FORK

1. Basic fork

1 #include <stdio.h> //perror

2 #include <stdlib.h>

3 #include <unistd.h> // fork

4 #include <sys/wait.h> // wait

5

6 int main(){

7 int p, i;

8 p=fork();

9 if (p == -1) {perror("fork impossible!"); exit(1);}

10 if (p == 0) { // child process

11 for (i = 0; i < 10; i++){

12 // pid=process id, ppid=parent process id

13 printf("Fiu: i=%d,, pid=%d,, ppid-%d\n", i, getpid(), getppid());}

14 exit(0);

15 } else { // parent process0

16 for (i = 0; i < 10; i++)

17 printf("Parinte: i=%d,, pid=%d,, ppid=%d\n", i, getpid(), getppid());

18 wait(0);

19 }

20

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

22 return 0;

23 }

wait -> o sa fie copil zombie daca nu punem

exit -> sa se opreasca codul pt procesul respectiv

SIGNALS

1 #include <stdio.h>

2 #include <stdlib.h>

3 #include <unistd.h>

4 #include <signal.h>

5 #include <string.h>

6

7 void f(int sgn){

8 char s[32];

9 printf("Are you sure you want me to stop [y/N]? ")

10 scanf("%s", s);

11 if(strcmp(s, "y") == 0){

12 exit(0);

13 }

14 }

15

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

17 signal(SIGINT, f);

18 while(1);

19 return 0;

20 }

PROCESSES

p[0] – reading end of the pipe

p[1] – writing end of the pipe

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

2 int a[4] = {1, 2, 3, 4};

3 int pid, p[2];

4

5 pipe(p);

6 pid = fork();

7 if(pid == 0) {

8 close(p[0]); // close the reading end of childe

9 a[2] += a[3];

10 write(p[1], &a[2], sizeof(int)); //write, from the adress of a[2], size of an integer

11 close(p[1]); // close the writing end of childe

12 exit(0);

13 }

14 close(p[1]); // close the writing end of parent

15 a[0] += a[1];

16 read(p[0], &a[2], sizeof(int)); // read and replace at the adress of a[2], size of an integer

