Lab 10

Anthony Thap

Brian Ackley

Excercise 0:

dup2(): makes newfd be the copy of oldfd but if old fd is a valid descriptor and newfd has the same value as oldfd, then it does nothing and just returns newfd. Also if the oldfd is not a valid file descriptor then it just fails.

read: one line is read from the standard input, and then the first word is assigned, second word is assigned to second name and so on.

Write: allows you to communicate with other users, by copying lines from your terminal to theirs.

Pipe: daemon processes requests from the Postfix queue manager to deliver messages to external commands.

Fprintf: writes output to the given output stream

atoi: converts the initial portion of the string pointed to by nptr to int.

Getpid: returns the process ID of the calling process.

Wait: Wait for each specified process and return its termination status. Each n may be a process ID or a job specification; if a job spec is given, all processes in that job's pipeline are waited for. If n is not given, all currently active child processes are waited for, and the return status is zero. If n specifies a non-existent process or job, the return status is 127. Otherwise, the return status is the exit status of the last process or job waited for.

Excercise 1:

```
int n = 98;
for ( int i = 0; i < 10; i++ ) {
write ( STDOUT_FILENO, &i, sizeof(i) );
read ( STDIN_FILENO, &n, sizeof (n) );
fprintf( stderr, "%d\n", n );
```

this code will count 10 times starting with 98. the output then would be:

98,99,100,101,102,103,104,105,106,107

excersise 2a:

```
int n = 98;
for ( int i = 0; i < 10; i++ ) {
read ( STDIN_FILENO, &n, sizeof (n) );
write ( STDOUT_FILENO, &i, sizeof(i) );
```

```
fprintf( stderr, "%d\n", n );
```

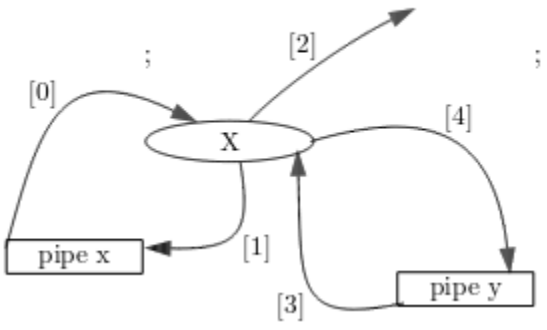
this code would do the same thing as the previous code but since the fact that each one is trying to reference the others directory it ends up not being able to process through and just hanging

excersise 2b:

```
int n = 98;
for ( int i = 0; i < 10; i++ ) {
printf("%d ", i );
scanf ( "%d", &n );
fprintf( stderr, "%d\n", n );
```

The program hangs on scanf because the pipe reading and writing are fully bu ered. The funciton printf does not write anything to the pipe until the bu er is full. Put a ush (std) statement after the printf to get output.

Excersise 3:



	Process X fd table
[0]	pipe x read
[1]	pipe x write
[2]	standard error
[3]	pipe y read
[4]	pipe y write

The connection to the parent process of Exercise 3 after the second pipe (fd) has been executed. The process (X) has created two pipes.

Excercise 4:

.1 - //ring.cpp

```
#include <unistd.h>

#include <string.h>

#include <errno.h>

#include <stdlib.h>

#include <iostream>

using namespace std;

int main( int argc, char *argv[] )

{

int fd[2];
```

```
int childpid; //process id of child

int nprocs; //total number of processes in the ring

int error; //return value from dup2 call

int i;

if ( argc < 2 || (nprocs = atoi( argv[1])) <= 0 ){

cout << "Usage: " << argv[0] << " nprocesses" << endl;

exit ( 1 );

}

if ( pipe ( fd ) == -1 ){

perror("Could not create pipe");

exit ( 1 );

}
```

```
if ( dup2 ( fd[0], STDIN_FILENO ) == -1 ||

dup2 ( fd[1], STDOUT_FILENO ) == -1 ) {

perror ("Could not dup pipes");

exit ( 1 );

}

close ( fd[0] );

close ( fd[1] );

for ( i = 1; i < nprocs; i++ ) {

5if ( pipe( fd ) == -1 ) {

fprintf(stderr, "Could not create pipe %d: %s\n",

i, strerror ( errno ) );

exit ( 1 );

}
```

```
if ( ( childpid = fork() ) == -1 ) {

fprintf(stderr, "Could not create child %d: %s\n",

i, strerror ( errno ) );

exit ( 1 );

}

if ( childpid > 0 ) //for parent, reassign stdout

error = dup2 ( fd[1], STDOUT_FILENO );

else //for child, reassign stdin

error = dup2 ( fd[0], STDIN_FILENO );

if ( error == -1 ) {

fprintf(stderr, "Could not dup pipes for iteration %d: %s\n",

i, strerror( errno ) );
```

```
exit ( 1 );

}

close ( fd[0] );

close ( fd[1] );

if ( childpid ) //parent done

break;

}

//say hello to the world

fprintf( stderr, "This is process %d with ID %d and parent ID is %d\n",

i, getpid(), getppid() );

return 0;
```

