CIS 452-10 Lab 3

Thomas Bailey
Parker Skarzynski

January 2019

1. The program prints, in order:

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

2. (a) Main runs, which calls signal(). Signal() chooses how the receipt of a signal will be handled. In our case, signal dictates that the function sigHandler(int sigNum) will handle a signal. The main function printfs "waiting...".

   (b) When we press ctrl-c, a signal interrupt is sent to the program, which is received by the signal() call. Then, sigHandler is called with our signum as a parameter. sigHandler() prints out "received an interrupt.". Then, sigHandler sleeps for 1 second, printfs "outta here.", and exits with status 0.

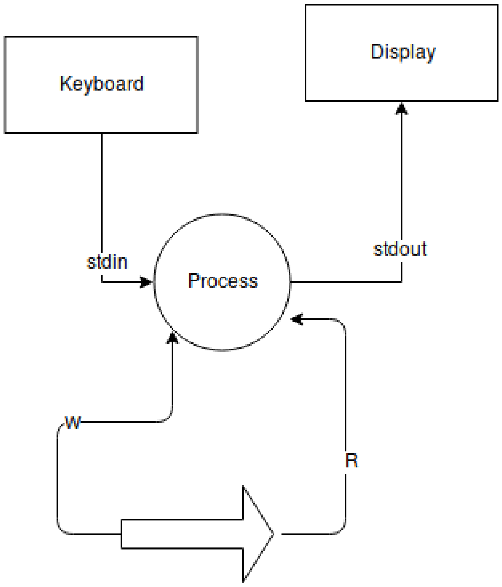
3. The standard output of the child process will go to the file named temp, because dup2() was called before the fork, meaning the child will have the same file descriptor table.

4. The standard output of the child process will go to the console, because fork() was called before dup2() and the child's file descriptor table is the same as the parent before dup2() was called.

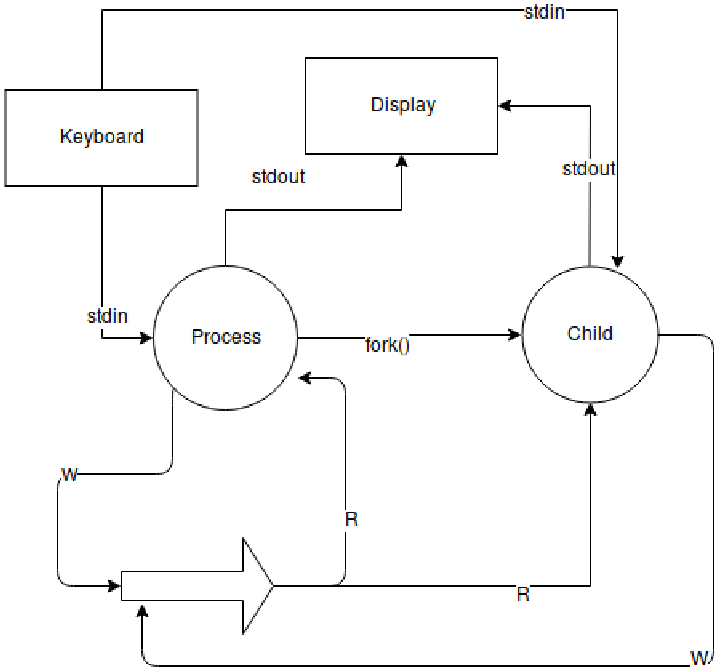
5. The process starts as a parent and forks() to create a child process. The child starts after the fork(), enters the child block (!pid) and calls dup2(fd[WRITE], STDOUT_FILENO). This copies the file descriptor fd[WRITE] (the beginning of the pipe) to STDOUT_FILENO, the output designated for in the file descriptor table for the child. Now, output from the child will go the the pipe. Now, the child closes both fd[READ] and fd[WRITE] which has no effect, because STDOUT_FILENO is already a reference to the beginning of the pipe. Fgets takes input from the keyboard, and then it is written to STDOUT_FILENO (the pipe). The child exits. After the child block, the parent executes dup2(fd[READ], STDIN_FILENO), which copies fd[READ] into STDIN_FILENO. Then, fd[READ] and fd[WRITE]

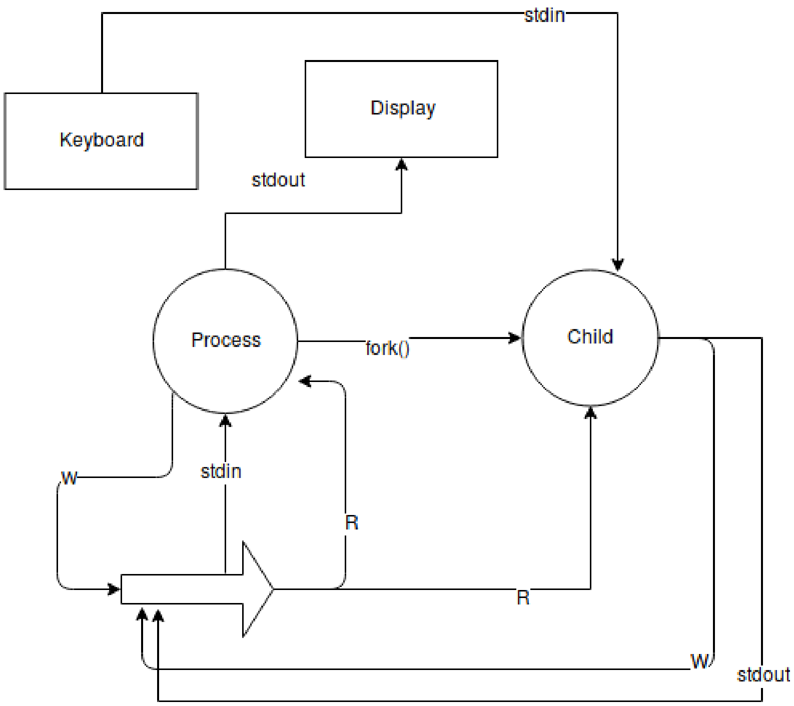
are closed. This doesn't matter because STDIN_FILENO is already the end of the pipe. When we read from STDIN_FILENO on the next line, we read out of the pipe, and when we puts(str) we print to STDOUT_FILENO (which is unmodified because we modified it in the child, not the parent). So it prints out on the console.

