PIPES:

1.Read numbers and display all divizors

#include <stdlib.h>

#include <unistd.h>

#include <stdio.h>

#include <time.h>

#include <sys/types.h>

#include <sys/wait.h>

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

int p2c[2], c2p[2];

if (0 > pipe(p2c)) {

perror("Error on pipe p2c");

exit(1);

}

if (0 > pipe(c2p)) {

perror("Error on pipe c2p");

exit(1);

}

int f = fork();

if (0 > f) {

perror("Error on fork");

exit(1);

} else if (0 == f) {

close(p2c[1]);

close(c2p[0]);

int n, div, count = 0;

while(1) {

if (0 > read(p2c[0], &n, sizeof(int))) {

perror("Error on child read");

}

if(n == -1)

break;

count = 0;

for (div = 1; div <= n; div++) {

if (n % div == 0) {

count++;

}

}

if (0 > write(c2p[1], &count, sizeof(int))) {

perror("Error on child write");

}

for (div = 1; div <= n; div++) {

if (n % div == 0) {

if (0 > write(c2p[1], &div, sizeof(int))) {

perror("Error on child write");

}

}

}

}

close(p2c[0]);

close(c2p[1]);

exit(0);

} else {

close(p2c[0]);

close(c2p[1]);

int n, i, size, div;

scanf("%d", &n);

printf("Number read %d\n", n);

while(n != 0) {

printf("Parent sending %d\n", n);

if (0 > write(p2c[1], &n, sizeof(int))) {

perror("Error on parent write");

}

if (0 > read(c2p[0], &size, sizeof(int))) {

perror("Error on parent read");

}

printf("Divizors of %d \n", n);

for (i = 0; i < size; i++) {

if (0 > read(c2p[0], &div, sizeof(int))) {

perror("Error on parent read");

}

printf("Divizor: %d ", div);

}

printf("\n");

scanf("%d", &n);

}

if(n == 0) {

int stop = -1;

if (0 > write(p2c[1], &stop, sizeof(int))) {

perror("Error on parent write");

}

}

wait(0);

close(p2c[1]);

close(c2p[0]);

}

return 0;

}

2.Generate random numbers

#include <stdlib.h>

#include <unistd.h>

#include <stdio.h>

#include <time.h>

#include <sys/types.h>

#include <sys/wait.h>

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

int p2c[2], c2p[2];

if (0 > pipe(p2c)) {

perror("Error on pipe p2c");

exit(1);

}

if (0 > pipe(c2p)) {

perror("Error on pipe c2p");

exit(1);

}

int f = fork();

if (0 > f) {

perror("Error on fork");

exit(1);

} else if (0 == f) {

close(p2c[1]);

close(c2p[0]);

srandom(time(0));

int num = random() % 900 + 101;

printf("Child generated %d\n", num);

int n,i=0;

while(1) {

if (0 > read(p2c[0], &n, sizeof(int))) {

perror("Error on child read");

}

printf("Child received %d ", n);

n-=num;i+=1;

printf("diff: %d\n", n);

if(abs(n) < 50)

{printf("i=%d\n",i);

if (0 > write(c2p[1], &i, sizeof(int))) {

perror("Error on child write");}

break;}

if (0 > write(c2p[1], &n, sizeof(int))) {

perror("Error on child write");

}

}

//if (0 > write(c2p[1], &i, sizeof(int))) {

// perror("Error on child write");

//}

close(p2c[0]);

close(c2p[1]);

exit(0);

} else {

close(p2c[0]);

close(c2p[1]);

srandom(time(0));

sleep(1);

int n = random() % 1000 + 51;int i=0;

// printf("Parent generated %d\n", n);

while(abs(n) > 50) {

// printf("Parent sending %d\n", n);

// i+=1;

if (0 > write(p2c[1], &n, sizeof(int))) {

perror("Error on parent write");

}

i++;

if (0 > read(c2p[0], &n, sizeof(int))) {

perror("Error on parent read");

}

//printf("Parent received %d\n", n);

if(abs(n)<50){

if (0 > read(c2p[0], &i, sizeof(int))) {

perror("Error on parent read");

}

printf("A generated %d numbers\n",i);

}

srandom(time(0));

n = random() % 1000 + 51;

sleep(1);

//printf("Parent generated %d\n", n);

}

wait(0);

close(p2c[1]);

close(c2p[0]);

}

return 0;

}

3.Read numbers and print all prime numbers less that the read number

#include <stdlib.h>

#include <unistd.h>

#include <stdio.h>

#include <time.h>

#include <sys/types.h>

#include <sys/wait.h>

int isPrime(int n) {

if (n < 2)

return 1;

if (n == 2)

return 0;

if (n % 2 == 0)

return 1;

int x;

for (x = 2; x <= n/2; x++)

if (n % x == 0)

return 1;

return 0;

}

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

int p2c[2], c2p[2];

if (0 > pipe(p2c)) {

perror("Error on pipe p2c");

exit(1);

}

if (0 > pipe(c2p)) {

perror("Error on pipe c2p");

exit(1);

