Примерни задачи за курс “Операционни системи”, СУ, ФМИ

1 януари 1980 г.

Семинарните упражнения на курса “Операционни системи” разглеждат следните теми, групирани по “Работа в GNU/Linux shell” и “Използване на системни примитиви в програми на C”:

1. Въведение (shell)
2. Файлова система и работа с файлове (shell)
3. Обработка на текст (shell)
4. Процеси (shell)
5. Командни интерпретатори и скриптове (shell)
6. Системни примитиви за вход и изход (C)
7. Системни примитиви за работа с процеси (C)
8. Системни примитиви за работа с pipe-ове (C)

Тук са събрани примерни практически задачи по различните теми, както и някои теоретични въпроси, с надеждата те да бъдат полезни на студентите при тяхната работа в курса. За практическите задачи (shell и C) се очаква студентите да са разгледали някои по-базови задачи по дадена тема преди да преминат към изложените тук.

Съдържание

1 Работа в GNU/Linux shell

Забележка: За всички задачи, освен ако не е указано друго, в имената на файловете и директориите няма специални символи. Във файловата система може да съществуват директории, до които нямате достъп.

1.1 Поточна обработка (теми 1,2,3)

Зад. 1 2016-SE-01 Даден е текстов файл с име `philip-j-fry.txt`. Напишете shell script и/или серия от команди, които извеждат броя редове, съдържащи поне една четна цифра и несъдържащи малка латинска буква от `a` до `w`.

Примерно съдържание на файла:

```
123abv123
123zz123
MMU_2.4
```

Примерен изход:

```
Броят на търсените редовете е 2
```

Зад. 2 2017-IN-01 Напишете серия от команди, извеждащи на екрана само броя на всички обекти във файловата система, чиито собственик е текущият потребител.

Забележка: Във файловата система със сигурност съществуват директории, до които нямате достъп.

Зад. 3 2017-IN-02 Напишете серия от команди, които изтриват:

а) всички файлове в текущата директория и нейните поддиректории, които са с нулева дължина.
б) 5-е най-големи файла в home директорията на текущия потребител и нейните поддиректории.

Зад. 4 2017-IN-03 Напишете серия от команди, които от файла `/etc/passwd` да вземат под-низ, състоящ се от втора и трета цифра на факултетния номер на студентите от специалност Информатика, чиито фамилии завършват на "а". Изведете коя комбинация от цифри се среща най-често и коя е тя.

Примерно съдържание на файла:

```
s45194:x:1255:502:Elizabet Mihaylova, Inf, k3, g1:/home/Inf/s45194:/bin/bash
s45139:x:1261:502:Vasilena Peycheva:/home/Inf/s45139:/bin/bash
s81257:x:1079:503:Vasilena Nikolova, KN, 2kurs, 5gr:/home/KN/s81257:/bin/bash
s81374:x:1117:503:Ivan Kamburov, KN, 2kurs, 7gr:/home/KN/s81374:/bin/bash
kiril:x:508:500:Kiril Varadinov:/home/kiril:/bin/bash
s61812:x:1128:504:Vladimir Genchev:/home/SI/s61812:/bin/bash
user:x:1000:99:Inactive user just to start UID from 1000:/home/user:/sbin/nologin
s81254:x:1077:503:Mariela Tihova, KN, 2kurs, 5gr:/home/KN/s81254:/bin/bash
s81386:x:1121:503:Daniela Ruseva, KN, 2kurs, 7gr:/home/KN/s81386:/bin/bash
s45216:x:1235:502:Aleksandar Yavashev, Inf, k3, g3:/home/Inf/s45216:/bin/bash
```

Примерен изход:

```
2 51
```

Зад. 5 2017-SE-01 Намерете имената на топ 5 файловете в текущата директория с най-много hardlinks.

Зад. 6 2018-SE-01 Променете правата на всички директории, намиращи се някъде *във* вашата home директория, така че да станат такива, каквито биха се получили, ако ги бяхте създали с маска `0022`.

Зад. 7 2018-SE-02 Напишете серия от команди, извеждащи на екрана *само* inode-а на най-скоро променения (по съдържание) файл, намиращ се в home директорията на потребител pesho (или нейните поддиректории), който има повече от едно име.

Зад. 8 2018-SE-03 При подреждане в нарастващ ред на числовите потребителски идентификатори (`UID`) на акаунтите, дефинирани в системата, 201-ят акаунт е от групата, запазена за акаунти от специалност СИ.

Изведете списък с имената (име и фамилия) и home директориите на всички акаунти от специалност СИ, подреден по факултетен номер.

За справка:

```
s61988:x:1219:504:Stoian Genchev,SI,2,5:/home/SI/s61988:/bin/bash
s81430:x:1234:503:Iordan Petkov, KN, k2, g7:/home/KN/s81430:/bin/bash
s61807:x:1248:504:Elica Venchova:/home/SI/s61807:/bin/bash
s62009:x:1254:504:Denitsa Dobreva, 2, 6:/home/SI/s62009:/bin/bash
s61756:x:1258:504:Katrin Kartuleva, SI, 4, 1:/home/SI/s61756:/bin/bash
s855287:x:1195:504:Vaska Kichukova,SI,2,5:/home/SI/s855287:/bin/bash
```

Примерен изход:

```
Katrin Kartuleva:/home/SI/s61756
Elica Venchova:/home/SI/s61807
Stoian Genchev:/home/SI/s61988
Denitsa Dobreva:/home/SI/s62009
Vaska Kichukova:/home/SI/s855287
```

Зад. 9 2019-SE-01 Даден е текстовият файл `planets.txt`, който съдържа информация за гравитационно закръглените обекти в дадена слънчева система. На всеки ред има информация за точно един обект в следните колони, разделени с ';':

- име на обекта
- тип на обекта (един знак)
  - `T` - земен тип
  - `G` - газов гигант
  - `I` - леден гигант
- средно разстояние на обекта до локалната звезда
- маса на обекта (относителна величина)
- обем на обекта (относителна величина)
- плътност (g/cm^3)
- средна орбитална скорост (km/s)

Първият ред във файла е header, който описва имената на колоните.

Данните за обектите не са сортирани.

Намерете и изведете разделени с таб името и масата на обекта, който е **едновременно**:

- най-близкият до локалната звезда
- от същия тип като типа на най-далечният до локалната звезда обект

Примерен входен файл:

```
name;type;distance;mass;volume;density;speed
earth;T;1.00000011;1;1;5.52;29.7859
mars;T;1.52366231;0.107;0.151;3.94;24.1309
saturn;G;9.53707032;95;763.62;0.7;9.6724
mercury;T;0.38709893;0.055;0.056;5.43;47.8725
venus;T;0.72333199;0.815;0.857;5.24;35.0214
jupiter;G;5.20336301;318;1321.3;1.33;13.0697
neptune;I;30.06896348;17;57.747;1.76;5.4778
uranus;I;19.19126393;14.5;63.102;1.3;6.8352
```

Зад. 10 2019-SE-02 Вие сте асистент по ОС. На първото упражнение казвате на студентите да си напишат данните на лист, взимате го и им правите акаунти. След упражнението обаче, забравяте да вземете

листа със себе си - сещате се половин час по-късно, когато трябва да въведете имената на студентите в таблица, но за зла беда в стаята вече няма ни помен от листа (вероятно иззет от спешния отряд на GDPR-полицията)

Сещате се, че в началото на упражнението UNIX-часовникът е показвал 1551168000, а в края 1551176100.

Напишете команда, която изкарва разделени с таб факултетните номера и имената на потребителите от специалност СИ, чиито home директории са променили статуса си (status change time) в зададения времеви интервал.

Приемете, че всички потребители от СИ имат home директории под /home/SI.

Примерен изход:

```
62198   Ivaylo Georgiev
62126   Victoria Georgieva
62009   Denitsa Dobreva
62208   Trayana Nedelcheva
```

Няколко реда от `/etc/passwd` *за справка:*

```
s62136:x:1302:503:Alexander Ignatov, SI, 2, 2:/home/KN/s62136:/bin/bash
s62171:x:1031:504:Deivid Metanov:/home/SI/s62171:/bin/bash
s62126:x:1016:504:Victoria Georgieva:/home/SI/s62126:/bin/bash
s62009:x:1170:504:Denitsa Dobreva,SI,3,3:/home/SI/s62009:/bin/bash
s62196:x:1221:504:Elena Tuparova,SI,2,1:/home/SI/s62196:/bin/bash
```

Зад. 11 2019-SE-03 От всички файлове в home директорията на потребителя velin, изведете дълбочината на файл, който:

- има същия inode като този на най-скоро променения файл сред тях
- има минимална дълбочина

Пояснение Под ”дълбочина”да се разбира дълбочина в дървото на файловата система: например файлът `/foo/bar/baz` има дълбочина 3.

Зад. 12 2020-SE-01 За всички файлове, които (едновременно):

- се намират някъде *във* вашата home директория;
- имат права, каквито биха имали, ако ги бяхте създали с маска `0022`

променете правата им така, че group owner-а на файла да има право да пише в него.

Зад. 13 2020-SE-02 Даден е текстовият файл `spacex.txt`, който съдържа информация за полети на ракети на SpaceX. На всеки ред има информация за един такъв полет в следните колони, разделени с вертикална черта ’|’:

- дата на полета в UNIX формат
- космодрум
- успешност на полета (две възможни стойности)
  - `Success` - успешен полет
  - `Failure` - неуспешен полет
- полезен товар

Първият ред във файла е header, който описва имената на колоните. Данните във файла не са сортирани.

Намерете и изведете разделени с двоеточие (’:’) успешността и информацията за полезния товар на най-скорощния полет, който е изстрелян от космодрума с най-много неуспешни полети.

Примерен входен файл:

```
date|launch site|outcome|payload
1291759200|CCAFS|Success|Dragon demo flight and cheese
1435438800|CCAFS|Failure|SpaceX CRS-7
1275666300|CCAFS|Success|Dragon Spacecraft Qualification Unit
```

```
1452981600|VAFB|Success|Jason-3
1498165200|KSC|Success|BulgariaSat-1
1473454800|CCAFS|Failure|Amos-6
1517868000|KSC|Success|Elon Musk's Tesla
1405285200|CCAFS|Success|Orbcomm
```

Зад. 14 2022-CE-01

Напишете серия от команди, които намират всички обикновени файлове, отговарящи едновременно на следните условия:

- намират се в home директорията на текущия потребител, но не и в нейните под-директории;
- собственик им е текущият потребител.

Серията от команди трябва да променя правата на намерените файлове така, че да станат такива, каквито биха се получили, ако при създаването на файловете ефективната маска е била 0002.

Гарантирайте, че серията от команди няма изход.

Зад. 15 2023-CE-01

Напишете серия от команди, която извежда пътищата до всички файлове (из цялата файлова система), чиито собственик е текущият потребител и имат име, завършващо на `.blend<число>`.

Примерен изход:

```
/home/students/ivan/scenes/landscape.blend1
/home/students/ivan/scenes/landscape.blend26
/home/students/ivan/scenes/landscape.blend42
/home/students/ivan/.local/share/Trash/lewd.blend1
/tmp/baba.blend5
```

Зад. 16 2023-CE-02

Разполагате с директория `/var/log/my_logs`, в която се намират log файлове. Log файл представлява обикновен файл, чието име има следния вид:

```
<server_name>_<unix_timestamp>.log
```

Името на сървър (`server_name`) може да съдържа букви, цифри и долни черти. Unix timestamp е число, означаващо брой изминали секунди след полунощ на първи януари 1970.

Считаме, че файловете в тази директория, чиито имена нямат този формат, **не са** log файлове.

Пример за име:

```
correlator_1692117813.log
```

Напишете серия от команди, намираща общия брой на срещанията на думата error във всички log файлове.

1.2 Скриптове (теми 1,2,3,4,5)

Зад. 17 2016-SE-01 Напишете shell скрипт, който по подаден един позиционен параметър, ако този параметър е директория, намира всички symlink-ове в нея и под-директориите ѝ с несъществуващ destination.

Зад. 18 2016-SE-02 Напишете shell скрипт, който приема един позиционен параметър - число. Ако скриптът се изпълнява като root, да извежда обобщена информация за общото количество активна памет *(RSS - resident set size, non-swapped physical memory that a task has used)* на процесите на всеки потребител. Ако за някой потребител обобщеното число надвишава подадения параметър, да изпраща подходящи сигнали за прекратяване на процеса с най-много активна памет на потребителя.

Забележка: Приемаме, че изхода в колоната RSS е число в същата мерна единица, като числото, подадено като аргумент. Примерен формат:

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0  15816  1884 ?        Ss   May12   0:03 init [2]
root         2  0.0  0.0      0     0 ?        S    May12   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S    May12   0:02 [ksoftirqd/0]
```

Забележка: Алтернативно може да ползвате изхода от `ps -e -o uid,pid,rss`

Зад. 19 2016-SE-03 Напишете shell скрипт който, ако се изпълнява от root, проверява кои потребители на системата нямат homedir или не могат да пишат в него.

Примерен формат:

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

Зад. 20 2016-SE-04 В текущата директория има само обикновени файлове (без директории). Да се напише bash script, който приема 2 позиционни параметъра – числа, който мести файловете от текущата директория към нови директории (a, b и c, които трябва да бъдат създадени), като определен файл се мести към директория 'a', само ако той има по-малко редове от първи позиционен параметър, мести към директория 'b', ако редове са между първи и втори позиционен параметър и в 'c' в останалите случаи.

Зад. 21 2016-SE-05 Файловете във вашата home директория съдържат информация за музикални албуми и имат специфична структура. Началото на всеки ред е годината на издаване на албума, а непосредствено, след началото на всеки ред следва името на изпълнителя на песента. Имената на файловете се състоят от една дума, която съвпада с името на изпълнителя.

Примерно съдържание на файл с име "Bonnie":

```
2005г. Bonnie - "God Was in the Water" (Randall Bramblett, Davis Causey) – 5:17
2005г. Bonnie - "Love on One Condition" (Jon Cleary) – 3:43
2005г. Bonnie - "So Close" (Tony Arata, George Marinelli, Pete Wasner) – 3:22
2005г. Bonnie - "Trinkets" (Emory Joseph) – 5:02
2005г. Bonnie - "Crooked Crown" (David Batteau, Maia Sharp) – 3:49
2005г. Bonnie - "Unnecessarily Mercenary" (Jon Cleary) – 3:51
2005г. Bonnie - "I Will Not Be Broken" - "Deep Water" (John Capek, Marc Jordan) – 3:58
```

Да се напише shell скрипт приемащ два параметъра, които са имена на файлове от вашата home директория. Скриптът сравнява, кой от двата файла има повече на брой редове, съдържащи неговото име (на файла). За файлът победител изпълнете следните действия:

- извлечете съдържанието му, без годината на издаване на албума и без името на изпълнителя
- сортирайте лексикографски извлеченото съдържание и го запишете във файл с име 'изпълнител.songs'

Примерен изходен файл (с име Bonnie.songs):

```
"Crooked Crown" (David Batteau, Maia Sharp) – 3:49
"God Was in the Water" (Randall Bramblett, Davis Causey) – 5:17
"I Will Not Be Broken" - "Deep Water" (John Capek, Marc Jordan) – 3:58
"Love on One Condition" (Jon Cleary) – 3:43
"So Close" (Tony Arata, George Marinelli, Pete Wasner) – 3:22
"Trinkets" (Emory Joseph) – 5:02
"Unnecessarily Mercenary" (Jon Cleary) – 3:51
```

Зад. 22 2016-SE-06 Имате текстов файл със следното съдържание (всяка книга е на един ред):

```
1979 г. - „Синият тайфун“ (сборник съветски научнофантастични разкази за морето)
1979 г. - „Двойната звезда“ - Любен Дилов
1979 г. - „Завръщане от звездите“ - Станислав Лем (Превод: Веселин Маринов)
1979 г. - „Среща с Рама“ - Артър Кларк (Превод: Александър Бояджиев)
```

