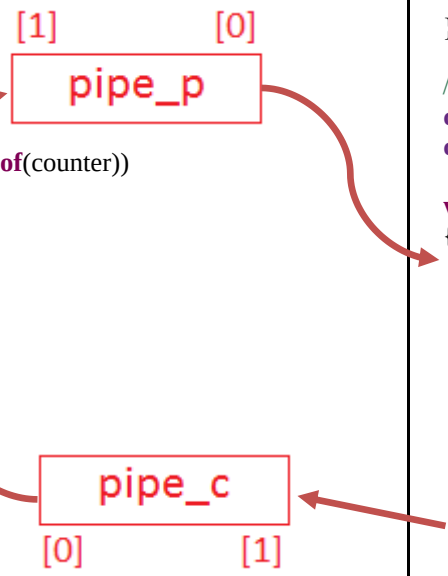
```
/*Copyright (C) ViGIR Vision-Guided and Intelligent Robotics Lab
* Written by Guilherme DeSouza, Luis Rivera
* This program is free software; you can redistribute it and/or modify  it under the terms of the GNU General Public License as published by  the Free Software Foundation, meaning:
* keep this copyright notice, do not try to make money out of it, it's distributed WITHOUT ANY WARRANTY, yada yada yada...*/
```

test_simple_pipes_1.c

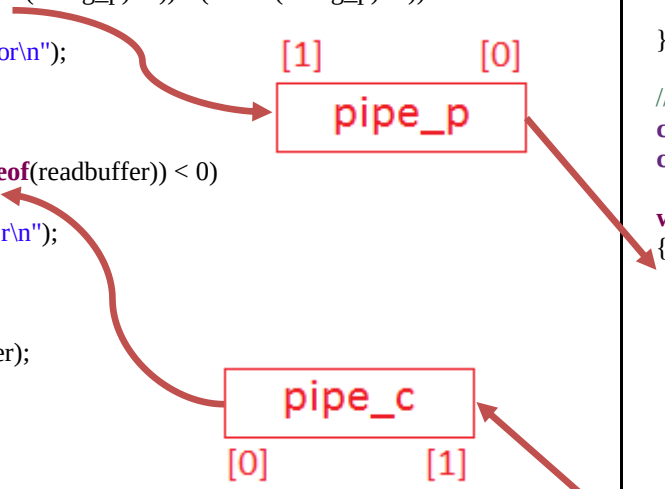
```
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>

// function called by the child process.
// Inputs: file descriptors associated to the pipes
void child(int pipe_p, int pipe_c)
{
  int counter = 0;

  while(counter < 100000)
  {
    sleep(1);
    counter++ ;
    printf("child:  %d \n", counter);

    if(write(pipe_p, &counter, sizeof(counter)) != sizeof(counter))
    {
      printf("child's pipe write error\n");
      exit(-1);
    }

    if(read(pipe_c, &counter, sizeof(counter)) < 0)
    {
      printf("child's pipe read error\n");
      exit(-1);
    }
  }
}
```




```
int main(void)
{
  int frkvar, counter = 0;
  int pipe_p[2], pipe_c[2];          // size has to be 2 (2 file descriptors)

  if(pipe(pipe_p) < 0)
  {
    printf("pipe creation error\n");
    exit(-1);
  }

  if(pipe(pipe_c) < 0)
  {
    printf("pipe creation error\n");
    exit(-1);
  }

  if((frkvar = fork()) < 0)
  {
    printf("fork error\n");
    exit(-1);
  }

  if(frkvar == 0)  // child process
  {
    close(pipe_p[0]);       // not needed, but it emphasizes that these sides
    close(pipe_c[1]);       // are not used by the child process in this example
    child(pipe_p[1], pipe_c[0]);
  }

  // parent process
  close(pipe_p[1]);         // not needed, but it emphasizes that these sides
  close(pipe_c[0]);         // are not used by the parent process in this example

  while(counter < 100000)
  {
    if(read(pipe_p[0], &counter, sizeof(counter)) < 0)
    {
      printf("parent's pipe read error\n");
      exit(-1);
    }

    counter++;
    sleep(1);

    if(write(pipe_c[1], &counter, sizeof(counter)) != sizeof(counter))
    {
      printf("parent's pipe write error\n");
      exit(-1);
    }

    printf("parent: %d \n", counter);
  }

  return 0;
}
```

