Practical 1

1.1 Aim: To implement cp command.

Software Requirements: GNU C Compiler(GNU Compiler Collection), <sys/types.h> ,<sys/stat.h> ,<fcntl.h>, <unistd.h>

Hardware Requirement:

Description: Copy program should take multiple files as an input from the command line

Algorithm: Processor, Memory, Standard Input, Standard Output.

1. If read file does not exist declare error message.
   1. Exit.
2. Open read file in read mode.
3. If write file does not exist create file.
   1. Open newly created file in write mode.
4. Open file in write mode.
5. If EOF declare copy complete.
6. Read the data in to buffer
7. Write buffer data on to file.
8. Go to step 5.

Source Code:

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

int main(int argc, char* argv[])
{
      int fdr[10],i,index,fdw;
      char buf;
      if(argc <= 1)
      {
            write ( 1 , "Pass arguments" , 15 );
            _exit(0);
      }
      for(index = 1; index <= argc-1; index++)
      {
            fdr[index] = open( argv[index]  , O_RDONLY );
      }
      fdw = open( argv[argc] , O_WRONLY | O_APPEND);
      index=1;
      while(index <= argc-1)
      {
            if( fdr[index] > 0 )
            {
                  while( (i = read ( fdr[index] , &buf , 1 )) > 0 )
                  {
                        write ( fdw , &buf , 1 );
                        }
                }

                else
                        write ( 1 , "ERR\n" , 5);
                index++;
        }

        for(index = 1; index < argc; index++)
                close(fdr[i]);
        return 0;
}
```

Input Output:

```
$cat a
abc
$cat b
$gcc copy.c -o copy
$./copy a b
$cat b
abc
$
```

1.2 Aim: To implement cat command.

Software Requirements:GNU C Compiler(GNU Compiler Collection), <sys/types.h> ,<sys/stat.h> ,<fcntl.h>, <unistd.h>

Hardware Requirement: Processor, Memory, Standard Input, Standard Output.

Description: cat program should write the content of the specified file on to the standard output.

Algorithm:

1. If read file does not exist declare error message.
    1. Exit.
2. Open read file in read mode.
3. If EOF declare copy complete.
4. Read the data in to buffer.
5. Write buffer data on to terminal.
6. Go to step 3.

Source Code:

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

```
int main(int argc, char* argv[])
{
      int fdr[10],i,index;
      char buf;
      if(argc <= 1)
      {
            write ( 1 , "Pass arguments" , 15 );
            _exit(0);
      }
      for(index = 1; index <= argc; index++)
      {
            fdr[index] = open( argv[index]  , O_RDONLY );
      }
      index=0;
      while(index <= argc)
      {
            if( fdr[index] > 0 )
            {
                  while( (i = read ( fdr[index] , &buf , 1 )) > 0 )
                  {
                        write ( 1 , &buf , 1 );
                  }
            }

            else
                  write ( 1 , "ERR\n" , 5);
            index++;
      }

      for(index = 1; index < argc; index++)
            close(fdr[i]);
      return 0;
}
```

Input Output:

```
$cat a
abc
$cat b
xyz
$gcc cat1.c -o cat1
$./cat1 a b
abc
xyz
$
```

Practical 2

2.1 Aim: To Demonstrate the basics of fork utility.

Software Requirements: GNU C Compiler(GNU Compiler Collection), <sys/types.h> ,<stdio.h>, <string.h>

Hardware Requirement: Processor, Memory, Standard Input, Standard Output.

Description: Implement basic function of fork which will fork a process in main and will print process id of parent and child.

Source Code:

```
#include  <stdio.h>
#include  <string.h>
#include  <sys/types.h>

#define   MAX_COUNT  200
#define   BUF_SIZE   100

void  main(void)
{
     pid_t  pid;
     int    i;
     char   buf[BUF_SIZE];

     fork();
     pid = getpid();
     for (i = 1; i <= MAX_COUNT; i++) {
          sprintf(buf, "This line is from pid %d, value = %d\n", pid, i);
          write(1, buf, strlen(buf));
     }
}
```

Input Output:

