Zad. 1.1

Napisz program `InputOutput` odczytujący z terminala i wypisujący na konsoli kolejno wartości typu: `int`, `float` i napis. Bufor napisu ma mieć rozmiar 10. Odczyt napisu nie może powodować przepełnienia bufora. Przykładowa sesja:

```
a = 5
int value a = 5

b = 5
float value b = 5.000000

c = 0123456789
string value c = 012345678
```

- dlaczego wypisany napis nie ma litery 9 ?

Zad. 1.1.1

Odczytaj z terminala napis `abcdefghij` przy pomocy funkcji `fgets` z wykorzystaniem poprzedniego bufora napisu i wypisz go na konsoli.

- dlaczego nie można pobrać drugiego napisu?

- dlaczego na konsoli wyświetla się literka 9?

- w jaki sposób wyczyścić zawartość strumienia wejściowego?

Zad. 1.1.2 *

Odczytaj i wypisz na konsoli wartość zmienno-przecinkową typu `double`.

Zad. 1.1.3 *

Odczytaj i wypisz na konsoli cały napis "ala ma kota” przy pomocy funkcji `scanf`.

Zad. 1.1.4 *

Odczytaj i wypisz na konsoli cały napis "ala ma kota” przy pomocy funkcji `fgets`.

Zad. 1.2 *

Napisz program `empty` sprawdzający czy strumień wejściowy jest pusty.

Zad. 1.3

Napisz program `Fibo` wyliczający wartości ciągu Fibonacciego przy pomocy trzech funkcji.

- podaj definicję ciągu Fibonacciego

```
0   1   2   3   4   5   6    indeksy
1   1   2   3   5   8   13   wartosci
```

Zad. 1.3.1

Funkcja `fibo1` - metoda dziel i zwyciężaj.

```
f(0) = 1
f(1) = 1
f(n) = f(n-1) + f(n-2)
```

- dokonaj analizy wywołania `fibo1(4)`.

- narysuj drzewo wywołań dla `fibo1(4)`.

Zad. 1.3.2

Funkcja `fibo2` - metoda programowania dynamicznego z ramką trójzębną.

```
r0  r1  r2
|---|---|
0   1   2   3   4   5   6    indeksy
1   1   2   3   5   8   13   wartości
    |---|---|
    r0  r1  r2
```

Przesunięcie ramki w prawo:

```
r0 = r1
r1 = r2
r2 = r1 + r0
```

- ile razy należy przesunąć ramkę w prawo, aby wyznaczyć wartość `n`-tego wyrazu ciągu Fibonacciego w funkcji `fibo2` dla `n >= 3` ?

- dokonaj analizy wywołania `fibo2(4)`.

- narysuj graf obliczeń dla `fibo2(4)`.

Zad. 1.3.3 *

Funkcja `fibo3` - metoda programowania dynamicznego z ramką dwuzębną.

```
r0  r1
|---|
0   1   2   3   4   5   6    indeksy
```

```
1   1   2   3   5   8   13   wartości
|   |---|
pom r0  r1
```

Przesunięcie ramki w prawo:

```
pom = r0
r0 = r1
r1 = r0 + pom
```

- ile razy należy przesunąć ramkę w prawo, aby wyznaczyć wartość `n`-tego wyrazu ciągu Fibonacciego w funkcji `fibo3` dla `n >= 2` ?

- dokonaj analizy wywołania `fibo3(4)`.

- narysuj graf obliczeń dla `fibo3(4)`.

- która funkcja ma mniejszą złożoność obliczeniową `fibo2` czy `fibo3` ?

Przykładowa sesja:

```
fibo1(4) = 5
fibo2(4) = 5
fibo3(4) = 5
```

Zad. 1.3.4 *

Podaj cztery inne funkcje wyliczające rekurencyjnie wartości ciągu Fibonacciego.

Zad. 1.4 *

Napisz program `Sequence` wyliczający wartości ciągu $\{a_n\}$ przy pomocy trzech funkcji. Ciąg zdefiniowany jest rekurencyjnie:

```
a(0) = 1
a(1) = 4
a(n) = 2*a(n-1) + 0.5*a(n-2)
```

- wylicz dziesięć pierwszych wyrazów ciągu $\{a_n\}$ w programie Excel

Zad. 1.4.1 *

Funkcja `a1` - metoda dziel i zwyciężaj.

- dokonaj analizy wywołania `a1(4)`.

- narysuj drzewo wywołań dla `a1(4)`.

Zad. 1.5 *

W pliku `liczba.txt` wyznacz funkcję `f(n)` wyliczającą liczbę cyfr dla dowolnej liczby całkowitej `n`. Wykorzystaj funkcję $\log_{10}(x)$.

Zad. 2.1

Napisz program `FiboTree` wypisujący, jak wyglądają kolejne wywołania funkcji `fibo1` razem z wartościami przez nie zwracanymi. Przykładowa sesja:

```
fibo1(4) = 5
fibo2(3) = 3
fibo3(2) = 2
fibo4(1) = 1
fibo5(0) = 1
fibo6(1) = 1
fibo7(2) = 2
fibo8(1) = 1
fibo9(0) = 1
```

- sprawdź czy drzewo wywołań z wcześniejszego zadania zostało poprawnie narysowane

Zad. 2.2

Napisz program `Sequence` wyliczający wartości ciągu $\{a_n\}$ przy pomocy trzech funkcji. Ciąg zdefiniowany jest rekurencyjnie:

```
a(0) = 1
a(1) = 4
a(n) = 2*a(n-1) + 0.5*a(n-2)
```

Zad. 2.2.1 *

Funkcja `a2` - metoda programowania dynamicznego z ramką trójzębną.

- narysuj schemat dla ramki trójzębnej analogicznie jak dla ciągu Fibonacciego

- ile razy należy przesunąć ramkę w prawo, aby wyznaczyć wartość `n`-tego wyrazu ciągu $\{a_n\}$ w funkcji `a2` dla `n >= 3` ?

- pętla przesuwająca ramkę tym razem musi startować od indeksu 1

- dokonaj analizy wywołania `a2(4)`.

- narysuj graf obliczeń dla `a2(4)`.

Zad. 2.2.2

Funkcja `a3` - metoda programowania dynamicznego z ramką dwuzębną.

- narysuj schemat dla ramki dwuzębnej analogicznie jak dla ciągu Fibonacciego

- ile razy należy przesunąć ramkę w prawo, aby wyznaczyć wartość `n`-tego wyrazu ciągu $\{a_n\}$ w funkcji `a3` dla `n >= 2` ?

- pętla przesuwająca ramkę tym razem musi startować od indeksu 1

- dokonaj analizy wywołania `a3(4)`.

- narysuj graf obliczeń dla `a3(4)`.

Przykładowa sesja:

```
a1(4) = 42.250000
a2(4) = 42.250000
a3(4) = 42.250000
```

Zad. 2.3 *

Napisz program `SequenceTree` wypisujący, jak wyglądają kolejne wywołania funkcji `a1` razem z wartościami przez nie zwracanymi. Przykładowa sesja:

```
a1(4) = 42.250000
a2(3) = 19.000000
a3(2) = 8.500000
a4(1) = 4.000000
a5(0) = 1.000000
a6(1) = 4.000000
a7(2) = 8.500000
a8(1) = 4.000000
a9(0) = 1.000000
```

- sprawdź czy drzewo wywołań z wcześniejszego zadania zostało poprawnie narysowane

Zad. 2.4 *

Zaproponuj metodę kodowanie drzewa katalogowego przy pomocy tablic trójwymiarowych w języku Java. Przetestuj ją dla następujących drzew.

