Tyler Paquet
Anthony Dowling
CIS 452
Due: 2/2/2017

Lab 3: Inter Process Communication

1. what does the program print, and in what order?

```
Waiting…
^C received an interrupt
Outta here
```

2. describe *exactly* what is happening to produce the answer observed for the above question

The program begins by printing waiting, and pausing the program. What happens when an interrupt is given is that the signal handler catches it and then prints “outta here” after handling the interrupt.

3. where does the standard output of the child process go? Explain.

The child process will run and it’s output will write to the temp file.

4. where does the standard output of the child process go? Explain.

The child’s output will still go to the temp file. The difference here is that the PARENT’S output will not be written to the temp file since dup2 is being run IN the child’s process

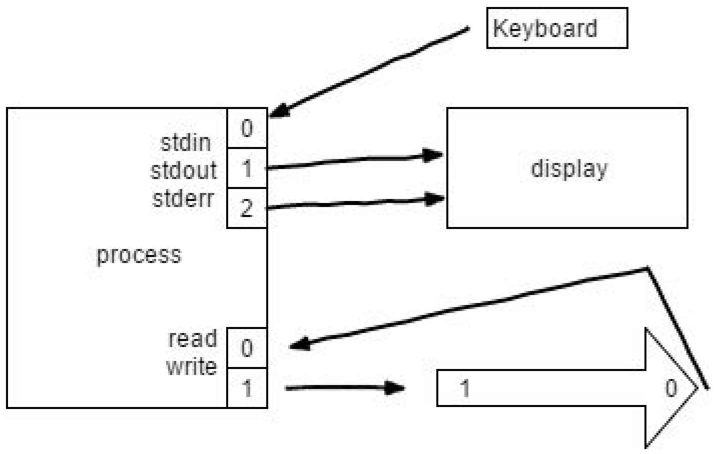
5. what exactly does the program do (i.e. describe its functionality)?

The program creates a pipe and checks to make sure that it runs correctly. If it does not run correctly then it prints “plumbing problem” and exits. If the pipe function runs correctly the program then forks and checks to make sure the fork function returns correctly. If it does not, then the program prints “fork failed”, and exits. If the fork call is executed properly then a child process is created. This child process then redirects the file descriptor for the write end of the pipe to the STDOUT_FILENO. The pipe is then closed and the process waits for the user to input a string. When the string is inputted, it is saved in the character array str. Str is then written to the STDOUT_FILENO stream and therefore enters the write side of the pipe. The child process then exits and the parents process redirects the file descriptor for the read end of the pipe to STDIN_FILENO. The parent process then reads the string in the pipe by calling the read function of the STDIN_FILENO stream and places it inside str. The parent process then prints the string to the stdout using puts and then the process returns and exits.

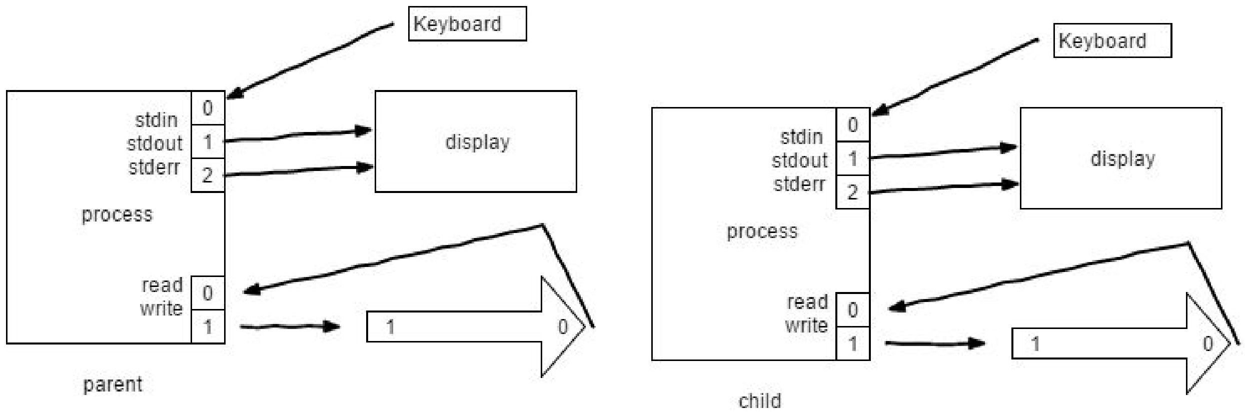
6. create a diagram that visually describes the input/output structure of the executing program. how processes and handles as in the pipe example diagrammed in class; show the file descriptor table as presented above in the File I/O section:

- At point A (after the pipe is created)
- At point B (after the parent forks)
- At point C (after the parent and child have duplicated descriptors)
- At point D (after parent and child close descriptors)

