UNIVERSIDADE FEDERAL DO MARANHÃO
Sistemas Operacionais II
prof. Antonio de Abreu Batista Júnior
Laboratório 1 - Pipes

Nome dos integrantes do grupo:

A pipe is a mechanism provided by the operating system that lets one process send a stream of bytes to another one. Since a child process inherits all open descriptors from the parent when it is created, we can create a pipe (which is a pair of connected descriptors) before creating the child process, thus allowing both processes access to the same pipe (see Figure). In order to create a pipe, you first need to create a *file descriptor*:

int fd[2];

The first integer in the array (element 0) is set up and opened for reading, while the second integer (element 1) is set up and opened for writing. Visually speaking, the output of **fd[1]** becomes the input for **fd[0]**. All data traveling through the pipe moves through the kernel. In case of a fork(), if the parent wants to receive data from the child, it should close **fd[1]**, and the child should close **fd[0]**. If the parent wants to send data to the child, it should close **fd[0]**, and the child should close **fd[1]**. Since descriptors are shared between the parent and child, we should always be sure to close the end of pipe we arent concerned with. On a technical note, the EOF will never be returned if the unnecessary ends of the pipe are not explicitly closed.

The following example should clarify the above description:

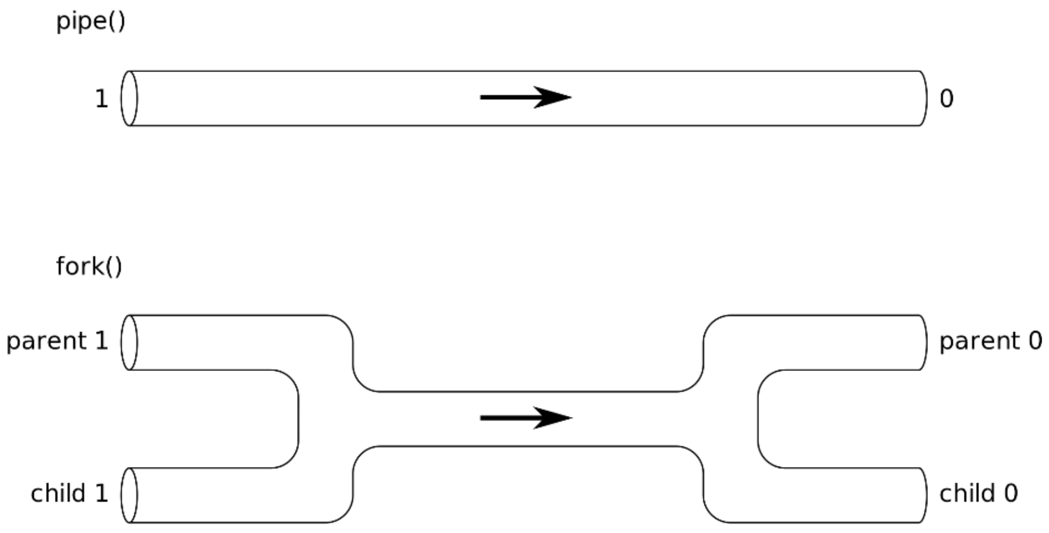

Figure 1: UNIX pipe

```
int main ( void ) {
   int fd [2] , nbytes ;
   pid_t childpid ;
   char string [] =" We love 3 SH3 !\ n ";
   char readbuffer [80];

   pipe ( fd ) ;
   if (( childpid = fork () ) = = −1) {
       perror (" fork ") ;
       exit (0) ;
   }
   if ( childpid = = 0) {
       close ( fd [0]) ;
       write ( fd [1] , string , ( strlen ( string ) +1) ) ;
       exit (0) ;
   }
   else {
       close ( fd [1]) ;
       nbytes = read ( fd [0] , readbuffer , sizeof (readbuffer ) ) ;
       printf (" Received string : % s " , readbuffer ) ;
   }
   return 0;
}
```

1. write a C program that:

- Creates a child and a parent process.
- The child process receives 1-byte integers from the keyboard one at a time. Upon getting each new input, it should send it to the parent process using a pipe until the input is '-1'.
- When the parent process receives '-1', it should (1) compute the sum of all the integers it has received, and (2) send the result to the child process using another pipe.
- When the child process receives the result, it terminates.