```
1979 г. - „Алиби“ - Димитър Пеев (криминален роман)
1979 г. - „Тайнственият триъгълник“ (сборник НФ разкази за морето)
1979 г. - „Второто нашествие на марсианците“ - Аркадий и Борис Стругацки
1979 г. - „Гробищен свят“ - Клифърд Саймък (Превод: Михаил Грънчаров)
1979 г. - „Чоки“ - Джон Уиндъм (Превод: Теодора Давидова)
1979 г. - „Спускане в Маелстрьом“ - Едгар Алан По (Превод: Александър Бояджиев)
1980 г. - „Допълнителна примамка“ - Робърт Ф. Йънг (Превод: Искра Иванова, ...)
1980 г. - „Кристалното яйце“ - Хърбърт Уелс (Превод: Борис Миндов, ...)
1980 г. - „Онирофилм“ (сборник италиански НФ разкази) (Превод: Никола Иванов, ...)
```

Напишете shell script (приемащ аргумент име на файл) който извежда:

- всеки ред от файла с добавен пореден номер във формат `"1. "`, `"2. "`, ... `"11. "` ...
- махат данните за годината на издаване
- сортират изхода по заглавие (лексикографски, възходящо)

Примерен изход (показани са само първите 4 реда):

```
5. „Алиби“ - Димитър Пеев (криминален роман)
7. „Второто нашествие на марсианците“ - Аркадий и Борис Стругацки
8. „Гробищен свят“ - Клифърд Саймък (Превод: Михаил Грънчаров)
2. „Двойната звезда“ - Любен Дилов
```

Зад. 23 2017-IN-01 Напишете скрипт, който приема три задължителни позиционни аргумента:

- име на фаил
- *низ1*
- *низ2*

Файлът е текстови, и съдържа редове във формат:

ключ=стойност

където *стойност* може да бъде:

- празен низ, т.е. редът е *ключ=*
- низ, състоящ се от един или повече термове, разделени с интервали, т.е., редът е *ключ=*$t_1\ t_2\ t_3$

Някъде във файла:

- се съдържа един ред с *ключ* първия подаден низ (*низ1*);
- и може да се съдържа един ред с *ключ* втория подаден низ (*низ2*).

Скриптът трябва да променя реда с *ключ низ2* така, че обединението на термовете на редовете с ключове *низ1* и *низ2* да включва всеки терм еднократно.

Примерен входен файл:

```
$ cat z1.txt
FOO=73
BAR=42
BAZ=
ENABLED_OPTIONS=a b c d
ENABLED_OPTIONS_EXTRA=c e f
```

Примерно извикване:

```
$ ./a.sh z1.txt ENABLED_OPTIONS ENABLED_OPTIONS_EXTRA
```

Изходен файл:

```
$ cat z1.txt
FOO=73
BAR=42
```

```
BAZ=
ENABLED_OPTIONS=a b c d
ENABLED_OPTIONS_EXTRA=e f
```

Зад. 24 2017-IN-02 Напишете скрипт, който приема задължителен позиционен аргумент - име на потребител *FOO*. Ако скриптът се изпълнява от root:

а) да извежда имената на потребителите, които имат повече на брой процеси от *FOO*, ако има такива;
б) да извежда средното време (в секунди), за което са работили процесите на всички потребители на системата (`TIME`, във формат *HH:MM:SS*);
в) ако съществуват процеси на *FOO*, които са работили над два пъти повече от средното време, скриптът да прекратява изпълнението им по подходящ начин.

За справка:

```
$ ps -e -o user,pid,%cpu,%mem,vsz,rss,tty,stat,time,command | head -5
USER       PID %CPU %MEM    VSZ   RSS TT       STAT     TIME COMMAND
root         1  0.0  0.0  15820  1920 ?        Ss   00:00:05 init [2]
root         2  0.0  0.0      0     0 ?        S    00:00:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S    00:00:01 [ksoftirqd/0]
root         5  0.0  0.0      0     0 ?        S<   00:00:00 [kworker/0:0H]
```

Зад. 25 2017-IN-03 Напишете скрипт, който извежда името на потребителския акаунт, в чиято home директория има най-скоро променен обикновен файл и кой е този файл. Напишете скрипта с подходящите проверки, така че да бъде валиден инструмент.

Зад. 26 2017-SE-01 Напишете скрипт, който получава задължителен първи позиционен параметър – директория и незадължителен втори – число. Скриптът трябва да проверява подадената директория и нейните под-директории и да извежда имената на:

а) при подаден на скрипта втори параметър – всички файлове с брой hardlink-ове поне равен на параметъра;
б) при липса на втори параметър – всички symlink-ове с несъществуващ destination (счупени symlink-ове).

*Забележка:*За удобство приемаме, че ако има подаден втори параметър, то той е число.

Зад. 27 2017-SE-02 Напишете скрипт, който приема три задължителни позиционни параметра - директория *SRC*, директория *DST* (която не трябва да съдържа файлове) и низ *ABC*. Ако скриптът се изпълнява от root потребителя, то той трябва да намира всички файлове в директорията *SRC* и нейните поддиректории, които имат в името си като под-низ *ABC*, и да ги мести в директорията *DST*, запазвайки директорийната структура (но без да запазва мета-данни като собственик и права, т.е. не ни интересуват тези параметри на новите директории, които скриптът би генерирал в *DST*).

Пример:

- в *SRC* (`/src`) има следните файлове:

  ```
  /src/foof.txt
  /src/1/bar.txt
  /src/1/foo.txt
  /src/2/1/foobar.txt
  /src/2/3/barf.txt
  ```

- *DST* (`/dst`) е празна директория
- зададения низ е `foo`

Резултат:

- в *SRC* има следните файлове:

  ```
  /src/1/bar.txt
  /src/2/3/barf.txt
  ```

- в *DST* има следните файлове:

```
/dst/foof.txt
/dst/1/foo.txt
/dst/2/1/foobar.txt
```

Зад. 28 2017-SE-03 Напишете скрипт, който ако се изпълнява от root потребителя:

а) извежда обобщена информация за броя и общото количество активна памет *(RSS - resident set size, non-swaped physical memory that a task has used)* на текущите процеси на всеки потребител;
б) ако процесът с най-голяма активна памет на даден потребител използва два пъти повече памет от средното за потребителя, то скриптът да прекратява изпълнението му по подходящ начин.

За справка:

```
$ ps aux | head -5
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0  15820  1052 ?        Ss   Apr21   0:06 init [2]
root         2  0.0  0.0      0     0 ?        S    Apr21   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S    Apr21   0:02 [ksoftirqd/0]
root         5  0.0  0.0      0     0 ?        S<   Apr21   0:00 [kworker/0:0H]
```

Алтернативно, може да ползвате изхода от `ps -e -o uid,pid,rss`

Зад. 29 2017-SE-04 Напишете shell script, който получава задължителен първи позиционен параметър – директория и незадължителен втори – име на файл. Скриптът трябва да намира в подадената директория и нейните под-директории всички symlink-ове и да извежда (при подаден аргумент файл – добавяйки към файла, а ако не е – на стандартния изход) за тях следната информация:

- ако destination-а съществува – *името на symlink-а* `->` *името на destination-а*;
- броя на symlink-овете, чийто destination не съществува.

Примерен изход:

```
lbaz -> /foo/bar/baz
lqux -> ../../../qux
lquux -> /foo/quux
Broken symlinks: 34
```

Зад. 30 2017-SE-05 Напишете скрипт, който получава два задължителни позиционни параметъра – директория и низ. Сред файловете в директорията би могло да има такива, чиито имена имат структура `vmlinuz-`*x*`.`*y*`.`*z*`-`*arch* където:

- `vmlinuz` е константен низ;
- тиретата "`-`" и точките "`.`" присъстват задължително;
- *x* е число, version;
- *y* е число, major revision;
- *z* е число, minor revision;
- наредената тройка *x.y.z* формира глобалната версия на ядрото;
- *arch* е низ, архитектура (платформа) за която е съответното ядро.

Скриптът трябва да извежда само името на файла, намиращ се в подадената директория (но не и нейните поддиректории), който:

- спазва гореописаната структура;
- е от съответната архитектура спрямо параметъра-низ, подаден на скрипта;
- има най-голяма глобална версия.

Пример:

- Съдържание на `./kern/`:

  ```
  vmlinuz-3.4.113-amd64
  vmlinuz-4.11.12-amd64
  vmlinuz-4.12.4-amd64
  vmlinuz-4.19.1-i386
  ```

- Извикване и изход:

```
$ ./task1.sh ./kern/ amd64
vmlinuz-4.12.4-amd64
```

Зад. 31 2017-SE-06 Напишете скрипт, който ако се изпълнява от root потребителя, намира процесите на потребителите, които не са root потребителя и е изпълнено поне едно от следните неща:

- имат зададена несъществуваща home директория;
- не са собственици на home директорията си;
- собственика на директорията не може да пише в нея.

Ако общото количество активна памет *(RSS - resident set size, non-swaped physical memory that a task has used)* на процесите на даден такъв потребител е по-голямо от общото количество активна памет на root потребителя, то скриптът да прекратява изпълнението на всички процеси на потребителя.

За справка:

```
$ ps aux | head -n 5
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0  15820  1052 ?        Ss   Apr21   0:06 init [2]
root         2  0.0  0.0      0     0 ?        S    Apr21   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S    Apr21   0:02 [ksoftirqd/0]
root         5  0.0  0.0      0     0 ?        S<   Apr21   0:00 [kworker/0:0H]
```

Алтернативно, може да ползвате изхода от `ps -e -o uid,pid,rss`

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
s61934:x:1177:504:Mariq Cholakova:/home/SI/s61934:/bin/bash
```

Зад. 32 2018-SE-01 Нека съществува програма за моментна комуникация (Instant messaging), която записва логове на разговорите в следния формат:

- има определена директория за логове (*LOGDIR*)
- в нея има директорийна структура от следния вид:
  LOGDIR/протокол/акаунт/приятел/
  като на всяко ниво може да има няколко екземпляра от съответния вид, т.е. няколко директории *протокол*, във всяка от тях може да има няколко директории *акаунт*, и във всяка от тях – няколко директории *приятел*
- във всяка от директориите *приятел* може да има файлове с имена от вида `yyyy-mm-dd-hh-mm-ss.txt` – година-месец-ден и т.н., спрямо това кога е започнал даден разговор
- всеки такъв файл представлява лог на даден разговор със съответния приятел, като всяка разменена реплика между вас е на отделен ред
- даден идентификатор *приятел* може да се среща няколко пъти в структурата (напр. през различни ваши акаунти сте водили разговори със същия приятел)

Напишете скрипт, който приема задължителен позиционен аргумент - име на лог директория (*LOGDIR*). Скриптът трябва да извежда десетимата приятели, с които имате най-много редове комуникация глобално (без значение протокол и акаунт), и колко реда имате с всеки от тях. Опишете в коментар как работи алгоритъмът ви.

Зад. 33 2018-SE-02 Напишете скрипт, който приема два позиционни аргумента – име на текстови файл и директория. Директорията не трябва да съдържа обекти, а текстовият файл (US-ASCII) е стенограма и всеки ред е в следния формат:

```
ИМЕ ФАМИЛИЯ (уточнения): Реплика
```

където:

- `ИМЕ ФАМИЛИЯ` присъстват задължително;
- `ИМЕ` и `ФАМИЛИЯ` се състоят само от малки/главни латински букви и тирета;
- `(уточнения)` не е задължително да присъстват;
- двоеточието ‘:’ присъства задължително;

- Репликата не съдържа знаци за нов ред;
- в стринга преди двоеточието ‘:’ задължително има поне един интервал между ИМЕ и ФАМИЛИЯ;
- наличието на други интервали където и да е на реда е недефинирано.

Примерен входен файл:

```
John Lennon (The Beatles): Time you enjoy wasting, was not wasted.
Roger Waters: I'm in competition with myself and I'm losing.
John Lennon:Reality leaves a lot to the imagination.
Leonard  Cohen:There is a crack in everything, that's how the light gets in.
```

Скриптът трябва да:

- създава текстови файл `dict.txt` в посочената директория, който на всеки ред да съдържа:

  ```
  ИМЕ ФАМИЛИЯ;НОМЕР
  ```

  където:
  - ИМЕ ФАМИЛИЯ е уникален участник в стенограмата (без да се отчитат уточненията);
  - НОМЕР е уникален номер на този участник, избран от вас.
- създава файл `НОМЕР.txt` в посочената директория, който съдържа всички (и само) редовете на дадения участник.

Зад. 34 2018-SE-03 Напишете скрипт, който приема два позиционни аргумента – имена на текстови файлове в CSV формат:

```
8,foo,bar,baz
2,quz,,foo
12,1,3,foo
3,foo,,
5,,bar,
7,,,
4,foo,bar,baz
```

Валидни са следните условия:

- CSV файловете представляват таблица, като всеки ред на таблицата е записан на отделен ред;
- на даден ред всяко поле (колона) е разделено от останалите със запетая;
- броят на полетата на всеки ред е константа;
- в полетата не може да присъства запетая, т.е., запетаята винаги е разделител между полета;
- ако във файла присъстват интервали, то това са данни от дадено поле;
- първото поле на всеки ред е число, което представлява идентификатор на реда (`ID`).

Примерно извикване: `./foo.sh a.csv b.csv`

Скриптът трябва да чете `a.csv` и на негова база да създава `b.csv` по следния начин:

- някои редове във файла се различават само по колоната `ID`, и за тях казваме, че формират множество A_i
- за всяко такова множество A_i да се оставя само един ред - този, с най-малка стойност на `ID`-то;
- редовете, които не са членове в някое множество A_i се записват в изходния файл без промяна.

Зад. 35 2019-SE-01 Напишете два скрипта (по един за всяка подточка), които четат редове от `STDIN`. Скриптовете трябва да обработват само редовете, които съдържат цели положителни или отрицателни числа; останалите редове се игнорират. Скриптовете трябва да извежда на `STDOUT`:

а) всички уникални числа, чиято абсолютна стойност е равна на максималната абсолютна стойност сред всички числа
б) всички най-малки уникални числа от тези, които имат максимална сума на цифрите си

Примерен вход:

```
We don't
n11d n0
educat10n
```

```
12.3
6
33
-42
-42
111
111
-111
```

Примерен изход за а):

```
-111
111
```

Примерен изход за б):

```
-42
```

Зад. 36 2019-SE-02 Напишете шел скрипт, който приема множество параметри. Общ вид на извикване:

```
./foo.sh [-n N] FILE1...
```

В общия случай параметрите се третират като имена на (`.log`) файлове, които трябва да бъдат обработени от скрипта, със следното изключение: ако първият параметър е стрингът `-n`, то вторият параметър е число, дефиниращо стойност на променливата N, която ще ползваме в скрипта. Въвеждаме понятието *идентификатор на файл* (ИДФ), което се състои от името на даден файл без разширението `.log`. За удобство приемаме, че скриптът:

- ще бъде извикван с аргументи имена на файлове, винаги завършващи на `.log`
- няма да бъде извикван с аргументи имена на файлове с еднакъв ИДФ.

Лог файловете са текстови, като всеки ред има следния формат:

- време: timestamp във формат `YYYY-MM-DD HH:MM:SS`
- интервал
- данни: поредица от символи с произволна дължина

За удобство приемаме, че редовете във всеки файл са сортирани по време възходящо.

Примерно съдържание на даден лог файл:

```
2019-05-05 06:26:54 orthanc rsyslogd:  rsyslogd was HUPed
2019-05-06 06:30:32 orthanc rsyslogd:  rsyslogd was HUPed
2019-05-06 10:48:29 orthanc kernel: [1725379.728871] Chrome_~dThread[876]: segfault
```

Скриптът трябва да извежда на `STDOUT` последните N реда (ако N не е дефинирано - 10 реда) от всеки файл, в следния формат:

- timestamp във формат `YYYY-MM-DD HH:MM:SS`
- интервал
- ИДФ
- интервал
- данни

Изходът трябва да бъде глобално сортиран по време възходящо.