Zad. 3.1

Napisz program `Fractions` wedle poniższego schematu i przetestuj jego działanie. Rozbuduj program o pozostałe działania na ułamkach, sugerowane nazwy funkcji: `sum`, `sub`, `mul`, `quo`. Przed każdą dodaną funkcją należy podać wyprowadzenie dla działania.

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

struct Fraction {
    int num, den;
};

/*
a/b + c/d = (a*d)/(b*d) + (c*b)/(d*b) = (a*d + c*b) / (b*d)
*/

struct Fraction sum(struct Fraction x, struct Fraction y) {
    int a = x.num;
    int b = x.den;

    int c = y.num;
    int d = y.den;

    struct Fraction z;

    z.num = a*d + c*b;
    z.den = b*d;

    return z;
}

void main() {
    struct Fraction x = {2, 3};
    struct Fraction y = {1, 4};

    struct Fraction z = sum(x, y);

    printf("%d/%d + %d/%d = %d/%d\n", x.num, x.den,
                                      y.num, y.den, z.num, z.den);
}
```

Zad. 3.1.1

Dodaj do programu funkcję `print` wypisującą działanie dla podanych ułamków i operatora.

```
void print(struct Fraction x, struct Fraction y, const char op);
```

Przykładowa sesja:

```
2/3 + 1/4 = 11/12
2/3 - 1/4 = 5/12
2/3 * 1/4 = 2/12
2/3 / 1/4 = 8/3
2/3 : 1/4 = 8/3
$ - nieznane dzialanie
```

Zad. 3.1.2 *

Dodaj do programu funkcję `printFraction` wypisującą ułamek wedle schematu:

```
printFraction((struct Fraction){2,0});  // NaN
printFraction((struct Fraction){0,2});  // 0
printFraction((struct Fraction){2,4});  // 1/2
printFraction((struct Fraction){-1,2}); // -1/2
printFraction((struct Fraction){1,-2}); // -1/2
printFraction((struct Fraction){4,-2}); // -2
printFraction((struct Fraction){5,-2}); // -2 1/2
```

Prototyp funkcji:

```
void printFraction(struct Fraction x);
```

Zad. 3.2

Napisz program `Strings` implementujący i testujący następujące funkcje:

Zad. 3.2.1

Funkcja `indexOf` działająca analogicznie do tej samej funkcji z klasy `String` w języku Java. W implementacji należy wykorzystać zmienne z poniższego schematu.

```
/*                  index
                    |
          0    1    2    3    4      indexes
str -> ['9']['9']['$'][' ']['\0']
       |         |
       str       ptr                 pointers
*/

int indexOf(const char *str, int c);
```

Zad. 3.2.2 *

Funkcja `identity` zwracająca imię i nazwisko oddzielone spacją.

```
char *identity(const char *name, const char *surname);
```

Zad. 3.2.3 *

Funkcja `login` tworząca login użytkownika na podstawie pierwszej litery imienia i całego nazwiska. Należy pamiętać, że login na systemie Linux składa się z maksymalnie 32 znaków.

```
char *login(const char *name, const char *surname);
```

Zad. 3.3 *

Podaj słowny opis funkcji rekurencyjnej, która utworzy drzewo katalogowe dla tablicy trójwymiarowej z zad. 2.4 * z poprzednich laboratoriów. Można użyć funkcji tworzących plik i katalog oraz funkcji zmiany katalogu i przejścia do katalogu poziom wyżej.

Zad. 4.1

Napisz pełną implementację programu `Fractions` z odczytem danych z terminala i obsługą błędów.

Zad. 4.1.1

Dodaj funkcję `isNumber` sprawdzającą, czy napis `s` przechowuje liczbę całkowitą. Wykorzystaj funkcję `isdigit` z biblioteki standardowej.

```
/*
       0    1    2    3      index
s -> ['-']['3']['5']['\0']
     |    |    |    |        pointer
     s    s+1  s+2  s+3
*/

int isNumber(const char *s);
```

Zad. 4.1.2

Dodaj funkcję `trim` usuwającą z napisu `s` początkowe i końcowe znaki białe. Wykorzystaj funkcję `isspace` z biblioteki standardowej.

```
/*
       i         i'        j'        j
       0    1    2    3    4    5    6    7
s -> [' '][' ']['a']['l']['a'][' '][' '][' ']['\0']
       k
*/

char *trim(char *s);
```

Zad. 4.1.3

Dodaj funkcję `getOperator` określającą operator z napisu `s`. Funkcja zwraca prawdę, jeśli napis jest poprawnym operatorem arytmetycznym.

```
int getOperator(char *op, const char *s);
```

Zad. 4.1.4

Dodaj funkcję `getFraction` określającą strukturę ułamkową z napisu `s`. Funkcja zwraca prawdę, jeśli napis jest ułamkiem zwykłym lub liczbą całkowitą. Napis może mieć postać:

liczba
liczba / liczba

```
/*               index
                 |
       0    1    2    3    4      indexes
s -> ['3']['7']['/']['5']['\0']
     |         |
     s         slash              pointers
*/

int getFraction(struct Fraction *x, const char *s);
```

Zad. 4.1.5

Zaimplementuj odczyt z terminala zgodnie z poniższym przykładem. Jeśli wprowadzony napis ma niepoprawny format, to jego odczyt należy powtórzyć. Przykładowa sesja:

```
a/b = 2/3
c/d = 1/4

op = +

2/3 + 1/4 = 11/12
```

Zad. 4.1.6 *

Zaimplementuj bezpieczny odczyt z terminala bez możliwości przepełnienia bufora linii.

Laboratorium 5

Zad. 5.1

Przepisz program `Files` otwierający plik do odczytu oraz implementujący i testujący następujące funkcje:

Zad. 5.1.1

Funkcja `printChars` odczytuje zawartość pliku bajt po bajcie.

```
void printChars(FILE *fp);
```

Zad. 5.1.2

Funkcja `printLines` odczytuje zawartość pliku linia po linii.

```
void printLines(FILE *fp);
```

Zad. 5.1.3 *

Funkcja `copy` kopiuje pliki analogicznie do komendy `cp`.

```
void copy(const char *addr1, const char *addr2);
```

Zad. 5.2

Napisz program `CountWords` odczytujący plik tekstowy oraz obliczający liczbę słów w tym pliku przy pomocy funkcji:

Zad. 5.2.1

Funkcja `countWords1` oblicza liczbę słów przy pomocy zmiennej stanu `inside`, gdzie wartość 0 oznacza, że zamierzamy odczytać znak poza słowem, a wartość 1 oznacza, że zamierzamy odczytać znak w słowie.

```
int countWords1(FILE *fp);
```

Zad. 5.2.2 *

Funkcja `countWords2` oblicza liczbę słów bez pomocy zmiennej stanu.

```
int countWords2(FILE *fp);
```

Zad. 5.3

Napisz program `DisplayWords` odczytujący plik tekstowy oraz wypisujący poszczególne słowa tego pliku przy pomocy funkcji `strtok` z biblioteki standardowej.

