CIS 452
Lab 3 Report

Ashley Hendrickson
Muna Gigowski
Fall 2019

Signals

Question One

What does the program print, and in what order?

Waiting…

^C received an interrupt.
Outa here.

Question Two

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

When the program runs, it prints out “waiting…” and then executes pause() while it waits to receive the interrupt ctrl+C. Once the user types in ctrl+C, ^C prints out onto the screen and the program’s signal handler gets called and executes, immediately printing “received an interrupt.” after the displayed command. Then the code tells the program to sleep for one second, before finally printing "outta here.” and exiting.

File I/O

Question Three

Where does the standard output of the child process go? Explain.

The standard output of the child process points to the file named “temp,” just as the parent’s does, because when a process forks its child process inherits the file descriptors of the parent, and the parent process had already executed the dup2() redirect to the “temp” file before the fork occurred . Furthermore, since the table that holds the file descriptor entries is outside of the process code and data it is not affected (overridden) by the exec() command.

Question Four

Where does the standard output of the child process go? Explain.

The standard output of the child will remain the normal standard output (display), since it inherited the file descriptors of its parent when the parent forked, and at that

time it had not used any redirect logic to change its file descriptors yet. Even though after forking the parent used dup2() to redirect its file descriptors, that logic is done in the parent-specific code so it does not affect the child's file descriptors.

Pipes

Question Five

What *exactly* does the program do (i.e. describe its high-level functionality)?

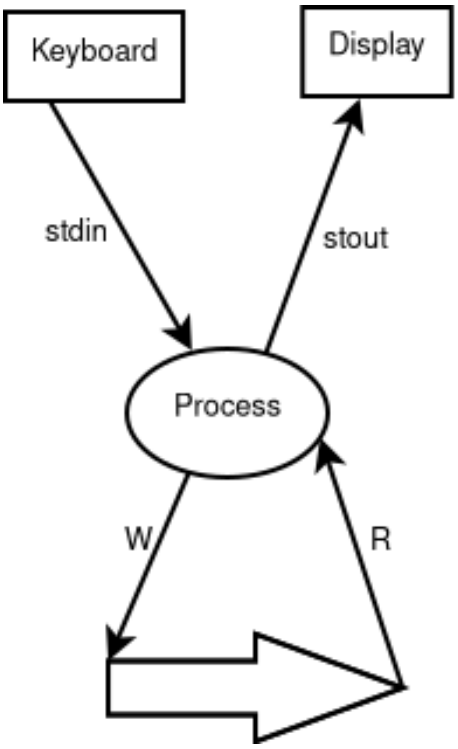
When the program executes, it executes and creates a pipe and then forks. Then it checks to see if it is in the child process, and if so it executes a dup2() redirect to write to the pipe that was created, instead of the standard output. Then it closes the read and write file descriptors. Finally, still within the child process code, the code reads in user console input and writes it to the pipe before exiting.

Underneath the conditional that checks for the child process, within the parent process code, a dup2() redirect is executed to read in from the pipe instead of the standard input (keyboard). Then it closes the read and write file descriptors. It then reads in from the pipe and, assuming the number of characters is less than the MAX allowed, it uses puts() to write what was in the pipe to the screen, before returning successfully.

Question Six

Create a diagram that visually describes the input/output structure of the executing program. Show 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.