Зад. 37 2019-SE-03 За удобство приемаме, че разполагате със системен инструмент sha256sum, който приема аргументи имена на файлове като за всеки файл пресмята и извежда уникална хеш стойност, базирана на съдържанието на файла. Изходът от инструмента е текстови, по един ред за всеки подаден като аргумент файл, в следния формат:

- хеш стойност с дължина точно 64 знака
- два интервала
- име на файл

Примерна употреба и изход:

```
$ sha256sum /var/log/syslog /var/log/user.log README.md
b2ff8bd882a501f71a144b7c678e3a6bc6764ac48eb1876fb5d11aac11014b78  /var/log/syslog
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855  /var/log/user.log
e4702d8044b7020af5129fc69d77115fd4306715bd678ba4bef518b2edf01fb9  README.md
```

Напишете скрипт, който приема задължителен параметър име на директория (ДИР1). Някъде в директорията ДИР1 може да съществуват архивни файлове с имена `NAME_report-TIMESTAMP.tgz`, където:

- `NAME` е низ, който не съдържа символ `'_'`
- `TIMESTAMP` е във формат Unix time (POSIX time/UNIX Epoch time)

На всяко пускане на скрипта се обработват само новосъздадените или модифицираните по съдържание спрямо предното пускане на скрипта архивни файлове от горния тип. За всеки такъв архивен файл се изпълнява следното:

- ако архивният файл съдържа файл с име meow.txt, то този текстови файл да бъде записан под името `/extracted/NAME_TIMESTAMP.txt`, където `NAME` и `TIMESTAMP` са съответните стойности от името на архивния файл.

Зад. 38 2020-SE-01 Напишете shell скрипт, който получава два задължителни позиционни параметъра - име на файл (`bar.csv`) и име на директория. Директорията може да съдържа текстови файлове с имена от вида `foobar.log`, всеки от които има съдържание от следния вид:

Пример 1 (`loz-gw.log`):

```
Licensed features for this platform:

  Maximum Physical Interfaces    : 8
  VLANs                          : 20
  Inside Hosts                   : Unlimited
  Failover                       : Active/Standby
  VPN-3DES-AES                   : Enabled
  *Total VPN Peers                : 25
  VLAN Trunk Ports               : 8

This platform has an ASA 5505 Security Plus license.

Serial Number: JMX00000000

Running Activation Key: 0e268e0c
```

Пример 2 (`border-lozenets.log`):

```
Licensed features for this platform:

  Maximum Physical Interfaces    : 4
  VLANs                          : 16
  Inside Hosts                   : Unlimited
  Failover                       : Active/Active
  VPN-3DES-AES                   : Disabled
  *Total VPN Peers                : 16
  VLAN Trunk Ports               : 4

This platform has a PIX 535 license.

Serial Number: PIX5350007

Running Activation Key: 0xd11b3d48
```

Имената на лог файловете (`loz-gw`, `border-lozenets`) определят даден hostname, а съдържанието им дава детайли за определени параметри на съответният хост.

Файлът `bar.csv`, който трябва да се генерира от вашия скрипт, е т.н. CSV (comma separated values) файл, тоест текстови файл - таблица, на който полетата на всеки ред са разделени със запетая. Първият ред се ползва за определяне на имената на колоните.

Скриптът трябва да създава файла `bar.csv` на база на log файловете в директорията. Генерираният CSV файл от директория, която съдържа само `loz-gw.log` и `border-lozenets.log` би изглеждал така:

```
hostname,phy,vlans,hosts,failover,VPN-3DES-AES,peers,VLAN Trunk Ports,license,SN,key
loz-gw,8,20,Unlimited,Active/Standby,Enabled,25,8,ASA 5505 Security Plus,JMX00000000,0e268e0c
border-lozenets,4,16,Unlimited,Active/Active,Disabled,16,4,PIX 535,PIX5350007,0xd11b3d48
```

Полетата в генерирания от скрипта CSV файл не трябва да съдържат излишни trailing/leading интервали. За улеснение, приемете, че всички whitespace символи във входните файлове са символа "интервал".

Зад. 39 2020-SE-02 Напишете shell скрипт, който приема задължителен параметър - име на файл. Файлът е log файл на HTTP сървър, в който се записват всички получени от сървъра request-и, които клиентите са изпратили. Файлът е текстови, като на всеки ред има информация от следния вид:

```
35.223.122.181 dir.bg - [03/Apr/2020:17:25:06 -0500] GET / HTTP/1.1 302 0 "-" "Zend_Http_Client"
94.228.82.170 del.bg - [03/Apr/2020:17:25:06 -0500] POST /auth HTTP/2.0 400 153 "foo bar" "<UA>"
```

Всеки ред на файла се състои от полета, разделени с интервал. Описание на полетата с пример спрямо първият ред от горните:

- адрес на клиент - `35.223.122.181`
- име на виртуален хост (сайт) - `dir.bg`
- име на потребител - `-`
- timestamp на заявката - `[03/Apr/2020:17:25:06 -0500]`
- заявка - `GET / HTTP/1.1` - състои се от три компонента, разделени с интервал: метод на заявката (за удобство приемаме, че може да има само GET и POST заявки), ресурсен идентификатор, и протокол (приемаме, че може да има само HTTP/1.0, HTTP/1.1 и HTTP/2.0 протоколи)
- код за статус на заявката - `302`
- брой байтове - `0`
- referer - `"-"` - ограден в двойни кавички, подава се от HTTP клиента, произволен низ
- user agent - `"Zend_Http_Client"` - ограден в двойни кавички, подава се от HTTP клиента, произволен низ

За всеки от top 3 сайта, към които има най-много заявки, скриптът трябва да изведе в долния формат:

- брой на HTTP/2.0 заявките
- брой на не-HTTP/2.0 заявките
- top 5 клиента, направили най-много заявки, завършили с код, по-голям от 302 (и броя на съответните им зявки)

```
dir.bg HTTP/2.0: 0 non-HTTP/2.0: 5
del.bg HTTP/2.0: 5 non-HTTP/2.0: 0
      5 94.228.82.170
      2 34.73.112.204
      1 185.217.0.138
```

Зад. 40 2020-SE-03 Под *пакет* ще разбираме директория, която има следната структура:

```
<name>
  |-- version
  |-- tree
        |...
```

Където *<name>* е името на пакета, `version` е текстов файл, който съдържа низ от вида `1.2.3-4` и нищо друго, а `tree` е директория с произволно съдържание.

За да получим *архив на пакет*, архивираме (`tar`) и компресираме (`xz`) съдържанието на директорията `tree`.

Под *хранилище* ще разбираме директория, която има следната структура:

```
<repo name>
  |-- db
  |-- packages
        |...
```

Където *<repo name>* е името на хранилището, `db` е текстов файл, чиито редове имат вида `<package name>-<package version> <package checksum>` и са сортирани лексикографски. Директорията `packages` съдържа архиви с имена `<package checksum>.tar.xz`, които съответстват на редове в `db`. Под *<package checksum>* имаме предвид sha256 сумата на архива на пакета.

Напишете скрипт `repo_add.sh`, който приема два аргумента - път до хранилище и път до пакет, който добавя пакета в хранилището. Ако същата версия на пакет вече съществува, архивът се заменя с новата версия. В противен случай, новата версия се добавя заедно с другите.

Заб: Първо си проектирайте общия алгоритъм на работа.

Примерно хранилище:

```
myrepo
  |-- db
  |-- packages
        |-- 6e3549438bc246b86961b2e8c3469321ca22eabd0a6c487d086de7a43a0ef766.tar.xz
        |-- 66b28e48161ba01ae25433b9ac4086a83b14d2ee49a62f2659c96514680ab6e8.tar.xz
        |-- 99c934ad80bd9e49125523c414161e82716b292d4ed2f16bb977d6db7e13d9bc.tar.xz
```

Със съдържание на `db`:

```
glibc-2.31-2 6e3549438bc246b86961b2e8c3469321ca22eabd0a6c487d086de7a43a0ef766
zlib-1.1.15-8 66b28e48161ba01ae25433b9ac4086a83b14d2ee49a62f2659c96514680ab6e8
zlib-1.2.11-4 99c934ad80bd9e49125523c414161e82716b292d4ed2f16bb977d6db7e13d9bc
```

Примерен пакет:

```
zlib
  |-- version        # contains '1.2.11-3'
  |-- tree
        |...
```

Съдържание на хранилището след изпълнение на `./repo-add.sh myrepo zlib`

```
myrepo
  |-- db
  |-- packages
        |-- 6e3549438bc246b86961b2e8c3469321ca22eabd0a6c487d086de7a43a0ef766.tar.xz
        |-- 66b28e48161ba01ae25433b9ac4086a83b14d2ee49a62f2659c96514680ab6e8.tar.xz
        |-- b839547ee0aed82c74a37d4129382f1bd6fde85f97c07c5b705eeb6c6d69f162.tar.xz
        |-- 99c934ad80bd9e49125523c414161e82716b292d4ed2f16bb977d6db7e13d9bc.tar.xz
```

Със съдържание на `db`:

```
glibc-2.31-2 6e3549438bc246b86961b2e8c3469321ca22eabd0a6c487d086de7a43a0ef766
zlib-1.1.15-8 66b28e48161ba01ae25433b9ac4086a83b14d2ee49a62f2659c96514680ab6e8
zlib-1.2.11-3 b839547ee0aed82c74a37d4129382f1bd6fde85f97c07c5b705eeb6c6d69f162
zlib-1.2.11-4 99c934ad80bd9e49125523c414161e82716b292d4ed2f16bb977d6db7e13d9bc
```

Зад. 41 2020-SE-04 Напишете скрипт, който приема два аргумента - имена на директории. Първата (*SRC*) съществува, докато втората (*DST*) трябва да бъде създадена от скрипта. Директорията *SRC* и нейните поддиректории може да съдържат файлове, чиито имена завършат на `.jpg`. Имената на файловете може да съдържат интервали, както и поднизове, оградени със скоби, например:

```
A single (very ugly) tree (Outdoor treks) 2.jpg
Falcons.jpg
Gorgonzola (cheese).jpg
Leeches (two   different ones) (Outdoor treks).jpg
Pom   Pom   Pom.jpg
```

За даден низ ще казваме, че е *почистен*, ако от него са премахнати leading и trailing интервалите и всички последователни интервали са сведени до един.

За всеки файл дефинираме следните атрибути:

- *заглавие* - частта от името преди `.jpg`, без елементи оградени в скоби, почистен. Примери:

  ```
  A single tree 2
  Falcons
  Gorgonzola
  Leeches
  Pom Pom Pom
  ```

- *албум* - последният елемент от името, който е бил ограден в скоби, почистен. Ако *албум* е празен стринг, ползваме стойност по подразбиране `misc`. Примери:

  ```
  Outdoor treks
  misc
  cheese
  Outdoor treks
  misc
  ```

- *дата* - времето на последна модификация на съдържанието на файла, във формат YYYY-MM-DD
- *хеш* - първите 16 символа от sha256 сумата на файла. *Забележка: приемаме, че в тази идеална вселена първите 16 символа от sha256 сумата са уникални за всеки файл от тези, които ще се наложи да обработваме.*

Скриптът трябва да създава в директория *DST* необходимата структура от под-директории, файлове и symlink–ове, така че да са изпълнени следните условия за всеки файл от *SRC*:

- *DST*`/images/хеш.jpg` - копие на съответния файл
- следните обекти са относителни symlink–ове към `хеш.jpg`:
  - *DST*`/by-date/дата/by-album/албум/by-title/заглавие.jpg`
  - *DST*`/by-date/дата/by-title/заглавие.jpg`
  - *DST*`/by-album/албум/by-date/дата/by-title/заглавие.jpg`
  - *DST*`/by-album/албум/by-title/заглавие.jpg`
  - *DST*`/by-title/заглавие.jpg`

Зад. 42 2020-SE-05 Напишете shell скрипт, който приема 3 позиционни аргумента – две имена на файлове и име на директория. Примерно извикване:

```
$ ./foo.sh foo.pwd config.cfg cfgdir/
```

В директорията `cfgdir/` и нейните под-директории може да има файлове с имена завършващи на `.cfg`. За да са *валидни*, тези файлове трябва да съдържат редове само в следните формати (редовете започващи с # са коментари):

```
# internal laboratory
{ no-production };
```

```
{ volatile };

# meow configs
{ run-all; };
```

Във файла `foo.pwd` има описани потребителски имена (username) и MD5 хеш суми на паролите им, с по един запис на ред, в следният формат:

```
username:password_hash
```

Също така, разполагате с команда `pwgen`, която генерира и извежда на `STDOUT` случайни пароли, и знаете, че поддържа следните два аргумента:

```
$ pwgen [ password_length ] [ number_of_passwords ]
```

Вашият скрипт трябва да валидира cfg файловете в директорията, и за тези, които не са валидни, да извежда на `STDOUT` името на файла и номерирани редовете, които имат проблем, в следния формат:

```
Error in filename.cfg:
Line 1:XXXX
Line 37:YYYY
```

където *XXXX* и *YYYY* е съдържанието на съответния ред.

За валидните файлове, скриптът трябва да:

- генерира `config.cfg` като обединение на съдържанието им;
- името на файла, без частта `.cfg` дефинира потребителско име. Ако във файла с паролите не съществува запис за този потребител, то такъв да се добави и на стандартния изход да се изведе потребителското име и паролата (поне 16 символа) разделени с един интервал.

Зад. 43 2020-SE-06 Под *конфигурационен файл* ще разбираме файл, в който има редове от вида `key=value`, където `key` и `value` могат да се състоят от букви, цифри и знак "долна черта" ("_"). Освен това, във файла може да има празни редове; може да има произволен whitespace в началото и в края на редовете, както и около символа "=". Също така са допустими и коментари в даден ред: всичко след символ "#" се приема за коментар.

Под `<date>` ще разбираме текущото време, върнато от командата `date` без параметри; под `<user>` ще разбираме името на текущият потребител.

Напишете shell скрипт `set_value.sh`, който приема 3 позиционни аргумента – име на конфигурационен файл, ключ (`foo`) и стойност (`bar`). Ако ключът:

- присъства във файла с друга стойност, скриптът трябва да:
  - да закоментира този ред като сложи # в началото на реда и добави в края на реда `# edited at <date> by <user>`
  - да добави нов ред `foo = bar # added at <date> by <user>` точно след стария ред
- не присъства във файла, скриптът трябва да добави ред от вида `foo = bar # added at <date> by <user>` на края на файла

Примерен `foo.conf`*:*

```
# route description

from = Sofia
to   = Varna   # my favourite city!
type = t2_1
```

Примерно извикване:

```
./set_value.sh foo.conf to Plovdiv
```

Съдържание на `foo.conf` *след извикването:*

```
# route description

from = Sofia
# to   = Varna   # my favourite city! # edited at Tue Aug 25 15:48:29 EEST 2020 by human
to = Plovdiv # added at Tue Aug 25 15:48:29 EEST 2020 by human
type = t2_1
```

Зад. 44 2021-SE-01

Разполагате с машина, на която е инсталиран специализиран софтуер, който ползва два потребителски акаунта – `oracle` и `grid`. Всеки от потребителите *би трябвало* да има environment променлива `ORACLE_HOME`, която указва абсолютен път до директория във формат `/path/to/dir`. В под-директория `bin` на зададената директория *би трябвало* да има изпълним файл с име `adrci`. Всеки от двата потребителя има собствена директория *diag_dest*, която е във вида `/u01/app/потребител`. Когато някой от потребителите изпълни неговото копие на командата `adrci` с параметър `exec="show homes"` може да получи на STDOUT един от следните два изхода:

- вариант 1: (неуспех): `No ADR homes are set`
- вариант 2: (успех):

  ```
  ADR Homes:
   diag/rdbms/orclbi/orclbi1
  diag/rdbms/orclbi/orclbi2
  ```

И в двата случая командата приключва с exit code `0`. Ако командата се изпълни успешно, тя връща списък с един или повече ADR Homes, които са релативни имена на директории спрямо *diag_dest* на съответният потребител.

Напишете скрипт, който може да се изпълнява само от някой от тези два акаунта, и извежда на STDOUT размера в мегабайти и абсолютният път на всеки ADR Home.

Примерен изход:

```
0    /u01/app/oracle/diag/rdbms/orclbi/orclbi1
389  /u01/app/oracle/diag/rdbms/orclbi/orclbi2
```

Зад. 45 2021-SE-02

Един от често използваните DNS сървъри е BIND9, при който описанието на DNS зоните обикновенно стои в текстови файлове, наричани *зонални файлове*. За улеснение, в рамките на задачата, ще ползваме опростено описание на зоналните файлове.