```
void printWords(FILE *fp);
```

Zad. 5.4

Napisz program `NewLine` testujący jakie znaki określają koniec linii w pliku tekstowym na systemie Windows i Linux. Porównaj działanie programu na obu systemach. Zaimplementuj funkcje:

Zad. 5.4.1

Funkcja `printHex` wypisuje plik w postaci dwucyfrowych liczb szesnastkowych.

```
void printHex(FILE *fp);
```

Zad. 5.4.2

Funkcja `printChar` wypisuje plik w postaci znaków umieszczonych w apostrofach.

```
void printChar(FILE *fp);
```

Zad. 5.4.3

Funkcja `printLinesHex` wypisuje plik linia po linii analogicznie do `printHex`.

```
void printLinesHex(FILE *fp);
```

Zad. 5.4.4

Funkcja `printLinesChar` wypisuje plik linia po linii analogicznie do `printChar`.

```
void printLinesChar(FILE *fp);
```

Przykładowa sesja:

```
1 linux.txt
2 windows.txt

Choose file: 1

linux.txt

61 6C 61 0A 6B 6F 74 0A

'a' 'l' 'a' '\n' 'k' 'o' 't' '\n'

61 6C 61 0A
6B 6F 74 0A

'a' 'l' 'a' '\n'
'k' 'o' 't' '\n'
```

Pliki do testów można pobrać ze strony:

```
http://balois.pl/file/pliki.zip
```

- dlaczego rozmiary tych plików są różne?

Dla systemu Linux program można uruchomić na stronie:

```
https://cocalc.com
```

- jakie znaki określają koniec linii w systemie Linux, Windows i Mac OS?

```
https://pl.wikipedia.org/wiki/Koniec_linii
```

Zad. 5.5 *

Obsługa plików i katalogów w języku C dla systemu Windows. Referat dla dwóch osób na za dwa tygodnie.

Zad. 5.6 *

Obsługa plików i katalogów w języku C dla systemu Linux. Referat dla dwóch osób na za trzy tygodnie.

Zad. 5.7 *

Obsługa plików i katalogów w języku Java. Referat dla dwóch osób na za cztery tygodnie.

Projekt 1

Napisz w asemblerze i w języku C program ilustrujący wywołanie funkcji systemowej (ang. system call). W programie w C należy użyć wrapera funkcji systemowej lub funkcji `syscall`, jeśli funkcja systemowa nie ma wrapera. Jedna połowa grupy wykonuje projekt dla kodu 32 bitowego, a druga połowa grupy wykonuje projekt dla kodu 64 bitowego. Sugerowana nazwa programu to nazwa funkcji systemowej.

Deadline: 20.11.2023

Projekty należy wysłać na maila w spakowanym katalogu:

```
Nazwisko Imie
  nazwa.c
  nazwa.asm
```

W nazwie katalogu nie używamy polskich znaków.

Laboratorium 6

Zad. 6.1

1. Sprawdzić czy zainstalowany jest kompilator GCC?

2. Zainstaluj GCC jeśli nie jest zainstalowany.

3. Zaloguj się na drugim terminalu `ALT+F2`.

3. W katalogu domowym utwórz katalog `programy`.

4. W katalogu `programy` wykonaj `nano hello.c`

5. Napisz program `HelloWorld` i wyjdź z edytora.

Zad. 6.2

1. Na pierwszym terminalu sprawdź czy w katalogu domowym istnieje plik `.nanorc`

2. Sprawdź w jaki sposób w edytorze Nano ustawić rozmiar tabulacji na 4 i włączyć opcje tworzenia tabulacji z użyciem spacji.

3. W katalogu domowym otwórz plik `.nanorc` do edycji w edytorze Nano.

4. Dokonaj konfiguracji rozmiaru tabulacji oraz opcji tabulacji za pomocą spacji.

5. Na drugim terminalu otwórz plik `hello.c` i sprawdź, czy działają nowe ustawienia.

6. Na pierwszym terminalu przejdź do katalogu `programy` i skompiluj `hello.c`

```
gcc hello.c -o hello
```

7. Uruchom program `hello`.

8. Dla pliku `hello.c` ustaw uprawnia tylko do odczytu. *

9. Kiedy można uruchomić program `hello` przez podanie jego nazwy?

Zad. 6.3

Przy pomocy komendy `whoami` odczytaj nazwę bieżącego użytkownika.

Zad. 6.4

Przy pomocy komendy `getent passwd` odczytaj wpis dla bieżącego użytkownika.

Zad. 6.5

Przy pomocy komendy `getent group` odczytaj identyfikator grupy `staff`.

Zad. 6.6

Przy pomocy komendy `delgroup` usuń grupę `staff`.

Zad. 6.7

Przy pomocy komendy `addgroup` utwórz grupę `staff` z poprzednim identyfikatorem.

- w pliku `/etc/group` znajdź wpis dla grupy `staff`

- jaki identyfikator ma grupa `staff` ?

`https://www.computerhope.com/unix/adduser.htm`

Zad. 6.8

Przy pomocy komendy `adduser` do grupy `staff` dodaj użytkownika `dyrektor`.

- w pliku `/etc/passwd` znajdź wpis dla użytkownika `dyrektor`

- jaki katalog domowy ma użytkownik `dyrektor` ?

- jaka powłoka obsługuje terminal użytkownika `dyrektor` ?

Zad. 6.9

Przy pomocy komendy `adduser` do grupy `student` dodaj użytkownika `jkowalski` z opisem `jan kowalski` oraz hasłem `haslo1`.

- w pliku `/etc/passwd` przejrzyj wpis dla użytkownika `jkowalski`

- jaki katalog domowy ma użytkownik `jkowalski` ?

- jaka powłoka obsługuje terminal użytkownika `jkowalski` ?

Zad. 6.10

Przy pomocy komendy `deluser` usuń użytkownika `jkowalski`.

Zad. 6.11

Usuń katalog domowy użytkownika `jkowalski`.

- jaka opcja dla komendy `deluser` usuwa użytkownika razem z jego katalogiem domowym?

Zad. 6.12

Przy pomocy komendy `openssl passwd -crypt` zaszyfruj napis `haslo1`. Powtórz komendę trzy razy. Czy szyfrowanie jest jednoznaczne?

- ile znaków maksymalnie może mieć hasło dla tej komendy?

Zad. 6.13

Przy pomocy komendy `useradd` do grupy `student` dodaj użytkownika `jkowalski` z opisem `jan kowalski` oraz hasłem `haslo1`. Parametry dla komendy należy dobrać w taki sposób, aby użytkownik został utworzony analogicznie jak w zadaniu 6.9.

- w pliku `/etc/passwd` przejrzyj wpis dla użytkownika `jkowalski`

- jaki katalog domowy ma użytkownik `jkowalski` ?

- jaka powłoka obsługuje terminal użytkownika `jkowalski` ?

`https://www.computerhope.com/unix/useradd.htm`

Zad. 6.14

Przy pomocy komendy `userdel` usuń użytkownika `jkowalski` razem z jego katalogiem domowym.

Zad. 6.15

Przepisz i przeanalizuj program `crypt-gnu.c` ilustrujący szyfrowanie haseł.

```
http://balois.pl/file/crypt-gnu.c
```

Zad. 6.16 *

Przepisz i przeanalizuj program `crypt-xopen.c` ilustrujący szyfrowanie haseł.