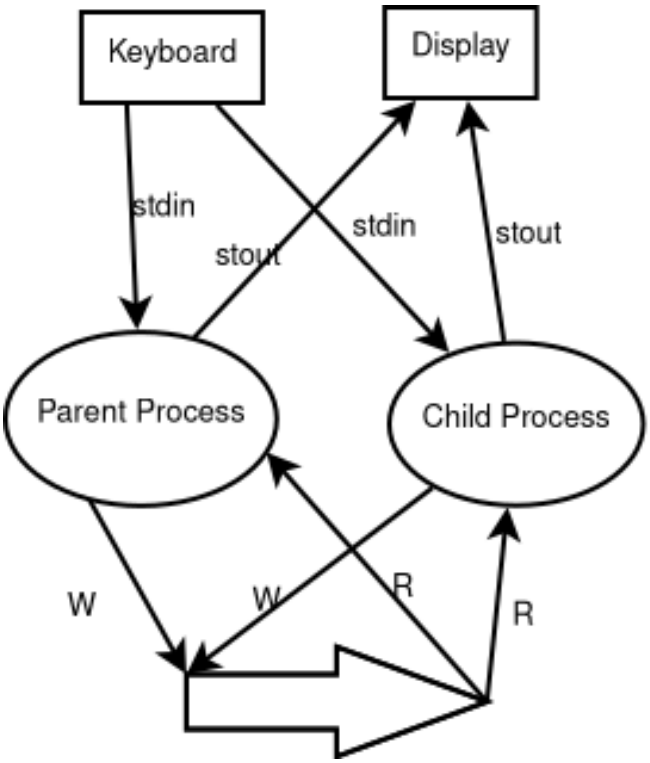
At Point A:



File Descriptors

0 -> stdin
1 -> stdout
2 -> stderr
r -> pipe
w -> pipe

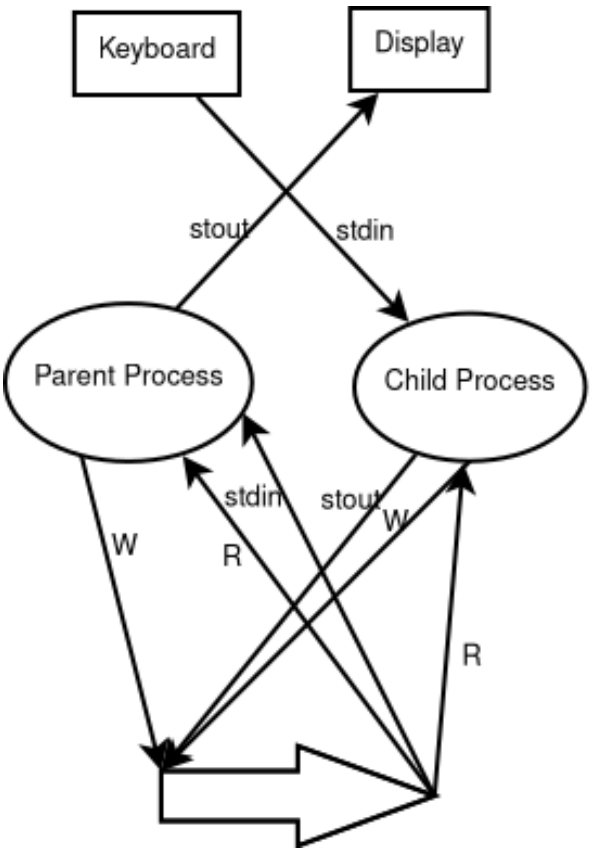
At Point B:



File Descriptors

0 -> stdin
1 -> stdout
2 -> stderr
r -> pipe
w -> pipe

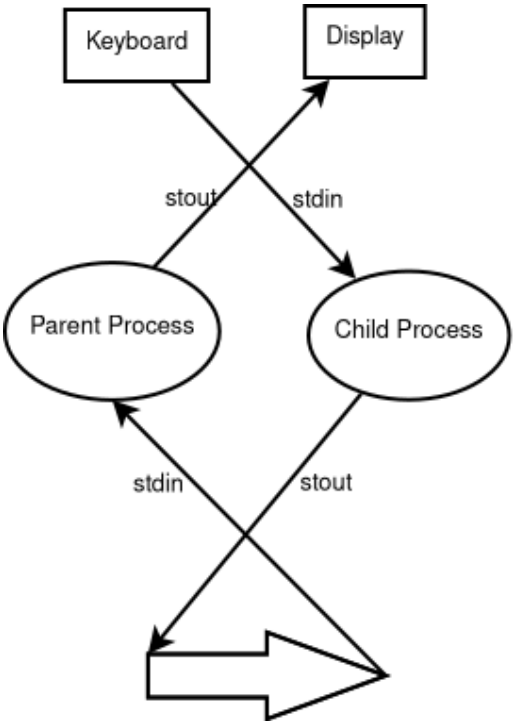
At Point C:



