Pipes

1. pipe - Link 1
    - `int pipe(int pipefd[2])`
    - pipe() creates a pipe, a unidirectional data channel that can be used for interprocess communication. The array pipefd is used to return two file descriptors referring to the ends of the pipe. pipefd[0] refers to the read end of the pipe. pipefd[1] refers to the write end of the pipe.

2. write - Link 1
    - `ssize_t write(int fd, const void buf[.count], size_t count)`
    - write() writes up to *count* bytes from the buffer starting at *buf* to the file referred to by the file descriptor *fd*. On success, the number of bytes written is returned. On error, -1 is returned, and errno is set to indicate the error.
3. read - Link 1
    - `ssize_t read(int fd, const void buf[.count], size_t count)`
    - read() attempts to read up to *count* bytes from file descriptor *fd* into the buffer starting at *buf*. On success, the number of bytes read is returned (zero indicates the end of the file), and the file position is advanced by this number.
4. close - Link 1, Link 2(Highly recommended read)
    - `int close(int fd)`
    - close() closes a file descriptor so that it no longer refers to any file and may be reused. close() returns zero on success. On error, -1 is returned, and errno is set to indicate the error.
5. execlp -
    - `int execlp(const char *file, const char *arg, ...)`
    - The file argument is the path name of an executable file to be executed. arg is the string we want to appear as argv[0] in the executable. By convention, argv[0] is just the executable file name; normally, it's set to the same as the file. The ... are now the additional arguments to give to the executable.
6. dup2 - Link 1, Link 2
    - `int dup(int oldfd), int dup2(int oldfd, int newfd)`
    - The dup() system call allocates a new file descriptor that refers to the same open file description as the descriptor oldfd. The dup2() system call performs the same task as dup(), but instead of using the lowest-numbered unused file descriptor, it uses the file descriptor number specified in newfd. In other words, the file descriptor newfd is adjusted to now refer to the same open file description as oldfd.
7. wait - Link 1,
    - `pid_t wait(int *_Nullable wstatus)`
    - Kindly go through this

Problem 0

Write a C program that creates a pipe, writes a message to one end, and reads and displays it from the other end.

Solution

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

int main(void) {
  int pfds[2];
  char buf[30];

  // Create pipe
  if (pipe(pfds) == -1) {
     perror("Error in creating pipe\n");
     exit(1);
  }

  printf("Writing to the file descriptor #%d\n", pfds[1]);
  write(pfds[1], "test", 5);
  printf("Reading from file descriptor #%d\n", pfds[0]);
  read(pfds[0], buf, 5);
  printf("We have read: \"%s\"\n", buf);
  return 0;
}
```