}

int f = fork();

if (0 > f) {

perror("Error on fork");

exit(1);

} else if (0 == f) {

close(p2c[1]);

close(c2p[0]);

int n, div, count = 0;

while(1) {

if (0 > read(p2c[0], &n, sizeof(int))) {

perror("Error on child read");

}

if(n == -1)

break;

count = 0;

for (div = 0; div < n; div++) {

if (isPrime(div) == 0) {

count++;

}

}

if (0 > write(c2p[1], &count, sizeof(int))) {

perror("Error on child write");

}

for (div = 0; div < n; div++) {

if (isPrime(div) == 0) {

if (0 > write(c2p[1], &div, sizeof(int))) {

perror("Error on child write");

}

}

}

}

close(p2c[0]);

close(c2p[1]);

exit(0);

} else {

close(p2c[0]);

close(c2p[1]);

int n, i, size, div;

scanf("%d", &n);

printf("Number read %d\n", n);

while(n != 0) {

printf("Parent sending %d\n", n);

if (0 > write(p2c[1], &n, sizeof(int))) {

perror("Error on parent write");

}

if (0 > read(c2p[0], &size, sizeof(int))) {

perror("Error on parent read");

}

printf("Prime numbers less than %d \n", n);

for (i = 0; i < size; i++) {

if (0 > read(c2p[0], &div, sizeof(int))) {

perror("Error on parent read");

}

printf(" %d ", div);

}

printf("\n");

scanf("%d", &n);

}

if(n == 0) {

int stop = -1;

if (0 > write(p2c[1], &stop, sizeof(int))) {

perror("Error on parent write");

}

}

wait(0);

close(p2c[1]);

close(c2p[0]);

}

return 0;

}

FIFOS:

Header:

#define SIZE 200

char \*gog\_fifo = "./gog\_fifo";

Program A:

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

#include <unistd.h>

#include <errno.h>

#include <signal.h>

#include "common.h"

void makeFifoIfNotExist(char\* fifo\_name) {

if (mkfifo(fifo\_name, 0600) < 0) {

perror("Error on create fifo");

exit(1);

}

}

void writeToFifo(int my\_fifo, char\* buf) {

int length = strlen(buf);

if (write(my\_fifo, &length, sizeof(int)) < 0) {

perror("Error on write size to fifo");

}

if (write(my\_fifo, buf, length \* sizeof(char)) < 0) {

perror("Error on write string to fifo");

}

}

void handler(int sig) {

printf("\nSIGINT received; cleaning up\n");

if (unlink(gog\_fifo) < 0) {

perror("Unlink failed");

exit(1);

}

exit(0);

}

int main(int argc, char\*\* argv) {

int my\_fifo, k;

FILE\* f;

char\* cmd = (char\*)malloc(SIZE \* sizeof(char));

memset(cmd, 0, SIZE \* sizeof(char));

char\* buf = (char\*)malloc(SIZE \* sizeof(char));

memset(buf, 0, SIZE \* sizeof(char));

signal(SIGINT, handler);

makeFifoIfNotExist(gog\_fifo);

if ((my\_fifo = open(gog\_fifo, O\_WRONLY)) < 0) {

perror("Error on opening fifo");

exit(1);

}

while(1) {

if ((k = read(0, cmd, SIZE \* sizeof(char))) < 0) {

perror("Error of reading command");

}

cmd[k-1] = 0;

if (strcmp(cmd, "stop") == 0) {

printf("Done!\n");

break;

}

if ((f = popen(cmd, "r")) == NULL) {

perror("Error on popen");

}

else {

while (fread(buf, 1, SIZE-1, f) > 0) {

writeToFifo(my\_fifo, buf);

memset(buf, 0, SIZE \* sizeof(char));

}

}

pclose(f);

memset(cmd, 0, SIZE \* sizeof(char));

}

int stop = -1;

if (write(my\_fifo, &stop, sizeof(int)) < 0) {

perror("Eroor on writing to fifo");

}

close(my\_fifo);

free(cmd);

free(buf);

if (unlink(gog\_fifo) < 0) {

perror("Unlink failed");

exit(1);

}

return 0;

}

Program B:

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

#include <unistd.h>

#include <errno.h>

#include "common.h"

int main(int argc, char\*\* argv) {

int my\_fifo, dim, i, k;

char\* buf = (char\*)malloc(SIZE \* sizeof(char));

memset(buf, 0, SIZE \* sizeof(char));

if ((my\_fifo = open(gog\_fifo, O\_RDONLY)) < 0) {

perror("Error on opening fifo");

exit(1);

}

while(1) {

if (read(my\_fifo, &dim, sizeof(int)) < 0) {

perror("Error on reading size from fifo");

}

if (dim < 0) {

break;

}

i = 0;

while (i < dim) {

int size = (dim - i > SIZE - 1) ? SIZE - 1 : dim - i;

if ((k = read(my\_fifo, buf, size)) < 0 ) {

perror("Error on read message from fifo");

