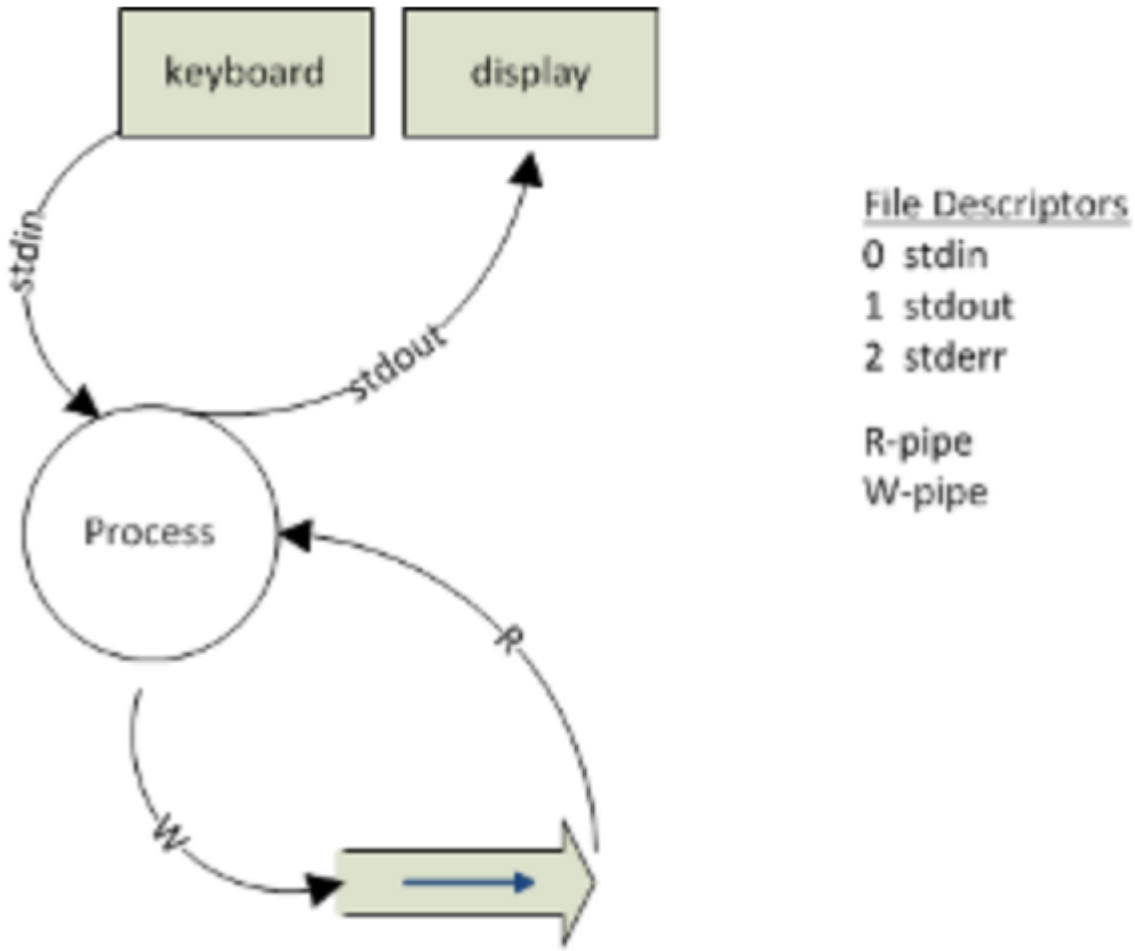