```
http://balois.pl/file/crypt-xopen.c
```

Poczytaj strony:

```
https://linux.die.net/man/3/crypt_r
https://manpages.ubuntu.com/manpages/bionic/pl/man3/crypt.3.html
```

Zad. 6.17

Napisz program `addusers` tworzący użytkowników na podstawie bazy w pliku `baza.txt` zawierającej:

```
jan;kowalski;haslo1
piotr;nowak;haslo2
grzegorz;adamski;haslo3
```

Użytkowników tworzymy analogicznie jak w zadaniu 6.13. Program ma działać poprawnie dla pliku `baza.txt` zapisanego zarówno w formacie Linux, jak i Windows.

Zad. 6.18

Wejdź na poniższą stronę i obejrzyj program:

`https://dpaste.com/897JG9S9L`

Pobierz program na Linux komendą:

```
wget dpaste.com/897JG9S9L.txt -O hello.c
```

- jakie rozszerzenie podajemy po adresie strony z programem?

- jaka opcja służy do określenia nazwy pobieranego programu?

Wyświetl, skompiluj i uruchom pobrany program.

Zad. 6.19

Umieść na stronie `dpaste.com` program `addusers` i pobierz go na Linux.

Zad. 6.20 *

Napisz program `delusers` usuwający użytkowników podanych na liście w pliku.

Zad. 7.1

Napisz program `check` sprawdzający na jakim systemie został skompilowany. Wykorzystać kompilację warunkową. Przykładowa sesja:

```
Program compiled on: Windows
```

Zad. 7.2

Napisz program `arguments` wypisujący kolejno:

- wartość zmiennej `argc`
- adres programu
- nazwę programu *
- przekazane parametry

Przykładowa sesja:

```
argc = 2

addr: D:\Dev-C\arguments\arguments.exe

name: arguments.exe

params: *.c
```

Zad. 7.3

Napisz program `list` wywołujący komendę `dir` dla systemu Windows lub `ls` dla systemu Linux. Parametry podane przy uruchomieniu programu są przekazywane do obu komend.

Zad. 7.4

Napisz program `cyfry` wyliczający liczbę cyfr danej liczby całkowitej przy pomocy funkcji:

- funkcja `cyfry1` wykorzystuje funkcję `log10`. *

- funkcja `cyfry2` wykorzystuje funkcję `snprintf`.

- funkcja `cyfry3` wykorzystuje dzielenie przez `10`. *

Przykładowa sesja:

```
cyfry1(-3579) = 4
cyfry2(-3579) = 4
cyfry3(-3579) = 4
```

Zad. 7.5 *

Napisz program `version` wypisujący wersję biblioteki `libc` na 3 różne sposoby dla systemu Linux i Windows.

Zad. 7.6 *

Napisz program `dodawanie` realizujący dodawanie pisemne. Interfejs programu musi wyglądać dokładnie tak samo, jak w programie `dodawanie.exe` na stronie autora. Zakładamy, że dane wejściowe mają postać:

```
Z: a = 0, 1, 2, ...
   b = 0, 1, 2, ...
```

- w jakich przypadkach program może dawać niepoprawne wyniki?

Przykładowa sesja:

```
a = 9237
b = 1267

 1 11
  9237
+ 1267
------
 10504
```

Zad. 7.7 *

Dodaj do programu `dodawanie` kontrolę poprawności danych wejściowych i wyniku dla przypadków:

- dane wejściowe pobierane są jako liczby całkowite
- dane wejściowe pobierane są jako łańcuchy znaków

Zad. 8.1 *

Napisz program `regexp` ilustrujący wyrażenia regularne POSIX'a.

Zad. 8.2

Napisz program `misc` implementujący i testujący następujące funkcje:

- funkcja `losuj` zwraca wartość losową z przedziału `[a, b)`

```
int losuj(int a, int b);
```

- funkcja `wariacje1` wylicza metodą kombinatoryczną liczbę ciągów 2-elementowych, jakie można utworzyć ze znaków określonych wyrażeniem regularnym `[a-zA-Z0-9./]`

```
int wariacje1();
```

- funkcja `wariacje2` wylicza metodą brutalnej siły liczbę ciągów 2-elementowych, jakie można utworzyć z wartości określonych wyrażeniem regularnym `[a-zA-Z0-9./]` *

```
int wariacje2();
```

- funkcja `set` w sposób losowy określa wartość parametru `salt` dla funkcji `crypt_r`

```
void set(char salt[2]);
```

```
https://www.man7.org/linux/man-pages/man3/crypt_r.3.html
```

- funkcja `errnoExample` ilustruje wypisanie błędu otwarcia pliku przy pomocy zmiennej globalnej `errno`

```
void errnoExample();
```

- funkcja `perrorExample` ilustruje wypisanie błędu otwarcia pliku przy pomocy funkcji `perror` *

```
void perrorExample();
```

Projekt 2

Projekty należy wykonać w języku C. Interfejs programu powinien być zgodny z przykładową sesją. Dokumentacja projektu powinna zawierać również raport z testów. Wykonanie projektu obejmuje:

- plan aplikacji
- research
- implementację
- testowanie
- dokumentację
- obronę projektu
- poprawę błędów

Deadline: 23.01.2024

Projekty należy wysłać na maila w spakowanym katalogu:

```
Nazwisko Imie
```

```
  nazwa.c
  nazwa.docx  ; dokumentacja i raport z testów
```

W nazwach katalogu i plików nie używamy polskich znaków.

Projekt 1 (3.5)

Napisz program `split`, który podzieli plik podany jako pierwszy parametr na pliki o rozmiarze podanym jako drugi parametr. Można też używać rozmiarów predefiniowanych, np.: `cd`, `dvd`, `blue-ray`. Nazwy plików wynikowych powinny składać się z nazwy pliku wejściowego i odpowiedniego przyrostka, np.:

```
plik_part1
plik_part2
plik_part3
```

Projekt 2 (4.0)

Napisz program `line-end` odnajdujący znak lub sekwencję znaków oznaczającą koniec linii. W dokumentacji opisz dokładnie algorytm odnajdywania końca linii. Program należy przygotować w jednym pliku źródłowym do skompilowania na systemie Linux i Windows. Przykładowa sesja:

```
System Windows

char newline: '\r' '\n'