test_simple_pipes_2.c

```c
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// function called by the child process.
// Inputs: file descriptors associated to the pipes
void child(int pipe_p, int pipe_c)
{
  char string_p[] = "Hello parent!";          // message to be sent to the parent
  char readbuffer[80];

  while(1)
  {
    sleep(1);

    if(write(pipe_p, string_p, (strlen(string_p)+1))!=(strlen(string_p)+1))
    {
      printf("child's pipe write error\n");
      exit(-1);
    }

    if(read(pipe_c, readbuffer, sizeof(readbuffer)) < 0)
    {
      printf("child's pipe read error\n");
      exit(-1);
    }

    printf("child: %s\n", readbuffer);
  }
}

int main(void)
{
  int frkvar;
  int pipe_p[2], pipe_c[2];                  // size has to be 2 (2 file descriptors)

  char string_c[] = "Hello child!";          // message to be sent to the child
  char readbuffer[80];

  if(pipe(pipe_p) < 0)
  {
    printf("pipe creation error\n");
    exit(-1);
  }

  if(pipe(pipe_c) < 0)
  {
    printf("pipe creation error\n");
    exit(-1);
  }

  if((frkvar = fork()) < 0)
  {
    printf("fork error\n");
    exit(-1);
  }

  if(frkvar == 0)  // child process
  {
    close(pipe_p[0]);        // not needed, but it emphasizes that these sides
    close(pipe_c[1]);        // are not used by the child process (in this example)
    child(pipe_p[1], pipe_c[0]);
  }

  // parent process
  close(pipe_p[1]);          // not needed, but it emphasizes that these sides
  close(pipe_c[0]);          // are not used by the parent process (in this example)

  while(1)
  {
    if(read(pipe_p[0], readbuffer, sizeof(readbuffer)) < 0)
    {
      printf("parent's pipe read error\n");
      exit(-1);
    }

    printf("parent: %s\n", readbuffer);

    sleep(1);

    if(write(pipe_c[1],string_c,(strlen(string_c)+1)) != (strlen(string_c)+1))
    {
      printf("parent's pipe write error\n");
      exit(-1);
    }
  }

  return 0;
}
```



test_named_pipes_A.c

```c
#include <unistd.h>  #include <sys/types.h>  #include <fcntl.h>  #include <stdio.h>  #include <stdlib.h>

int main(void)
{
  int counter, dummy;
  int pipe_BtoA, pipe_AtoB;            // for file descriptors

  dummy = system("mkfifo BtoA");       // could be done separately in each task,
  dummy = system("mkfifo AtoB");       // or in a terminal directly

  if((pipe_BtoA = open("BtoA", O_RDONLY)) < 0)  {
    printf("pipe BtoA error\n");
    exit(-1);
  }

  if((pipe_AtoB = open("AtoB", O_WRONLY)) < 0)  {
    printf("pipe AtoB error\n");
    exit(-1);
  }

  while(1)  {
    sleep(1);

    if(read(pipe_BtoA, &counter, sizeof(counter)) < 0)  {
      printf("BtoA pipe read error\n");
      exit(-1);
    }

    counter++;

    if(write(pipe_AtoB, &counter, sizeof(counter)) != sizeof(counter))  {
      printf("AtoB pipe write error\n");
      exit(-1);
    }

    printf("Count TaskA: %d \n", counter);
  }

  return dummy;
}
```

test_named_pipes_B.c

```c
#include <unistd.h>  #include <sys/types.h>  #include <fcntl.h>  #include <stdio.h>  #include <stdlib.h>

int main(void)
{
  int counter = 0;
  int pipe_BtoA, pipe_AtoB;            // for file descriptors

  if((pipe_BtoA = open("BtoA", O_WRONLY)) < 0)  {
    printf("pipe BtoA error\n");
    exit(-1);
  }

  if((pipe_AtoB = open("AtoB", O_RDONLY)) < 0)  {
    printf("pipe AtoB error\n");
    exit(-1);
  }

  while(1)  {
    sleep(1);

    if(write(pipe_BtoA, &counter, sizeof(counter)) != sizeof(counter))  {
      printf("BtoA pipe write error\n");
      exit(-1);
    }

    if(read(pipe_AtoB, &counter, sizeof(counter)) < 0)  {
      printf("AtoB pipe read error\n");
      exit(-1);
    }

    counter++;
  }

  return 0;
}
```