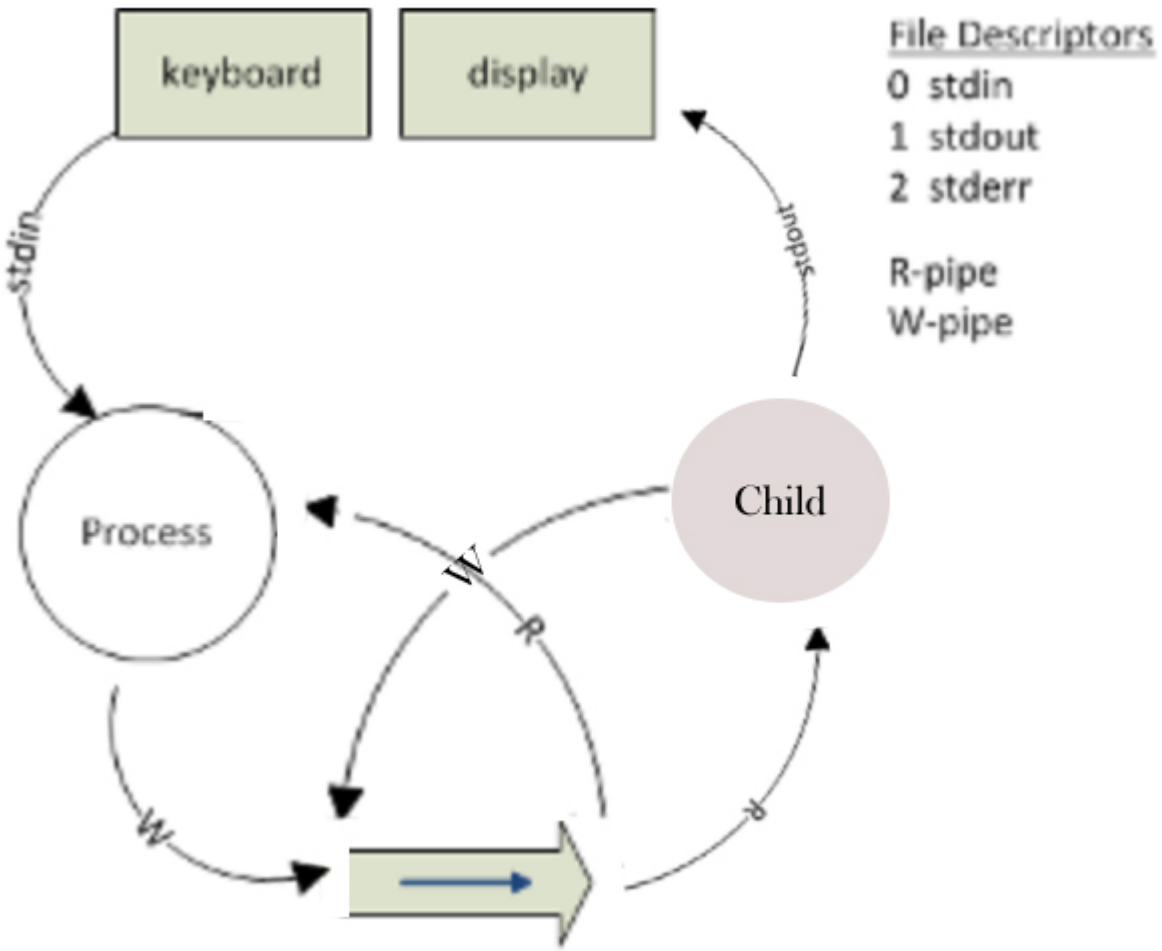
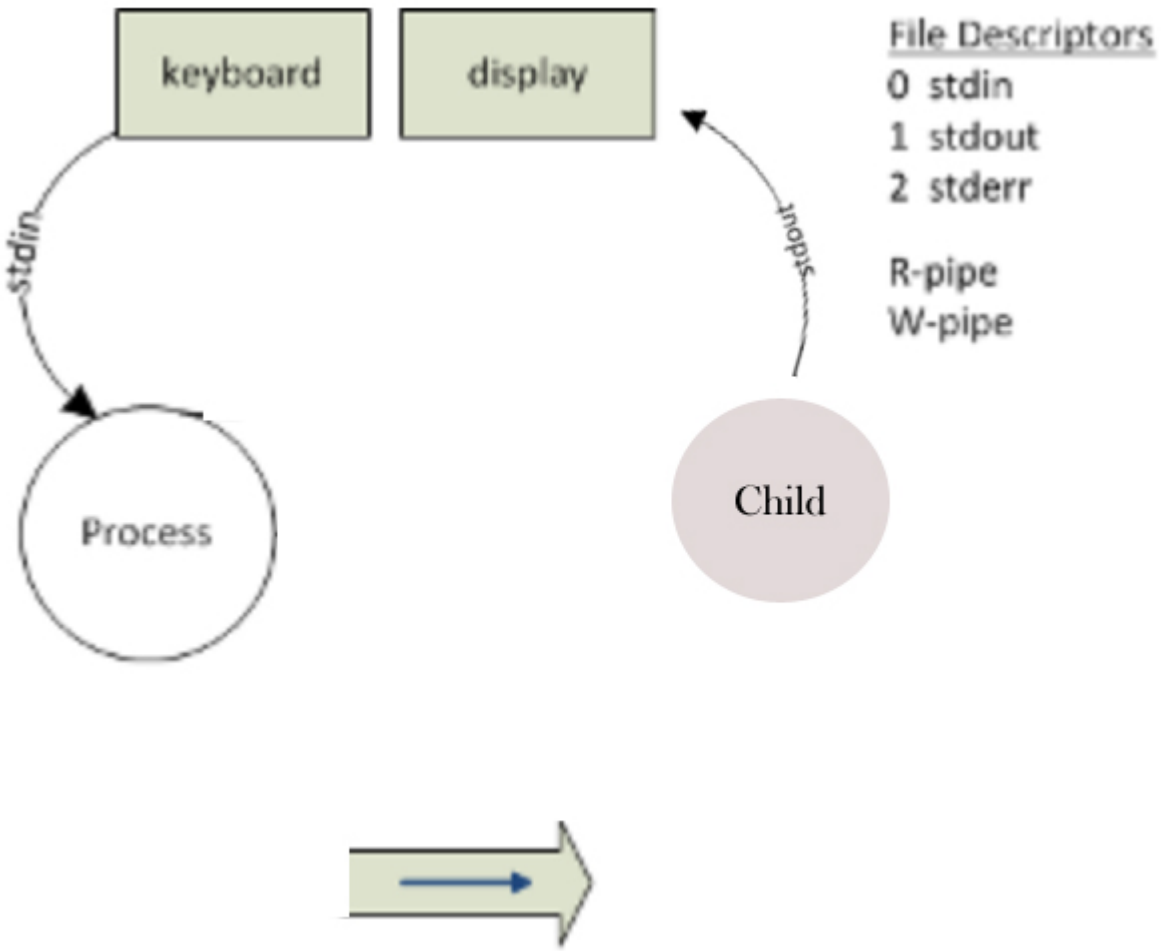