6. (a)

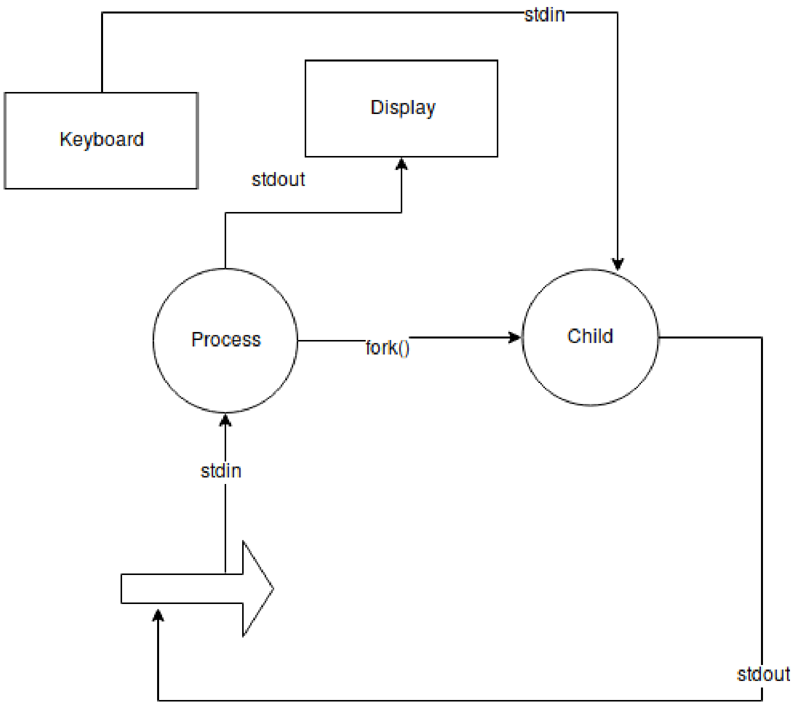


(b)




stdin
Keyboard
Display
stdout
Process
fork()
Child
W
stdin
R
R
W
stdout

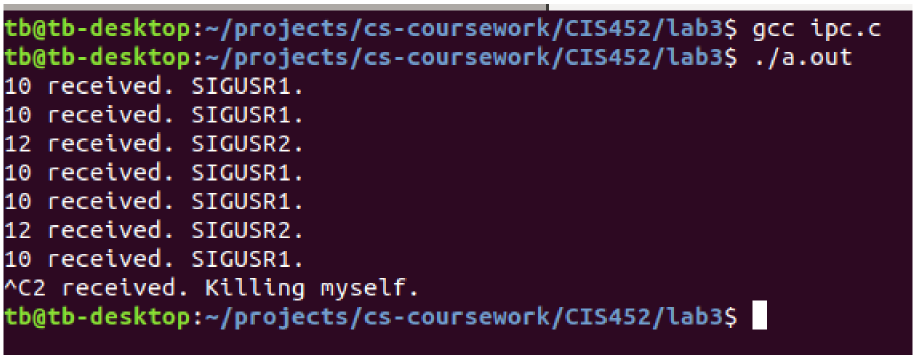
(c)


stdin
Keyboard
Display
stdout
Process
fork()
Child
stdin
stdout

(d)

7.

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>
#include <time.h>
void handleSig(int);
int main() {
  pid_t pid;
  // Spawn off a child process
  if ((pid = fork()) < 0) {
    perror("fork failed");
    exit(1);
  } else if (!pid) {
    fflush(stdout);
    // We're a child
    srand(time(NULL));
    int r;
    pid_t ppid = getppid();
    while(1) {
      r = (rand()%5)+1;
      fflush(stdout);
      sleep(r);
      if(r%2==0) {
  kill(ppid, SIGUSR1);
      } else {
  kill(ppid, SIGUSR2);
      }
    }
    exit(0);
  }
  // We're a parent
  fflush(stdout);
  while(1) {
    // Catch signals
    signal(SIGINT, handleSig);
    signal(SIGUSR1, handleSig);
    signal(SIGUSR2, handleSig);
    pause();
  }
  return 0;
}
void handleSig(int sig) {
  printf("%d received. ", sig);
  if (sig == SIGINT) {
    printf("Killing myself.\n");
    exit(0);
  } else if (sig == SIGUSR1) {
    printf("SIGUSR1.\n");
  } else if (sig == SIGUSR2) {
    printf("SIGUSR2.\n");
  } else {
    printf("Something is wrong.\n");
  }
  fflush(stdout);
}
```



```
tb@tb-desktop:~/projects/cs-coursework/CIS452/lab3$ gcc ipc.c
tb@tb-desktop:~/projects/cs-coursework/CIS452/lab3$ ./a.out
10 received. SIGUSR1.
10 received. SIGUSR1.
12 received. SIGUSR2.
10 received. SIGUSR1.
10 received. SIGUSR1.
12 received. SIGUSR2.
10 received. SIGUSR1.
^C2 received. Killing myself.
tb@tb-desktop:~/projects/cs-coursework/CIS452/lab3$ 
```