hex newline: 0D 0A
```

Projekt 3 (4.5)

W programie `jiffies` zaimplementuj moduł jądra, który tworzy plik `/proc/jiffies` podający bieżącą wartość zmiennej `jiffies`, gdy plik ten jest czytany, na przykład za pomocą polecenia `cat /proc/jiffies`. Po usunięciu modułu należy usunąć utworzony plik.

Projekt 4 (4.5)

W programie `seconds` zaimplementuj moduł jądra, który tworzy plik `/proc/sekundy` podający liczbę sekund, które upłynęły od momentu załadowania modułu jądra, gdy plik ten jest czytany, na przykład za pomocą polecenia `cat /proc/seconds`. W programie należy użyć wartości `jiffies` oraz `HZ`. Po usunięciu modułu należy usunąć utworzony plik.

Projekt 5 (5.0)

Napisz w języku Java program `Drzewo`, który rekurencyjnie utworzy drzewo katalogowe zakodowane w tablicy trójwymiarowej. Można użyć komend tworzących plik i katalog oraz komend zmiany katalogu i przejścia do katalogu poziom wyżej. Zakoduj w tablicach trójwymiarowych poniższe drzewa katalogowe i przetestuj program.

Sposób kodowania drzewa w tablicy i funkcja rekurencyjna tworząca drzewo muszą być dokładne opisane w dokumentacji.

Projekt 6 (5.0)

Napisz program `add2path` dla systemu Linux, który do zmiennej systemowej `PATH` w sposób permanentny:

- dodaje bieżący katalog, jeśli go tam nie ma
- usuwa bieżący katalog, jeśli on tam jest
- dodaje katalog przekazany jako parametr, jeśli go tam nie ma
- usuwa katalog przekazany jako parametr, jeśli on tam jest
- wypisuje na terminalu wykonaną operację
- katalogi mogą być przekazywane przez adresy względne i bezwzględne

Projekt 7 (5.0)

Napisz program `add2path` dla systemu Windows, który do zmiennej systemowej `PATH` w sposób permanentny:

- dodaje bieżący katalog, jeśli go tam nie ma
- usuwa bieżący katalog, jeśli on tam jest
- dodaje katalog przekazany jako parametr, jeśli go tam nie ma
- usuwa katalog przekazany jako parametr, jeśli on tam jest
- wypisuje na terminalu wykonaną operację
- katalogi mogą być przekazywane przez adresy względne i bezwzględne

Laboratorium 9

Zad. 9.1

Przepisz program `copy` oraz przeanalizuj jego działanie.

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

#define MAX 512

int main(int argc, char* argv[]) {
    char buf[MAX];
    int desc_zrod, desc_cel;
    int lbajt;
```

```
    if (argc < 3) {
        fprintf(stderr, "Za malo argumentow. Uzyj:\n");
        fprintf(stderr, "%s <plik zrodlowy> <plik docelowy>\n", argv[0]);
        exit(1);
    }

    desc_zrod = open(argv[1], O_RDONLY);

    if (desc_zrod == -1) {
        perror("Blad otwarcia pliku zrodlowego.\n");
        exit(1);
    }

    desc_cel = creat(argv[2], 0640);

    if (desc_cel == -1) {
        perror("Blad utworzenia pliku docelowego.\n");
        exit(1);
    }

    while ((lbajt = read(desc_zrod, buf, MAX)) > 0) {
        if (write(desc_cel, buf, lbajt) == -1) {
            perror("Blad zapisu pliku docelowego.\n");
            exit(1);
        }
    }

    if (lbajt == -1) {
        perror("Blad odczytu pliku zrodlowego.\n");
        exit(1);
    }

    if (close(desc_zrod) == -1 || close(desc_cel) == -1) {
        perror("Blad zamkniecia pliku.\n");
        exit(1);
    }

    exit(0);
}
```

Zad. 9.2

Przepisz program `size` oraz przeanalizuj jego działanie.

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

int main(int argc, char* argv[]) {
    int desc;
    long rozm;

    if (argc < 2) {
        fprintf(stderr, "Za malo argumentow. Uzyj:\n");
        fprintf(stderr, "%s <nazwa pliku>\n", argv[0]);
        exit(1);
    }
```

```
    desc = open(argv[1], O_RDONLY);

    if (desc == -1) {
        perror("Blad otwarcia pliku");
        exit(1);
    }

    rozm = lseek(desc, 0, SEEK_END);

    if (rozm == -1) {
        perror("Blad w pozycjonowaniu");
        exit(1);
    }

    printf("Rozmiar pliku %s: %ld\n", argv[1], rozm);

    if (close(desc) == -1) {
        perror("Blad zamkniecia pliku");
        exit(1);
    }

    exit(0);
}
```

Zad. 9.3 *

Napisz program kopiujący zawartość pliku o nazwie podanej jako pierwszy parametr, do pliku którego nazwa podana jest jako drugi parametr.

Zad. 9.4 *

Napisz program zmieniający kolejność znaków w każdej linii pliku o nazwie podanej jako parametr.

Zad. 9.5 *

Napisz procedurę kopiowania ostatnich 10 znaków, słów i ostatnich 10 linii jednego pliku do innego.

Zad. 9.6 *

Napisz program do rozpoznawania czy plik o podanej nazwie jest plikiem tekstowym, plik tekstowy zawiera znaki o kodach 0-127, można w tym celu użyć funkcji `isascii`.

Zad. 9.7 *

Napisz program, który w pliku o nazwie podanej jako ostatni argument zapisze połączoną zawartość wszystkich plików, których nazwy zostały podane w linii poleceń przed ostatnim argumentem.

Zad. 9.8 *

Napisz program do porównywania plików o nazwach przekazanych jako argumenty. Wynikiem działania programu ma być komunikat, że pliki są identyczne, pliki różnią się od znaku nr <nr znaku> w linii <nr znaku linii> lub - gdy jeden z plików zawiera treść drugiego uzupełnioną o jakieś dodatkowe znaki - plik <nazwa> zawiera <liczba> znaków więcej niż zawartość pliku <nazwa>.

Zad. 10.1

Przepisz program `fork` oraz przeanalizuj jego działanie.

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

int main() {
    printf("Poczatek\n");

    fork();

    printf("Koniec\n");

    return 0;
}
```

- dlaczego napis `Poczatek` wyświetla się tylko raz?
- dlaczego napis `Koniec` wyświetla się dwa razy?

Zad. 10.2

Przepisz program `fork2` oraz przeanalizuj jego działanie.

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

int main() {
    printf("Poczatek\n");

    pid_t pid = fork();

    if (pid < 0) fprintf(stderr, "Blad tworzenia procesu potomnego.\n");

    if (pid == 0) printf("Proces dziecko, zmienna pid = %d\n", pid);

    if (pid > 0) printf("Proces rodzic, zmienna pid = %d\n", pid);

    printf("Koniec\n");

    return 0;
}
```

- jaką wartość przechowuje zmienna `pid` dla procesu potomnego?
- jaką wartość przechowuje zmienna `pid` dla procesu macierzystego?

Zad. 10.3

Napisz program `fork3` na podstawie `fork2` w taki sposób, aby proces rodzic oczekiwał na zakończenie procesu dziecko.

Zad. 10.4

Napisz program `fork4` na podstawie `fork3` w taki sposób, aby proces rodzic i proces dziecko dodatkowo wyświetlał swój pid przy pomocy funkcji `getpid()`.

Zad. 10.5 *

Napisz program `fork5` w którym proces dziecko wyświetla własny `pid` oraz `pid` rodzica.

Zad. 10.6

Przepisz program `execlp` oraz przeanalizuj jego działanie.

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

int main() {
    printf("Poczatek\n");

    execlp("ls", "ls", "-l", NULL);

    printf("Koniec\n");

    return 0;
}
```

- dlaczego napisz `Koniec` się nie wyświetla?

Zad. 10.7

Przepisz program `execlp2` oraz przeanalizuj jego działanie.

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

int main() {
    printf("Poczatek\n");

    if (fork() == 0) {
       execlp("ls", "ls", "-a", NULL);

       perror("Blad uruchmienia programu.");

       exit(1);
    }
```

```
    wait(NULL);

    printf("Koniec\n");

    return 0;
}
```

- dlaczego tym razem napis `Koniec` się wyświetla?

Zad. 10.8 *

Przepisz program `execlp3` oraz przeanalizuj jego działanie.

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

int main() {
    printf("Poczatek\n");

    pid_t pid = fork();

    if (pid == -1) {
        perror("Blad tworzenia procesu potomnego.");

        return 1;
    }

    if (pid == 0) {
        execlp("ls", "ls", "-l", NULL);

        perror("Blad uruchmienia programu.");

        exit(1);
    }

    if (wait(NULL) == -1)
        perror("Blad w oczekiwaniu na zakonczenie potomka.");

    printf("Koniec\n");

    return 0;
}
```

- opisz na podstawie dokumentacji `man` wartości zwracane przez funkcję `fork()` oraz funkcję `wait()`.