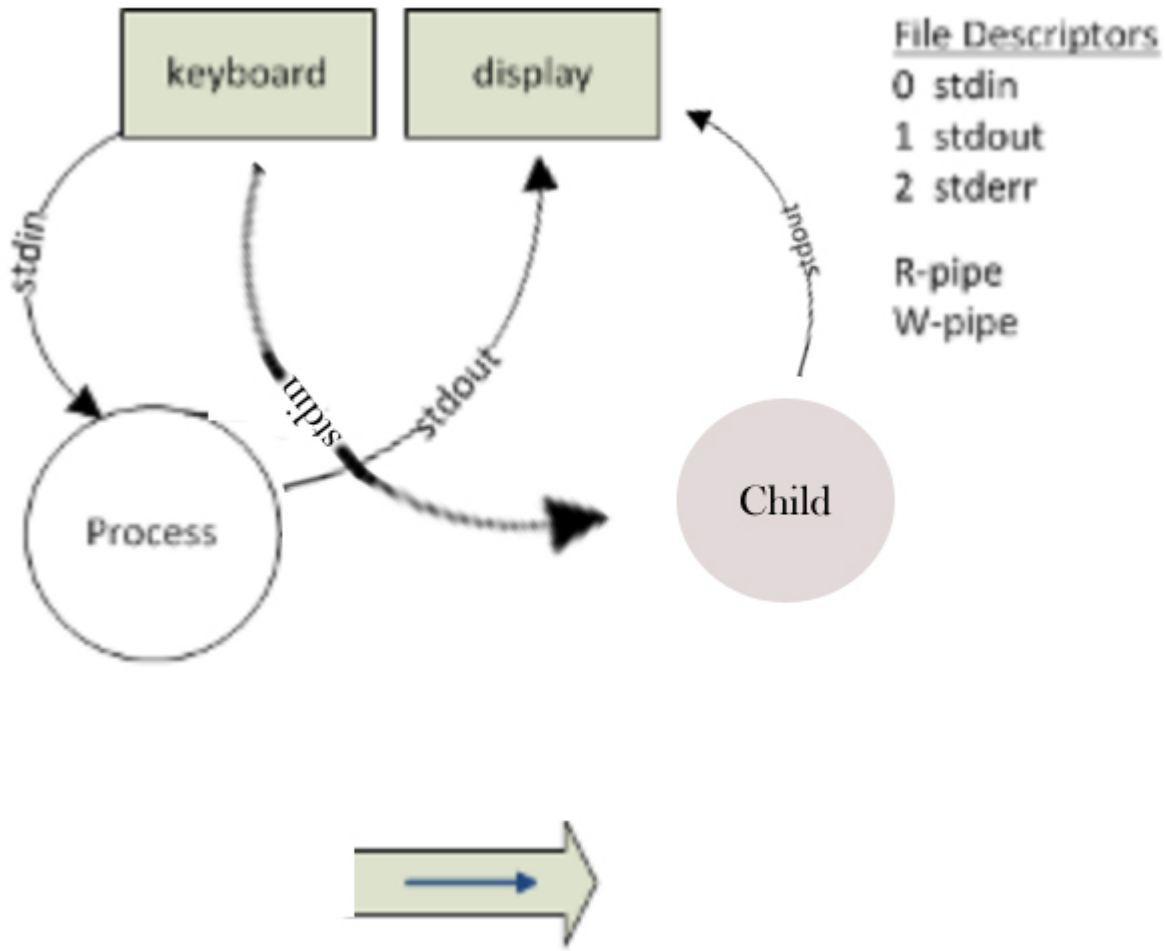
Lab 3