```
     ................
This line is from pid 3456, value 13
This line is from pid 3456, value 14
     ................
This line is from pid 3456, value 20
This line is from pid 4617, value 100
This line is from pid 4617, value 101
     ................
This line is from pid 3456, value 21
This line is from pid 3456, value 22
     ................
```

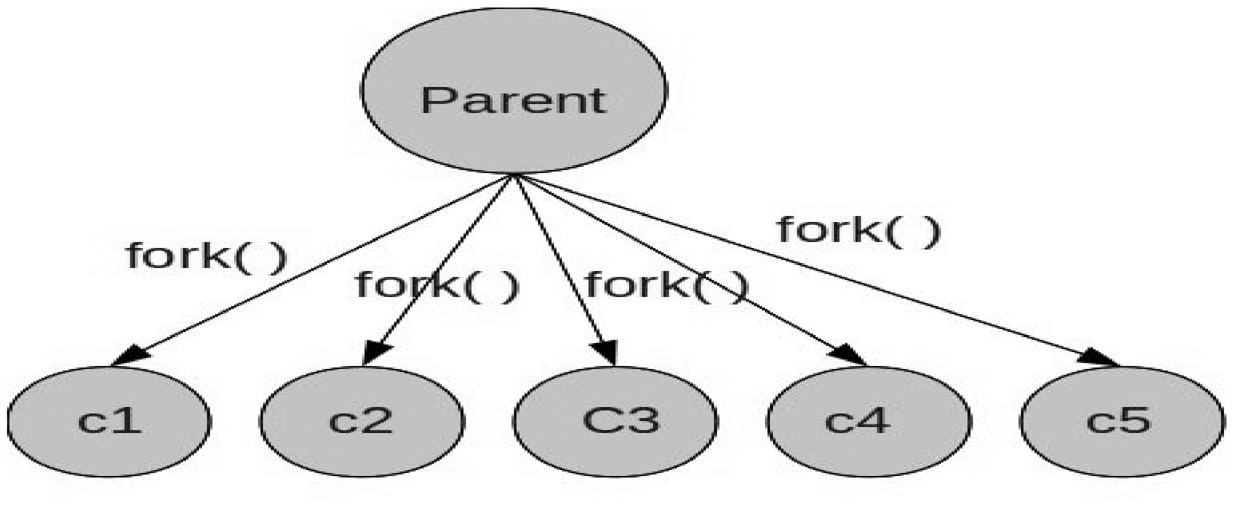
2.2 Aim: To implement Fanning and Chaining.

Software Requirements: GNU C Compiler(GNU Compiler Collection), <sys/types.h> ,<sys/stat.h> ,<fcntl.h>, <unistd.h>

Hardware Requirement: Processor, Memory, Standard Input, Standard Output.

Description:

Fanning and chaining:



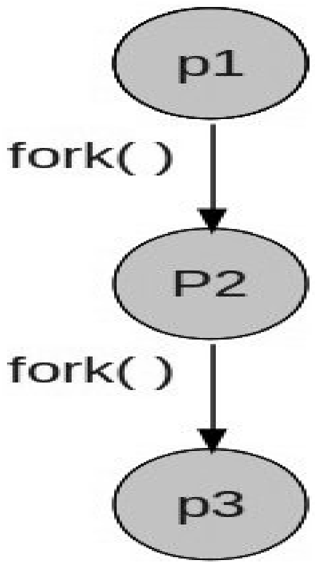


Illustration 1: 1. Fanning 2. Chaining

Source Code:

chaining:

```
#include<stdio.h>
#include<stdlib.h>
int main(int argc,char* argv[])
{
      int i=11,j=0;
       int retval,flag=0;
      retval = fork();
      if(retval==0)
      {
            for(i=0;i<atoi(argv[1]);i++)
            {
                  if(retval>0)
                  {
                  }
                  else
                  {
                  printf("child id: %d and  my parent is %d\n",getpid(),getppid());
                  sleep(1);
                  retval = fork();
                  }
            }
      }
return 0;
}
```

Fanning:

```
#include<stdio.h>
#include<stdlib.h>
int main(int argc,char* argv[])
{
      int i=11,j=0;
      int retval;
      retval = fork();

if(retval>0)
{
      for(i=0;i<atoi(argv[1]);i++)
      {
            if(retval==0)
                  {