17 close(p[0]); // close reading end

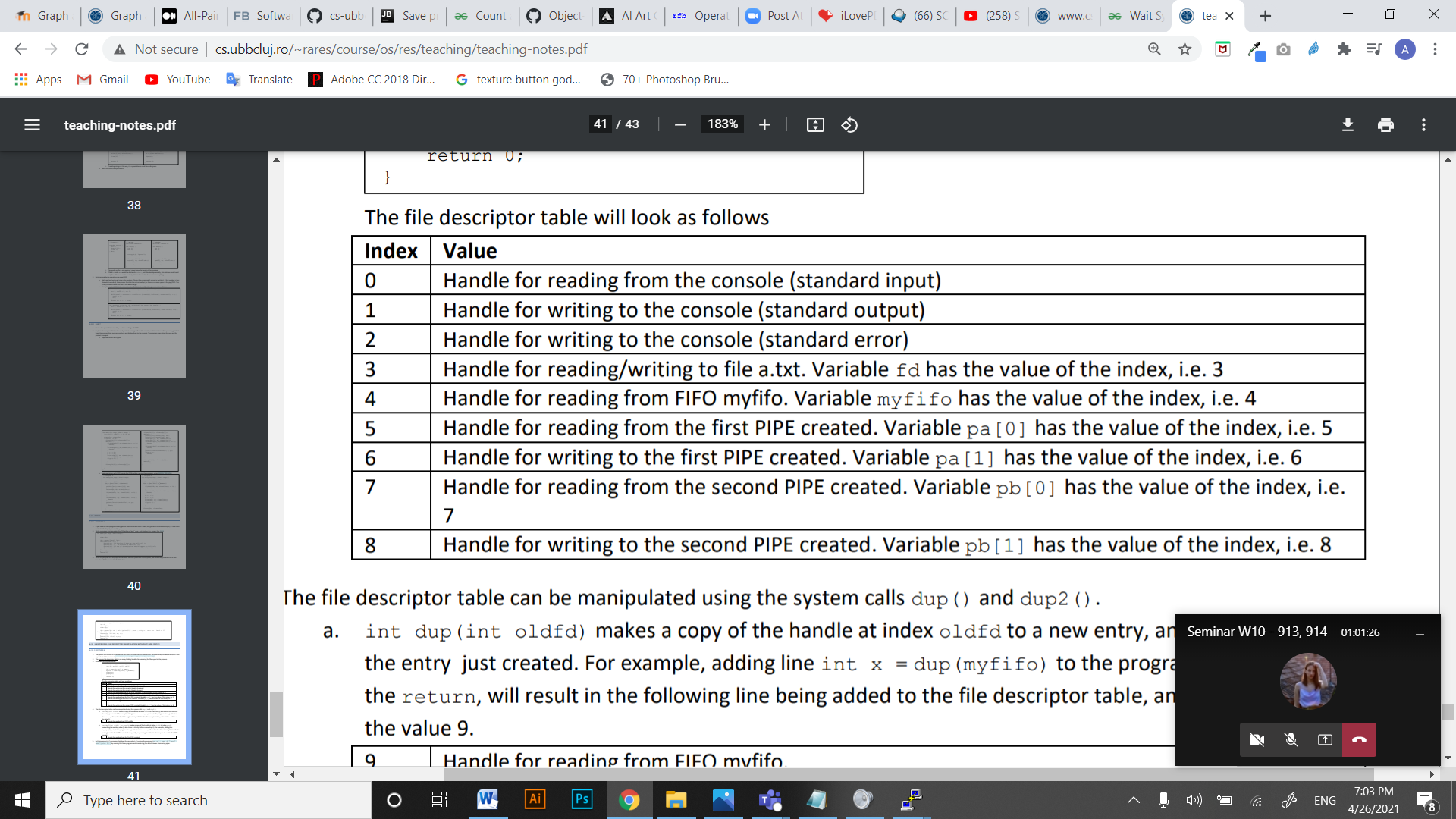
18 wait(0); // wait for childe

19 a[0] += a[2];

20 printf("%d\n", a[0]);

21 return 0;

22 }



1. Write a C program that creates a child process. The parent process reads integer numbers from the keyboard and sends them to the child process via pipe. For each received number, the child process determines its divisors and it sends them back to the parent process via pipe, which prints them. The processes stop when the parent reads the number 0. The C program will perform all the necessary checks.

PRIN PIPES:

#include <unistd.h>

2 #include <stdio.h>

3 #include <stdlib.h>

4 #include <sys/wait.h>

5

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

7 int p2c[2], c2p[2], n, d, p, i;

8 // make the pipes

9 pipe(p2c); pipe(c2p);

10

11 p=fork();

12 if (p == -1) {perror("Fork impossible!"); exit(1);}

13 if (p == 0) { // child process

14 // the child READS the parent and WRITES TO parent

15 close(p2c[1]); // in here parent doesn't write to child

16 close(c2p[0]); // in here child is not read by parent

17

18 while(1){

19 // what you read in the parent (p2c[0]) goes to child in n

20 if(read(p2c[0],&n,sizeof(int)) <= 0){

21 break;

22 }

23

24 // algorithm for divisors

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

26 if(n%i == 0){

27 write(c2p[1], &i, sizeof(int));

28 }

29 }

30

31 // if n == 0

32 if(n == 0){

33 break;

34 }

35 }

36 close(p2c[0]); close(c2p[1]);

37 exit(0);

38 }

39 // parent process

40 // The parent WRITES TO child and READ FROM child

41 close(p2c[0]);

42 close(c2p[1]);

43 while(1){

44 // read number

45 printf("n= \n");

46 scanf("%d", &n);

47 if(n == 0){

48 break;}

49

50 // write in child the number n

51 write(p2c[1], &n, sizeof(int));

52 printf("The divisors of %d are: \n", n);

53 while(read(c2p[0], &d, sizeof(int)) > 0){

54 // read divisors and print them. Stop at the last divisor

55 printf("%d \n", d);

56 if(d == n)

57 break;

58 }

59 printf("Input new number: \n");

60 }

61

62 close(p2c[1]);

63 close(c2p[0]);

64

65 wait(0);

66 return 0;

67 }

**PRIN FIFO(2 programs)**

1 #include <unistd.h>

2 #include <stdio.h>

3 #include <stdlib.h>

4 #include <sys/wait.h>

5 #include <sys/stat.h>

6 #include <sys/types.h>

7 #include <fcntl.h>

8

9 // B IS CHILD, A IS PARENT

10 //

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

12 int a2b, b2a, n, d;

13

14 // the parent WRITES TO child and READS FROM child

15 a2b = open("a2b", O\_WRONLY);

16 b2a = open("b2a", O\_RDONLY);

17

18 while(1) {

19 printf("n= \n");

