Exercise 19:

/\*19.

\* Write a C program that receives any number of strings as command line arguments.

\* The program creates two child processes, which inherit the parent's command line arguments

\* (ie. no need to send the arguments via pipe/fifo to the children for this problem).

\* Each child process creates a thread for each of the command line arguments.

\* Each thread created by the first child will extract the vowels from its argument

\* and will append them to a string shared among the threads.

\* Each thread created by the second child process will extract the digits from its argument

\* and will add them to a sum shared among the threads.

\* Both child processes wait for their respective threads to finish and send the result to the parent via pipe.

\* The parent displays the results.\*/

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <stdio.h>

#include <time.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <pthread.h>

#include <fcntl.h>

typedef struct {

char\* str;

char\* res;

int\* pos;

pthread\_mutex\_t\* mtx;

} data1;

typedef struct {

char\* str;

int\* sum;

pthread\_mutex\_t\* mtx;

} data2;

void initData1(data1\* d, char\* givenStr, char\* givenRes, int\* givenPos, pthread\_mutex\_t\* givenMtx) {

d->str = givenStr;

d->res = givenRes;

d->pos = givenPos;

d->mtx = givenMtx;

}

void initData2(data2\* d, char\* givenStr, int\* givenSum, pthread\_mutex\_t\* givenMtx) {

d->str = givenStr;

d->sum = givenSum;

d->mtx = givenMtx;

}

int writeToPipe(int fd, char\* buf) {

int nr = strlen(buf);

if (0 > write(fd, &nr, sizeof(int))) {

perror("Error on writing size of string to pipe");

return -1;

}

if (0 > write(fd, buf, sizeof(char) \* nr)) {

perror("E§rror on writing string to the pipe");

return -1;

}

return 0;

}

void readFromPipe(int fd, int len, char\* buf) {

int readBytes = 0;

while (readBytes < len) {

int k;

if ((k = read(fd, buf + readBytes, (len - readBytes) \* sizeof(char))) > 0)

readBytes += k;

}

}

void\* threadA(void\* a) {

data1 d = \*((data1\*)a);

int i;

for (i = 0; i < strlen(d.str); i++) {

if (d.str[i] == 'a' || d.str[i] == 'e' || d.str[i] == 'i' || d.str[i] == 'o' || d.str[i] == 'u'

|| d.str[i] == 'A' || d.str[i] == 'E' || d.str[i] == 'I' || d.str[i] == 'O' || d.str[i] == 'U') {

pthread\_mutex\_lock(d.mtx);

d.res[\*d.pos] = d.str[i];

\*d.pos += 1;

pthread\_mutex\_unlock(d.mtx);

}

}

return NULL;

}

void\* threadB(void\* b) {

data2 d = \*((data2\*)b);

int i;

for (i = 0; i < strlen(d.str); i++) {

if (d.str[i] >= '0' && d.str[i] <= '9') {

pthread\_mutex\_lock(d.mtx);

\*d.sum += (d.str[i] - '0');

pthread\_mutex\_unlock(d.mtx);

}

}

return NULL;

}

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

int a2p[2], b2p[2];

if (argc < 2) {

printf("Please provide at least one string argument");

exit(1);

}

if (0 > pipe(a2p)) {

perror("Error on pipe a2cp");

exit(1);

}

if (0 > pipe(b2p)) {

perror("Error on pipe b2p");

exit(1);

}

int n = argc - 1;

int i;

int fa = fork();

if (0 > fa) {

perror("Error on creating first child process");

exit(1);

}

else if (fa == 0) {

close(b2p[0]);

close(b2p[1]);

close(a2p[0]);

int len = 0;

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

len += strlen(argv[i + 1]);

/\*in the case all given arguments contain only vowels, we need one more character

\* so the it can have a NULL terminator\*/

len++;

pthread\_mutex\_t\* mtx = malloc(sizeof(pthread\_mutex\_t));

pthread\_mutex\_init(mtx, NULL);

char\* result = malloc(sizeof(char) \* len);

memset(result, 0, sizeof(char) \* len);

int\* position = malloc(sizeof(int));

\*position = 0;

data1\* args = malloc(sizeof(data1) \* n);

pthread\_t\* T = malloc(sizeof(pthread\_t) \* n);

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

initData1(&args[i], argv[i + 1], result, position, mtx);

if (0 > pthread\_create(&T[i], NULL, threadA, (void\*)&args[i])) {

perror("Error creating threads in first child process");

exit(1);

}

}

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

pthread\_join(T[i], NULL);

if (-1 == writeToPipe(a2p[1], result)) {

free(T);

free(args);

free(position);

free(result);

free(mtx);

close(a2p[1]);

exit(1);

}

free(T);

free(args);

free(position);

free(result);

free(mtx);

close(a2p[1]);

exit(1);

return 0;