}

else {

i += k;

printf("%s", buf);

memset(buf, 0, SIZE \* sizeof(char));

}

}

fflush(stdout);

}

free(buf);

close(my\_fifo);

return 0;

}

#include <stdlib.h>

#include <unistd.h>

#include <stdio.h>

#include <time.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <errno.h>

#include <signal.h>

#include <string.h>

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

int p2c[2], c2p[2];

if (0 > pipe(p2c)) {

perror("Error on pipe p2c");

exit(1);

}

if (0 > pipe(c2p)) {

perror("Error on pipe c2p");

exit(1);

}

int f = fork();

if (0 > f) {

perror("Error on fork");

exit(1);

} else if (0 == f) {

close(p2c[1]);

close(c2p[0]);

int i, j, n, count;

int\* sir = (int\*)malloc(100 \* sizeof(int));

memset(sir, 0, 100 \* sizeof(int));

while(1) {

if (0 > read(p2c[0], &n, sizeof(int))) {

perror("Error on child read");

}

if(n == 0)

break;

count = 0;

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

if (n % i == 0) {

count++;

}

}

if (0 > write(c2p[1], &count, sizeof(int))) {

perror("Error on child write");

}

i = 0;

for (j = 1; j <= n; j++) {

if (n % j == 0) {

sir[i] = j;

i += 1;

}

}

if (0 > write(c2p[1], sir, count \* sizeof(int))) {

perror("Error on child write");

}

}

free(sir);

close(p2c[0]);

close(c2p[1]);

exit(0);

} else {

close(p2c[0]);

close(c2p[1]);

int n, i, size;

int\* sir = (int\*)malloc(100 \* sizeof(int));

memset(sir, 0, 100 \* sizeof(int));

scanf("%d", &n);

printf("Number read %d\n", n);

while(n != 0) {

printf("Parent sending %d\n", n);

if (0 > write(p2c[1], &n, sizeof(int))) {

perror("Error on parent write");

}

if (0 > read(c2p[0], &size, sizeof(int))) {

perror("Error on parent read");

}

printf("Divizors of %d \n", n);

if (0 > read(c2p[0], sir, size \* sizeof(int))) {

perror("Error on parent read");

}

for (i = 0; i < size; i++) {

printf("Divizor: %d ", sir[i]);

}

printf("\n");

scanf("%d", &n);

}

free(sir);

close(p2c[1]);

close(c2p[0]);

wait(0);

}

return 0;

}

PIPES MODEL:

#include <stdlib.h>

#include <unistd.h>

#include <stdio.h>

#include <time.h>

#include <sys/types.h>

#include <sys/wait.h>

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

int p2c[2], c2p[2];

if (0 > pipe(p2c)) {

perror("Error on pipe p2c");

exit(1);

}

if (0 > pipe(c2p)) {

perror("Error on pipe c2p");

exit(1);

}

int f = fork();

if (0 > f) {

perror("Error on fork");

exit(1);

}

else if (0 == f) {

/\*close the pipes that you don't need\*/

close(p2c[1]);

close(c2p[0]);

/\* insert variables\*/

while(1) {

if (0 > read(p2c[0], &n, sizeof(int))) {

perror("Error on child read");

}

/\* insert condition to break\*/

/\* if you want to send an array of bytes first count the numbers, send the counter and then send each number individually\*/

if (0 > write(c2p[1], &count, sizeof(int))) {

perror("Error on child write size");

}

/\* here you have the loop and the consition you are required to respect\*/

if (0 > write(c2p[1], &div, sizeof(int))) {

perror("Error on child write");

}

}

/\* close the pipes you just used\*/

close(p2c[0]);

close(c2p[1]);

exit(0);

}

else {

/\* close the pipes that you don't need\*/

close(p2c[0]);

close(c2p[1]);

/\* insert variables

either read from the console, use scanf or generate random numbers

how to generate the seed : srandom(time(0)); and

how to compute the reandom number : int n = random() % (upper\_bound - lower\_bound) + lower\_bound + 1; \*/

while(/\* insert desired condition\*/) {

printf("Parent sending %d\n", n);

if (0 > write(p2c[1], &n, sizeof(int))) {

perror("Error on parent write");

}

/\* in the case when you have an array of bytes, first read the lenght and then the numbers\*/

if (0 > read(c2p[0], &size, sizeof(int))) {

perror("Error on parent read");

}

/\* print if needed\*/

for (i = 0; i < size; i++) {

if (0 > read(c2p[0], &div, sizeof(int))) {

perror("Error on parent read");

}

/\* print if needed\*/

}

/\* in the case when you have to read only 1 number\*/

if (0 > read(c2p[0], &n, sizeof(int))) {

perror("Error on parent read");

}

/\* read another number from console or generate another random one\*/

}

close(p2c[1]);

close(c2p[0]);

wait(0);

}

return 0;

}

Read with spaces:

1.

char str[MAX\_LIMIT];

fgets(str, MAX\_LIMIT, stdin);

2.

char str[20];

scanf("%[^\n]%\*c", str);