EXP	THEORY QUESTIONS
1	1. Write about BIOS and MBR stages of a typical Linux boot process. 7. Write about the Grub and Kernel stages of a typical Linux boot process 9. Write about INIT and run-level stages of a typical Linux boot process.
2	5. Write about the Linux File system. 11. Write about Vi Editors.
4	20. Write about basic Linux commands 3. Write about File permissions in Basic Linux commands.
14,15	13. Write about the OS161 file system 26. State the Steps to Build the Software from the Source file 15. Write the OS161 Installation steps 17. Understanding the OS161 file system and working with test programs

---

2. Write a program to send a message (pass-through command-line arguments) into a message queue. Send a few messages with unique message numbers. (Same as 22)

Writer Process:

Step 1 - Create a message queue or connect to an already existing message queue (msgget())

Step 2 – specify the message type as 1.

Step 3- Write into message queue (msgsnd())

Step 4- terminate the process

Code:

```
#include<stdio.h>

#include<string.h>

#include<stdlib.h>
```

```
#include<sys/ipc.h>

#include<sys/msg.h>

int main (int argc, char *argv [ ])

{int len, mid,i=1;

        struct buffer

        {long mtype;

        char buf[50];

        }x;

mid=msgget((key_t)6,IPC_CREAT|0666);

x.mtype=atoi(argv[1]);

strcpy(x.buf,argv[2]);

len=strlen(x.buf);

msgsnd(mid,&x,len,0);

printf("Message of size %d sent successfully \n",len);

return 0;

}
```

OUTPUT:

$./a.out 1 welcome (note: 1 is messageId and welcome is message)

Message of size 7 sent successfully

---

4. Write a program to receive a particular message from the message queue. Use the message number to receive the particular message. (Same as 23)

Reader Process:

Step 1 - Create a message queue or connect to an already existing message queue (msgget())

Step 2 – specify the message type as 1.

Step 3 – Read from the message queue (msgrev())

Step 4 - Perform control operations on the message queue (msgctl())

Step 5 – terminate the reader process

Code:

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<sys/ipc.h>
#include<sys/msg.h>
int main(int argc, char *argv[ ])
{
 int len,mid,i=1;
            struct buffer
              { long mtype;
              char buf[50];
               }x;
  mid=msgget((key_t)6,0666);
   x.mtype=atoi(argv[1]);
   len=atoi(argv[2]);
  msgrcv(mid, &x,len,x.mtype,0);
   printf("The message is:%s\n",x.buf);
   return 0;
}
```

OUTPUT:

$ ./a.out 1 7 (note: 1 is messageId and 7 is size of the message)

The message is: welcome

---

6. Write a command to sort the file os and write the output into the file f22. Also, eliminate duplicate lines. to display the unique lines of the sorted file f21. Also, display the number of occurrences of each line.

To sort file 'os' and write output into file f22:

$ sort os > f22 (or) sort -o f22 os

To sort and eliminate duplicate lines, and write the output into file f22:

$ sort os | uniq > f22

To display unique lines of sorted file f21:

$ uniq f21

To display number of occurrences of each line:

$ uniq -c f21

8.Write a command to sort the file /etc/passwd in descending order to sort the file /etc/passwd by user-id numerically. (Similar to 25th)

To sort /etc/passwd in descending order:

$ sort -r /etc/passwd

To sort /etc/passwd by user id:

$ sort -n /etc/passwd

To display the names of nologin users

grep -v "/sbin/nologin$" /etc/passwd | cut -d: -f1

ls -ld /etc/s* | wc -l

---

10. Write a command

a.to cut 5 to 8 characters of the file f1.

$ cut -c 5-8 testfile.txt

b.to display the user-id of all the users in your system.

$ cat /etc/passwd | cut -d: -f1

---

12. Write a program to perform process synchronization in producer-consumer problem

```
#include <stdio.h>

#include <stdlib.h>


int mutex = 1;

int full = 0;

int empty = 10, x = 0;

void producer()

{

        --mutex;

        // Increase the number of full slots by 1
```

```
        ++full;
        // Decrease the number of empty slots by 1
        --empty;
        // Item produced
        x++;
        printf("\nProducer produces item %d",x);
        // Increase mutex value by 1
        ++mutex;
}
void consumer()
{
        // Decrease mutex value by 1
        --mutex;
        // Decrease the number of full slots by 1
        --full;
        // Increase the number of empty slots by 1
        ++empty;
        printf("\nConsumer consumes item %d",x);
        x--;
        // Increase mutex value by 1
        ++mutex;
}
int main()
{
        int n, i;
        printf("\n1. Press 1 for Producer"
                "\n2. Press 2 for Consumer"
                "\n3. Press 3 for Exit");
        for (i = 1; i > 0; i++) {
```

```
		printf("\nEnter your choice: ");
		scanf("%d", &n);
		switch (n) {
		case 1:
			if ((mutex == 1)
				&& (empty != 0)) {
				producer();
			}
			else {
				printf("Buffer is full!");
			}
			break;
		case 2:

			if ((mutex == 1)
				&& (full != 0)) {
				consumer();
			}
			else {
				printf("Buffer is empty!");
			}
			break;
		case 3:
			exit(0);
			break;
		}
	}
}
```

OUTPUT:

1. Press 1 for Producer

2. Press 2 for Consumer

3. Press 3 for Exit

Enter your choice: 1

Producer produces item 1

Enter your choice: 1

Producer produces item 2

Enter your choice: 2

Consumer consumes item 2

Enter your choice: 2

Consumer consumes item 1

Enter your choice: 2

Buffer is empty!

Enter your choice: 3

---

14. Write a Program to demonstrate the concept of process creation.