Под *whitespace* разбираме произволна комбинация от табове и интервали.

Под *FQDN* разбираме низ, който има допустими символи малки латински букви, цифри и точка; не може да започва с точка, не може да има две или повече съседни точки, задължително завършва с точка.

Зоналните файлове съдържат *ресурсни записи*, по един на ред. Общият вид на даден ресурсен запис е `<ключ> <TTL> <клас> <тип> <RDATA>`, разделени с whitespace, например:

```
astero.openfmi.net. 3600 IN A 185.117.82.99
```

Където:

- ключ (`astero.openfmi.net.`) – FQDN
- TTL (`3600`) – цифри; полето може да липсва
- клас (`IN`) - главни латински букви; класът винаги е `IN`
- тип (`A`) - главни латински букви; някое от `SOA`, `NS`, `A`, `AAAA`
- RDATA (`185.117.82.99`) - данни на записа; различни за различните типове ресурсни записи; всичко след типа до края на реда.

Знакът точка-и-запетая `;` е знак за коментар, и всичко след него до края на реда се игнорира.

Във всеки зонален файл трябва да има точно един SOA запис, и той трябва да е първият запис във файла. Пример за едноредов SOA запис:

```
openfmi.net. 3600 IN SOA nimbus.fccf.net. root.fccf.net. 2021041901 86400 7200 3024000 3600
```

RDATA-та на SOA запис се състои от два FQDN-а и пет числа, разделени с whitespace.

Въпреки, че горното е валиден SOA запис, за прегледност в зоналните файлове често се ползва следният синтаксис (многоредов SOA запис, еквивалентен на горния):

```
openfmi.net. 3600 IN SOA nimbus.fccf.net. root.fccf.net. (
                         2021041901 ; serial
                         86400      ; refresh
                         7200       ; retry
                         3024000    ; expire
                         3600       ; negative TTL
                       )
```

т.е., поредицата от числа се разбива на няколко реда, оградени в обикновенни скоби, и за всяко число се слага коментар какво означава.

Първото от тези числа (serial) представлява серийният номер на зоната, който трябва да се увеличава всеки път, когато нещо в зоналният файл се промени. Изключително важно е това число само да нараства, и никога да не намалява.

Един от често използваните формати за сериен номер, който показва кога е настъпила последната промяна в зоналния файл представлява число във вида YYYYMMDDTT, т.е., четири цифри за година, две цифри за месец, две цифри за дата и още две цифри за поредна промяна в рамките на описания ден. За последните две цифри (TT) има ограничение да са от 00 до 99 (естествено, така не може да има повече от 100 промени в рамките на един ден).

За удобство приемаме, че конкретен сериен номер (точната поредица цифри) се среща само на едно място в зоналния файл.

Напишете шел скрипт, който по подадени имена на зонални файлове променя серийният номер в SOA записа на всеки файл по следният алгоритъм:

- ако датата в серийният номер е по-стара от днешната, новият сериен номер трябва да е от вида `днешнадата00`
- ако датата в серийният номер е равна на днешната, серийният номер трябва да се увеличи с единица

Важат следните условия:

- скриптът трябва да може да обработва и едноредови, и многоредови SOA записи
- за всеки зонален файл, който не е успял да обработи, скриптът трябва да вади съобщение за грешка, което включва и името на зоналния файл. Съобщенията трябва да са лесно обработваеми с познатите инструменти за обработка на текст.

Зад. 46 2021-SE-03 Напишете shell скрипт, който приема два позиционни параметъра – имена на файлове. Примерно извикване:

```
$ ./foo.sh input.bin output.h
```

Файлът `input.bin` е двоичен файл с елементи `uint16_t` числа, създаден на *little-endian* машина.

Вашият скрипт трябва да генерира C хедър файл, дефиниращ масив с име `arr`, който:

- съдържа всички елементи от входния файл;
- няма указана големина;
- не позволява промяна на данните.

Генерираният хедър файл трябва да:

- съдържа и `uint32_t` променлива `arrN`, която указва големината на масива;

- бъде валиден и да може да се `#include`-ва без проблеми от C файлове, очакващи да "виждат" `arr` и `arrN`.

За да е валиден един входен файл, той трябва да съдържа не повече от 524288 елемента.

За справка, dump на съдържанието на примерен input.bin:

```
00000000: 5555 5655 5955 5a55 6555 6655 6955 6a55  UUVUYUZUeUfUiUjU
00000010: 9555 9655 9955 9a55 a555 a655 a955 aa55  .U.U.U.U.U.U.U.U
```

Зад. 47 2021-SE-04

Разполагате с машина, на която е инсталиран специализиран софтуер, който ползва два потребителски акаунта – `oracle` и `grid`. За всеки от двата акаунта съществува директория, която ще наричаме *diag_dest* и тя е от вида `/u01/app/потребител`.

Всеки от потребителите *би трябвало* да има environment променлива `ORACLE_HOME`, която указва абсолютен път до директория във формат `/път/до/дир`. В поддиректорията `bin` на зададената директория *би трябвало* да има изпълним файл с име `adrci`.

Всеки от потребителите може да подаде серия от подкоманди, които неговото копие на `adrci` да изпълни, като го извика с параметър `exec="cmd1; cmd2; cmd3"`. Отделните подкоманди се разделят с точка и запетая (`;`). Командата `adrci` винаги приключва с exit code `0`.

Нека дефинираме следните подкоманди:

- `SET BASE` – за да се гарантира правилна работа на командата `adrci`, при всяко нейно извикване първата подкоманда трябва да бъде от вида `SET BASE diag_dest`, където *diag_dest* е абсолютният път на съответната директория за дадения потребител
- `SHOW HOMES` – подкоманда `SHOW HOMES` извежда на STDOUT един от следните два изхода:
  - вариант 1: (неуспех): `No ADR homes are set`
  - вариант 2: (успех):

    ```
    ADR Homes:
    diag/rdbms/orclbi/orclbi1
    diag/rdbms/orclbi/orclbi2
    ```

    Ако командата се изпълни успешно, тя връща списък с един или повече ADR Homes, които са релативни имена на директории спрямо *diag_dest* на съответният потребител.
    От полученият списък с релативни пътища *интересни* за нас са само тези, които за име на директория на второ ниво имат една от следните директории: `crs`, `tnslsnr`, `kfod`, `asm` или `rdbms`.
- `SET HOMEPATH` – подкоманда `SET HOMEPATH път` задава *активна* работна директория, спрямо която ще се изпълняват последващи подкоманди в рамките на същото изпълнение на `adrci`; *път* е релативен път до директория, спрямо изхода на `SHOW HOMES`
- `PURGE` – подкоманда `PURGE -AGE минути` изтрива определени файлове в текущо активната работна директория, по-стари от дефинираното време в минути. *Забележка: изтриват се само безопасни файлове, т.е. такива, чието изтриване няма да доведе до проблем. Дефиницията на безопасни файлове е извън обхвата на тази задача.*

Напишете shell скрипт, който може да се изпълнява само от някой от указаните два акаунта и приема задължителен първи позиционен аргумент число (в часове, минимална стойност 2 часа). Скриптът трябва да почиства безопасните файлове по-стари от зададените часове в интересните ADR Home-ове на съответния потребител.

Зад. 48 2022-CE-01

Описание на формат на CSV (текстови) файл:

- CSV файлът представлява таблица, като всеки ред на таблицата е записан на отделен ред;
- на даден ред всяко поле (колона) е разделено от останалите със запетая;
- в полетата не може да присъства запетая, т.е. запетаята винаги е разделител между полета;
- броят на полетата на всеки ред е константа;
- първият ред във файла е header, който описва имената на колоните.

Разполагате с два CSV файла със следното примерно съдържание:

- `base.csv`:

  ```
  unit name,unit symbol,measure
  ```

```
second,s,time
metre,m,length
kilogram,kg,mass
ampere,A,electric current
kelvin,K,thermodynamic temperature
mole,mol,amount of substance
candela,cd,luminous intensity
```

- `prefix.csv`:

```
prefix name,prefix symbol,decimal
tera,T,1000000000000
giga,G,1000000000
mega,M,1000000
mili,m,0.001
nano,n,0.000000001
```

Където смисълът на колоните е:

- за `base.csv`
  - `unit name` – име на мерна единица
  - `unit symbol` – съкратено означение на мерната единица
  - `measure` – величина
- за `prefix.csv`
  - `prefix name` – име на представка
  - `prefix symbol` – означение на представка
  - `decimal` – стойност

Забележка: Във файловете може да има и други редове, освен показаните в примера. Приемаме, че файловете спазват описания по-горе формат, т.е. не е необходимо да проверявате формата.

Напишете shell скрипт, който приема три задължителни параметъра: *число*, *prefix symbol* и *unit symbol*.

Скриптът, ако може, трябва да извежда числото в основната мерна единица без представка, добавяйки в скоби величината и името на мерната единица.

Примерен вход и изход:

```
$ ./foo.sh 2.1 M s
2100000.0 s (time, second)
```

Забележка: За изчисления може да ползвате `bc`.

Зад. 49 2022-CE-02 Съвременните компютри могат да влизат в различни режими за енергоспестяване (suspend) и излизат от този режим (wake-up) при настъпването на определени събития. Linux kernel предоставя специален виртуален файл `/proc/acpi/wakeup`, чрез който може да се проверява или променя настройката за "събуждане" при при настъпване на различни събития. Тъй като този файл е интерфейс към ядрото, "четенето" от файла и "писането" във файла изглеждат различно.

За улеснение на задачата ще ползваме опростено описание на този файл.

Под *whitespace* разбираме произволна комбинация от табове и интервали.

При четене от файла изходът представлява четири колони, разделени с whitespace, в полетата не може да има whitespace; първият ред е header на съответната колона.

Примерно съдържание на файла:

```
Device    S-state      Status       Sysfs node
GLAN      S4           *enabled     pci:0000:00:1f.6
XHC       S3           *enabled     pci:0000:00:14.0
XDCI      S4           *disabled
LID       S4           *enabled     platform:PNP0C0D:00
HDAS      S4           *disabled    pci:0000:00:1f.3
RP01      S4           *enabled     pci:0000:00:1c.0
```

където:

- първата колона дава уникално име на устройство, което може да събуди машината, като името е ограничено до четири знака главни латински букви и цифри;
- третата колона описва дали това устройство може да събуди машината. Възможните стойности са `enabled`/`disabled`, предхождани от звездичка;
- втората и четвъртата колона ги игнорираме в рамките на задачата.

При записване на име на устройство във файла `/proc/acpi/wakeup`, неговото състояние се променя от `disabled` на `enabled` или обратно.

Например, ако `/proc/acpi/wakeup` изглежда както примера по-горе, при изпълнение на командата `echo XHC > /proc/acpi/wakeup`, третият ред ще се промени на:

```
XHC       S3           *disabled     pci:0000:00:14.0
```

При изпълнение на командата `echo HDAS > /proc/acpi/wakeup`, шестият ред ще се промени на:

```
HDAS      S4           *enabled      pci:0000:00:1f.3
```

Напишете shell скрипт, който при подаден първи параметър име на устройство (напр. `LID`) настройва съответното устройство да **не може** да събуди машината.

Зад. 50 2022-IN-01

Както знаете, при отваряне на файл с редактора `vi`, той създава в същата директория временен файл с име в следния формат: точка, името на оригиналния файл, точка, swp. Например, при редактиране на файл с име `foo.txt` ще се създаде временен файл с име `.foo.txt.swp`.

Напишете shell скрипт, който приема два задължителни позиционни аргумента – имена на директории. Примерно извикване: `./foo.sh ./dir1 /path/to/dir2/`

В `dir1` може да има файлове/директории, директорията `dir2` трябва да е празна.

Скриптът трябва да копира всички обикновенни файлове от `dir1` (и нейните под-директории) в `dir2`, запазвайки директорийната структура, но без да копира временните файлове, създадени от редактора `vi` (по горната дефиниция).

Забележка: За удобство приемаме, че не ни вълнува дали метаданните на обектите (owner, group, permissions, etc.) ще се запазят.

Примерни обекти:

```
dir1/
dir1/a
dir1/.a.swp
dir1/b
dir1/c/d
dir1/c/.bar.swp
```

Обекти след изпълнението:

```
dir2/
dir2/a
dir2/b
dir2/c/d
dir2/c/.bar.swp
```

Зад. 51 2022-IN-02

Името на дадена машина можете да вземете с командата `hostname -s`.

Разполагате с машина, на която е инсталиран специализиран софтуер, който ползва два потребителски акаунта – `oracle` и `grid`.

Всеки от потребителите *би трябвало* да има environment променлива `ORACLE_BASE`, която указва абсолютен път до директория във формат `/път/до/дир`.

Всеки от потребителите *би трябвало* да има environment променлива `ORACLE_HOME`, която указва абсолютен път до директория във формат `/път/до/дир`. В поддиректорията `bin` на зададената директория *би трябвало* да има изпълним файл с име `sqlplus`.

Всеки от потребителите *би трябвало* да има environment променлива `ORACLE_SID` с някакъв низ като стойност.

Ако горните три environment променливи съществуват, всеки от потребителите може да изпълнява неговото копие на командата `sqlplus` със следните параметри: `sqlplus -SL "/ as роля" @foo.sql` където `роля` трябва да бъде низът `SYSDBA` при изпълнение от `oracle` и `SYSASM` при изпълнение от `grid`. И в двата случая `sqlplus` изпълнява SQL заявките от файла (`foo.sql`, името на файла няма значение) и извежда изхода на stdout. Ако съдържанието на sql файла е:

```
SELECT value FROM v$parameter WHERE name = 'diagnostic_dest';
EXIT;
```

изходът ще бъде стойността на търсения параметър `diagnostic_dest` в следния вид:

```
oracle@astero:~$ sqlplus -SL "/ as sysdba" @a.sql

VALUE
--------------------------------------------------------------------------------
/u01/app/oracle

oracle@astero:~$
```

Параметърът `diagnostic_dest` може да няма стойност, в който случай изведеният низ ще е празен. Изходът винаги е 5 реда, стойността винаги е на 4-и ред. Ако командата `sqlplus` не се изпълни успешно, тя ще върне ненулев exit code.

За всеки от двата акаунта съществува директория, която ще наричаме *diag_base*. Конкретната директория е:

- същата като `ORACLE_BASE`, ако `diagnostic_dest` няма стойност
- същата като `diagnostic_dest`, ако `diagnostic_dest` има стойност

За всеки от двата акаунта *би трябвало* да съществува под-директория на *diag_base* с име `diag`, която ще наричаме *diag_dir*.

Съществуват три множества *интересни* за нас файлове:

- множество `crs` – за потребител `grid`, в *diag_dir* има под-директория `crs`, в която има под-директория с името на машината, в която има под-директория `crs`, в която има под-директория `trace`. *Интересни* за нас файлове в тази директория са такива, чието име завършва на подчертавка-число и имат разширение `trc` или `trm`, например `foo_356.trc`, `bar_40001.trm`.
- множество `tnslsnr` – за потребител `grid`, в *diag_dir* има под-директория `tnslsnr`, в която има под-директория с името на машината, в която има *няколко* директории с различни имена. Във всяка от тези директории има под-директории `alert` и `trace`. *Интересни* за нас са файловете в `alert`, чието име завършва на подчертавка-число и имат разширение `xml` (напр. `baz_78.xml`) и файловете в `trace`, чието име завършва на подчертавка-число и имат разширение `log` (напр. `qux_88231.log`).
- множество `rdbms` – за потребител `oracle`, в *diag_dir* има под-директория `rdbms`, в която има *няколко* под-директории, във всяка от които може да има *няколко* под-директории. *Интересни* за нас са *само* файловете в тези директории на второ ниво, чието име завършва на подчертавка-число и имат разширение `trc` или `trm`, например `corge_27.trc`, `grault_1024.trm`.

Напишете скрипт, който може да се изпълнява само от някой от тези два акаунта, и приема задължителен първи позиционен аргумент число (в дни). В зависимост от това кой потребител изпълнява скрипта, той трябва да извежда на stdout за всяко множество на съответния потребител общият размер (в килобайти) на описаните по-горе *интересни* файлове за които времето на последната промяна (по съдържание) на файла е по-голямо от зададеното като параметър на скрипта.