Zad. 10.9

Przepisz program `sierota` oraz przeanalizuj jego działanie.

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

int main() {
```

```
    if (fork() == 0) {
        printf("child pid = %d\n", getpid());

        sleep(20);

        exit(0);
    }

    exit(0);
}
```

- sprawdź stan programu `sierota` na liście procesów

Zad. 10.10

Przepisz program `zombie` oraz przeanalizuj jego działanie.

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

int main() {
    if (fork() == 0) {
        printf("child pid = %d\n", getpid());
        exit(0);
    }

    sleep(20);

    wait(NULL);

    exit(0);
}
```

- sprawdź stan programu `zombie` na liście procesów

Zad. 10.11

Przepisz program `status` oraz przeanalizuj jego działanie.

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

int main() {
    pid_t pid1 = fork();

    if (pid1 == 0) exit(7); // proces dziecko

    printf("pid rodzica = %d\n", getpid());
    printf("pid dziecka = %d\n\n", pid1);

    int status;
```

```
    pid_t pid2 = wait(&status);

    printf("Proces: %d\n", pid2);
    printf("Status: %x\n", status);
}
```

Zad. 10.12

Przepisz program `status2` oraz przeanalizuj jego działanie.

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

int main() {
    pid_t pid1 = fork();

    if (pid1 == 0) { // proces dziecko
        sleep(10);
        exit(7);
     }

    printf("pid rodzica = %d\n", getpid());
    printf("pid dziecka = %d\n\n", pid1);

    kill(pid1, 9);

    int status;

    pid_t pid2 = wait(&status);

    printf("Proces: %d\n", pid2);
    printf("Status: %x\n", status);
}
```

Zad. 10.13

Przepisz program `redirect` oraz przeanalizuj jego działanie.

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char* argv[]) {
    close(1);

    creat("ls.txt", 0600);

    execvp("ls", argv);
}
```

- opisz na podstawie dokumentacji `man` działanie funkcji `creat` *

- w jaki sposób wynik programu `ls` przekierowywany jest do pliku `ls.txt` ?

https://www.computerhope.com/jargon/f/file-descriptor.htm

Zad. 10.14 *

Napisz program tworzący dwa procesy. Każdy ze stworzonych procesów powinien utworzyć proces - potomka. Należy wyświetlać identyfikatory procesów rodziców i potomków po każdym wywołaniu funkcji `fork`.

Zad. 10.15 *

Napisz program tworzący równocześnie trzy procesy zombi.

Zad. 10.16 *

Napisz program, którego rezultatem będzie wydruk zawartości bieżącego katalogu poprzedzony napisem `Poczatek` a zakończony napisem `Koniec`.

Laboratorium 11

Zad. 11.1

Przepisz program `pipe` oraz przeanalizuj jego działanie.

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

int main() {
    int pdesk[2];

    if (pipe(pdesk) == -1) {
        perror("Blad tworzenia potoku.\n");
        exit(1);
    }

    pid_t pid = fork();

    if (pid == -1) {
        perror("Blad tworzenia procesu potomnego.\n");
        exit(1);
    }

    if (pid == 0) {
        if (write(pdesk[1], "Hallo!", 7) == -1) {
            perror("Blad zapisu do potoku.\n");
            exit(1);
        }

        exit(0);
    }

    if (pid > 0) {
        char buf[10];
```

```
        if (read(pdesk[0], buf, 10) == -1) {
            perror("Blad odczytu z potoku.\n");
            exit(1);
        }

        printf("Odczytano z potoku: %s\n", buf);
    }

    return 0;
}
```

Zad. 11.2

Przepisz program `pipe2` oraz przeanalizuj jego działanie.

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

int main() {
    int pdesk[2];

    pipe(pdesk);

    pid_t pid = fork();

    if (pid == 0)
        write(pdesk[1], "Hallo!", 7);

    if (pid > 0) {
        char buf[10];

        read(pdesk[0], buf, 10);
        read(pdesk[0], buf, 10);

        printf("Odczytano z potoku: %s\n", buf);
    }

    return 0;
}
```

- dlaczego program się blokuje?

Zad. 11.3

Przepisz program `pipe3` oraz przeanalizuj jego działanie.

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

#define MAX 512

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

```
    if (pipe(pdesk) == -1) {
        perror("Blad tworzenia potoku.\n");
        exit(1);
    }

    pid_t pid = fork();

    if (pid == -1) {
        perror("Blad tworzenia procesu potomnego.\n");
        exit(1);
    }

    if (pid == 0) {
        dup2(pdesk[1], 1);

        execvp("ls", argv);

        perror("Blad uruchomienie programu ls.\n");
        exit(1);
    }

    if (pid > 0) {
        close(pdesk[1]);

        char buf[MAX];
        int lb, i;

        while ((lb = read(pdesk[0], buf, MAX)) > 0) {
            for (i = 0; i < lb; i++)
                buf[i] = toupper(buf[i]);

            if (write(1, buf, lb) == -1) {
                perror("Blad zapisu na standardowe wyjscie.\n");
                exit(1);
            }
        }

        if (lb == -1) {
            perror("Blad odczytu z potoku.\n");
            exit(1);
        }
    }

    return 0;
}
```

Zad. 11.4

Przepisz program `pipe4` oraz przeanalizuj jego działanie.

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

#define MAX 512

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

```
    if (pipe(pdesk) == -1) {
        perror("Blad tworzenia potoku ");
        exit(1);
    }

    pid_t pid = fork();

    if (pid == -1) {
        perror("Blad tworzenia procesu potomnego ");
        exit(1);
    }

    if (pid == 0) {
        dup2(pdesk[1], 1);

        execvp("ls", argv);

        perror("Blad uruchomiania programu ls ");
        exit(1);
    }

    if (pid > 0) {
        close(pdesk[1]);

        dup2(pdesk[0], 0);

        execlp("tr", "tr", "a-z", "A-Z", NULL);

        perror("Blad uruchomiania programu tr ");
    }

    return 0;
}
```

Zad. 11.5

Przepisz program `fifo` oraz przeanalizuj jego działanie.

```
#include <fcntl.h>
#include <sys/stat.h>

int main() {
    mkfifo("kolFIFO", 0600);

    open("kolFIFO", O_RDONLY);

    return 0;
}
```

- dlaczego program się blokuje?

Zad. 11.6

Przepisz program `fifo2` oraz przeanalizuj jego działanie.

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
```

```
#include <sys/stat.h>
#include <unistd.h>

int main() {
    int pdesk;

    if (mkfifo("/tmp/fifo", 0600) == -1) {
        perror("Blad tworzenia kolejki FIFO.\n");
        exit(1);
    }

    pid_t pid = fork();

    if (pid == -1) {
        perror("Blad tworzenia procesu potomnego.\n");
        exit(1);
    }

    if (pid == 0) {
        pdesk = open("/tmp/fifo", O_WRONLY);

        if (pdesk == -1){
            perror("Blad otwarcia potoku do zapisu.\n");
            exit(1);
        }

        if (write(pdesk, "Hallo!", 7) == -1){
            perror("Blad zapisu do potoku.\n");
            exit(1);
        }

        exit(0);
    }

    if (pid > 0) {
        pdesk = open("/tmp/fifo", O_RDONLY);

        if (pdesk == -1){
            perror("Blad otwarcia potoku do odczytu.\n");
            exit(1);
        }

        char buf[10];

        if (read(pdesk, buf, 10) == -1){
            perror("Blad odczytu z potoku.\n");
            exit(1);
        }

        printf("Odczytano z potoku: %s\n", buf);
    }

    return 0;
}
```

