Week : 30th march to 3rd April

• Lab assignment 1 :

For coperating processes, they must communicate to accomplish one goal, because without communica co-operation is not possible in any case.

So,

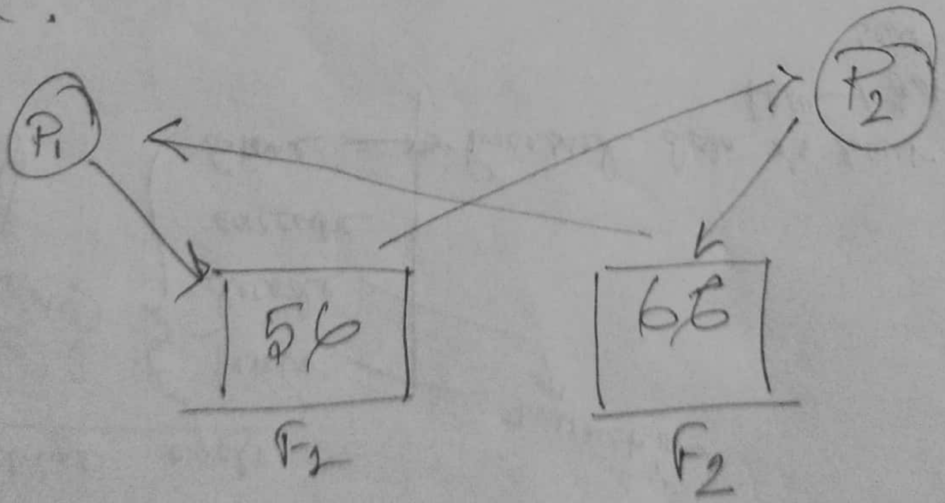
1. Processes can communicate using Files.

Ex(1.) communication between Parent & child Process ; It may be passing any value.

Ex (2). Parent Process creates 2 files before forking child process.

Ex (3). child process inherits file descriptors from parent and they share the file pointers

Ex (4) Can use one for parent to write and child to read other for child to write & parent to read.



2. OS supports something called Pipe.

→ Pipe provides two file descriptors (int fd[2])

→ Read from fd[0] accesses data written to fd[1] in FIFO (first in first out) order and vice-versa.

→ This idea is somewhat like communication using files ~~$~~. But rather than using file descriptors to read from and write into files, it short circuits the mechanism, bypasses the disc and directly does the communication through the operating system. So, same communication can be done without the overhead of involving the hard-disc (as files are not accessed).

Scanned with CamScanner

Two types of pipes are there :

1. unnamed Pipe
2. Named pipe

1. unnamed Pipe

i) These are created by the shell
ii) unidirectional
iii) used between the child & Parent Process.

2. Named Pipe

i) created using mkfifo command
ii) can be accessed by two any unrelated Processes.
iii) Bidirectional.

Example : Named Piple (writer Program) ⇒

```
#include<stdio.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<fcntl.h>
#include<unistd.h>
int main()
{ int fd, retval;
  char buffer[10] = "STUDENT.ID";
  fflush(stdin);
  retval = mkfifo("/tmp/myfifo", 0666);
  fd = open("/tmp/myfifo", O_WRONLY);
  write(fd, buffer, sizeof(buffer));
  close(fd);
 return 0;
```

Two types of pipes are there :

1. unnamed Pipe }
2. named pipe }

1. unnamed Pipe

i) These are created by the shell
ii) unidirectional
iii) used between the child & Parent Process.

2. Named Pipe

i) created using mkfifo command
ii) can be accessed by two any unrelated Processes.
iii) Bidirectional.

Example : Named Piple (writer Program) ⇒

```
#include<stdio.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<fcntl.h>
#include<unistd.h>
int main()
{ int fd, retval;
  char buffer[10] = "STUDENT.ID";
  fflush(stdin);
  retval = mkfifo("/tmp/myfifo", 0666);
  fd = open("/tmp/myfifo", O_WRONLY);
  write(fd, buffer, sizeof(buffer));
  close(fd);
 return 0;
}
```

Example : Named Pipe (Reader Program) →

```
# include <stdio.h>
# include <sys/stat.h>
# include <sys/types.h>
# include <fcntl.h>
# include <unistd.h>
int main ()
{ int fd, retval ;
  char buffer [10] ;
  fd = open (" /tmp/myfifo ", O_RDONLY );
  // as we have already create the fifo , we just
  / need to open it .
  retval = read (fd , buffer , sizeof (buffer)) ;
  fflush ( stdin ) ;
  write (1, buffer , size (buffer));
  // we are now reading the content out from the
  //buffer & writting it to the screen ,(as you
  // know '1' = Standard output for fd value)
  close (fd) ;
  return 0 ;
}
```

Scanned with CamScanner

Program to use concept of Pipe : (1) [UNNAMED PIPE]
writer = Child

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

int main(int argc, char *argv[])
{
    Pid_t pid;
    int mypipefd[2];
    int ret;
    char buf[20];
    ret = pipe(mypipefd);
    if (ret == -1)
     {
         Perror("pipe");
         exit(1);
     }
    pid = fork();
    if (pid == 0)
     {   /* child Process */
         Printf("child Process\n");
         write(mypipe[1], "hello there!", 12);
     }
     else
     {  /* Parent process */
         printf("Parent process\n");
         read(mypipefd[0], buf, 15);
         printf("buf: %s\n", buf);
     }
  return 0;
}
```

Example of Pipe: (2) [UNNAMED PIPE]
Writer = Parent

```
# include <stdio.h>
# include <unistd.h>
int main (void)
{
      int pipefd[2];
      int ret;
      char buffer [15];
      pipe (pipefd);
      ret = fork();
      if (ret > 0) {
      fflush (stdin); // clean the standard i/p line
      printf ("Parent Process\n");
      write (Pipefd[1], "Hello Students!", 15);
      }
      if (ret == 0)
      { sleep (5);
        fflush (15);
        printf ("Child Process");
        read (Pipefd[0], buffer, sizeof(buffer));
        write (1, buffer, sizeof(buffer));
       }
      return 0;
}
```

Scanned with CamScanner