semid = semget(key, 1, 0);
printf(“\n lock the CS before entering in CS\n”);
semop(semid,&buf,1);
printf(“Inside the critical section\n”);
printf(“Press enter to unlock the critical section \n”);
```

```
getchar();
buf.sem_op=1;
semop(semid, &buf,1);
printf("Crtical section unlocked \n");
}
```

Output:

```
key = -1
Lock the CS before entering in CS
Inside the critical section
Press enter to unlock the critical section
Critical section unlocked.
```

7. Program to demonstrate the mutex concept.

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
void *function();
pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
int counter = 0;
main()
{
int re1,re2;
pthread_t thread1, thread2;
if((re1 = pthread_create(&thread1,NULL,&function,NULL)))
{
printf("Thread creation failed: %d\n",re1);
}
if((re2=pthread_create(&thread2,NULL,&function,NULL)))
{
printf("Thread creation failed: %d\n",re2);
}
pthread_join(thread1, NULL);
pthread_join(thread2, NULL);
exit(EXIT_SUCCESS);
}
void *function()
{
pthread_mutex_lock(&mutex1);
counter ++;
```

```
printf(“counter value: %d\n”,counter);
pthread_mutex_unlock(&mutex1);
}
```

Output
Counter value :1
Counter value : 2