}

int fb = fork();

if (0 > fb) {

perror("Error on creating first child process");

exit(1);

}

else if (fb == 0) {

close(a2p[0]);

close(a2p[1]);

close(b2p[0]);

pthread\_mutex\_t\* mtx = malloc(sizeof(pthread\_mutex\_t));

pthread\_mutex\_init(mtx, NULL);

int\* sum = malloc(sizeof(int));

\*sum = 0;

data2\* args = malloc(sizeof(data2) \* n);

pthread\_t\* T = malloc(sizeof(pthread\_t) \* n);

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

initData2(&args[i], argv[i + 1], sum, mtx);

if (0 > pthread\_create(&T[i], NULL, threadB, (void\*)&args[i])) {

perror("Error creating threads in second child process");

free(T);

free(args);

free(sum);

free(mtx);

close(b2p[1]);

exit(1);

}

}

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

pthread\_join(T[i], NULL);

if (0 > write(b2p[1], sum, sizeof(int))) {

perror("Error writing sum to pipe");

free(T);

free(args);

free(sum);

free(mtx);

close(b2p[1]);

exit(1);

}

free(T);

free(args);

free(sum);

free(mtx);

close(b2p[1]);

exit(0);

}

close(a2p[1]);

close(b2p[1]);

char\* buf;

int len;

if (0 > read(a2p[0], &len, sizeof(int)))

perror("Error on reading string size from pipe");

buf = malloc(sizeof(char) \* (len + 1));

memset(buf, 0, sizeof(char) \* (len + 1));

readFromPipe(a2p[0], len, buf);

int sum;

if (0 > read(b2p[0], &sum, sizeof(int)))

perror("Error on reading sum from pipe");

close(a2p[0]);

close(b2p[0]);

wait(0);

wait(0);

printf("All vowels: %s\n", buf);

printf("Sum of all digits: %d\n", sum);

free(buf);

return 0;

}

Exercise 20:

/\*20.

\* Write a C program that creates N threads and one child process (N given as a command line argument).

\* Each thread will receive a unique id from the parent.

\* Each thread will generate two random numbers between 1 and 100 and will print them together with its own id.

\* The threads will send their generated numbers to the child process via pipe or FIFO.

\* The child process will calculate the average of each pair of numbers received from a thread and will print it alongside the thread id.

\* Use efficient synchronization.

\*/

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <pthread.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <sys/stat.h>

typedef struct {

int idx;

pthread\_mutex\_t\* mtx;

int fd;

} data;

typedef struct {

int idx;

int nr1;

int nr2;

} response;

void initData(data\* d, int givenIdx, pthread\_mutex\_t\* givenMtx, int\* givenFd) {

d->idx = givenIdx;

d->mtx = givenMtx;

d->fd = \*givenFd;

}

void\* thread(void\* a) {

data d = \*((data\*)a);

response r;

r.idx = d.idx;

r.nr1 = random() % 101;

r.nr2 = random() % 101;

printf("Thread %d - generated %d and %d\n", r.idx, r.nr1, r.nr2);

pthread\_mutex\_lock(d.mtx);

if (0 > write(d.fd, &r, sizeof(response)))

perror("Error on sending data");

pthread\_mutex\_unlock(d.mtx);

return NULL;

}

void makeFifoIfNotExists(char\* fifoName) {

if (0 > mkfifo(fifoName, 0600)) {

perror("Error on creating fifo");

exit(1);

}

}

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

if (argc != 2) {

perror("Please provide 1 integer argument");

exit(1);

}

char\* fifoName = "./theFifo";

makeFifoIfNotExists(fifoName);

int n = atoi(argv[1]);

int f = fork();

if (f < 0) {

perror("Error on creating child");

exit(1);

}

else if (f == 0) {

int i;

int fd = open(fifoName, O\_RDONLY);

if (0 > fd) {

perror("Error on opening fifo");

exit(1);

}

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

response r;

if (0 > read(fd, &r, sizeof(response)))

perror("Error on reading data");

printf("Thread %d - average: %f\n", r.idx, (double)(r.nr1 + r.nr2) / 2);

}

if (0 > close(fd))

perror("Error on closing fifo");

exit(0);

}

int i;

int fd = open(fifoName, O\_WRONLY);

if (0 > fd) {

perror("Error on opening fifo");

exit(1);

}

pthread\_mutex\_t\* mtx = malloc(sizeof(pthread\_mutex\_t));

pthread\_mutex\_init(mtx, NULL);

data\* args = malloc(sizeof(data) \* n);

pthread\_t\* T = malloc(sizeof(pthread\_t) \* n);

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

initData(&args[i], i, mtx, &fd);

if (0 > pthread\_create(&T[i], NULL, thread, (void\*)&args[i])) {

perror("Error on creating thread");

wait(0);

free(args);

free(T);

free(mtx);

close(fd);

if (0 > unlink(fifoName))

perror("Error on unlinking fifo");

exit(1);

}

}

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

pthread\_join(T[i], NULL);

wait(0);

free(args);

free(T);

free(mtx);

if (0 > close(fd))

perror("Error on closing fifo");

if (0 > unlink(fifoName))

perror("Error on unlinking fifo");

return 0;

}

Exercise 21:

/\*21.

