CO-1

PART-C

6)

6A) A) (i) dup system call

Syntax:

int dup(int oldfd)

old file descriptor whose copy is created.

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
int main()
{
    int fd = open("f1.txt", O_WRONLY | O_APPEND);
    if(fd < 0)
        perror("Error in opening f1\n");
    int fd1 = dup(fd);
    write(fd1, "Hello world\n", 46);
    write(fd, "Helloworld\n", 51);
    return 0;
}
```

(i) pipe is one way communication we can use pipe such that one process write to the pipe and the other process reads from the pipe. The pipe can be used to create child processes for reading and writing.

Each end of the pipe is closed individually using normal close() system call.

when Reading from pipe read() will return (end of file) when the write end of the pipe is closed. if writer end of the is still open and there is no data, read() will sleep until input becomes available.

Writing to pipes:

if read end of pipe is closes, a write() will fail and process will be sent SIGPIPE signal. Default SIGPIPE handler terminates.

6. B)

(i) echo.c

```
#include "types.h"
#include "stat.h"
#include "user.h"
int main(int argc, char *argv[])
{  int i;
   for(int i = 1; i < argc; i++)
      printf(1, "%s %s", argv[i], i+1 < argc ? " " :
                                                "\n");
   exit();
}
```

(ii) rm.c

```
#include "types.h"
#include "stat.h"
#include "user.h"
int main (int argc, char *argv[])
{  int i;
   if(argc < 2){
      printf(2, "usage: rm files...\n");
   exit();
```

```
for (i=1; i<argc; i++){
    if (unlink(argv[i])<0) {
        printf(2, "rm: %s failed to delete\n",
                                    argv[i]);
        break;
    }
}
exit();
}
```