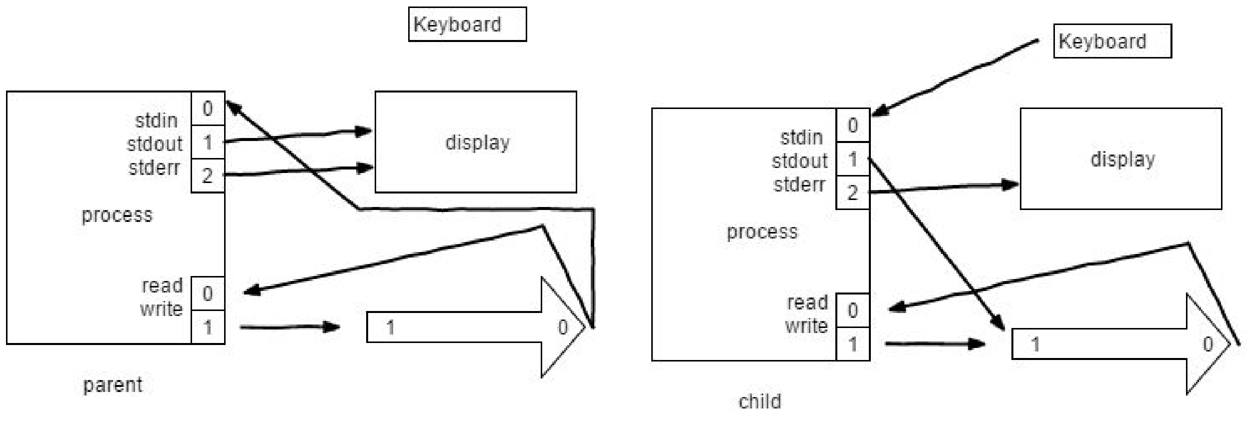
Point A



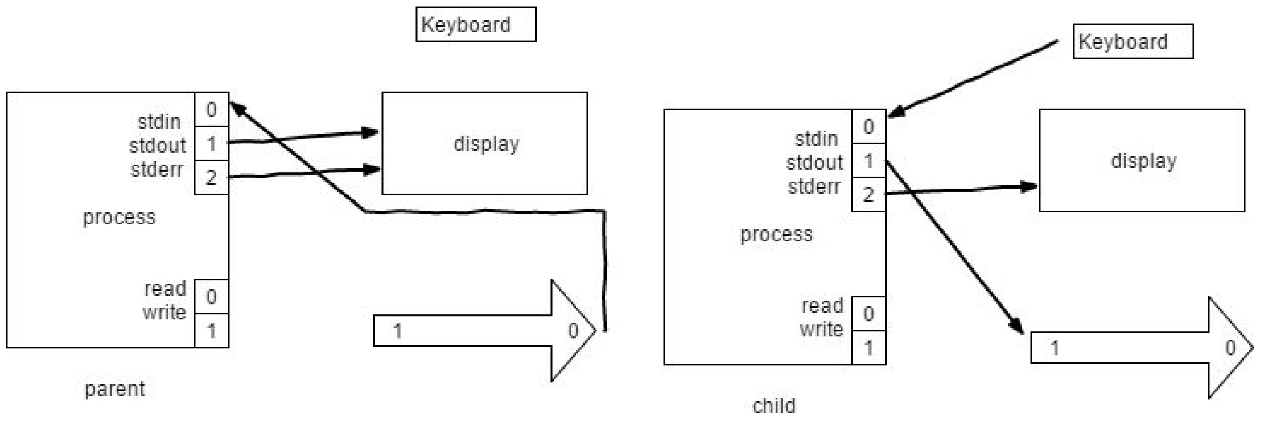
Point B



Point C


Keyboard
stdin
stdout
stderr
0
1
2
display
process
read
write
0
1
1
0
parent
Keyboard
stdin
stdout
stderr
0
1
2
display
process
read
write
0
1
1
0
child

Point D


Keyboard
stdin
stdout
stderr
0
1
2
display
process
read
write
0
1
1
0
parent
Keyboard
stdin
stdout
stderr
0
1
2
display
process
read
write
0
1
1
0
child

```
/*
* Lab 3
* Tyler Paquet
* Anthony Dowling
* Due: 2/2/2017
*/

#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <time.h>
#include <stdlib.h>

void handler1(int sig);
void handler2(int sig);
void handler3(int sig);
void handler4(int sig);

int exitprog = 0;

int main()
{
        pid_t pid1;
        signal(SIGUSR1, handler1);
        signal(SIGUSR2, handler2);
        pid1 = fork();
        if(pid1 == 0)
  {
        signal(SIGINT, handler4);
        while(pid1 == 0)
        {
                srand(time(NULL));
                int randtime = (rand() % 5) + 1;
                        sleep(randtime);
                int randsig = (rand() % 2) + 1;

                if(randsig == 1)
                        {
                                kill(getppid(), SIGUSR1);
                }
                        if(randsig == 2)
                        {
                                kill(getppid(), SIGUSR2);
                }
            }
    }
        else
        {
            signal(SIGINT, handler3);
            printf("spawned child PID# %d\n", pid1);
            while(exitprog == 0)
            {
                printf("waiting... ");
                fflush(stdout);
                pause();
            }
            printf("That's it, I'm shutting you down...\n");
            kill(pid1, SIGINT);
        }
        return 0;
}

void handler1(int sig)
{
        printf("received a SIGUSR1 signal\n");
}

void handler2(int sig)
{
        printf("received a SIGUSR2 signal\n");
}

void handler3(int sig)
{
        printf(" received.\n");
        exitprog = 1;
}

void handler4(int sig)
{
    exit(0);
}
```

Sample Output:

```
spawned child PID# 4767
waiting... received a SIGUSR1 signal
waiting... received a SIGUSR2 signal
waiting... received a SIGUSR1 signal
waiting... received a SIGUSR2 signal
waiting... received a SIGUSR1 signal
waiting... received a SIGUSR1 signal
waiting... received a SIGUSR1 signal
waiting... received a SIGUSR2 signal
waiting... received a SIGUSR2 signal
waiting... received a SIGUSR2 signal
waiting... received a SIGUSR2 signal
waiting... received a SIGUSR2 signal
waiting... received a SIGUSR1 signal
waiting... ^C received.
That's it, I'm shutting you down...
```