Примерно изпълнение и изход:

```
oracle@astero:~$ ./foo.sh 42
rdbms: 14400
grid@astero:~$ ./foo.sh 73
crs: 28800
tnslsnr: 33600
```

Забележка: Правилното ползване на временни файлове е разрешено.

Зад. 52 2022-IN-03 Вашите колеги от съседната лаборатория работят по проект, в който до сега се е ползвала *нестандартна* имплементация на командата `java`. По време на миграцията с цел използване на *стандартна* имплементация на командата се оказало, че на много места командата се вика по стария, грешен начин, който вече е невалиден. Вашата задача е да напишете shell скрипт `iwojima.sh`, който работи подобно на старата имплементация на `java`, но вътрешно вика новата имплементация по правилния начин.

Файловият формат JAR (Java ARchive) представлява пакет, който обединява няколко Java class файлове, допълнителни мета-данни и различни ресурси. Файлове в този формат обикновенно (но не задължително) имат разширение `.jar`. Част от метаданните във файла указват кой е главният клас, чийто метод main() се изпълнява по подразбиране. Въпросният main() метод ще получи списък с аргументите, указани при извикването на командата `java` (подобно на main() функцията в езика C).

Едно от извикванията на *стандартната* имплементация на командата `java` (единственото, което разглеждаме в тази задача) има следният синтаксис: `java [options] -jar filename [args]` където:

- `options` – незадължителна поредица от опции на командата `java`, разделени с интервал; всяка опция задължително започва с тире;
- `filename` – име на JAR файл, който да бъде изпълнен, използва се винаги с опция `-jar`;
- `args` – незадължителни аргументи, с които ще бъде стартиран main() методът на класа по подразбиране (аргументи на самата Java програма).

Като пример разполагаме с JAR файл (`app.jar`), с един клас в него, чийто main() метод реализира единствено функционалност, еквивалента на следният bash код:

```
if [ -z "${qux}" ]; then
    qux=default
fi
echo "${qux}"
```

Общият вид на изпълнение с командата `java` би бил:

```
$ java -jar ./app.jar
default
```

Една от стандартните опции на командата `java` е `-Dproperty=value`, с което се дефинира променлива с име `property` и със стойност `value`. При нужда от няколко променливи, за всяка се ползва отделна опция `-D`. Пример за правилно ползване на опцията в *стандартната* имплементация:

```
$ java -Dqux=foo -jar /path/to/app.jar
foo
```

За съжаление, *нестандартната* имплементация на `java`, която е ползвана до момента, работи по друг начин. Пример:

```
$ java -jar ./app.jar -Dqux=foo
default

$ java -Dqux=foo -jar ./app.jar -hoge fuga
default

$ java -jar -Dqux=foo app.jar
foo
```

Ако трябва да обобщим как работи *нестандартната* имплементация:

- всичко след името на JAR файла са аргументи за main() метода (напр. `-hoge` и `fuga`);
- името на JAR файла е *някъде* след опцията `-jar`;
- опции за `java` командата може да има и преди, и след опцията `-jar`;
- опцията `-Dproperty=value` работи само ако е след опцията `-jar`.

Забележки: Вашият скрипт `iwojima.sh` няма нужда да поддържа извиквания, които не подават опция `-jar`. За удобство приемаме, че опциите (вкл. `property` и `value`), имената на JAR файловете и аргументите не могат да съдържат интервали.

Зад. 53 2022-IN-04

Вашите колеги от съседната лаборатория работят със специализирана система Fuga, която се базира на два текстови файла – за автентификация (`foo.pwd`) и основен конфигурационен файл (`foo.conf`). Двата файла би трябвало да се съхраняват в главната директория на системата, която ще наричаме `fuga`.

На всеки ред във първия файл има потребителско име (малки латински букви) и паролата за този потребител, разделени с двоеточие, като паролата не е в чист вид, а е *хеширана*.

Форматът на конфигурационния файл е недефиниран. Тъй като конфигурационният файл е твърде голям за удобна работа, вашите колеги са решили да го разделят на части в отделни *малки* конфигурационни файлове, които държат в директория `fuga/cfg` и нейните под-директории. Всеки такъв файл има име във формат `bar.cfg`, където `bar` е име на потребител на системата. Колегите ви са написали скрипт (`fuga/validate.sh`), който приема един задължителен позиционен аргумент – име на конфигурационен файл. Скриптът проверява указания конфигурационен файл за валиден синтаксис и може да приключи с някой от следните exit code-ове:

- 0 – файлът е валиден;
- 1 – файлът не е валиден;
- 2 – настъпила е грешка при изпълнение на скрипта.

Ако файлът не е валиден, скриптът извежда на stdout редовете, на които има грешка, предхождани от `Line x:` където x е номера на реда. В останалите случаи скриптът няма изход.

Разполагате с команда `pwgen`, която генерира и извежда на stdout случайни пароли (букви и цифри), и знаете, че поддържа следните два аргумента: `pwgen [password_length] [number_of_passwords]`

Също така разполагате с командата `mkpasswd`, която по подаден аргумент – парола в чист вид извежда нейният *хеш* на stdout.

Помогнете на колегите си, като напишете шел скрипт, който приема параметър – име на директория `fuga`. Скриптът трябва да:

- извежда на stderr грешните редове от *малките* конфигурационни файлове, във формата извеждан от валидиращия скрипт, като всеки ред трябва да започва с името на конфигурационния файл и знак двоеточие ':';
- (ре-)генерира `foo.conf` като конкатенация на валидните *малки* конфигурационни файлове;
- провери за всеки валиден *малък* конфигурационен файл дали потребителят съществува в автентификационния файл и ако не – да го добави по подходящия начин, като изведе на stdout потребителското име и паролата в чист вид, разделени с двоеточие.

Зад. 54 2022-SE-01 Съвременните компютри могат да влизат в различни режими за енергоспестяване (suspend) и излизат от този режим (wake-up) при настъпването на определени събития. Linux kernel предоставя специален виртуален файл `/proc/acpi/wakeup`, чрез който може да се проверява или променя настройката за "събуждане" при настъпване на различни събития, в общия случай - при активност на някое устройство. Тъй като този файл е интерфейс към ядрото, "четенето" от файла и "писането" във файла изглеждат различно.

За улеснение на задачата ще ползваме опростено описание на този файл.

Под *whitespace* разбираме произволна комбинация от табове и интервали. При четене от файла изходът представлява четири колони, разделени с whitespace, в полетата не може да има whitespace; първият ред е header на съответната колона. Примерно съдържание на файла:

```
Device    S-state      Status       Sysfs node
GLAN      S4           *enabled     pci:0000:00:1f.6
```

```
XHC       S3          *enabled     pci:0000:00:14.0
XDCI      S4          *disabled
LID       S4          *enabled     platform:PNP0C0D:00
HDAS      S4          *disabled    pci:0000:00:1f.3
RP01      S4          *enabled     pci:0000:00:1c.0
```

където:

- първата колона дава уникално име на устройство, което може да събуди машината, като името е ограничено до четири знака главни латински букви и цифри;
- третата колона описва дали това устройство може да събуди машината. Възможните стойности са `enabled`/`disabled`, предхождани от звездичка;
- втората и четвъртата колона ги игнорираме в рамките на задачата.

При записване на име на устройство във файла `/proc/acpi/wakeup`, неговото състояние се променя от `disabled` на `enabled` или обратно. Например, ако файлът изглежда както примера по-горе, при запис на `XHC` в него, третият ред ще се промени на:

```
XHC       S3          *disabled    pci:0000:00:14.0
```

При запис на `HDAS`, шестият ред ще се промени на:

```
HDAS      S4          *enabled     pci:0000:00:1f.3
```

Дефиниран е формат на конфигурационен файл, който описва желан комплект от настройки на wakeup събития. Примерен файл:

```
# comment bar
GLAN    disabled
LID     enabled    # comment foo
```

където:

- знакът диез (#) е знак за коментар до края на реда;
- редовете *би трябвало* да са комбинация от име на устройство и желаното състояние на настройката за събуждане при събития от това устройство, разделени с whitespace.

Напишете скрипт, който при подаден първи параметър име на конфигурационен файл в горния формат осигурява исканите настройки за събуждане. Ако във файла има ред за устройство, което не съществува, скриптът да извежда предупреждение. Обърнете внимание на обработката за грешки и съобщенията към потребителя – искаме скриптът да бъде удобен и валиден инструмент.

Зад. 55 2022-SE-02 Дефинирана е система за бекъпване на сървъри, която държи направените архиви в главна директория (която ще наричаме *fubar*), в която има четири под-директории за различни *класове* бекъпи:

- `0` – съдържа годишни бекъпи
- `1` – съдържа месечни бекъпи
- `2` – съдържа седмични бекъпи
- `3` – съдържа дневни бекъпи

Всяка директория съдържа архивни файлове с имена във формат `hostname-area-YYYYMMDD.tar.xz`, където:

- `hostname` е името на някаква машина, която е бекъпвана;
- `area` е типът бекъп за съответната машина;
- `YYYYMMDD` е датата, на която е направен бекъпа;
- никое от горните полета не може да съдържа тире или точка;
- някои от файловете могат да са symlink-ове.

Примерни имена на файлове:

```
astero-etc-20220323.tar.xz   stratios-etc-20220428.tar.xz   nestor-db-20220404.tar.xz
gila-srv-20220405.tar.xz     catalyst-var-20220406.tar.xz   drake-home-20220404.tar.xz
dominix-var-20220404.tar.xz
```

Комбинацията от `hostname` и `area` дефинира уникален *обект* за бекъпване. Всички архивни файлове са *пълноценни* бекъпи на даден *обект* и са достатъчни за неговото възстановяване при нужда (*заб. извън обхвата на задачата*).

Ако даден файл е symlink, то той може да е валиден или счупен. Symlink-овете са създадени за удобство и не ги считаме за *пълноценни* бекъпи.

Политиката ни за бекъп казва, че за да имаме *валиден* бекъп на даден *обект*, за него трябва да имаме минимум 1 годишен, 2 месечни, 3 седмични и 4 дневни *пълноценни* бекъпа.

Важност:

- *обектите* са равни по важност помежду си;
- важността на *класовете* бекъпи е във възходящ ред по горния списък, т.е. при равни други условия дневните бекъпи са по-ценни от седмичните и т.н.;
- в рамките на един *клас* бекъпи по-новите бекъпи са по-важни от по-старите.

Напишете shell скрипт, който приема два два задължителни позиционни аргумента – име на директория и число в интервала $[1, 99]$. Примерно извикване: `./foo.sh ./bar/ 30` където:

- директорията представлява главна директория (*fubar*) на описаната система за бекпъване;
- числото дефинира колко процента е максималното допустимо използвано място на файловата система, в която се намира подадената директория.

За удобство приемаме, че директорията *fubar* и всички обекти в нея се намират в една и съща файлова система.

Упътване: Командата `df` извикана с аргумент име на файл/директория връща информация за файловата система, в която той се намира. Пример:

```
$ df ./README.md
Filesystem        1K-blocks     Used Available Use% Mounted on
/dev/mapper/o7-hm 100663296 61735732  37288764  63% /home
```

Скриптът трябва да изтрива *минимален* брой *пълноценни* архивни файлове така, че:

- всеки *обект* да има *валиден* бекъп;
- обръща внимание на описаните по-горе важности;
- процентите използвано място върху файловата система да е *не повече* от подаденият параметър (а ако това е невъзможно, скриптът да освободи колкото може повече място, без да нарушава *валидностите* на бекъпите на *обектите*);
- не е допустимо след работата на скрипта да останат счупени symlink–ове.

Зад. 56 2023-CE-01

Трябва да напишете shell script, който работи с данни от наблюдения на звезди.

Скриптът приема два параметъра:

- Име на CSV файл, съдържащ данните
- Име на тип звезда (по-долу ще обясним за какво го ползваме)

Част от CSV файла изглежда така:

```
Hen 2-99,000-BDB-257,13 52 30.68 -66 23 26.6,Cir,PN,--,13.1
Nova Car 1971,--,10 39 47.12 -63 14 07.2,Car,NON-CV,--,--
WR 21a,000-BJP-528,10 25 56.50 -57 48 43.5,Car,WR,31.673,12.82
HD 109857,000-BKQ-240,12 39 14.58 -75 22 14.1,Mus,HMXB,--,6.48
```

Колонките са разделени със запетайки и имат следните значения:

1. **Име на звездата**
2. Идентификатор
3. Координати
4. **Име на съзвездие**
5. **Тип на звездата**
6. Период

7. **Яркост (магнитуд)**
   - По-ярките звезди имат по-малка стойност за магнитуд

Скриптът трябва да извежда името на най-ярката звезда от съзвездието, в което има най-много звезди от типа, подаден като втори параметър.

Липсващите стойности са обозначени с две тирета.

Зад. 57 2023-CE-02

Вие сте част от екип от физици от бъдещето, които анализират данни, събрани от различни наблюдателни точки в космоса. Разполагате с файлове с данни, които съдържат информация за разстоянието от наблюдателна точка до различни черни дупки в космоса. Един ред от такъв файл има следния вид:

```
<име на черна дупка>: <разстояние> megaparsecs
```

Примерни файлове (`point1.opt` отляво и `point2.opt` отдясно)

```
Phoenix A: 1793 megaparsecs
H1821+643: 900 megaparsecs
Holmberg 15A: 216 megaparsecs
SMSS J215728.21-360215.1: 7665 megaparsecs
```

```
SMSS J215728.21-360215.1: 8211 megaparsecs
Holmberg 15A: 195 megaparsecs
Phoenix A: 1823 megaparsecs
Ton 618: 3211 megaparsecs
```

Вашата задача е да напишете скрипт, който приема три аргумента от командния ред (имената на два файла с данни от наблюдателни точки и името на черна дупка) и извежда коя от двете наблюдателни точки се намира по-близо до черната дупка.

Примерно извикване:

```
$ ./find_closest_point.sh point1.opt point2.opt 'Phoenix A'
point1.opt
```

Зад. 58 2023-IN-01

Вашите колеги от съседната лаборатория имат няколко GNU/Linux машини със специфични RAID контролери и ползват специализирана система за наблюдение Hugin. Помогнете на колегите си, като напишете shell скрипт, който взима температурата на дисковете, закачени към тези контролери, и генерира изход, удобен за ползване от системата за наблюдение.

За достъп до данните на дисковете, колегите ви казват, че root потребителя може да изпълни следната команда, която взима информация от даден контролер:

```
ssacli ctrl slot=x pd all show detail
```

където x е номер на PCI слот, в който е включен контролера. Примерен изход от командата:

```
Smart Array P420i in Slot 0 (Embedded)
   Array A
      physicaldrive 1I:2:1
         Rotational Speed: 15000
         Current Temperature (C): 35

      physicaldrive 1I:2:2
         Interface Type: SAS
         Current Temperature (C): 36
         SMR Support: None

      physicaldrive 1I:2:3
         Status: OK
         Drive Type: Spare Drive
         Current Temperature (C): 34
         Maximum Temperature (C): 42
```

Още един пример:

```
Smart Array P822 in Slot 1
   Array J
      physicaldrive 2E:1:19
         Current Temperature (C): 29

      physicaldrive 2E:1:20
         Current Temperature (C): 28

   Unassigned
      physicaldrive 2E:1:21
         Current Temperature (C): 30

      physicaldrive 2E:1:22
         Current Temperature (C): 29
```

Ако обобщим, изходът изглежда така:

- обща информация за контролера в съответния слот, и частност, *модел на контролера* (`P420i`/`P822`)
- една или повече секции на второ ниво, дефиниращи (нула или повече) масиви или (нула или една) секция с неизползвани дискове (`Array A`/`Array J`/`Unassigned`), като буквата (A, J, ...) дефинира *името на масива* в рамките на контролера
- във всяка такава секция има една или повече подсекции за дискове в съответния масив (`physicaldrive 1I:2:1`, `physicaldrive 2E:1:22`, и т.н.), където низът след physicaldrive е *име на този диск* в рамките на контролера
- във всяка подсекция за даден диск има *множество* параметри във вида *key name: value* като за колегите ви е интересен само параметърът `Current Temperature (C)`

Нека въведем следните термини:

- *идентификатор на диск* – низ във вида "`SSAcmmmappp`", където:
  - `SSA` – литерал
  - c – номер на слот
  - mmm – модел на контролер
  - a – име на масив или `UN` при неизползван диск
  - ppp – име на диск, без двуеточия
- *label на диск* – низ във вида "`SSAc mmm a qqq`", където
  - qqq – име на диск
  - останалите полета са по горната дефиниция
- *забележка:* кавичките не са част от идентификатора или от label-а на даден диск.

