CSCE-313 Quiz 4 [25 points]

Solutions must be posted by 11:59pm THU Mar 31 on eCampus to receive credit. Late submissions are not allowed. You can print, write, scan and upload to eCampus. Or you can type answers and upload. In either case, please upload your answers in PDF format.

NOTE1: For details regarding any system calls invoked in the code below: you can grab the usage from MAN pages in linux (example: http://linux.die.net/man or from command "man <cmd>" in shell).

NOTE2: Answers to questions below without explanation will only receive partial credits.

Q1. [5 points] What would this code print for file containing the text "aakash"? **Write your answer below along with an explanation.**

```
int main(int argc, char *argv[])
{
    int fd1, fd2, fd3;
    char c1, c2, c3;
    char *fname = argv[1];
    fd1 = Open(fname, O_RDONLY, 0);
    fd2 = Open(fname, O_RDONLY, 0);
    fd3 = Open(fname, O_RDONLY, 0);
    Dup2(fd2, fd3);
    Read(fd1, &c1, 1);
    Read(fd2, &c2, 1);
    Read(fd3, &c3, 1);
    printf("c1 = %c, c2 = %c, c3 = %c\n", c1, c2, c3);
    return 0;
}
```

> c1 = a, c2 = a, c3 = a

The code will print as shown above. The first file descriptor takes in the first A, and the second takes in the second A. Since the second file descriptor duplicates itself into the third, the third file descriptor will also read A.

Q2. [5 points] What would this code print for file containing the text "savannah"? **Write your answer below along with an explanation.**

```
int main(int argc, char *argv[])
{
    int fd1;
    int s = getpid() & 0x1;
    char c1, c2;
    char *fname = argv[1];
    fd1 = Open(fname, O_RDONLY, 0);
    Read(fd1, &c1, 1);
    if (fork()) {
        sleep(s);
        Read(fd1, &c2, 1);
        printf("Humphry: c1 = %c, c2 = %c\n", c1, c2);
    } else {
        sleep(1-s);
        Read(fd1, &c2, 1);
        printf("Bogart: c1 = %c, c2 = %c\n", c1, c2);
    }
    return 0;
}
```

> Bogart: c1 = s, c2 = a

> Humphry: c1 = s, c2 = v

Bogart prints first, since the first process iterated is the parent. This statement sleeps for the number of seconds of the process ID, reads the given file, and prints the line containing the first and second letters of the input file. The Humphrey statement is the child statement, and prints second. It sleeps for the number of seconds inverse of the process ID, reads the given file, and prints the line containing the first and the second letters of the file, as well. Since the letter 'A' has already been printed, the statement moves on and prints the third letter.

Q3. [5 points] What would be the contents of the resulting file from execution of the program below? Explain your reasoning. You can grab the usage of "open", "write" from MAN pages in linux (example: http://linux.die.net/man)

```
int main(int argc, char *argv[])
{
    int fd1, fd2, fd3;
    char *fname = argv[1];
    fd1 = Open(fname, O_CREAT|O_TRUNC|O_RDWR, S_IRUSR|S_IWUSR);
    Write(fd1, "pqrs", 4);
    fd3 = Open(fname, O_APPEND|O_WRONLY, 0);
    Write(fd3, "jklmn", 5);
    fd2 = dup(fd1);  /* Allocates descriptor */
    Write(fd2, "wxyz", 4);
    Write(fd3, "ef", 2);
    return 0;
}
```

> pqrswxyznef

The program first opens and writes the first statement to the given file descriptor FD1, and then appends the FD2 text after it. Since the original FD2 simply duplicated FD1, the program skips over the text that's already been written and writes the next FD2 text after it. The program then takes the last (5^{th}) character of FD3 (as permitted by the system call), and the final 2 characters of the FD3 after that to complete our output above.

Q4. [5 points] Consider the following code:

```
int main(int argc, char* argv[]) {
  char buf[] = "ab";
  int r = open("file.txt", O_RDONLY);
  int r1, r2, pid;
  r1 = dup(r);
  read(r, buf, 1);
  if((pid=fork())==0) {
    r1 = open("file.txt", O_RDONLY);
  }
  else{
    waitpid(pid, NULL, 0);
  }
  read(r1, buf, 1);
  printf("%s", buf);
  return 0;
}
```

Assume that the disk file "file.txt" contains the string of bytes 15213 . Also assume that all system calls succeed. What will be the output when this code is compiled and run? Explain your answer.

Reference: **(a) waitpid()** system call suspends execution of the calling process until a child specified by pid argument has changed state. **(b) dup(fd)** copies the given file descriptor fd into the lowest-numbered available file descriptor and then returns the new file descriptor.

> ***1b*5b**

The program takes in the bit string from the file, and first loads in the string "ab" into the buffer. Since the first process is a parent process, it takes the first character from the input file and prints it out after a preceding asterisk (*). Since there is another remaining space in the buffer, it then prints the element at the second index of the buffer, followed by an asterisk (*) again. It repeats this process for the second character from the input file, printing that element followed by the element at the second index of the buffer again.

Q5. [5 points] Consider the following code

```
int main(int argc, char* argv[]) {
  char buf[] = "abc";
  int r = open("file.txt", O_RDWR);
  int r1 = 0;
  int r2 = open("file.txt", O_RDWR);
  dup2(r, r1);
  read(r, buf, 1);
  read(r2, buf, 2);
  write(r, buf, 3);
  read(r2, buf, 1);
  write(r1, buf, 1);
  return 0;
}
```

Assume that the disk file ***file.txt*** originally contains the string of bytes **12345** . Also assume that all system calls succeed. **What will file.txt contain when this code is compiled and run? Explain your answer.**

> 112c2

The program writes to the file as shown above. The first and third r variables originally open the input file, while the second stores the value of 0. When the second r variable duplicates the file descriptor of the first, they are all now accessing this file, but a bit differently than one another. As we read the first 2 r variables from the buffer “abc”, we print the first character from the input file back to the same file. We do the same with the second variable in the third index, and since we have an open element of the buffer still, we print the third character of the buffer next. Finally, since the next r variable was previously duplicated, it will print again, resulting in the final string we get above.