\* Write a C program named A that creates 3 child processes named B, C and D.

\* A generates a random number between 10 and 20 and sends it to process D.

\* Processes B and C keep generating numbers between 1 and 200 and send them to process D

\* which calculates their difference.

\* The processes stop when the absolute difference between the numbers generated by B and C

\* is less or equal to the number generated by process A.

\* \*/

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

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

int a2d[2], b2d[2], c2d[2], d2b[2], d2c[2];

if (0 > pipe(a2d)) {

perror("Error on pipe a2d");

exit(1);

}

if (0 > pipe(b2d)) {

perror("Error on pipe b2d");

exit(1);

}

if (0 > pipe(c2d)) {

perror("Error on pipe c2d");

exit(1);

}

if (0 > pipe(d2b)) {

perror("Error on pipe d2b");

exit(1);

}

if (0 > pipe(d2c)) {

perror("Error on pipe d2c");

exit(1);

}

int fb = fork();

if (0 > fb) {

perror("Error on creating child b");

exit(1);

}

else if (fb == 0) {

close(a2d[0]);

close(a2d[1]);

close(c2d[0]);

close(c2d[1]);

close(d2c[0]);

close(d2c[1]);

close(d2b[1]);

close(b2d[0]);

srandom(getpid());

int b;

while (1) {

b = random() % 201;

printf("Process B - generated: %d\n", b);

if (0 > write(b2d[1], &b, sizeof(int))) {

perror("Error on writing data to pipe");

exit(1);

}

if (0 > read(d2b[0], &b, sizeof(int))) {

perror("Error on reading data from pipe");

exit(1);

}

if (b == -1)

break;

}

close(d2b[0]);

close(b2d[1]);

exit(0);

}

int fc = fork();

if (0 > fc) {

perror("Error on creating child b");

exit(1);

}

else if (fc == 0) {

close(a2d[0]);

close(a2d[1]);

close(d2b[0]);

close(d2b[1]);

close(b2d[0]);

close(b2d[1]);

close(c2d[0]);

close(d2c[1]);

srandom(getpid());

int c;

while (1) {

c = random() % 201;

printf("Process C - generated: %d\n", c);

if (0 > write(c2d[1], &c, sizeof(int))) {

perror("Error on writing data to pipe");

exit(1);

}

if (0 > read(d2c[0], &c, sizeof(int))) {

perror("Error on reading data from pipe");

exit(1);

}

if (c == -1)

break;

}

close(c2d[1]);

close(d2c[0]);

exit(0);

}

int fd = fork();

if (0 > fd) {

perror("Error on creating child b");

exit(1);

}

else if (fd == 0) {

close(a2d[1]);

close(b2d[1]);

close(d2b[0]);

close(c2d[1]);

close(d2c[0]);

int a;

if (0 > read(a2d[0], &a, sizeof(int))) {

perror("Error on reading data from pipe a2d");

exit(1);

}

int b, c;

int stop = 0;

while (stop == 0) {

if (0 > read(b2d[0], &b, sizeof(int))) {

perror("Error on writing data from pipe b2d");

exit(1);

}

if (0 > read(c2d[0], &c, sizeof(int))) {

perror("Error on writing data from pipe c2d");

exit(1);

}

int dif = abs(b - c);

printf("Process D: %d\n", dif);

if (dif <= a)

stop = -1;

if (0 > write(d2b[1], &stop, sizeof(int))) {

perror("Error on writing data to pipe d2b");

exit(1);

}

if (0 > write(d2c[1], &stop, sizeof(int))) {

perror("Error on writing data to pipe d2c");

exit(1);

}

}

close(a2d[0]);

close(d2b[1]);

close(b2d[0]);

close(c2d[0]);

close(d2c[1]);

exit(0);

}

close(a2d[0]);

close(b2d[0]);

close(c2d[0]);

close(d2b[0]);

close(d2c[0]);

close(b2d[1]);

close(c2d[1]);

close(d2b[1]);

close(d2c[1]);

srandom(getpid());

int a = random() % 11 + 10;

printf("Process A: %d\n", a);

if (0 > write(a2d[1], &a, sizeof(int))) {

perror("Error on writing data to pipe a2d");

exit(1);

}

wait(0);

wait(0);

wait(0);

close(a2d[1]);

return 0;

}