Възможно е да съществува *environment* променлива `CTRLSLOTS`, чиято стойност да е един или повече номера на PCI слотове, разделени с интервали, и в този случай скриптът трябва да извежда информация от тези контролери. Ако такава променлива не е дефинирана, скриптът по подразбиране да работи с контролера в слот 0.

Скриптът трябва да поддържа следните видове изпълнение:

- с един параметър `autoconf` – скриптът трябва да извежда низа `yes`
- с един параметър `config` – скриптът трябва да извежда следния изход:

```
graph_title SSA drive temperatures
graph_vlabel Celsius
graph_category sensors
graph_info This graph shows SSA drive temp
SSA0P420iA1I21.label SSA0 P420i A 1I:2:1
SSA0P420iA1I21.type GAUGE
SSA0P420iA1I22.label SSA0 P420i A 1I:2:2
SSA0P420iA1I22.type GAUGE
SSA0P420iA1I23.label SSA0 P420i A 1I:2:3
SSA0P420iA1I23.type GAUGE
```

```
SSA1P822J2E119.label SSA1 P822 J 2E:1:19
SSA1P822J2E119.type GAUGE
SSA1P822J2E120.label SSA1 P822 J 2E:1:20
SSA1P822J2E120.type GAUGE
SSA1P822UN2E121.label SSA1 P822 UN 2E:1:21
SSA1P822UN2E121.type GAUGE
SSA1P822UN2E122.label SSA1 P822 UN 2E:1:22
SSA1P822UN2E122.type GAUGE
```

- без параметри – скриптът трябва да извежда следния изход:

```
SSA0P420iA1I21.value 35
SSA0P420iA1I22.value 36
SSA0P420iA1I23.value 34
SSA1P822J2E119.value 29
SSA1P822J2E120.value 28
SSA1P822UN2E121.value 30
SSA1P822UN2E122.value 29
```

В горните изходи:

- низовете, започващи с `graph` са литерали
- низовете, в които се среща `.label`, `.type` или `.value` започват с *идентификатор на диск*
- низът след `.label` е *label на диск*
- `.type` винаги е `GAUGE`
- числото след `.value` е температурата на въпросния диск

Зад. 59 2023-IN-02

Вашите колеги от съседната лаборатория ползват специализиран софтуер за оптометрични изследвания, който записва резултатите от всяко измерване в отделен файл. Файловете имат уникално съдържание, по което се определя за кое измерване се отнася файла. За съжаление, тъй като колегите ви ползват бета версия на софтуера, той *понякога* записва по няколко пъти резултатите от дадено измерване в произволна комбинация от следните варианти:

- нула или повече отделни обикновенни файла с еднакво съдържание;
- нула или повече групи от hardlink-ове, като всяка група съдържа две или повече имена на даден файл с измервания.

Помогнете на колегите си, като напишете shell скрипт, който приема параметър – име на директория, съдържаща файлове с измервания. Скриптът трябва да извежда на стандартния изход списък с имена на файлове, кандидати за изтриване, по следните критерии:

- ако измерването е записано само в отделни файлове, трябва да остане един от тях;
- ако измерването е записано само в групи от hardlink-ове, всяка група трябва да се намали с едно име;
- ако измерването е записано и в групи, и като отделни файлове, за групите се ползва горния критерий, а всички отделни файлове се премахват.

Зад. 60 2023-SE-01

Напишете скрипт, който цензурира всички срещания на “забранени” думи в дадени текстове.

Примерно извикване: `./redact.sh bad_words.lst ./my_texts`.

Първият аргумент на скрипта е име на текстов файл, съдържащ по една забранена дума на ред:

```
cake
cakes
shake
banana
pine_apple42
shakinator
```

Вторият аргумент е име на директория: интересуват ни всички файлове в нея и в нейните поддиректории, чиито имена завършват на `.txt`.

Скриптът ви трябва да *подмени* всички срещания на забранени думи във въпросните файлове с брой звездички, съответстващ на дължината на думата. Подменят се само цели срещания на думи.

Например, ако имаме файл `./my_texts/shake.txt` със съдържание:

```
to make banana shake, we start by blending four bananas.
```

след изпълнение на скрипта, съдържанието му трябва да е:

```
to make ****** *****, we start by blending four bananas.
```

Под "дума" разбираме последователност от букви, цифри и долни черти.

За улеснение, може да приемете, че разглеждаме само малки букви (никъде не се срещат главни букви).

За бонус точки: премахнете улесненито, правейки операцията по цензуриране case-insensitive (файлът със забранени думи пак съдържа само малки букви, но в текстовете могат да се срещнат варианти на думите с произволни комбинации от малки и главни букви): ако в примерния текст се срещне думата `BaNaNa`, тя трябва да бъде заменена с `******`, защото `banana` е забранена дума.

Зад. 61 2023-SE-02

Задачата ви е да напишете скрипт `benchmark.sh`, който измерва средното време за изпълнение на дадена команда. Първият аргумент на скрипта е число (време за провеждане на експеримента, в секунди), а останалите аргументи на скрипта са измерваната команда и нейните аргументи.

Скриптът трябва да изпълнява подадената команда многократно, докато изтече подаденото време. Когато това се случи, скриптът трябва да изчака последното извикване на командата да приключи и да изведе съобщение, описващо броя направени извиквания, общото и средното време за изпълнение.

```
$ ./benchmark.sh 60 convert image.jpg result.png
Ran the command 'convert image.jpg result.png' 8 times for 63 seconds.
Average runtime: 7.88 seconds.

$ ./benchmark.sh 10 sleep 1.5
Ran the command 'sleep 1.5' 7 times for 10.56 seconds.
Average runtime: 1.51 seconds.
```

Забележки:

- Времената се извеждат в секунди, с точност два знака след запетайката.
- Приемете, че времето на изпълнение на частите от скрипта извън подадената команда е пренебрежимо малко.

Зад. 62 2023-SE-03

При статистическа обработка на текстове често се налага да имаме списък от думи (наречени "стоп-думи"), които се срещат прекалено често и не носят стойност за изследването. Такива думи са например "you", "then", "for" и т.н.

Напишете скрипт `stopword_candidates.sh`, който приема като аргумент име на директория и извежда 10-те думи, които най-много изглеждат като стоп-думи.

- За да бъде стоп-дума, трябва броят файлове, които я съдържат ≥ 3 пъти да е $\geq \frac{\text{общия брой файлове}}{2}$
- Една дума е по-добър кандидат от друга, ако има по-голям общ брой срещания във всички файлове

Забележки:

- Под "всички файлове" имаме предвид всички обикновени файлове в дадената директория и нейните поддиректории
- Под "дума" имаме предвид непрекъсната последователност от латински букви (a-z) - всички останали символи не са част от думите

- За улеснение може да приемете, че във файловете няма главни букви

Зад. 63 2023-SE-04

Напишете скрипт, който открива еднакви по съдържание файлове в дадена директория и използва тази информация, за да намали заетото място на диска.

Скриптът приема един параметър — име на директория. Примерно извикване: `./dedup.sh ./my-dir`

Скриптът трябва да направи две неща:

- ако има файлове с еднакво съдържание, да направи така, че имената на тези файлове да сочат към едно и също копие на съответните данни
- да изведе съобщение, съдържащо следната информация:
  - колко групи файлове са дедупликирани
  - колко байта от мястото на диска се е освободило

Забележки:

- считаме, че цялата дадена директория се намира върху една файлова система
- ако два файла имат еднакви хеш-суми, считаме, че са еднакви по съдържание

Зад. 64 2023-SE-05

Казваме, че командата `foo` заема памет X, когато X е сумата на заетата физическа памет (`rss`) на всички процеси, чиято команда (`comm`) е `foo`.

Напишете скрипт, който поглежда процесите на системата през една секунда, докато спрат да съществуват команди, чиято заета памет е над 65536.

След това, скриптът трябва да изведе всички команди, които са заемали памет над 65536 в поне половината "поглеждания".

Зад. 65 2023-SE-06

Напишете скрипт, който копира **снимки** от дадена директория (която ще наричаме **фотоапарата**) в нова директория (която ще наричаме **библиотеката**), спазвайки определена структура.

Скриптът приема пътищата на двете директории като аргументи, очаквайки, че **библиотеката** не съществува и трябва да я създаде.

Снимките са всички файлове в директорията на **фотоапарата** и нейните поддиректории, чиито имена завършват на `.jpg`. Под *дата* на снимка ще имаме предвид нейното време на модификация (mtime).

Библиотеката трябва да се състои от поддиректории с имена от вида `2023-04-20_2023-04-23`, означаващи **затворени** интервали от дни. Скриптът трябва да създаде такива поддиректории и да разположи снимките в тях, така че да са изпълнени следните свойства:

1. Всяка снимка се намира в такава директория, че датата на снимката да принадлежи на съответния интервал от дни
2. За всеки ден от всеки интервал съществува поне една снимка от този ден в този интервал
3. Няма припокриващи се и долепени интервали (между всеки два различни интервала има поне един ден без снимка)

Оригиналните имена на снимките не ни интересуват: скриптът трябва да им даде нови имена, използвайки техния mtime, с формат, изглеждащ така:

```
<библиотеката>/2023-04-20_2023-04-23/2023-04-21_13:04:33.jpg
```

Подсказка:

```
$ date -d '2000-02-29 + 1 day' +'%Y-%m-%d'
2000-03-01
```

Бонус за още няколко точки: направете скрипта така, че да може да работи при вече съществуваща библиотека, добавяйки снимките от фотоапарата към нея и запазвайки **всички** свойства. Ако снимка с дадено име вече съществува в библиотеката, нека скриптът я пропусне.

Зад. 66 2024-IN-01

Вашите колеги от съседната лаборатория се нуждаят от shell скрипт, с който да автоматизират синхронизацията на директории между различни сървъри. Скриптът трябва да е базиран на командата `rsync`, която има следният общ вид:

```
rsync [OPTION...] SRC DEST
```

В рамките на задачата ще ползваме следният опростен синтаксис за командата:

- при зададена опция `-a` и ако SRC и DEST са абсолютни пътища до директории завършващи с наклонена черта (`/path/to/dir/`), командата ще направи пълна синхронизация на всички обекти (файлове и директории) рекурсивно от SRC в DEST, запазвайки времената, правата и собствениците на обектите, които копира;
- двете директории SRC и DEST биха могли да са локални или *само една* от тях може да е директория на друг сървър, в който случай синтаксисът е `username@server:/path/to/dir/`
- синхронизацията винаги е от SRC към DEST, без значение дали и двете са локални директории или една от тях е отдалечена;
- при зададена опция `--delete` обекти, които съществуват в DEST, но не и в SRC, ще бъдат изтрити за да може DEST да е точно копие на SRC;
- при зададена опция `-v` командата извежда имената на обектите които синхронизира;
- при зададена опция `-n` реална синхронизация не се извършва, командата само пресмята кои обекти трябва да се синхронизират, и ако има указана и опция `-v` извежда имената им.

Напишете скрипт, който да може да автоматизира синхронизацията на директория между текущата машина (на която се изпълнява скрипта) и една или повече отдалечени машини. Работата на скрипта зависи от текстови конфигурационен файл, чието име *би трябвало* да е указано в environment променливата `ARKCONF`. Всеки ред на файла е във вид на `key=value` двойки, например:

```
WHAT="/home/amarr/sync"
WHERE="abaddon apostle archon augur"
WHO="heideran"
```

Където:

- `WHAT` дефинира абсолютен път до директорията, която трябва да се синхронизира, като дефиницията се отнася както за локалната директория (на машината на която се изпълнява скрипта), така и за отдалечените сървъри, т.е., директорията има еднакво абсолютно име на всички машини;
- `WHERE` дефинира списък със сървъри, с които да се извършва синхронизация, като е гарантирано, че имената на сървърите са само малки латински букви и първата буква е винаги ‘`a`’;
- `WHO` дефинира username, който да се ползва за достъп до отдалечените сървъри (всички сървъри ползват този username).

За нормална работа на вашия скрипт се изисква и трите ключа да са дефинирани, като за удобство приемаме, че стойностите им точно спазват примерния формат. Комбинацията от вашият скрипт и *валиден* конфигурационен файл ще наричаме *система* за синхронизация. Общ вид на извикване на скрипта: `ark.sh push|pull [-d] [server]` където:

- аргументите не са позиционни, т.е. нямат наредба;
- аргументите, оградени в квадратни скоби (`-d`, `server`) не са задължителни;
- ако е зададен *валиден за системата* параметър server, синхронизацията се извършва само между локалната машина и този сървър;
- ако не указан server, синхронизацията се извършва между локалната машина и всеки от сървърите *итеративно* (по сървърите);
- ако е зададена опция `-d`, синхронизацията трябва да се извършва с включена опция `--delete` на командата `rsync`;
- задължително трябва да е дефиниран точно един от аргументите `push` или `pull`, тъй като те са под-команди, които определят посоката на синхронизация:
  - при `push` – от локалната машина към отдалечените сървъри;
  - при `pull` – от отдалечените сървъри към локалната машина.

Примерни валидни извиквания:

```
ark.sh push -d
ark.sh -d abaddon pull
ark.sh apostle pull
```

Всяка синхронизация в рамките на работата на системата е тристъпкова – скриптът трябва да:

- изведе на потребителя какво ще бъде синхронизирано;
- изиска някакъв вид потвърждение от потребителя;
- извърши конкретната синхронизация.

Обърнете внимание на валидацията, искаме системата за синхронизация да е устойчива на невнимателно подадени параметри.

Зад. 67 2024-IN-02

Вашите колеги от съседната лаборатория се нуждаят от shell скрипт, с който да автоматизират създаването на графични диаграми, визуализиращи йерархии на наследяване на класове в C++ проекти. Самото създаване на диаграмите се извършва от инструмента `dag-ger`, който приема аргумент – име на текстови файл с описание на насочен ацикличен граф и генерира на стандартния изход визуализация на графа в SVG формат. Примерно съдържание на текстови файл с описание на граф:

```
Dog
Animal
Animal -> Dog
Object -> Dog
Thread -> Dog
Thread
Object
Alsatian
Dog -> Alsatian
```

Всеки ред на файла може да бъде в един от следните два формата:

- *node* – връх в графа, (`Dog`);
- *pnode* `->` *cnode* – ребро в графа, показващо връзката от родител (pnode) към наследник (cnode), (`Animal -> Dog`).

Всички файлове на даден C++ проект са в определена директория (и евентуално нейните под-директории) като *всеки клас* е деклариран в някой заглавен (header) файл. *Първият ред от декларацията на даден клас* е задължително в следния формат:

```
class CName : mode1 P1, mode2 P2,..., modeN PN
```

където:

- *CName* – име на класа; всичко след името на класа е незадължително;
- *mode1*, ..., *modeN* – спецификатор за достъп (`private`, `public` или `protected`);
- *P1*,...,*PN* – име на родителски клас.

Примерен първи ред *от декларацията на клас*:

```
class Dog : public Animal, private Object, protected Thread
```

Вашият скрипт трябва да приема два позиционни аргумента – път до директория, съдържаща (гарантирано валиден) C++ проект и име на изходен графичен файл за визуализирания граф. Графът трябва да визуализира пълната йерархия на наследяване на класовете в проекта.

Забележки: За удобство приемаме, че идентификаторите в C++ могат да съдържат малки/главни латински букви, цифри и подчертавка '_' и не могат да започват с цифра. За реализъм приемаме, че в кода няма коментари.

Зад. 68 2024-SE-01

Министерството на Истината ви поставя спешна задача: да напишете скрипт `replace.sh`, който замества думи в определени текстове.

`replace.sh` приема следните два типа валидни аргументи:

- Файл: произволен низ, не-започващ с тире, интерпретиран като име на файл
- Замяна: аргумент от вида `-R<дума1>=<дума2>`

Редът на аргументите няма значение, а броят им може да е произволен.