20 scanf("%d", &n);

21

22 if(n == 0){

23 break;

24 }

25 // write in child the number n

26 write(a2b, &n, sizeof(int));

27 printf("The divisors of %d are: \n", n);

28 while(read(b2a, &d, sizeof(int)) > 0){

29 //read divisors

30 printf("%d \n", d);

31 if (d == n)

32 break;

33 }

34 printf("Input new number: \n");

35 }

36 close(a2b);

37 close(b2a);

38 return 0;

39 }

**PRIN FIFO(1 PROGRAM)**

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

int main(){  
        int p, i;  
        int p2c, c2p;

        p = fork();  
        if(p<0) {  
                perror("fork impossible"); exit(1);  
        }  
        if(p==0)  
        {  
                printf("Starting child\n");  
                int nr;  
                p2c = open("p2c", O\_RDONLY);  
                c2p = open("c2p", O\_WRONLY);  
                while(1){  
                        read(p2c, &nr, sizeof(int));  
                        //if(nr==-1)  
                        //      break;

                        for(i = 1; i<=nr;i++){  
                                if(nr%i==0)  
                                        write(c2p, &i, sizeof(int));  
                        }  
                        i=-1;  
                        write(c2p, &i, sizeof(int));  
                }  
                close(p2c);  
                close(c2p);  
                exit(0);

        }  
        printf("Starting parent\n");  
        p2c = open("p2c", O\_WRONLY);  
        c2p = open("c2p", O\_RDONLY);

        while(1){  
                int nr;  
                printf("Nr= ");  
                scanf("%d", &nr);  
                if(nr==0){  
                        //nr=-1;

  //write(p2c, &nr,sizeof(int));  
                        break;  
                }  
                write(p2c, &nr, sizeof(int));

                while(1){  
                        read(c2p, &i, sizeof(int));  
                        if(i == -1)  
                                break;  
                        printf("%d ", i);

                }  
                printf("\n");  
        }

        close(p2c);  
        close(c2p);  
        wait(0);  
        return 0;  
}

**Problema test 1: Write s C program that creates a child process C, and the parent P communicates with it using pipes. Parent process sends to the child a string (word) of length >15. The child process removes the last four characters and sends the new word to the parent. The parent process adds 1 digit at the beginning of the word and sends the new word to the child. The child again removes the last three characters and so on , until the word has seven or less characters.**

1 #include <unistd.h>

2 #include <stdio.h>

3 #include <stdlib.h>

4 #include <sys/wait.h>

5 #include <string.h>

6

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

8 int p2c[2], c2p[2], p;

9 pipe(p2c); pipe(c2p);

10 p=fork();

11 if(p<0){perror("Fork impossible! \n"); exit(1);}

12 if(p==0){ //child

13 close(p2c[1]);

14 close(c2p[0]);

15 int len, count;

16 char word[200];

17 while(1) {

18 read(p2c[0], &count, sizeof(int));

19 read(p2c[0], word, sizeof(char)\*(count+1));

20

21 if(len <= 7) break;

22 word[count-4]='\0';

23 len = strlen(word);

24 printf("(C)-> %s\n", word);

25

26 write(c2p[1], &len, sizeof(int));

27 write(c2p[1], word, sizeof(char)\*(len+1));

28

29 }

30 close(p2c[0]);

31 close(c2p[1]);

32 exit(0);

33 }

34 // parent

35 close(p2c[0]);

36 close(c2p[1]);

37 char word[200];

38 printf("Word: \n");

**Problema test 2: Write a C program that creates 2 child processes called A and B. The parent process reads a number N from the keyboard, then reads N integers and stores them in an array. Once ALL the integers are read and stored in the array, the parent process sends them to both child processes. Process A determines the maximum of the numbers and process B determines the minimum of the numbers. Both child processes will print the number that they determined. The C program will perform all the necessary checks.**

#include <unistd.h>

2 #include <stdio.h>

3 #include <stdlib.h>

4 #include <sys/wait.h>

5

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

7 int p2a[2], p2b[2], a2p[2], b2p[2];

8 int pmax, pmin, i;

9

10 pipe(p2a); pipe(a2p);

11 pipe(p2b); pipe(b2p);

12

13 pmax = fork();

14 if(pmax<0){ perror("Failed to created max process\n"); exit(1);}

15 if(pmax==0){ // child process for max

16 close(p2a[1]);

17 close(a2p[0]);

18

19 int n;

