Assg1

Q1]

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

int main(){
int fd=open("file_hole.txt",O_WRONLY | O_CREAT, 0666);
if(fd==-1)
{
perror("Error in opening File!!!!!!");
return 1;
}
lseek(fd,1024,SEEK_SET);
const char* data= "The Entered DATA is After  the HOle is Created.\n";
write(fd,data,strlen(data));
close(fd);
return(0);
}
```

Q2]

```
#include<stdlib.h>
#include<stdio.h>
#include<fcntl.h>
#include<unistd.h>
int main(){
int fd = open("A1Q2.txt",O_RDONLY | O_CREAT);
if(fd == -1)
{
	printf("FILE  not created......\n");
	return 1;

}
printf("\nSleeping before Terminating.... ");
if (fd==1)
{
	sleep(15);
}
close(fd);
printf("\nACTIVE");
printf("\n");
return 0;
}
```

Q3]

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

```
int main() {
    struct dirent *entry;
    DIR *directory = opendir(".");
    int fileCount = 0;
if (directory == NULL) {
        printf("Error opening directory");
        return 1;
    }
   printf("Name\t\t\tSize (bytes)\n");
while ((entry = readdir(directory)) != NULL) {
        struct stat fileStat;

        if (stat(entry->d_name, &fileStat) == 0) {
            if (S_ISREG(fileStat.st_mode)) {
                printf("%s\t\t\t%ld\n", entry->d_name, (long)fileStat.st_size);
                fileCount++;
            }
        }
    }

    printf("Total files in the directory: %d\n", fileCount);
    closedir(directory);

    return 0;
}
```

Assg 2

Q1 a]

```
#include<stdio.h>
void main(){
int bal=0,deposit,withdraw,emi;
float ci;
char op;
 while(1){
printf("\n BANKING SYSTEM");
printf("\n------------------");
printf("\nD ->Deposit.");
printf("\nW ->Withdraw.");
printf("\nB ->Balance.");
printf("\nE ->EMI.");
printf("\nI ->Interest.");
printf("\nQ ->QUIT.");
printf("\nEnter an operation:");
scanf("%c",&op);
switch(op){
case 'D':
printf("\n Enter Deposit amt:");
scanf("%d",&deposit);
bal= bal+deposit;
break;
case 'W':
printf("\n Enter the amt to be withdrawn:");
scanf("%d",&withdraw);
bal=bal-withdraw;
if(bal<withdraw){
printf("\nInsufficient balance to withdraw..System going to SLEEP....");
sleep(10);
}
else { printf("Withdrawn amt %d ",&withdraw);}
break;
case 'B':
printf("\nBalance: %d",bal);
break;
case 'E':
printf("Enter the EMI amt to be paid :");
scanf("%d",&emi);
bal=bal-emi;
if(bal<emi){
printf("\nInsufficient Balance To pay EMI...System going to sleep");
sleep(10);
}
else{("EMI PAID AMT %d",&emi);}
break;
case 'I':
ci=(float)bal*4/100;
bal=bal+ci;
printf("\nInterest : %f",ci);
break;
case 'Q':
return;
default:
printf("\nInvalid Operation...");
}
}
}
```

Q1b]

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#include<semaphore.h>
#define NUM_THREADS 3
#define Initial_Bal 1000
int bal = Initial_Bal ;
sem_t withdraw_sem,deposit_sem,emi_sem;
void* deposit(void* arg)
{
int deposit_amt=*(int*) arg;
sem_wait(&deposit_sem);
bal=bal + deposit_amt;
printf("Deposit: %d,New Balance:%d\n",deposit_amt,bal);
sem_post(&emi_sem);
sem_post(&withdraw_sem);
return NULL;
```

```
}
void*  emi(void* arg)
{
int emi_amt= *(int*)arg;
sem_wait(&emi_sem);
if(bal<emi_amt)
{
printf("EMI :Insufficient balance. Going to sleep..\n");
sem_post(&emi_sem);
return NULL;
}
bal=bal-emi_amt;
printf("EMI: Deducted:%d,New Balance:%d\n",emi_amt,bal);
sem_post(&emi_sem);
return NULL;
}

void* withdraw(void* arg)
{
int withdraw_amt = withdraw_amt =*(int*)arg;
sem_wait(&withdraw_sem);
if(bal<withdraw_amt)
{
printf("\nWithdraw Amount Insufficient...");
printf("\nGoing to Sleep....");
sem_post(&withdraw_sem);
return NULL;
}
bal = bal - withdraw_amt;
printf("Withdraw:%d, New Balance:%d\n",withdraw_amt,bal);
sem_post(&emi_sem);
sem_post(&withdraw_sem);
return NULL;
}

int main()
{
int i;
pthread_t threads[NUM_THREADS];
int deposit_amt=0,emi_amt=0,withdraw_amt=0;
printf("Enter the amt to be deposit:");
scanf("%d",&deposit_amt);
printf("Enter the amt to be withdrawn:");
scanf("%d",&withdraw_amt);
printf("EMI :");
scanf("%d",&emi_amt);

sem_init(&deposit_sem,0,1);
sem_init(&emi_sem,0,0);
sem_init(&withdraw_sem,0,0);
pthread_create(&threads[0],NULL,deposit,&deposit_amt);
pthread_create(&threads[1],NULL,emi,&emi_amt);
pthread_create(&threads[2],NULL,withdraw,&withdraw_amt);
for(i=0;i<NUM_THREADS;i++)
{
pthread_join(threads[i],NULL);
}
sem_destroy(&deposit_sem);
sem_destroy(&emi_sem);
sem_destroy(&withdraw_sem);
return 0;
}
```