За всяка замяна, `replace.sh` трябва да промени всички файлове, така че всички срещания на `<дума1>` да бъдат заменени с `<дума2>`.

Забележки за поведението на скрипта:

- Приемаме, че думите съдържат само латински букви и цифри, а замяната счита главните и малките букви за различни (case sensitive).
- Заменят се само пълни думи
- Вече заменена дума не се заменя втори път. *Решения, не-спазващи това условие, се оценяват с най-много 8 точки.*
- Не може да разчитате, че фиксиран от вас низ не се среща в никой текст. Ако ви трябва низ, който гарантирано не се среща никъде, може да използвате `pwgen(1)`

Пример:

```
$ cat propaganda.txt
Our ally Eastasia is helping us in the war effort against Eurasia.

$ ./replace.sh -REastasia=Eurasia -REurasia=Eastasia propaganda.txt

$ cat propaganda.txt
Our ally Eurasia is helping us in the war effort against Eastasia.
```

Бонус: Скриптът да не извършва никаква замяна в редове, които започват със символа #

Зад. 69 2024-SE-02

Ваши колеги използват система за съхранение на файлове *PrevCloud*, която има собствена система за потребители, различна от тази за локални потребители на операционната система (`/etc/passwd`). Колегите ви са ви **дали на готово** компилирана програма `occ`, с която се управляват потребителите на *PrevCloud*, и ви молят да напишете скрипт `sync-users.sh`, който обновява потребителите на *PrevCloud* спрямо локалните потребители.

Ще разграничаваме:

- Потребители на *PrevCloud*, до които имаме достъп чрез `occ`
- Локални потребители - потребители на операционната система с UID ≥ 1000
- Системни потребители - потребители на операционната система с UID < 1000 (те не се използват в задачата)

Документация на подкомандите на `occ`, предоставена от вашите колеги:

- `./occ user:list` – дава списък от *PrevCloud* потребители във вида `- USERNAME: DISPLAYNAME`, като в задачата се интересуваме единствено от *USERNAME*. Примерен изход:

  ```
  - rms: Richard Stallman
  - dennis: Dennis Ritchie
  - brian: Brian Kernighan
  ```

- `./occ user:info USERNAME` – дава информация за *PrevCloud* потребителя *USERNAME*. Примерен изход:

  ```
  - user_id: rms
  - display_name: Richard Stallman
  - email: rms@gnu.org
  - enabled: true
  - quota: none
  ```

- `./occ user:add USERNAME` – създава нов *PrevCloud* потребител с име *USERNAME*; няма ефект и завършва неуспешно, ако той вече съществува

- `./occ user:disable USERNAME` – заключва профила на *PrevCloud* потребителя *USERNAME* (това съответства на полето `enabled` в изхода на `user:info`); няма ефект и завършва неуспешно, ако той вече е заключен
- `./occ user:enable USERNAME` – отключва профила на *PrevCloud* потребителя *USERNAME* (това съответства на полето `enabled` в изхода на `user:info`); няма ефект и завършва неуспешно, ако той вече е отключен

Вашата задача:

Скриптът `sync-users.sh` трябва:

- Ако за даден локален потребител няма съответен *PrevCloud* потребител със същия *USERNAME*, да създаде *PrevCloud* потребител за него.
- Ако за даден локален потребител съществува *PrevCloud* потребител, но той е заключен, да отключи съответния *PrevCloud* потребител.
- Ако за даден *PrevCloud* потребител не съществува локален потребител, да заключи съответния *PrevCloud* потребител.

Забележки за поведението на скрипта:

- С цел по-лесно тестване, ако съществува environment променливата `PASSWD`, нека скриптът използва файла, чието име е посочено в нея, вместо `/etc/passwd`, за да вземе локалните потребители.
- Пълен брой точки ще получат решения, в които `occ` командите за добавяне, заключване и отключване на потребител не се извикват излишно (напр. не се извиква `./occ user:enable` за *PrevCloud* потребител, който не е заключен).
- Програмата `occ` се намира в същата директория като скрипта `sync-users.sh`. Подсигурете скриптът `sync-users.sh` да може да бъде изпълняван от произволна текуща директория.

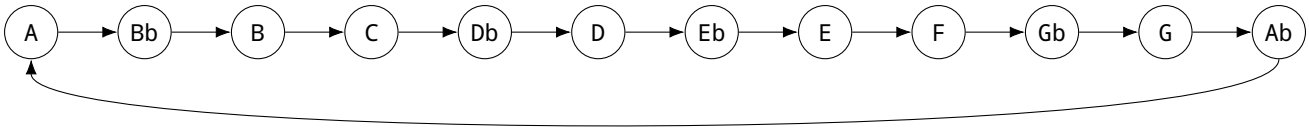
Зад. 70 2024-SE-03

ChordPro е текстов формат за представяне на текстове на песни, анотирани с акорди. Примерен откъс:

```
For [Fmaj7]here am I [Em]sitting in a tin can
[Fmaj7]Far above the [Em]world
[Bb]Planet Earth is [Am]blue and there's [G]nothing I can [F]do
```

Всяка поредица от символи, оградена в квадратни скоби, ще наричаме *акорд*. Първите един или два символа на акорда наричаме *основен тон*.

Напишете скрипт `transpose.sh`, който приема един аргумент (неотрицателно число N). Скриптът трябва да чете текст във формàта *ChordPro* от `stdin` и да изписва аналогичен текст на `stdout`, заменяйки единствено основните тонове по следната схема:



Числото N задава брой преходи по ребрата за замяна. Например, при изпълнение на скрипта с $N = 3$ върху горния пример, резултатът би бил:

```
For [Abmaj7]here am I [Gm]sitting in a tin can
[Abmaj7]Far above the [Gm]world
[Db]Planet Earth is [Cm]blue and there's [Bb]nothing I can [Ab]do
```

Забележки:

- Моля, избягвайте повторение на код.
- Ако свирите на тромпет, използвайте $N = 3$. За валдхорна, използвайте $N = 8$.

Зад. 71 2024-SE-04

Напишете скрипт `bake.sh`, който приема един задължителен аргумент - име на файл. Скриптът трябва да реализира build система, която *изгражда* дадения файл по следния алгоритъм.

Правила за изграждане на файлове

Скриптът се управлява от файл с име `bakefile`, който *би трябвало* да съществува в текущата директория, от която извикваме `bake.sh`. Файлът `bakefile` описва правила, по които се изграждат различните файлове, които `bake.sh` може да изгражда. Всеки ред дефинира правило за някой файл във вида:

```
<файл>:<зависимост 1> <зависимост 2> ... <зависимост N>:<команда>
```

Където:

- `<файл>`: името на файла, за който се отнася този ред
- `<зависимост ...>`: имена на файлове, които трябва да бъдат изградени преди да може да се изгради файлът `<файл>`. Може да приемете, че зависимостите няма да образуват цикъл.
- `<команда>`: произволна команда, която изгражда файла. Може да приемете, че след извикването на тази команда, ще е бил създаден файл с име `<файл>`.

Алгоритъм за изграждане на файл

- Ако за файла **има** запис в `bakefile`:
  1. Изграждаме всички негови зависимости по *същия* алгоритъм
  2. Изпълняваме изграждащата команда, но **само ако** ако някоя от зависимостите е по-нова от самия файл
- Ако за файла **няма** запис в `bakefile`:
  - Ако файлът съществува, `bake.sh` го приема за изграден и не прави нищо
  - Ако файлът не съществува, това води до грешка и няма как да продължим нататък

Пример

Нека е даден следният `bakefile`:

```
chapter1.pdf:chapter1.md:pandoc -o chapter1.pdf chapter1.md
chapter2.pdf:chapter2.md drawing42.svg:pandoc -o chapter2.pdf chapter2.md
book.pdf:chapter1.pdf chapter2.pdf:pdfunite chapter1.pdf chapter2.pdf book.pdf
drawing42.svg:drawing42.dia:dia_to_svg drawing42.dia > drawing42.svg
```

Ако извикваме `./bake.sh book.pdf`, скриптът ще трябва да изгради `book.pdf`. За целта първо трябва да изгради зависимостите на `book.pdf` (`chapter1.pdf` и `chapter2.pdf`, които пък от своя страна имат други зависимости), след което да изпълни командата `pdfunite chapter1.pdf chapter2.pdf book.pdf`, за която приемаме, че ще създаде `book.pdf` (без да се интересуваме каква е командата и какво точно прави, стига да е завършила успешно).

Зад. 72 2024-SE-05

Напишете скрипт `baseline.sh`, който уведомява потребителя, ако текущата стойност на дадена величина се различава много от историческите ѝ стойности, измерени предни седмици.

`baseline.sh` трябва да приема 2 аргумента:

- външна команда, измерваща търсената стойност: при изпълнение, тази команда изписва число с десетична точка
- път до файл, в който `baseline.sh` трябва да съхранява историческите данни във формат по ваш избор

Приемете, че потребителят е подсигурил, че `baseline.sh` ще бъде изпълняван на всеки час със еднакви аргументи. Скриптът трябва да изпълни външната команда, съхранявайки върнатата стойност и текущото време във файла с исторически данни. За всяко такова изпълнение, разглеждаме историческите стойности за същия ден от седмицата и час:

- Ако командата е завършила неуспешно, скриптът не извежда нищо и приключва със статус 3
- Ако текущата стойност се различава повече от двойно от средната историческа стойност, скриптът трябва да приключи със статус 2, след като изведе съобщение от вида:
  `YYYY-MM-DD HH: VVV.VVVV abnormal`
  където `YYYY-MM-DD HH` са датата и часа на измерването, а `VVV.VVVV` е измерената стойност
- В противен случай, скриптът не извежда нищо

Например, ако в момента е четвъртък, `2024-08-22 15:08`, скриптът трябва да провери средната стойност на всички измервания, направени в четвъртъци между 15 и 16 часа. Ако тази средна стойност е α, а текущата стойност е ξ, и не е вярно, че $\frac{\alpha}{2} \leq \xi \leq 2\alpha$, скриптът трябва да изведе съобщение.

Зад. 73 2024-SE-06

Напишете скрипт `reconcile.sh`, който по подаден файл описващ желано състояние на файловата система, осигурява, че то е спазено.

Скриптът ви трябва да приема един аргумент - път до файл. Всеки ред в този файл **ще бъде** в един от следните формати:

- <filename> file <permissions>
  Скриптът ви трябва да подсигури, че съществува обикновеният файл <filename> с права за достъп <permissions>.
- <dirname> dir <permissions>
  Скриптът ви трябва да подсигури, че съществува директорията <dirname> с права за достъп <permissions>.
- <linkname> symlink <target>
  Скриптът ви трябва да подсигури, че съществува файлът <linkname> от тип symbolic link, който да сочи към файла <target>.
- <filename> nonexistant
  Скриптът ви трябва да подсигури, че файлът <filename> не съществува.

Преди всяка поява на <permissions> е възможно да е подадена и незадължителна двойка <user owner>:<group owner>, които представляват потребител собственик и група собственик за съответния файл. Скриптът ви трябва да подсигури, че файлът ще притежава именно тези собственици.

Примерно съдържание на файла:

```
/home dir root:users 0755
/home/pesho dir pesho:pesho 0700
/home/pesho/some_file file 0644
/home/pesho/some_dir dir 0755
/home/pesho/some_dir/some_link symlink ../some_file
/home/pesho/some_dir file 0755
/home/pesho/some_other_dir nonexistant
```

За всеки <filename>, <dirname> и <linkname>, ако съответният файл:

- вече съществува, но е от друг тип, то следва той да бъде изтрит и пресъздаден с желания тип и метаданни.
- вече съществува и е от желания тип, то трябва единствено метаданните да бъдат обновени, ако е нужно.
- не съществува, то той трябва да бъде създаден. Ако съответната директория, в която се намира, не съществува, тя трябва да бъде създадена спрямо текущата маска.

Възможно е няколко реда във файла да реферират към едно и също име във файловата система. В такъв случай приемаме, че последният ред е финалното желано състояние.

ВАЖНО: Скриптът ви може да бъде изпълняван от всякакъв потребител във всякаква среда. Важна е проверката за грешки и обратната връзка към потребителя при неуспешна операция. При неуспешна операция, скриптът ви не трябва да приключва работа.

Зад. 74 2025-IN-01

Вашите колеги от съседната лаборатория трябва периодично да променят правата за достъп до обекти във файловата система на база на текстови файл с описание на необходимата конфигурация. Всеки ред на файла дефинира *правило*, което гарантирано се състои от три низа, разделени с whitespace. Примерно съдържание на файла:

```
/tmp/one R 700
/tmp/two A 700
/tmp/four A 750
/tmp/twenty T 750
```

Всяко правило описва на кои обекти трябва да се променят правата, като трите низа определят:

- *начална директория* в която да се търсят обекти за това правило; не съдържа специални символи;
- *тип* – една от следните три букви, която дефинира как се интерпретират битовете в третия низ:
  - `R` (raw) – обектът трябва да има точно тези битове за права
  - `A` (any) – обектът трябва да има вдигнат поне един от указаните битове
  - `T` (target) – обектът трябва да има вдигнати всичките указани битове
- девет *бита*, дефиниращи права за достъп във файлова система, представени по стандартния начин като трицифрено число в осмична бройна система.

Напишете bash скрипт `perm.sh`, който по подаден аргумент – име на файл с конфигурация, ако се изпълнява от root потребителя, извършва необходмите промени на правата при следните изисквания:

- правилата се обработват и прилагат линейно;
- като обекти във файловата система се обработват само обикновени файлове и директории;
- всички обекти биха могли да имат специални символи в името си;
- началните директории не се третират като обекти, чиито права трябва да се променят.

Правата на намерените обекти се **променят** на каквито биха били, ако съответния обект е бил създаден с ефективна `umask`-а съответно:

- `022` за директории и
- `002` за файлове.

Зад. 75 2025-IN-02

Вашите колеги от съседната лаборатория имат нужда да автоматизират периодичното създаване на определен вид конфигурационни файлове на база на съществуващ текстови файл (`map.txt`) с изисквания. Всеки ред на файла гарантирано се състои от три думи (малки латински букви), разделени с whitespace. Примерно съдържание на файла:

```
alfa  haydn degas
bravo rossini renoir
charlie schubert  monet
delta berlioz monet
echo  chopin  renoir
foxtrot liszt monet
```

Семантично думите на всеки ред дефинират:

- име на *сървър* (hostname) – `alfa`, `bravo`, `charlie`...
- име на *зона* (composer) – `haydn`, `rossini`, `schubert`...
- име на *екип* (artist) – `degas`, `renoir`, `monet`...

Общо файлът `map.txt` описва множество от екипи, като всеки екип съдържа един или повече сървъра; а всеки сървър има точно една зона.

Помогнете на колегите си, като напишете bash скрипт `foo.sh`, който приема два позиционни параметъра – низ, дефиниращ *domain* и име на файл с описание.

Скриптът трябва да извежда на стандартния изход редове, съдържащи релация за всяка зона към всеки сървър на съответния екип в указания по-долу формат. Имената на сървърите са допълнени с подадения domain и завършват с точка, като редовете с релациите за даден екип се предхождат от ред, започващ със знак за коментар (`;`) с името на екипа.

Примерно извикване и изход на база горния файл:

```
$ foo.sh example.net map.txt

; team degas
haydn  IN  NS  alfa.example.net.
; team monet
schubert  IN  NS  charlie.example.net.
schubert  IN  NS  delta.example.net.
```

```
schubert  IN  NS  foxtrot.example.net.
berlioz  IN  NS  charlie.example.net.
berlioz  IN  NS  delta.example.net.
berlioz  IN  NS  foxtrot.example.net.
liszt  IN  NS  charlie.example.net.
liszt  IN  NS  delta.example.net.
liszt  IN  NS  foxtrot.example.net.
; team renoir
rossini  IN  NS  bravo.example.net.
rossini  IN  NS  echo.example.net.
chopin  IN  NS  bravo.example.net.
chopin  IN  NS  echo.example.net.
```

Зад. 76 2025-SE-01

Ваш бивш съученик, с когото не сте се чували от 11 години, се свързва с вас и спешно иска помощ с автоматизация на използването на външния инструмент *ANTLR*, който вие имате вече инсталиран на компютъра си.