Zad. 11.7

Przepisz program `fifo3` oraz przeanalizuj jego działanie.

```
#include <stdio.h>
```

```
#include <stdlib.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <unistd.h>

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

    if (mkfifo("/tmp/fifo", 0600) == -1) {
        perror("Blad tworzenia kolejki FIFO.\n");
        exit(1);
    }

    pid_t pid = fork();

    if (pid == -1) {
        perror("Blad tworzenia procesu potomnego.\n");
        exit(1);
    }

    if (pid == 0) {
        close(1);

        pdesk = open("/tmp/fifo", O_WRONLY);

        if (pdesk == -1){
            perror("Blad otwarcia potoku do zapisu.\n");
            exit(1);
        }

        if (pdesk != 1){
            perror("Niewlasciwy deskryptor do zapisu.\n");
            exit(1);
        }

        execvp("ls", argv);

        perror("Blad uruchmienia programu.\n");
        exit(1);
    }

    if (pid > 0) {
        close(0);

        pdesk = open("/tmp/fifo", O_RDONLY);

        if (pdesk == -1){
            perror("Blad otwarcia potoku do odczytu.\n");
            exit(1);
        }

        if (pdesk != 0){
            perror("Niewlasciwy deskryptor do odczytu.\n");
            exit(1);
        }

        execlp("tr", "tr", "a-z", "A-Z", NULL);

        perror("Blad uruchomienia programu tr.\n");
    }
```

```
    return 0;
}
```

Zad. 11.8

Przepisz program `pipe3err` oraz przeanalizuj jego działanie.

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

#define MAX 512

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

    if (pipe(pdesk) == -1) {
        perror("Blad tworzenia potoku.\n");
        exit(1);
    }

    pid_t pid = fork();

    if (pid == -1) {
        perror("Blad tworzenia procesu potomnego.\n");
        exit(1);
    }

    if (pid == 0) {
        dup2(pdesk[1], 1);

        execvp("ls", argv);

        perror("Blad uruchomienie programu ls.\n");
        exit(1);
    }

    if (pid > 0) {
        close(pdesk[1]);

        wait(NULL);

        char buf[MAX];
        int lb, i;

        while ((lb = read(pdesk[0], buf, MAX)) > 0) {
            for (i = 0; i < lb; i++)
                buf[i] = toupper(buf[i]);

            if (write(1, buf, lb) == -1) {
                perror("Blad zapisu na standardowe wyjscie.\n");
                exit(1);
            }
        }

        if (lb == -1) {
            perror("Blad odczytu z potoku.\n");
            exit(1);
```

```
        }
    }

    return 0;
}
```

- w jakim przypadku może dojść do zakleszczenia programu?

Zad. 11.9

Przepisz program `fifo3err` oraz przeanalizuj jego działanie.

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

#define MAX 512

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

    if (mkfifo("/tmp/fifo", 0600) == -1) {
        perror("Blad tworzenia kolejki FIFO.\n");
        exit(1);
    }

    pid_t pid = fork();

    if (pid == -1) {
        perror("Blad tworzenia procesu potomnego.\n");
        exit(1);
    }

    if (pid == 0) {
        close(1);

        pdesk = open("/tmp/fifo", O_WRONLY);

        if (pdesk == -1){
            perror("Blad otwarcia potoku do zapisu.\n");
            exit(1);
        }

        if (pdesk != 1){
            perror("Niewlasciwy deskryptor do zapisu.\n");
            exit(1);
        }

        execvp("ls", argv);

        perror("Blad uruchmienia programu.\n");
        exit(1);
    }

    if (pid > 0) {
```

```
        wait(NULL);

        pdesk = open("/tmp/fifo", O_RDONLY);

        char buf[MAX];
        int lb, i;

        while ((lb = read(pdesk, buf, MAX)) > 0) {
            for (i = 0; i < lb; i++)
                buf[i] = toupper(buf[i]);

            if (write(1, buf, lb) == -1) {
                perror("Blad zapisu na standardowe wyjscie.\n");
                exit(1);
            }
        }

        if (lb == -1) {
            perror("Blad odczytu z potoku.\n");
            exit(1);
        }
    }

    return 0;
}
```

- dlaczego program się zakleszcza?

Zad. 11.10 *

Napisz program który tworzy trzy procesy - proces macierzysty i jego dwa procesy potomne. Pierwszy z procesów potomnych powinien zapisać do potoku napis „hello!", a drugi proces potomny powinien ten napis odczytać.

Zad. 11.11 *

Napisz program który tworzy trzy procesy, z których dwa zapisują do potoku, a trzeci odczytuje z niego i drukuje otrzymane komunikaty.

Zad. 11.12 *

Napisz programy realizujące następujące potoki:

```
Ls | wc
finger | cut -d' ' -f1
ls -l | grep ^d | more
ps -ef| tr -s ' '  :| cut -d: -f1 |sort| uniq -c |sort n
cat /etc/group | head -5 > grupy.txt
```

Zad. 11.13 *

Napisz program tworzący dwa procesy: klienta i serwera. Serwer tworzy ogólnodostępną kolejkę FIFO, i czeka na zgłoszenia klientów. Każdy klient tworzy własną kolejkę, poprzez którą będą przychodzić odpowiedzi serwera. Zadaniem klienta jest przesłanie nazwy stworzonej

przez siebie kolejki, a serwera odesłaniem poprzez kolejkę stworzoną przez klienta wyniku polecenia `ls`.

Zad. 11.14 *

Zmodyfikować poprzedni program, tak, by kolejka utworzona przez klienta była dwukierunkowa, klient publiczną kolejką powinien przesyłać nazwę stworzonej przez siebie kolejki. Dalsza wymiana komunikatów powinna odbywać się poprzez kolejkę stworzoną przez klienta. Klient kolejką tą powinien wysyłać polecenia, zadaniem serwera jest wykonywanie tych poleceń i odsyłanie wyników.

Zad. 1

Przepisz program `writer` tworzący kolejkę komunikatów, przeanalizuj jego działanie oraz prześlij do kolejki dwa dowolne komunikaty.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/msg.h>

#define N 80

struct msgbuf {
    long mtype;
    char mvalue[N];
};

int main() {
    int msgid = msgget(45281, IPC_CREAT | 0600);

    struct msgbuf msg;

    printf("Message to be sent: ");

    fgets(msg.mvalue, N, stdin);

    msg.mtype = 1;

    msgsnd(msgid, &msg, sizeof(msg.mvalue), 0);

    system("ipcs -q");

    return 0;
}
```

- w jaki sposób tworzymy kolejkę komunikatów?

- jaki typ ma wysyłany komunikat?

- jaki maksymalny rozmiar może mieć wysyłany komunikat?

- jaka komenda podaje informacje o kolejkach komunikatów?

- jaki klucz ma utworzona kolejka komunikatów?

- jaki identyfikator ma utworzona kolejka komunikatów?

- ile komunikatów znajduje się w kolejce?

Zad. 2

Przepisz program `reader` oraz przeanalizuj jego działanie. Odbierz komunikaty przesłane do kolejki i za każdym razem sprawdzaj status kolejki.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/msg.h>

#define N 80

struct msgbuf {
    long mtype;
    char mvalue[N];
};

int main() {
    int msgid = msgget(45281, IPC_CREAT | 0600);

    struct msgbuf msg;

    msgrcv(msgid, &msg, sizeof(msg.mvalue), 1, 0);

    printf("Message received: %s\n", msg.mvalue);

    system("ipcs -q");

    //msgctl(msgid, IPC_RMID, NULL);

    return 0;
}
```

- jaki typ ma odbierany komunikat?