Execute command :

cc ‘filename.c’ -o ‘new file name’ -lpthread -lrt

./’new file name’

Assg 3

```
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>
#include<time.h>
#include<sys/times.h>
#include<stdio.h>
#include<stdlib.h>
int main(void)
{
        int n;
        printf("Enter N");
        scanf("%d",&n);
        int i, status;
        pid_t pid;
        time_t currentTime; //current time
        struct tms cpuTime; //return info abt process time
        if((pid = fork())==-1)
        {
        perror("\nfork error");
        exit(EXIT_FAILURE);
        }
        else if(pid==0)
        //child process
        {
        time(&currentTime);
        printf("\nChild process started at %s",ctime(&currentTime));
        for(i=0;i<n;i++)
        {
        printf("\nCounting= %dn",i);
        sleep(1);
        }
        time(&currentTime);
        printf("\nChild process ended at %s",ctime(&currentTime));
        exit(EXIT_SUCCESS);
        }
        else
        {
        //Parent process
        time(&currentTime);
        // gives normal time
        printf("\nParent process started at %s ",ctime(&currentTime));
        if(wait(&status)== -1)
        //wait for child process
        perror("\n wait error");
        if(WIFEXITED(status))
        printf("\nChild process ended normally");
        else
        printf("\nChild process did not end normally");
        if(times(&cpuTime)<0)
         //Get process time
        perror("\nTimes error");
        else
        {
        // _SC_CLK_TCK: system configuration time: seconds clock tick
        printf("\nParent process user time= %fn",((double)
        cpuTime.tms_utime));
        printf("\nParent process system time = %fn",((double)
        cpuTime.tms_stime));
        printf("\nChild process user time = %fn",((double)
        cpuTime.tms_cutime));
         printf("\nChild process system time = %fn",((double)
        cpuTime.tms_cstime));
        }
        time(&currentTime);
        printf("\nParent process ended at %s",ctime(&currentTime));
        exit(EXIT_SUCCESS);
        }
}
```

2]

```
#include<stdio.h>
#include<stdlib.h>
void first(void)
{
        printf("This is a beautiful,");
        }
        void second(void){
  printf("Wonderful life");
        }
int main(){
        atexit(second);
        atexit(first);
        return 0;
}
```

3]

