Assignment : 2

List of Topics: Basic File I/O, fork() syscall

Introduction to File I/O

- creat(pathname, mode): creates file.
  creat(“./newfile.txt”, 0777);

- int fd=open(pathname, flag): A file is opened or created by calling the open function.
  Flags: O_RDONLY: Open for reading only.
  O_WRONLY: Open for writing only.
  O_RDWR: Open for reading and writing.
  Returns fd(file descriptor, which is an integer value).

- close(fd): An open file is closed by calling the close function.

- lseek(filedes, offset, whence): it changes the positions of the read/write pointer within the file. In every file any read or write operations happen at the position pointed to by the pointer. lseek() system call helps us to manage the position of this pointer within a file.
  Whence: SEEK_SET: from beginning of the file
  SEEK_CUR: from the current position in the file
  SEEK_END: from the end of the file

- read(fd, buf, nbytes): Data is read from an open file with the read function. ●

write(fd, buf, nbytes): Data is written to an open file with the write function.

- dup(): The dup() system call creates a copy of a file descriptor (to the lowest available file descriptor.).

```
int dup(int oldfd);
```

- dup2() : The dup2() system function is used to create a copy of an existing

file descriptor.

```
int dup2(int oldfd, int newfd);
```

FD: 0:STDIN
1:STDOUT
2:STDERR

Reference:

https://www.geeksforgeeks.org/input-output-system-calls-c-create-open-close-read-write/

Example: 1

```
#include <stdio.h>
#include <unistd.h> //dup, dup2
#include <fcntl.h> //read, write

int main()
{
        int fd;
        char msg[50]="This is the first line of the file.";

        fd=open("text.txt",O_RDWR);

        if(fd!=-1)
        {
              printf("Unable to open file");
        }
        else{
              read(fd,msg,sizeof(msg));
              write(fd,msg,sizeof(msg));
              close(fd);
        }
        return 0;
}
```

Example: 2

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

```
#include <unistd.h>
#include <fcntl.h>

int main(void) {
  int number1;
  int save_stdin = dup(0); /* save original fd */
  int input_fds = open("./numbers", O_RDONLY);
  if(dup2(input_fds, 0) < 0) {
    printf("Unable to duplicate file descriptor.");
    exit(EXIT_FAILURE);
  }
  scanf("%d", &number1);
  printf("%d",number1);
  return EXIT_SUCCESS;
}
```

Exercise:

1) Continuing **Example: 1,** write a C program to append - “This is appended line.” to the same file. And fetch the appended statement from the open file. 2) Write a program, to read the first 10 characters of the file **text. (attach two different outputs 1) O_RDONLY 2) O_WRONLY)**

3) Write a program which reads numbers from an input file and performs addition of the numbers.(use dup2()).

Basic Process control program:

1. wait()
2. exit()

---

3. fork()

4. Exec Family

1. wait()

A call to wait() blocks the calling process until one of its child processes exits or a signal is received. After the child process terminates, the parent continues its execution after the wait system call instruction. Child can send a signal to the parent by exec, exit, etc.

Syntax for wait:

pid_t wait(int *status);
pid_t waitpid(pid_t pid, int *status, int options); //If the parent wants to wait for a specific pid.

By using the status, the child can give a signal to the parent, so that parent can do further procedure accordingly.

Example:

```
int main()
{
      pid_t cpid;
      if (fork()== 0)
                  exit(0); /* terminate child */
      else
            cpid = wait(NULL); /* reaping parent */
      printf("Parent pid = %d\n", getpid());
      printf("Child pid = %d\n", cpid);
```

```
    return 0;
}
```

2. exit()

exit() terminates the process normally.

---

status: Status value returned to the parent process. Generally, a status value of 0 or EXIT_SUCCESS indicates success, and any other value or the constant EXIT_FAILURE is used to indicate an error. exit() performs following operations.

- Flushes unwritten buffered data.
- Closes all open files.
- Removes temporary files.
- Returns an integer exit status to the operating system.

Library: #include<stdlib.h>

Introduction to Fork()

Fork system call is used for creating a new process, which is called child process, which runs concurrently with the process that makes the fork() call (parent process). After a new child process is created, both processes will execute the next instruction following the fork() system call. A child process uses the same pc(program counter), same CPU registers, same open files which use in the parent process.

Library: #include<unistd.h>

Below are different values returned by fork().

- Negative Value: creation of a child process was unsuccessful.
- Zero: Returned to the newly created child process.
- Positive value: Returned to parent or caller. The value contains the process ID of the newly created child process.

How to get the process id of any processes?

- pid_t getpid() : Process id of current process
- pid_t getppid(): Process id of the parent process

Understanding fork using examples

A call to fork() might fail with a return value of -1. For each problem below, assume that fork() succeeds at every call.

---

1. Enumerate all possible outputs of the following program.

```
int main() {
     int x = 3;
     if (fork() != 0)
          printf("x=%d\n", ++x);
     printf("x=%d\n", --x);
     exit(0);
}
```

2. How many “hello” output lines does this program print?

```
void doit() {
     if (fork() == 0) {
          fork();
          printf("hello\n");
          exit(0);
     }
     return;
}

int main() {
     doit();
     printf("hello\n");
     exit(0);
}
```

3. How many “hello” output lines does this program print?

```
void doit() {
     if (fork() == 0) {
          fork();
          printf("hello\n");
          return;
     }
```

```
        return;
}
int main() {
        doit();
        printf("hello\n");
        exit(0);
}
```

Excercise

1

```
int main() {
        int i, nchildren = 0;
        pid_t pid;
        for (i = 1; i <= 2; i++) {
                pid = fork();
                if (pid > 0)
                        nchildren++;
        }
        printf("I had %d children\n", nchildren);
        return nchildren;
}
```

(a) How many new processes (in addition to the initial process) are created by this code? Draw a process graph that illustrates the processes at run-time. Hint: It helps to write down the values of n children for each process.

(b) I executed this program once and generated the following output.

I had 1 children
I had 0 children
I had 2 children
I had 1 children

Adding the numbers suggests that 4 child processes were created. Explain why this differs from your answer for part (a).

2

How many total processes and child processes will be created in the following code?

```
void main()
{
int i, n;
for ( i=1; i<=n; i++ )
fork();
}
```

3

What will be the value of the abc? Explain it with the proper diagram.

```
void main(){
     int a=1, b=2, c=3;
```

```
     c+=a+=b+=c*2;

     print(a,b,c);
     if(fork() == 0){
          a+=b+=c+=3;
          print(a,b,c);
          c++;
     }
     else{
          b*=a*=c+=3;
          print(a,b,c);
          c - -;
     }
     c+=a+=b+1;
```

```
    print(c);
}
```

4

Predict the output for the following code and also illustrate it using graph.

```
#include <stdio.h>

#include <unistd.h>



int main()

{
    int pid;
    float a, b, c;
    pid = fork();
    if (pid == 0)
        a=1+1;
    else
        wait();

    b=2+2;
    c=a+b;
    printf("c=%f\n", c);
    return 0;
}
```

Submission:

1. Submit assignment with the report consists of an input file and output file with proper explanation of each output of all the exercises in pdf format. 2. Add all the outputs and a brief description of the commands used in the given demo scripts in the report.

3. Submitted code in a report or in a .sh file should be well commented. 4. Submit a zip file with your student id which will consist of the folder for each script & respective outputs. one common report in a parent directory.

Eg:

- IDNO.zip
    - report.pdf
    - PR1
        - Script
        - Its respective output
    - PR2

5. Submission Deadline: (Monday), 23:59:00
6. Late submission will result in 0 scores.