by Jarred Parr and Alexander Fountain

1. The order is waiting, received an interrupt, outta here
2. The program starts and prints that it is waiting, from there it pauses the main thread and puts a signal handler on the program. The program is now paused until it detects a system interrupt which, in this case, is handled via a `ctrl-c` on the keyboard. From there, it catches the interrupt and says that it was received. This leads to the final print statement of "outta here" which leads the system to exit the program gracefully.
3. The child process will output to `stdout`. This is because `dup2` was called before the child was initialized, so it will print to the default, which is `stdout`.
4. The child process will now output to `tmp`. This is because the dup2 remapped the output after the child was initialized and all of the subprocesses will print as a result of that as well.
5. The program starts and awaits input on `stdin`. It then sends it across the pipe to the waiting file descriptor to be read. After the data is read, it prints the data. Both of the pipe ends then close their descriptors and store the output to a `char*` variable which is then printed to `stdout` at the end. The parent process then prints an empty string because it did not catch the input from the `stdin` that the child received as the descriptor was not pointed at it properly.
6. 


keyboard
display
File Descriptors
0 stdin
1 stdout
2 stderr
R-pipe
W-pipe
stdin
stdout
Process
Child
W
R
W
R


keyboard
display
File Descriptors
0 stdin
1 stdout
2 stderr
R-pipe
W-pipe
stdin
stdout
Process
Child



Programming Assignment

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <string.h>
#include <signal.h>
#include <time.h>
#include <wait.h>

#define READ 0
#define WRITE 1
#define MAX 1024

void child_process();
void sig_handler(int signum);

void child_process(pid_t pid) {
  srand(time(NULL));
  // Our random 5 second wait time
  for (;;) {
    int random_time = rand() % 6;
```

```
    int choice = rand() % 2;
    printf("waiting...");

    sleep(random_time);
    if (choice == 1)
      kill(pid, SIGUSR1);
    else
      kill(pid, SIGUSR2);
  }
}

void sig_handler(int signum) {
  if (signum == SIGUSR1) {
    printf("Received a SIGUSR1 signal\n");
  } else if (signum == SIGUSR2) {
    printf("Received a SIGUSR2 signal\n");
  } else if (signum == SIGINT) {
    printf("Oh we got a wise guy here eh? Shutting it down!\n");
    exit(EXIT_SUCCESS);
  } else {
    printf("What do dat one do");
  }
}

int main(int argc, char** argv) {
  pid_t pid;

  struct sigaction sa;
  sa.sa_handler = sig_handler;
  sigemptyset(&sa.sa_mask);
  sa.sa_flags = SA_RESTART;

  sigaction(SIGINT, &sa, NULL);
  sigaction(SIGUSR1, &sa, NULL);
  sigaction(SIGUSR2, &sa, NULL);
  /** spawn child **/

  if ((pid = fork()) < 0) {
    perror("Fork machine broke");
    return EXIT_FAILURE;
  } else if (pid == 0) {
    child_process(pid);
  } else {
    printf("Spawned child pid# %d\n", pid);
    wait(&pid);
  }
}
```