- ile komunikatów znajduje się w kolejce?

Zad. 3

Przepisz program `ipc` oraz przeanalizuj jego działanie.

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

#define N 80
```

```
struct msgbuf {
    long mtype;
    char mvalue[N];
};

int main() {
    int msgid = msgget(IPC_PRIVATE, IPC_CREAT | 0600);

    if (msgid == -1) {
        perror("Utworzenia kolejki komunikatow.\n");

        exit(1);
    }

    pid_t pid = fork();

    if (pid == 0) {
        struct msgbuf msg;

        msg.mtype = 1;
        strcpy(msg.mvalue, "Message from the child.");

        sleep(5);

        if (msgsnd(msgid, &msg, sizeof(msg.mvalue), 0) == -1) {
            perror("Wysylania komunikatu.\n");

            exit(1);
        }
    }

    if (pid > 0) {
        struct msgbuf msg;

        if (msgrcv(msgid, &msg, sizeof(msg.mvalue), 1, 0) == -1) {
            perror("Odbierania komunikatu.\n");

            exit(1);
        }

        printf("Parent received: %s\n", msg.mvalue);

        if (msgctl(msgid, IPC_RMID, NULL) == -1) {
            perror("Usuwania kolejki.\n");

            exit(1);
        }
    }

    exit(0);
}
```

- co oznacza wartość `IPC_PRIVATE`?

- czy procesy rodzic i dziecko wymagają synchronizacji?

- w jaki sposób usuwamy kolejkę komunikatów?

Zad. 4

Przepisz program `mwrite` tworzący segment pamięci współdzielonej oraz przeanalizuj jego działanie.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/shm.h>

#define MAX 5

int main() {
    int shmid = shmget(45281, MAX*sizeof(int), IPC_CREAT | 0600);

    if (shmid == -1) {
        perror("Utworzenia segmentu pamieci wspoldzielonej.\n");

        exit(1);
    }

    int *mem = (int*)shmat(shmid, NULL, 0);

    if (mem == (void*)-1) {
        perror("Przylaczenia segmentu pamieci wspoldzielonej.\n");

        exit(1);
    }

    printf("mwrite : mem = %p\n\n", mem);

    int i;
    for (i = 0; i < 2*MAX; i++) mem[i % MAX] = i;

    printf("Data written to shared memory.\n");

    shmdt(mem);

    return 0;
}
```

- w jaki sposób tworzymy segment pamięci współdzielonej?

- w jaki sposób dowiązujemy segment pamięci współdzielonej do procesu?

- jaki jest adres dowiązania segmentu pamięci współdzielonej dla procesu `mwrite` ?

- w jaki sposób rozpoznajemy błąd dowiązania segmentu pamięci współdzielonej?

- w jaki sposób odłączamy segment pamięci współdzielonej?

- jaki problem występuje przy zapisie do bufora cyklicznego?

Zad. 5

Przepisz program `mread` oraz przeanalizuj jego działanie.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/shm.h>

#define MAX 5

int main() {
    int shmid = shmget(45281, MAX*sizeof(int), IPC_CREAT | 0600);

    if (shmid == -1) {
        perror("Utworzenia segmentu pamieci wspoldzielonej.\n");

        exit(1);
    }

    int *mem = (int*)shmat(shmid, NULL, 0);

    if (mem == (void*)-1) {
        perror("Przylaczenia segmentu pamieci wspoldzielonej.\n");

        exit(1);
    }

    printf("mread : mem = %p\n\n", mem);

    printf("Data received:");

    int i;
    for (i = 0; i < MAX; i++) printf(" %d", mem[i]);

    printf("\n");

    shmdt(mem);

    shmctl(shmid, IPC_RMID, (struct shmid_ds*)0);

    system("ipcs -m");

    return 0;
}
```

- jaki jest adres dowiązania segmentu pamięci współdzielonej dla procesu `mread` ?

- co możemy powiedzieć o adresach logicznych segmentów shm dla obu programów?

- co możemy powiedzieć o adresach fizycznych segmentów shm dla obu programów?

- jaka komenda podaje informacje o segmentach pamięci współdzielonej?

- w jaki sposób usuwamy segment pamięci współdzielonej?

- w jakim przypadku proces `mread` odczyta same zera?

Zad. 6

Przepisz program `shm` oraz przeanalizuj jego działanie.

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

#define N 20000
#define MAX 4

int main() {
    int shmid = shmget(IPC_PRIVATE, MAX*sizeof(int), IPC_CREAT | 0600);

    if (shmid == -1) {
        perror("Utworzenia segmentu pamieci wspoldzielonej.\n");

        exit(1);
    }

    pid_t pid = fork();

    if (pid == 0) {
        int *mem = (int*)shmat(shmid, NULL, 0);

        int i;
        for (i = 0; i < N; i++) mem[i % MAX] = i;

        printf("Child finished writing to shared memory.\n");

        shmdt(mem);
    }

    if (pid > 0) {
        int *mem = (int*)shmat(shmid, NULL, 0);

        printf("Parent read: ");

        int i;
        for (i = 0; i < MAX; i++) printf("%d ", mem[i]);

        printf("\n");

        shmdt(mem);

        shmctl(shmid, IPC_RMID, (struct shmid_ds*)0);
    }

    exit(0);
}
```

- uruchom program kilka razy i omów wyniki

- w jakim przypadku proces `shm` odczyta same zera?

- czy procesy rodzic i dziecko wymagają synchronizacji?

Zad. 7

Napisz program `shm2` eliminujący błędy programu `shm`.

Laboratorium 13

Zad. 1

Przepisz program `thread` oraz przeanalizuj jego działanie.

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

void *hello(void *arg) {
    int i;
    for (i = 0; i < 5; i++) {
        printf("watek, i = %d\n", i);

        sleep(1);
    }

    return NULL;
}

int main() {
    pthread_t mythread;

    if (pthread_create(&mythread, NULL, hello, NULL)) {
       printf("Blad przy tworzeniu watku.\n");
       abort();
    }

    if (pthread_join( mythread, NULL)) {
        printf("Blad przy konczeniu watku.\n");
        exit(1);
    }

    exit(0);
}
```

Suplement

Zad.

Napisz program `scheduler.c` ilustrujący różne algorytmy szeregowania procesów. Program implementuje funkcje:

```
void burstTime(struct Process p[], int size);
```

- wypisuje czas wykonania procesów

```
void waitingTime(struct Process p[], int size);
```

- wypisuje czas oczekiwania procesów na przydział procesora

```
double avgWaitingTime(struct Process p[], int size);
```

- oblicza średni czas oczekiwania procesów na przydział procesora

```
void ganttChart(struct Process p[], int size);
```

- wypisuje diagram Gantta *

```
void applySJF(struct Process p[], int size);
```

- szereguje procesy według algorytmu SJF (ang. Shortest Job First)

Metody