20 read(p2a[0], &n, sizeof(int));

21 int arr[n];

22 read(p2a[0], arr, sizeof(int)\*n);

23 int maximum=arr[0];

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

25 if(maximum<arr[i]) maximum=arr[i];

26 }

27 write(a2p[1], &maximum, sizeof(int));

28

29 close(p2a[0]);

30 close(a2p[1]);

31 exit(0);

32 }

33 close(a2p[1]); close(p2a[0]);

34

35 pmin = fork();

36 if(pmin<0){ perror("Failed to created min process\n"); exit(1);}

37 if(pmin==0){ // child process for min

38 close(p2b[1]);

39 close(b2p[0]);

40

41 int n;

42 read(p2b[0], &n, sizeof(int));

43 int arr[n];

44 read(p2b[0], arr, sizeof(int)\*n);

45 int minimum=arr[0];

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

47 if(minimum>arr[i]) minimum=arr[i];

48 }

49 write(b2p[1], &minimum, sizeof(int));

50

51 close(p2b[0]);

52 close(b2p[1]);

53 exit(0);

54 }

55

56 close(b2p[1]); close(p2b[0]);

57

58 int n;

59 printf("How many numbers?\n");

60 scanf("%d", &n);

61 write(p2a[1], &n, sizeof(int));

62 write(p2b[1], &n, sizeof(int));

63

64 int arr[n];

65 printf("Input array:\n");

66 for(i=0;i<n;i++) scanf("%d", &arr[i]);

67 write(p2a[1], arr, n\*sizeof(int));

68 write(p2b[1], arr, n\*sizeof(int));

69

70 int max, min;

71 read(a2p[0], &max, sizeof(int));

72 read(b2p[0], &min, sizeof(int));

73 printf("Maximum is: %d\n", max);

74 printf("Minimum is: %d\n", min);

75 close(a2p[0]), close(p2a[1]);

76 close(b2p[0]), close(p2b[1]);

77 wait(0);

78 wait(0);

79 return 0;

80 }

\*\*Vlad:

Write a C program that creates a child process. The parent reads strings (that do not contain spaces) from the keyboard and sends them to the child process via pipe. The child process prints a message for each string that contains only digits. The parent and the child terminate once the parents reads “stop” (lowercase letters only). The C program will perform all the necessary checks.

#include <stdio.h>  
#include <unistd.h>  
#include <stdlib.h>  
#include <sys/wait.h>  
#include <string.h>  
int main(){

        int pid;  
        int p2c[2], c2p[2];  
        pipe(p2c);  
        pipe(c2p);

        pid = fork();

        if(pid < 0)  
        {  
                perror("fork didnt work");  
                exit(1);  
        }  
        if(pid==0){  
                close(p2c[1]);  
                close(c2p[0]);  
                char word[30];  
                while(1){  
                        int count;  
                        read(p2c[0], &count, sizeof(int));  
                        read(p2c[0], word, sizeof(char)\*(count+1));  
                        if(strcmp(word, "stop")==0)  
                                break;

                        int i;  
                        int found = 0;  
                        for(i=0;i<count-1;i++){  
                                if(word[i] < '0' || word[i] > '9'){  
                                        found =1;  
                                        break;  
                                }  
                        }  
                        word[count] = '\0';  
                        if(found == 0){

                                printf("\nWord ul %s has only digits.\n", word);  
                        }

                }

                close(p2c[0]);

               close(c2p[1]);  
                exit(0);

        }

        // parent  
        close(p2c[0]);  
        close(c2p[1]);

        while(1){  
                char word[30];  
                sleep(1);  
                printf("\nWord: ");  
                word[0] = '\0';  
                scanf("%s", word);

                int count;  
                count = strlen(word);  
                //printf("P: %d \n", coun

                write(p2c[1], &count, sizeof(int));  
                write(p2c[1], word, sizeof(char)\* strlen(word));

                if(strcmp(word, "stop") == 0)  
                        break;

        }

        close(p2c[1]);  
        close(c2p[0]);  
        wait(0);  
        return 0;  
}

**Write a C program that creates a child processes C and communicates with it using fifos. Parent process P sends to the childe process C a number n>50 multiple of 3. Process C divides number by 3 and sends this value back to the parent. Parent process receives the number n and add to it a value s={0,1,2} such that it is a multiple of 3, then send it to C. Child receives new number n, divides it by 3, and sends it to the parent and so on until the number is <=5**

**(P) -> 60 (C)->20 P->(21) (C) ->7 (P) ->9 (C)->3**

Program A:

#include <stdio.h>  
#include <unistd.h>  
#include <stdlib.h>  
#include <sys/wait.h>  
#include <string.h>  
#include <sys/types.h>  
#include <sys/stat.h>  
#include <fcntl.h>

int main(){  
        int p2c, c2p;

        int n;  
        p2c = open("p2c", O\_WRONLY);  
        c2p = open("c2p", O\_RDONLY);  
        printf("n= \n");  
        scanf("%d", &n);

        while(n%3 != 0 || n<50){  
                printf(" Wrong input try again bitch n= \n");  
                scanf("%d", &n);  
        }  
        while(n>5){  
                printf("(P) -> %d\n", n);  
                write(p2c, &n, sizeof(int));

                read(c2p, &n, sizeof(int));

                if(n%3!=0)  
                        n= n+ 3- (n%3);

        }

        close(p2c);  
        close(c2p);  
        return 0;

}

Program B:

#include <stdio.h>  
#include <unistd.h>  
#include <stdlib.h>  
#include <sys/wait.h>  
#include <string.h>  
#include <sys/types.h>  
#include <sys/stat.h>  
#include <fcntl.h>

int main(){  
        int p2c, c2p;

        p2c = open("p2c", O\_RDONLY);  
        c2p = open("c2p", O\_WRONLY);  
        int n;  
        read(p2c, &n, sizeof(int));  
        while(n>5){

                n=n/3;  
                printf("(C) -> %d \n", n);

                write(c2p, &n, sizeof(int));

               read(p2c, &n, sizeof(int));

        }

        close(p2c);  
        close(c2p);  
        return 0;

}

|  |
| --- |
| **/\*** |
|  | **1. Write a C program that will read from keyboard an array of N numbers and it will generate 3 child processes. The main process will send to each of the child processes the array which will do the following:** |
|  | **- c1: compute maximum** |
|  | **- c2: compute minimum** |
|  | **- c3: compute average** |
|  | **The results of the child processes will be returned to the part that will display them on the screen.** |
|  | \*/ |
|  |  |
|  |  |
|  | #include <stdio.h> |
|  | #include <unistd.h> |
|  | #include <sys/types.h> |
|  | #include <stdlib.h> |
|  |  |
|  | int main() { |
|  | int n; |
|  | scanf("%d", &n); |
|  | int arr[n]; |
|  | for (int i = 0; i < n; ++i) { |
|  | scanf("%d", &arr[i]); |
|  | } |
|  | int parentToChildren[2]; |
|  | pipe(parentToChildren); |
|  |  |
|  | int minToParent[2]; |
|  | pipe(minToParent); |
|  |  |
|  | int maxToParent[2]; |
|  | pipe(maxToParent); |
|  |  |
|  | int averageToParent[2]; |
|  | pipe(averageToParent); |
|  |  |
|  | int pMin = fork(); |
|  | if (pMin == -1) { |
|  | perror("failed to create min process"); |
|  | exit(1); |
|  | } else if (pMin == 0) { // min process |
|  | read(parentToChildren[0], arr, n \* sizeof(int)); |
|  | int min = arr[0]; |
|  | for (int i = 1; i < n; ++i) { |
|  | if (arr[i] < min) { |
|  | min = arr[i]; |
|  | } |
|  | } |
|  | write(minToParent[1], &min, sizeof(int)); |
|  | close(minToParent[1]); |
|  | } else { // parent |
|  | int pMax = fork(); |
|  | if (pMax == -1) { |
|  | perror("failed to create max process"); |
|  | exit(1); |
|  | } else if (pMax == 0) { // max process |
|  | read(parentToChildren[0], arr, n \* sizeof(int)); |
|  | int max = arr[0]; |
|  | for (int i = 1; i < n; ++i) { |
|  | if (arr[i] > max) { |
|  | max = arr[i]; |
|  | } |
|  | } |
|  | write(maxToParent[1], &max, sizeof(int)); |
|  | close(maxToParent[1]); |
|  | } else { // parent |
|  | int pAverage = fork(); |
|  | if (pAverage == -1) { |
|  | perror("failed to create average process"); |
|  | exit(1); |
|  | } else if (pAverage == 0) { // average process |
|  | read(parentToChildren[0], arr, n \* sizeof(int)); |
|  | double average = 0; |
|  | for (int i = 0; i < n; ++i) { |
|  | average += arr[i]; |
|  | } |
|  | average /= n; |
|  | write(averageToParent[1], &average, sizeof(double)); |
|  | close(averageToParent[1]); |
|  | } else { // parent |
|  | write(parentToChildren[1], arr, n \* sizeof(int)); |
|  | write(parentToChildren[1], arr, n \* sizeof(int)); |
|  | write(parentToChildren[1], arr, n \* sizeof(int)); |
|  | close(parentToChildren[1]); |
|  |  |
|  | int min; |
|  | read(minToParent[0], &min, sizeof(int)); |
|  | close(minToParent[0]); |
|  | printf("min is %d\n", min); |
|  |  |
|  | int max; |
|  | read(maxToParent[0], &max, sizeof(int)); |
|  | close(maxToParent[0]); |
|  | printf("max is %d\n", max); |
|  |  |
|  | double average; |
|  | read(averageToParent[0], &average, sizeof(double)); |
|  | close(averageToParent[0]); |
|  | printf("average is %f\n", average); |
|  |  |
|  | close(parentToChildren[0]); |
|  | wait(0); |
|  | wait(0); |
|  | wait(0); |
|  | } |
|  | } |
|  | } |
|  | } |