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

int main() {
   int pipefds1[2], pipefds2[2];
   int returnstatus1, returnstatus2;
   int pid;
   char pipe1writemessage[20] = "Hi";
   char pipe2writemessage[20] = "Hello";
   char readmessage[20];
   returnstatus1 = pipe(pipefds1);

   if (returnstatus1 == -1) {
      printf("Unable to create pipe 1 \n");
      return 1;
   }
   returnstatus2 = pipe(pipefds2);

   if (returnstatus2 == -1) {
      printf("Unable to create pipe 2 \n");
      return 1;
   }
   pid = fork();

   if (pid != 0){
         // Parent process
      close(pipefds1[0]); // Close the unwanted pipe1 read side
      close(pipefds2[1]); // Close the unwanted pipe2 write side
      printf("In Parent: Writing to pipe 1 – Message is %s\n", pipe1writemessage);
      write(pipefds1[1], pipe1writemessage, sizeof(pipe1writemessage));
         read(pipefds2[0], readmessage, sizeof(readmessage));
         printf("In Parent: Reading from pipe 2 – Message is %s\n", readmessage);
   }
   else {
         //Child Process
      close(pipefds1[1]); // Close the unwanted pipe1 write side
      close(pipefds2[0]); // Close the unwanted pipe2 read side
      read(pipefds1[0], readmessage, sizeof(readmessage));
      printf("In Child: Reading from pipe 1 – Message is %s\n", readmessage);
      printf("In Child: Writing to pipe 2 – Message is %s\n", pipe2writemessage);
      write(pipefds2[1], pipe2writemessage, sizeof(pipe2writemessage));
   }
   return 0;
}
```

Assg 4.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/shm.h>
#include<string.h>
int main()
{
        int i;
        void *shared_memory;
        char buff[100];
        int shmid;
        //shmget() used to return shared memory identifier with Associated Key
        shmid=shmget((key_t)2345, 1024, 0666|IPC_CREAT);
        printf("Key of shared memory is %d\n",shmid);

        //attaches the shared memory segment associated with the shared memory identifier
        shared_memory=shmat(shmid,NULL,0);

        //process attached to shared memory segment
        printf("Process attached at %p\n",shared_memory);

        //this prints the address where the segment is attached with this process
        printf("Enter some data to write to shared memory\n");

        //get some input from user
        read(0,buff,100);

        //data written to shared memory
        strcpy(shared_memory,buff);
        printf("You wrote : %s\n",(char *)shared_memory);

        shmid=shmget((key_t)2345, 1024, 0666);
        printf("Key of shared memory is %d\n",shmid);

        //process attached to shared memory segment
        shared_memory=shmat(shmid,NULL,0);
        printf("Process attached at %p\n",shared_memory);
        printf("Data read from shared memory is : %s\n",(char *)shared_memory);
}
```

Assg 5.

```
#include<stdio.h>
#include<signal.h>
#include<stdlib.h>
void sighup();
void sigint();
void sigquit();
main()
{
int pid,i,j,k;
if ((pid=fork())<0)
{
perror("fork");
exit(1);
}
if(pid==0)
{
signal(SIGHUP,sighup);
signal(SIGINT,sigint);
signal(SIGQUIT,sigquit);
for(;;);
}
else
{
j=0;
for(i=1;i<=5;i++)
{
j++;
printf("PARENT: sending SIGHUP signal:%d\n",j);
kill(pid,SIGHUP);
sleep(3);
printf("PARENT:sending SIGHUP signal:%d\n",j);
kill(pid,SIGINT);
sleep(3);
}
sleep(3);
printf("PARENT:sending SIGQUIT");
kill(pid,SIGQUIT);
}
}
void sighup()
{
signal(SIGHUP,sighup);
printf("child:i have received sighup\n");
}
```

2]
```
#include<stdio.h>
#include<signal.h>
#include<stdlib.h>
void sigstop()
{
	printf(" Suspended\n");
}
void sigcont()
{
	printf(" Its Back\n");
}
void sigint()
{
	printf(" resuming\n");
	exit(0);
}
int main(int argc, char **argv){
	printf("Starting the program\n");
	signal(SIGTSTP,sigstop);
	signal(SIGCONT,sigcont);
	signal(SIGINT, sigint);
	while(1)
	{
	sleep(2);
	}
      return 0;
}
```

3]
```
#include<stdio.h>
#include<signal.h>
#include<stdlib.h>
	int flag =0;
	void sigintHandler(int sig_num)
{
	if(flag==1)
	exit(0);
	{
	signal(SIGINT,sigintHandler);
	printf("\n ctrl+c signal catched for first time");
	flag = 1;
	//clear the output buffer and move the buffered data to console
	fflush(stdout);
	}
}
int main()
	{
	signal (SIGINT,sigintHandler);
	while(1){
	}
	return 0;
}
```