```c
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <unistd.h>

int main(int argc, char* argv[]) {
    int PC[2], CP[2];
    printf("%d\n", atoi(argv[1]));
    if (pipe(PC) < 0) { perror("Error on pipe"); exit(1); }

    if (pipe(CP) < 0) {
                perror("Error on pipe");
                exit(1);
        }

    int f = fork();
    if (f<0) {
        perror("Error on fork");
        exit(2);
    }
    if (f == 0) { //child
        close(PC[1]);
        close(CP[0]);
        int x = atoi(argv[1]);
        while (x > 0) {
            int n = rand()%10 + 1;
            x -= n;
            write(CP[1], &x, sizeof(int));
            printf("%d %s\n", x, "child");
            if (read(PC[0], &x, sizeof(int)) < 0) {
                                exit(0);
                    }
        }
        exit(0);
    }
    if (f > 0) { //parent
        int x;
        read(CP[0], &x, sizeof(int));
        close(PC[0]);
        close(CP[1]);
        while (x > 0) {
            int n = rand()%10 + 1;
            x -= n;
            write(PC[1], &x, sizeof(int));
            printf("%d %s\n", x, "parent");
            if (read(CP[0], &x, sizeof(int)) < 0 ) {

                                exit(0);
                    }
        }

        exit(0);
    }
    return 0;
}
```

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

int main(int argc, char* argv[])
{
    int p2c[2];

    int res = 0;

    // create pipe
    pipe(p2c);

    // create child process
    res = fork();
    switch(res)
    {
        case -1:
            perror("fork failed!");
            exit(0);
        case 0:
            // child process

            // read words from pipe
            for(int i=1; i< argc - 1; i++) {
         read(p2c[0], &argv[i], sizeof(char));
             printf("Word is: %s\n", argv[i]);}

            // close pipes
            close(p2c[0]);
            close(p2c[1]);

            // child is done
            exit(0);
        default:
            // parent process

            // send chars to child
         for(int i=1; i<argc - 1 ; i++) {
             write(p2c[1], &argv[i], sizeof(char));
         }

            // wait for child process to terminate
            wait(0);

            // close pipes
            close(p2c[0]);
            close(p2c[1]);
    }

    return 0;
}
```

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

int main() {
    int p, i;

    p = fork();

    if (p == -1) {
        perror("Fork failed");
    }

    if (p == 0) {
        for (i=0; i<10; i++)
            printf("Child: i=%d pid=%d ppid=%d\n", i, getpid(), getppid());
        exit(0);
    }
    else {
        for (i=0; i<10; i++)
            printf("Parent: i=%d pid=%d ppid=%d\n", i, getpid(), getppid());
        wait(0);
    }

    printf("Finished; pid=%d ppid=%d\n", getpid(), getppid());

    return 0;
}
```

```
//
// ex_pipe.c
//
// Sa se implementeze un proces care creeaza un proces copil cu care comunica
// prin pipe. Procesul parinte trimite prin pipe procesului copil doua numere
// intregi, iar procesul copil returneaza prin pipe suma lor.

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>


int main(int argc, char* argv[])
{
  int c2p[2];                     // child to parent file descriptors
  int res = pipe(c2p);
  if (res == -1)                  // fail to create pipe
  {
    perror("pipe(c2p) error: ");
    exit(EXIT_FAILURE);
  }

  int p2c[2];                     // parent to child file descriptors
  res = pipe(p2c);
  if (res == -1)                  // fail to create pipe
  {
    perror("pipe(p2c) error: ");
    exit(EXIT_FAILURE);
  }

  int pid = fork();
  if (pid == -1)                  // fail to create child
  {
    perror("fork() error: ");
    exit(EXIT_FAILURE);
  }

  // in the child process
  if (pid == 0)
  {
    // close the unused file descriptors
    close(c2p[0]);
    close(p2c[1]);

    while(1)
    {
      // read first integer
      int a;
      read(p2c[0], &a, sizeof(int));
      printf("\t[In CHILD] a: %d\n", a);

      // stop
      if (a == 0)
        break;

      // read the second integer
      int b;
      read(p2c[0], &b, sizeof(int));
      printf("\t[In CHILD] b: %d\n", b);

      // send the sum to parent
      int sum = a + b;
```

```
      write(c2p[1], &sum, sizeof(int));

      printf("\t[In CHILD] Sum: %d\n", sum);
    }

    // close the file descriptors
    close(c2p[1]);
    close(p2c[0]);

    exit(EXIT_SUCCESS);
  }

  // close the unused file descriptors
  close(c2p[1]);
  close(p2c[0]);

  while(1)
  {
    // read first integer
    int a;
    printf("[In PARENT] a: ");
    scanf("%d", &a);

    // send to the child
    write(p2c[1], &a, sizeof(int));

    if (a == 0)
      break;

    sleep(2);

    // read the second integer
    int b;
    printf("[In PARENT] b: ");
    scanf("%d", &b);

    // send to the child
    write(p2c[1], &b, sizeof(int));

    // read the sum from child
    int sum = 0;
    read(c2p[0], &sum, sizeof(int));

    printf("[In PARENT] Sum: %d\n", sum);
  }

  // wait for child
  int status;
  wait(&status);
  printf("\n[In PARENT] Child has finished with exit status: %d\n", status);

  // close the file descriptors
  close(c2p[0]);
  close(p2c[1]);

  return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char* argv[])
{
    if (argc < 2)
    {
        printf("no file name given\n");
        return 1;
    }

    FILE *file = fopen(argv[1], "r");
    fseek(file, 0, SEEK_END);
    int size = ftell(file);
    rewind(file);

    char* buffer = malloc(size);
    if (buffer == NULL)
    {
        printf("can't malloc(%d)\n", size);
        return 2;
    }

    if (fread(buffer, 1, size, file) != size)
    {
        printf("can't fread() %d bytes\n", size);
        return 3;
    }

    int p2c[2];
    int c2p[2];

    pipe(p2c);
    pipe(c2p);

    if (fork())
    {
        // parent

        // write the buffer
        int s = 0;
        while (s < size)
        {
            int x = write(p2c[1], (char*)((size_t)buffer + s), size-s);
            printf("parent written %d bytes\n", x);
            s += x;
        }

        // read the buffer and show on screen
        s = 0;
        while (s < size)
        {
            int x = read(c2p[0], (char*)((size_t)buffer + s), size-s);
            printf("parent read %d bytes\n", x);
            s += x;
        }

        for (s=0; s<size; s++)
        {
            printf("%c", buffer[s]);
        }
```

```
        // cleanup

        free(buffer);
        fclose(file);
    }
    else
    {
        // child
        int s = 0;

        // alloc a new buffer for child
        char* buff = malloc(size);
        if (buff == NULL)
        {
            printf("child can't malloc(%d)\n", size);
            exit(4);
        }

        // read the buffer from parent
        s = 0;
        while (s < size)
        {
            int x = read(p2c[0], (char*)((size_t)buff + s), size-s);
            printf("parent read %d bytes\n", x);
            s += x;
        }

        // transform to capital case
        for (s=0; s<size; s++)
        {
            if ((buff[s] >= 'a') && (buff[s] <= 'z'))
            {
                buff[s] -= 'a' - 'A';
            }
        }

        // write the buffer to parent
        s = 0;
        while (s < size)
        {
            int x = write(c2p[1], (char*)((size_t)buff + s), size-s);
            printf("parent written %d bytes\n", x);
            s += x;
        }


        // cleanup
        free(buff);
    }

    return 0;
}
```