Test2aa

3. [**25**] Write a C program which will create a **total** of 35 processes all of which will have their own separate pipe, i.e. all processes will be able to write to all other processes but each process will only read from its own pipe. Each process will be assigned a unique index 0, 1, ..., 34. Each process **with an odd pid** will write it's index and pid to every other process (but not to itself). The processes with even pids will not write any messages. Then each process (odd and even) will read all the messages written to its pipe and print on stdout, for each message, a line similar to:
Process 35894 index 24 heard from process 35992 index 17
Be careful that your code doesn't hang!

Test2bb

3. [**25**] Write a C program which will create a **total** of 40 processes all of which will have their own separate pipe, i.e. all processes will be able to write to all other processes but each process will only read from its own pipe. Each process will be assigned a unique index 0, 1, ..., 39. Each process **with an odd pid** will write it's index and pid to process 0 (possibly including process 0). The processes with even pids will write their index and pid to all other processes (but not to themselves). Then each process (odd and even) will read all the messages written to its pipe and print on stdout, for each message, a line similar to:
Process 35894 index 24 heard from process 35992 index 17
Be careful that your code doesn't hang!

Test2cc

3. [**25**] Write a C `program` which will fork **num** (where num is a command-line argument) children. Every child (but not the original parent) will write his pid to every other child (but not parent). Each child will then read every message written to him, one at a time, and then write each received pid along with his pid to the original parent using the following message structure.

```
struct message {
  pid_t my_pid;
  pid_t received_pid;
}
```

On receipt of each message, the parent process will print to stdout a message similar to:

```
Process 1234 has received a message from Process 4567
```

Test2e

4. [**25**] Write a C program which will fork 40 children. Each child will send back to the parent process, via a single pipe, either one or two messages depending on whether the child's pid is odd or even. If the child's pid is even the child will send the single message "`one`" while a child with an odd pid will send the message "`one`" followed by a second message "`two`" The parent process must not terminate until all messages have been read.

Test2dd

3. **[25]** Write a C program which will create 23 children processes and two pipes. All processes should know about both pipes. The original parent process will write 1000 messages to the pipe each consisting of a single integer starting with 0 up to 999. Each of the 23 child processes will read as many messages (one at a time) from the pipe as possible, printing out on stdout a message containing the child's pid and the message number. The general form of the message is:
   `Child 12475 received message 134`
   Each child will then write to the parent a message consisting of the number of messages that that child received. The parent will add up the total number of messages that the children claim to have received and, if the total is 1000 will print the following message to stdout:

   `All messages are accounted for`

   and if the total is less that 1000 (for example say 945) will print:

   `945 messages are accounted for`

Test2ee same as Test2aa
Test2ff

3. **[25]** Write a C `program` which will fork 30 children assigning each process a unique index from 0 to 30. There should be a common pipe shared by all the processes. The processes with an even index will write the numbers 1 to 1000, each number together with the processes pid, to the pipe. The odd processes (those with an odd index) will read as many messages as they can, adding up the numbers read (not the pids) and when the pipe is empty each odd process will write its pid and sum to stdout in the following format:

   `Process 47836 total 14327`

Test2gg

3. **[25]** Write a C `program` which will fork 23 children assigning each process a unique index from 0 to 23. After creation the original process (process 0) will write his pid to each child process. Each child process (processes 1, ..., 23) will read the message from process 0 and those with an **odd pid** will write a message back to process 0 that consists of the received pid and the child's pid. Process 0 will read all the messages written to him and will print on stdout a string similar to `"process 23456 received message from 98765\n"`.

Test2hh same as Test2cc
Test2ii same as Test2aa

Test2jj

3. [**25**] Write a C program which will create a **total** of 33 processes sharing two pipes. Each process must be assigned a unique index 0, 1, . . . , 32. The processes will be divided into two groups based on their pid, those with an even pid will be in the Even group and those with an odd pid in the Odd group. Each process in the Even group will write it's index and pid to the pipe of the Odd group while each process in the Odd group will write their index and pid to the pipe of the Even group. All processes will then read as many messages as possible from their group's read pipe printing on stdout, for each message read, a line similar to:

   ```
   Process 35894 index 24 heard from process 35991 index 17
   ```

   Be careful that your code doesn't hang and messages don't get mixed up!

Test2kk.1 similar to Test2aa (only even pid writes this time)
Test2kk.2 same as Test2aa
Test2l

4. [**25**] Write a C program which will create a **total** of 35 processes all of which will have their own separate pipe, i.e. all processes will be able to write to all other processes but each process will only read from its own pipe. Each process will be assigned a unique index 0, 1, . . . , 34. Each process will write it's index and pid to every other process (but not to itself). Then each process will read all the messages written to its pipe and print on stdout, for each message, a line similar to:
   Process 35894 index 24 heard from process 35992 index 17
   Be careful that your code doesn't hang!

Test2ll same as Test2gg
Test2mm same as Test2cc
Test2nn same as Test2jj

Test2oo

3. [**25**] Write a C program which will create a **total** of 33 processes, each process having its own pipe (i.e. one that it will read from). Each process must be assigned a unique index 0, 1, . . . , 32. The original process will send an integer generated by **rand()** to processes, 1, 2, . . . , 32. Each child will read the number from its pipe and, if even, will write to the parent (process 0) the random number read, its pid and its index. The original parent (process 0) will read each message and print a message similar to

   ```
   Original Process 0 read random number 3157 from process 35991 index 17
   ```

   Be careful that your code doesn't hang and messages don't get mixed up!