```
#include <stdio.h>
 #include<unistd.h>
 int main()
{
        fork();
        printf("SRMIST\n");
        return 0;
}
```

O/P *// SRMIST will be printed 2^1times*

SRMIST

---

16. Write a program to find the factorial of a given number.

```
echo "Enter a number"
read num
fact=1
for((i=2;i<=num;i++))
{
  fact=$((fact * i))
}
echo $fact
```

---

18. Write a program using execl(). Rewrite the same using execlp() and execv() functions.

execl()

```
#include <unistd.h>
int main(void)
{
  char *binaryPath = "/bin/ls";
  char *arg1 = "-lh";
  char *arg2 = "/home";
  execl(binaryPath, binaryPath, arg1, arg2, NULL);
  return 0;
}
```

execlp()

```
#include <unistd.h>
int main(void)
{
char *programName = "ls";
char *arg1 = "-lh";
char *arg2 = "/home";
execlp(programName, programName, arg1, arg2, NULL);
 return 0;
}
```

execv()

```
#include <unistd.h>

int main(void)

{

  char *binaryPath = "/bin/ls";

  char *args[] = {binaryPath, "-lh", "/home", NULL};

  execv(binaryPath, args);

  return 0;

}
```

---

19. Write a program to check all the files in the present working directory for a pattern (passed through the command line) and display the name of the file followed by a message stating that the pattern is available or not available.

```
for i in *
do
        if [-f $i]
        then
                grep $1 $i > /dev/null
                if [$? -eq 0]
                then
                        echo $i found
                else
                        echo $i not found
                fi
        fi
done
```

21.Given the following values num=10, x=*, y=`date` a="Hello, 'he said'".Execute and write the output of the following commands

echo num ,echo $num,echo $x,echo $(date).

Command Output	Output
echo num	num

echo $num	10
echo $x	*
echo $(date)	Mon 04 Jul 2022 9:00:26 AM EST
Extra	
echo ‘$x’	$x
echo “$x”	*
echo $y	date
echo $a	Hello, 'he said'
echo \$num	$num
echo $$num	$10

---

24. Write a program to do the following:

Create two processes, one is for writing into the shared memory (shm_write.c) and another is for reading from the shared memory (shm_read.c) In the shared memory, the writing process, creates a shared memory of size 1K (and flags) and attaches the shared memory. The writing process writes the data read from the standard input into the shared memory. The last byte signifies the end of the buffer Read process would read from the shared memory and write to the standard

Writer process:

Algorithm:

Step 1 - Create a shared memory using (shmget()) function.

Step 2 - attach the current process into created shared memory by calling shmat() function.

Step 3 - Write into shared memory after attaching it to it.

Step 4- After completing the write operation detach the process from the shared memory area.

Reader process:

Algorithm:

Step 1 - Create a shared memory using (shmget()) function.

Step 2 - attach the current process into created shared memory by calling shmat() function.

Step 3 - read the data which is already written by the reader process from shared memory after attaching it to it.

Step 4- Print the string and detach the process from the shared memory area.

Writer Program:

```
#include<stdio.h>
#include<sys/ipc.h>
#include<sys/shm.h>
int main()
{
 int shmid;
 char *str;
 shmid=shmget((key_t)9,1024,IPC_CREAT|0666);
 str=(char *)shmat(shmid,(char *)0,0);
 printf("Write data:");
 fgets(str,20,stdin);
 printf("Data written in memory : %s \n",str);
 shmdt(str);
 return 0;
}
```

Reader Program :

```
#include<stdio.h>
#include<sys/ipc.h>
#include<sys/shm.h>
int main()
{
        int shmid;
        char *str;
    shmid=shmget((key_t)6,1024,IPC_CREAT|0666);
        str=(char *)shmat(shmid,(char *)0,0);
    printf("Data read from memory : %s \n",str);
        shmdt(str);
        shmctl(shmid,IPC_RMID,NULL);
        return 0;
```

}

Output:

Writer.c

Write Data : Operating System Data

Written in memory: Operating System

Reader.c

Data read from memory: Operating System

---

27. Write a shell script to print a greeting as specified below. If the hour is greater than or equal to 0 (midnight) and less than or equal to 11 (up to 11:59:59), "Good morning" is displayed. If the hour is greater than or equal to 12 (noon) and less than or equal to 17 (up to 5:59:59 p.m.), "Good afternoon" is displayed. If neither of the preceding two conditions is satisfied, "Good evening" is displayed.

```
hour = $(date | cut -c12-13)
if [ “$hour” -ge 0 -a “$hour” -le 11]
then
        echo Good Morning
elif [$hour -le 17]
then
        echo Good Afternoon
else
        echo Good Evening
fi
```

---

28. Write a program to check whether the file has execute permission or not. If not, add the permission.

```
echo Enter the name of the file
read name
if [-x $name]
then
        echo Yes
else
        echo No
fi
```