/\* Create a C program that generates N random integers (N given at the command line).

\* It then creates a child, sends the numbers via pipe. The child calculates the average and sends the result back.

\*/

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <time.h>

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

if (argc != 2){

printf("Please provide exactly one argument.\n");

exit(1);

}

int p\_to\_c[2], c\_to\_p[2];

pipe(p\_to\_c);

pipe(c\_to\_p);

int f = fork();

if(-1 == f) {

perror("Error on fork ");

exit(1);

} else if (0 == f) {

// close unused pipe ends

close(p\_to\_c[1]);

close(c\_to\_p[0]);

int n = 0, i, nr;

float rez = 0;

if(0 > read(p\_to\_c[0], &n, sizeof(int))) {

perror("Error on read n from parent ");

close(p\_to\_c[0]);

close(c\_to\_p[1]);

exit(1);

}

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

if(0 > read(p\_to\_c[0], &nr, sizeof(int))) {

perror("Error on read from parent ");

close(p\_to\_c[0]);

close(c\_to\_p[1]);

exit(1);

}

rez += nr;

}

rez /= n;

if(0 > write(c\_to\_p[1], &rez, sizeof(float))) {

perror("Error on write result to parent ");

close(p\_to\_c[0]);

close(c\_to\_p[1]);

exit(1);

}

close(p\_to\_c[0]);

close(c\_to\_p[1]);

exit(0);

} else {

// close unused pipe ends

close(p\_to\_c[0]);

close(c\_to\_p[1]);

// convert the parameter from string to int

int n = atoi(argv[1]);

int i, nr;

float rez = -1;

// init random seed to prevent random from returning the same series

srandom(time(0));

// send n first

if(0 > write(p\_to\_c[1], &n, sizeof(int))) {

perror("Error on write n to child ");

close(p\_to\_c[1]);

close(c\_to\_p[0]);

wait(0);

exit(1);

}

// then send n integers

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

// limit the number generation between 0 and 99 just so we can easily check that the calculation is correct

nr = random() % 100;

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

if(0 > write(p\_to\_c[1], &nr, sizeof(int))) {

perror("Error on write number to child ");

close(p\_to\_c[1]);

close(c\_to\_p[0]);

}

}

wait(0);

if(0 > read(c\_to\_p[0], &rez, sizeof(float))) {

perror("Error on read result from child ");

}

printf("Average is %f\n", rez);

close(p\_to\_c[1]);

close(c\_to\_p[0]);

}

return 0;

}

6. Create two processes A and B. A generates a random number n between 50 and 200. If it is even, it sends it to B, if it is odd it sends n+1 to B. B receives the number and divides it by 2 and sends it back to A. The process repeats until n is smaller than 5. The processes will print the value of n at each step.

#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;

while(1) {

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

perror("Error on child read");

}

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

n /= 2;

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

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

perror("Error on child write");

}

if(n < 5)

break;

}

close(p2c[0]);

close(c2p[1]);

exit(0);

} else {

close(p2c[0]);

close(c2p[1]);

srandom(time(NULL));

int n = random() % 151 + 50;

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

while(n >= 5) {

if(n % 2 == 1)

n++;

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

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

perror("Error on parent write");

}

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

perror("Error on parent read");

}

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

}

wait(0);

close(p2c[1]);

close(c2p[0]);

}

return 0;

}