                  }
            else
            {
                  printf("parent %d and  my child is %d \n",getpid(),retval);
                  sleep(1);
                  retval = fork();
            }
      }
}
return 0;
}
```

Input Output:

Chaining:

```
$ gcc chaining.c -o chaining
$ ./chaining
$ ./chaining 10
$ child id: 2623 and  my parent is 2622
child id: 2624 and  my parent is 2623
child id: 2625 and  my parent is 2624
child id: 2626 and  my parent is 2625
child id: 2627 and  my parent is 2626
child id: 2628 and  my parent is 2627
child id: 2629 and  my parent is 2628
child id: 2630 and  my parent is 2629
child id: 2631 and  my parent is 2630
child id: 2632 and  my parent is 2631
```

Fanning:

```
$ gcc fanning.c -o fanning
$ ./fanning 10
parent 2665 and  my child is 2666
parent 2665 and  my child is 2667
parent 2665 and  my child is 2669
parent 2665 and  my child is 2670
parent 2665 and  my child is 2671
parent 2665 and  my child is 2672
parent 2665 and  my child is 2673
parent 2665 and  my child is 2674
parent 2665 and  my child is 2675
parent 2665 and  my child is 2676
```

<u>**Practical 3**</u>

3.1 Aim: To Demonstrate basics of threading mechanism in C.

Software Requirements: GNU C Compiler(GNU Compiler Collection), <sys/types.h> ,<sys/stat.h> ,<fcntl.h>, <unistd.h>

Hardware Requirement: Processor, Memory, Standard Input, Standard Output.

Source Code:

```
#include<pthread.h>
#include<stdio.h>
void ful()
{
      printf(“Thread prog\n”);
}
int main()
{
      pthread_t tid[5];
      int i,t;
      for(i=0;i<5;i++)
      {
            if((t=pthread_ctrate(&tid[i]),NULL,(void *)ful,NULL)<)
                  printf(“Error”);
      }
      for(i=0;i<5;i++)
            pthread_join(&tid[i],NULL);
}
```

Input Output:

```
$gcc myth.c –o myth –lpthread
$./myth
Thread prog
Thread prog
Thread prog
Thread prog
Thread prog
```

3.2 Aim: Matrix multiplication.

Software Requirements: GNU C Compiler(GNU Compiler Collection), <iostream> , <stdio.h> , <pthread.h> , <stdlib.h>

Hardware Requirement: Processor, Memory, Standard Input, Standard Output.

Description: Multiplication of the matrix can be done using thread as no element of the resulting matrix is dependent on the previous element so all can be calculated parallel.

Source Code:

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

using namespace std;


const int   MAX = 3;
const int   num_of_thr = 4;
int         matrix_A[MAX][MAX];
int         matrix_B[MAX][MAX];
int         c[MAX][MAX];



//worker thread
void* matrix_multi(void*)
{
    for(int i = 0; i < MAX; i++)
    {
        for(int j = 0; j < MAX; j++)
        {
            c[i][j] = 0;
            for(int k = 0; k < MAX; k++)
            {
                c[i][j] += matrix_A[i][k] * matrix_B[k][j];
            }
        }
    }
    pthread_exit(0);
}

int main()
{

    pthread_t thr_id[MAX][MAX];




   //Filling the Matrices
    for(int i = 0; i < MAX; i++)
    {
        for(int j = 0; j < MAX; j++)
        {
            matrix_A[i][j]=  1;
            matrix_B[i][j] =  1;
        }
    }


    //create the threads
    for(int i = 0; i < num_of_thr/2; i++)
    {
        for(int j = 0; j < num_of_thr/2; j++)
        {
            pthread_create(&thr_id[i][j],NULL,matrix_multi, NULL);
        }
    }

    //joining the threads
    for(int i = 0; i < num_of_thr/2; i++)
    {
        for(int j = 0; j < num_of_thr/2; j++)
        {
        pthread_join(thr_id[i][j],NULL);
        }
    }
      for(int i = 0; i < MAX; i++)
      {
        for(int j = 0; j < MAX; j++)
        {
            printf("%d\t",c[i][j]);
        }
      printf("\n");
    }
     return 0;
}
```

Input Output:

```
$ gcc matrix.c -o matrix -lpthread

$ ./matrix
 3 3 3
 3 3 3
 3 3 3
$
```