File Descriptors (parent)
0 -> fd[READ]
1 -> stdout
2 -> stderr
r -> pipe
w -> pipe

File Descriptors (child)
0 -> stdin
1 -> fd[WRITE]
2 -> stderr
r -> pipe
w -> pipe

At Point D:



File Descriptors (parent)
0 -> fd[READ]
1 -> stdout
2 -> stderr

File Descriptors (child)
0 -> stdin
1 -> fd[WRITE]
2 -> stderr

Programming Assignment

Source Code

```
#define _POSIX_SOURCE
#include <stdio.h>
#include <signal.h>
#include <sys/wait.h>
#include <errno.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
void handler(int signal);
void errorChecker();
void quit();
int i, pid1, pid2, status;
int main(){
        int sig, slp;
        pid1=getpid();
        printf("Parent pid=%d\n", pid1);
        if((pid2=fork())==0)//child
        {
                printf("Child pid =%d\n", pid2);
        while(1){
                sig=rand() % 2;
                slp=rand()%5+1;

                if(sig==1){
                printf("Child: sending parent SIGUSR1 ..........");
                fflush(stdout);
                kill(pid1, SIGUSR1);
                sleep(slp);
                }
                else{
                printf("Child: sending parent SIGUSR2 ..........");
                fflush(stdout);
                kill(pid1, SIGUSR2);
                sleep(slp);
                }
        }

        }
        else{
                if(signal(SIGINT, quit)==SIG_ERR){
                printf("Unable to install a signal hander for
SIGINT\n");
                }
                // Installing signal handlers
        if(signal(SIGUSR1, handler)==SIG_ERR){
                printf("Unable to install a signal handler for SIGUSR1\n");
        }
        if(signal(SIGUSR2, handler)==SIG_ERR){
                printf("Unable to install a signal handler for SIGUSR2\n");
        }
```

```
                //parent
                wait(&status);
                for(;;);
        }
}

void handler(int signo){
        if(signo==SIGUSR1){
                printf("received a SIGUSR1 signal %d \n", signo);

                //Reinstalling handlers
                if(signal(SIGUSR1, handler)==SIG_ERR)
                        printf("Unable to install a signal handler for
SIGUSR1\n");
}

        if(signo==SIGUSR2){
                printf("received a SIGUSR2 signal %d \n", signo);}
        //Reinstalling handlers
        if(signal(SIGUSR2, handler)==SIG_ERR)
                printf("Unable to install a signal handler for SIGUSR2\n");


        return;
}

void quit(){
if(pid1==getpid()){
//parent;
printf("\nending program .... \n");
exit(0);
}
else{
fflush(stdout);
        printf("...");
sleep(1);
exit(0);
}
}
```

Sample Output

```
[gigowskm@eos04 lab3]$ ./interrupts
Parent pid=2173
Child pid =0
Child: sending parent SIGUSR1 ..........received a SIGUSR1 signal 10
Child: sending parent SIGUSR1 ..........received a SIGUSR1 signal 10
Child: sending parent SIGUSR1 ..........received a SIGUSR1 signal 10
Child: sending parent SIGUSR2 ..........received a SIGUSR2 signal 12
Child: sending parent SIGUSR1 ..........received a SIGUSR1 signal 10
Child: sending parent SIGUSR2 ..........received a SIGUSR2 signal 12
^C
ending program ....
[gigowskm@eos04 lab3]$
```