System Programming Assignment #1

Student Name: 林楷恩
Student ID: b07902075

1 File Redirection

(a)
- Redirect a.out's standard input to read from the file named infile.
- Redirect a.out's standard error to standard output.
- Redirect a.out's standard output to write to the file named outfile.

(b)
```c
int main(int argc, char *argv[])
{
    int fd1, fd2;
    fd1 = open (infile, O_RDONLY);
    fd2 = open (outfile, O_WRONLY | O_CREAT, 0666);

    // redirect standard input to infile
    dup2(fd1, 0);
    // redirect standard error to standard output
    dup2(1, 2);
    // redirect standard output to outfile
    dup2(fd2, 1);

    execlp("./a.out", "./a.out", (char *)0);
    return 0;
}
```

2 Atomic operation

(a) **Yes, function `write_to_fd` should be an atomic operation.** Since the function accepts a file descriptor, the file is opened before the function is called. If the process fork a child process after opening the file and before calling this function, then there exists another process that inherits the descriptor table, whose `fd` is pointed to the same file table which containing the current file offset. Between the time the parent process calls `lseek(fd, offset, SEEK_SET)` and `write(fd, buf, nbytes)`, if the CPU context switch to that child process and it calls `lseek(fd, 0, SEEK_SET)` or `read(fd, &x, 1)`, i.e. anything that changes the current file offset, then after the CPU context switch back to parent process, the current file offset is not at the specified `offset` and thus it write to wrong position when calling `write(fd, buf, nbytes)`. Thus, we should make `write_to_fd` be an atomic operation to prevent it from misbehaving.

(b) **No, function `write_to_fn` is not necessarily be an atomic operation.**. Every time this function is called, it use `open` function to create a brand new open file table entry and file table, which have its own current file offset. Though there might be another process write to the same file between `lseek` and `write`, the function will definitely write to the requested offset, no matter what contents are overwritten.