.2 - all: betacalcs

```
rings : main.o
        g++ main.o -o rings
```

```
main.o : main.cpp rings1.h rings2.h rings3.h
	g++ -c main.cpp
clean:
	rm rigns main.o
```

.3 - //ring2.cpp

```
#include <unistd.h>

#include <string.h>

#include <errno.h>

#include <stdlib.h>

#include <iostream>

using namespace std;

int main( int argc, char *argv[] )

{

int fd[2];
```

```
int childpid; //process id of child

int nprocs; //total number of processes in the ring

int error; //return value from dup2 call

int i;

if ( argc < 2 || (nprocs = atoi( argv[1])) <= 0 ){

cout << "Usage: " << argv[0] << " nprocesses" << endl;

exit ( 1 );

}

if ( pipe ( fd ) == -1 ){

perror("Could not create pipe");

exit ( 1 );

}

if ( dup2 ( fd[0], STDIN_FILENO ) == -1 ||
```

```
dup2 ( fd[1], STDOUT_FILENO ) == -1 ) {

perror ("Could not dup pipes");

exit ( 1 );

}

close ( fd[0] );

close ( fd[1] );

for ( i = 1; i < nprocs; i++ ) {

5if ( pipe( fd ) == -1 ) {

fprintf(stderr, "Could not create pipe %d: %s\n",

i, strerror ( errno ) );

exit ( 1 );

}
```

```
if ( ( childpid = fork() ) == -1 ) {

fprintf(stderr, "Could not create child %d: %s\n",

i, strerror ( errno ) );

exit ( 1 );

}

if ( childpid > 0 ) //for parent, reassign stdout

error = dup2 ( fd[1], STDOUT_FILENO );

else //for child, reassign stdin

error = dup2 ( fd[0], STDIN_FILENO );

if ( error == -1 ) {

fprintf(stderr, "Could not dup pipes for iteration %d: %s\n",

i, strerror( errno ) );

exit ( 1 );
```

```
}

close ( fd[0] );

close ( fd[1] );

if ( childpid ) //parent done

break;

}

//say hello to the world

fprintf( stderr, "This is process %d with ID %d and parent ID is %d\n",

i, getpid(), getppid() );

return 0;
```

.4 - //ring3.cpp

```
#include <unistd.h>

#include <string.h>
```

```
#include <errno.h>

#include <stdlib.h>

#include <iostream>

using namespace std;

int main( int argc, char *argv[] )

{

int fd[2];

int childpid; //process id of child

int nprocs; //total number of processes in the ring

int error; //return value from dup2 call

int i;

if ( argc < 2 || (nprocs = atoi( argv[1])) <= 0 ){
```

```
cout << "Usage: " << argv[0] << " nprocesses" << endl;

exit ( 1 );

}

if ( pipe ( fd ) == -1 ){

perror("Could not create pipe");

exit ( 1 );

}

if ( dup2 ( fd[0], STDIN_FILENO ) == -1 ||

dup2 ( fd[1], STDOUT_FILENO ) == -1 ) {

perror ("Could not dup pipes");

exit ( 1 );

}

close ( fd[0] );
```

```
close ( fd[1] );

for ( i = 1; i < nprocs; i++ ) {

5if ( pipe( fd ) == -1 ) {

fprintf(stderr, "Could not create pipe %d: %s\n",

i, strerror ( errno ) );

exit ( 1 );

}

if ( ( childpid = fork() ) == -1 ) {

fprintf(stderr, "Could not create child %d: %s\n",

i, strerror ( errno ) );

exit ( 1 );

}
```

```
if ( childpid > 0 ) //for parent, reassign stdout

error = dup2 ( fd[1], STDOUT_FILENO );

else //for child, reassign stdin

error = dup2 ( fd[0], STDIN_FILENO );

if ( error == -1 ) {

fprintf(stderr, "Could not dup pipes for iteration %d: %s\n",

i, strerror( errno ) );

exit ( 1 );

}

close ( fd[0] );

close ( fd[1] );

if ( childpid ) //parent done

break;
```

```
}

//say hello to the world

sprintt( stderr, "This is process %d with ID %d and parent ID is %d\n",

i, getpid(), getppid() );

return 0;
```