Напишете скрипт `build.sh`, който се управлява с конфигурационни файлове, имащи следния вид:

```
<език> <желани типове изходни файлове> '<базова изходна директория>'
```

Примерен такъв конфигурационен файл:

```
Go recognizer listener '/foo'
JavaScript visitor recognizer '/bar'
Java listener recognizer visitor '/baz'
```

`build.sh` трябва да приема два аргумента:

- Име на конфигурационен файл (подобен на примерния от по-горе)
- Име на входен файл (съдържанието му не ни интересува)

`build.sh` трябва да извика ANTLR по веднъж за всеки ред от конфигурационния файл, така че да създаде нужните изходни файлове в съответните изходни директории.

Извикването става по следния начин:

```
antlr4 -Dlanguage=<език> <опции> -o <изходна директория> <входен файл>
```

Изходната директория трябва да е поддиректория с име, съответстващо на името на входния файл, но без разширение, и намираща се в базовата изходна директория.

Инструментът ANTLR генерира изходни файлове спрямо следните правила:

- Recognizer файловете се генерират винаги
- Listener файловете се генерират ако **не е** подадена опция `-no-listener`
- Visitor файловете се генерират ако **е** подадена опция `-visitor`

Ако приемем, че конфигурационният файл се казва `Antlrfile`, примерното извикване

```
./build.sh Antlrfile grammar/Query.g4
```

трябва да доведе до извикване на следните команди:

```
antlr4 -Dlanguage=Go -o /foo/Query grammar/Query.g4
antlr4 -Dlanguage=JavaScript -no-listener -visitor -o /bar/Query grammar/Query.g4
antlr4 -Dlanguage=Java -visitor -o /baz/Query grammar/Query.g4
```

ANTLR сам ще се погрижи да създаде нужните файлове в съответните изходни директории.

Зад. 77 2025-SE-02

Напишете скрипт `manage.sh`, който управлява *сървиси*. Скриптът очаква да съществува environment променливата `SVC_DIR`, съдържаща име на директория, в която се намират описанията на сървисите. Всеки обикновен файл в `SVC_DIR` описва един сървис (имената на файловете не ни интересуват).

Файловете имат следния вид:

```
name: <име на сървис>
pidfile: <име на pid файл>
outfile: <име на файл>
comm: <shell команда>
```

Примерен файл (наредбата на редовете няма значение):

```
comm: while true; do echo ZzZ; sleep 1; done
outfile: /tmp/sleeper_out
pidfile: /tmp/sleeper_pid
name: sleeper
```

`manage.sh` трябва да може да се извиква по следните начини:

- `./manage.sh start <име на сървис>` – стартира дадения сървис, което означава:
  - Стартира процес: shell, изпълняващ командата (`comm`) на дадения сървис във фонов режим
  - Записва PID-а на стартирания процес в съответния `pidfile`
  - Пренасочва `stdout` и `stderr` в изходния файл (`outfile`)
  - Ако сървисът вече работи, вместо гореизброените стъпки, *start* не прави нищо
- `./manage.sh stop <име на сървис>` – убива процеса на сървиса със SIGTERM
- `./manage.sh running` – изписва имената на всички сървиси, чиито процеси (спрямо `pidfile`) още работят, по едно на ред, сортирани лексикографски
- `./manage.sh cleanup` – изтрива PID-файловете на всички сървиси, чиито процеси не съществуват

Примерно извикване: `./manage.sh start sleeper`

2 Използване на системни примитиви в програми на C

Забележки:

- Полезни man страници:

```
open(2)     close(2)    read(2)     write(2)    lseek(2)    stat(2)     printf(3)
malloc(3)   fork(2)     wait(2)     exec(3)     kill(2)     exit(3)     err(3)
pipe(2)     dup(2)      mkfifo(3)   time(2)     sleep(3)
strlen(3)   strcmp(3)   qsort(3)    strtol(3)
```

- Задачите ви трябва да се компилират с осигурения `Makefile`.
- Всички функции от `<stdio.h>` освен `snprintf(3)` са **забранени**
- Обърнете внимание на коментарите, именуването на променливи и подреждането на кода.
- За всички задачи, освен ако не е указано друго:
  - програмата трябва да проверява за грешки при всички системни извиквания освен `close(2)`.
  - не е нужно да се освобождават ресурси преди извикване на `err(3)`
  - за удобство приемаме, че работим само с Little-endian машини;
  - не е гарантирано, че разполагате с достатъчно памет, в която да заредите всички данни.

2.1 Вход и изход (тема 6)

Зад. 78 2016-SE-01 Напишете програма на C, която приема параметър - име на (двоичен) файл с байтове. Програмата трябва да сортира файла.

Зад. 79 2016-SE-02 Двоичните файлове f1 и f2 съдържат 32 битови числа без знак (`uint32_t`). Файлът f1 е съдържа n двойки числа, нека i-тата двойка е $< x_i, y_i >$. Напишете програма на C, която извлича интервалите с начало x_i и дължина y_i от файла f2 и ги записва залепени в изходен файл f3.

Пример:

- f1 съдържа 4 числа (2 двойки): 30000, 20, 19000, 10
- програмата записва в f3 две поредици 32-битови числа, взети от f2 както следва:
- най-напред се записват числата, които са на позиции 30000, 30001, 30002, ... 30019.
- след тях се записват числата от позиции 19000, 19001, ... 19009.

Забележка: С пълен брой точки ще се оценяват решения, които работят със скорост, пропорционална на размера на изходния файл f3.

Зад. 80 2016-SE-03 Напишете програма на C приемаща параметър – име на (двоичен) файл с `uint32_t` числа. Програмата трябва да сортира файла. Ограничения:

- Числата биха могли да са максимум 100 000 000 на брой.
- Програмата трябва да работи на машина със същия endianness, както машината, която е създала файла.
- Програмата трябва да работи на машина с 256 MB RAM и 8 GB свободно дисково пространство.

Зад. 81 2017-IN-01 Напишете програма на C, която приема четири параметъра – имена на двоични файлове.

Примерно извикване:

```
$ ./main f1.dat f1.idx f2.dat f2.idx
```

Първите два (`f1.dat` и `f1.idx`) и вторите два (`f2.dat` и `f2.idx`) файла са *входен* и *изходен комплект* със следния смисъл:

- `DAT`-файловете (`f1.dat` и `f2.dat`) представляват двоични файлове, състоящи се от байтове (`uint8_t`);
- `IDX`-файловете представляват двоични файлове, състоящи се от наредени тройки от следните елементи (и техните типове), които дефинират поредици от байтове (низове) от съответния `DAT` файл:
  - *отместване* `uint16_t` – показва позицията на първия байт от даден низ спрямо началото на файла;
  - *дължина* `uint8_t` – показва дължината на низа;
  - *запазен* `uint8_t` – не се използва.

Първата двойка файлове (`f1.dat` и `f1.idx`) съществува, а втората трябва да бъде създадена от програмата по следния начин:

- трябва да се копират само низовете (поредици от байтове) от входния комплект, които започват с главна латинска буква (`A - 0x41, Z - 0x5A`).
- ако файловете са неконсистентни по някакъв начин, програмата да прекратява изпълнението си по подходящ начин.

Забележка: За удобство приемаме, че DAT файлът съдържа текстови данни на латински с ASCII кодова таблица (един байт за буква).

Примерен вход и изход:

```
$ xxd f1.dat
00000000: 4c6f 7265 6d20 6970 7375 6d20 646f 6c6f  Lorem ipsum dolo
00000010: 7220 7369 7420 616d 6574 2c20 636f 6e73  r sit amet, cons
00000020: 6563 7465 7475 7220 6164 6970 6973 6369  ectetur adipisci
00000030: 6e67 2065 6c69 742c 2073 6564 2064 6f20  ng elit, sed do
00000040: 6569 7573 6d6f 6420 7465 6d70 6f72 2069  eiusmod tempor i
00000050: 6e63 6964 6964 756e 7420 7574 206c 6162  ncididunt ut lab
00000060: 6f72 6520 6574 2064 6f6c 6f72 6520 6d61  ore et dolore ma
00000070: 676e 6120 616c 6971 7561 2e20 5574 2065  gna aliqua. Ut e
00000080: 6e69 6d20 6164 206d 696e 696d 2076 656e  nim ad minim ven
00000090: 6961 6d2c 2071 7569 7320 6e6f 7374 7275  iam, quis nostru
000000a0: 6420 6578 6572 6369 7461 7469 6f6e 2075  d exercitation u
000000b0: 6c6c 616d 636f 206c 6162 6f72 6973 206e  llamco laboris n
000000c0: 6973 6920 7574 2061 6c69 7175 6970 2065  isi ut aliquip e
000000d0: 7820 6561 2063 6f6d 6d6f 646f 2063 6f6e  x ea commodo con
000000e0: 7365 7175 6174 2e20 4475 6973 2061 7574  sequat. Duis aut
000000f0: 6520 6972 7572 6520 646f 6c6f 7220 696e  e irure dolor in
00000100: 2072 6570 7265 6865 6e64 6572 6974 2069   reprehenderit i
00000110: 6e20 766f 6c75 7074 6174 6520 7665 6c69  n voluptate veli
00000120: 7420 6573 7365 2063 696c 6c75 6d20 646f  t esse cillum do
00000130: 6c6f 7265 2065 7520 6675 6769 6174 206e  lore eu fugiat n
00000140: 756c 6c61 2070 6172 6961 7475 722e 2045  ulla pariatur. E
00000150: 7863 6570 7465 7572 2073 696e 7420 6f63  xcepteur sint oc
00000160: 6361 6563 6174 2063 7570 6964 6174 6174  caecat cupidatat
00000170: 206e 6f6e 2070 726f 6964 656e 742c 2073   non proident, s
00000180: 756e 7420 696e 2063 756c 7061 2071 7569  unt in culpa qui
00000190: 206f 6666 6963 6961 2064 6573 6572 756e   officia deserun
000001a0: 7420 6d6f 6c6c 6974 2061 6e69 6d20 6964  t mollit anim id
000001b0: 2065 7374 206c 6162 6f72 756d 2e0a        est laborum..

$ xxd f1.idx
00000000: 0000 0500 4f01 0200 4e01 0300            ....O...N...

$ xxd f2.dat
00000000: 4c6f 7265 6d45 78                        LoremEx

$ xxd f2.idx
00000000: 0000 0500 0500 0200                      ........
```

Зад. 82 2017-SE-01 Напишете програма на C, която приема три параметъра – имена на двоични файлове.

Примерно извикване:

```
$ ./main f1.bin f2.bin patch.bin
```

Файловете `f1.bin` и `f2.bin` се третират като двоични файлове, състоящи се от байтове (`uint8_t`). Файлът `f1.bin` е "оригиналният" файл, а `f2.bin` е негово копие, което е било модифицирано по някакъв

начин (извън обхвата на тази задача). Файлът `patch.bin` е двоичен файл, състоящ се от наредени тройки от следните елементи (и техните типове):

- *отместване* (`uint16_t`) – спрямо началото на `f1.bin`/`f2.bin`
- *оригинален байт* (`uint8_t`) – на тази позиция в `f1.bin`
- *нов байт* (`uint8_t`) – на тази позиция в `f2.bin`

Вашата програма да създава файла `patch.bin`, на базата на съществуващите файлове `f1.bin` и `f2.bin`, като описва вътре само разликите между двата файла. Ако дадено отместване съществува само в едния от файловете `f1.bin`/`f2.bin`, програмата да прекратява изпълнението си по подходящ начин.

Примерен `f1.bin`:

```
00000000: f5c4 b159 cc80 e2ef c1c7 c99a 2fb0 0d8c  ...Y......../...
00000010: 3c83 6fed 6b46 09d2 90df cf1e 9a3c 1f05  <.o.kF.......<..
00000020: 05f9 4c29 fd58 a5f1 cb7b c9d0 b234 2398  ..L).X...{...4#.
00000030: 35af 6be6 5a71 b23a 0e8d 08de def2 214c  5.k.Zq.:......!L
```

Примерен `f2.bin`:

```
00000000: f5c4 5959 cc80 e2ef c1c7 c99a 2fb0 0d8c  ..YY......../...
00000010: 3c83 6fed 6b46 09d2 90df cf1e 9a3c 1f05  <.o.kF.......<..
00000020: 05f9 4c29 fd58 a5f1 cb7b c9d0 b234 2398  ..L).X...{...4#.
00000030: afaf 6be6 5a71 b23a 0e8d 08de def2 214c  ..k.Zq.:......!L
```

Примерен `patch.bin`:

```
00000000: 0200 b159 3000 35af                      ...Y0.5.
```

Зад. 83 2017-SE-02 Напишете програма на C, която да работи подобно на командата `cat`, реализирайки само следната функционалност:

- общ вид на изпълнение: `./main [OPTION] [FILE]...`
- ако е подаден като **първи** параметър `-n`, то той да се третира като опция, което кара програмата ви да номерира (глобално) всеки изходен ред (започвайки от 1).
- програмата извежда на `STDOUT`
- ако няма подадени параметри (имена на файлове), програмата чете от `STDIN`
- ако има подадени параметри – файлове, програмата последователно ги извежда
- ако някой от параметрите е тире (`-`), програмата да го третира като специално име за `STDIN`

Примерно извикване:

```
$ cat a.txt
a1
a2

$ cat b.txt
b1
b2
b3

$ echo -e "s1\ns2" | ./main -n a.txt - b.txt
     1 a1
     2 a2
     3 s1
     4 s2
     5 b1
     6 b2
     7 b3
```

Забележка: Погледнете `setbuf(3)` *и* `strcmp(3)`.

Зад. 84 2017-SE-03 Напишете програма на C, която приема три параметъра, имена на двоични файлове.

Примерно извикване:

```
$ ./main patch.bin f1.bin f2.bin
```

Файловете `f1.bin` и `f2.bin` се третират като двоични файлове, състоящи се от байтове (`uint8_t`). Файлът `patch.bin` е двоичен файл, състоящ се от наредени тройки от следните елементи (и техните типове):

- *отместване* `uint16_t`
- *оригинален байт* `uint8_t`
- *нов байт* `uint8_t`

Програмата да създава файла `f2.bin` като копие на файла `f1.bin`, но с отразени промени на базата на файла `patch.bin`, при следния алгоритъм:

- за всяка наредена тройка от `patch.bin`, ако на съответното *отместване* (в байтове) спрямо началото на файла е записан байта *оригинален байт*, в изходния файл се записва *нов байт*. Ако не е записан такъв *оригинален байт* или такова *отместване* не съществува, програмата да прекратява изпълнението си по подходящ начин;
- всички останали байтове се копират директно.

Забележка: Наредените тройки във файла `patch.bin` да се обработват последователно.

Примерен `f1.bin`:

```
00000000: f5c4 b159 cc80 e2ef c1c7 c99a 2fb0 0d8c  ...Y......../...
00000010: 3c83 6fed 6b46 09d2 90df cf1e 9a3c 1f05  <.o.kF.......<..
00000020: 05f9 4c29 fd58 a5f1 cb7b c9d0 b234 2398  ..L).X...{...4#.
00000030: 35af 6be6 5a71 b23a 0e8d 08de def2 214c  5.k.Zq.:......!L
```

Примерен `patch.bin`:

```
00000000: 0200 b159 3000 35af                      ...Y0.5.
```

Примерен `f2.bin`:

```
00000000: f5c4 5959 cc80 e2ef c1c7 c99a 2fb0 0d8c  ..YY......../...
00000010: 3c83 6fed 6b46 09d2 90df cf1e 9a3c 1f05  <.o.kF.......<..
00000020: 05f9 4c29 fd58 a5f1 cb7b c9d0 b234 2398  ..L).X...{...4#.
00000030: afaf 6be6 5a71 b23a 0e8d 08de def2 214c  ..k.Zq.:......!L
```

Зад. 85 2017-SE-04 Напишете програма на C, която да работи подобно на командата `cat`, реализирайки само следната функционалност:

- програмата извежда на `STDOUT`
- ако няма подадени параметри, програмата чете от `STDIN`
- ако има подадени параметри – файлове, програмата последователно ги извежда
- ако някой от параметрите започва с тире (`-`), програмата да го третира като специално име за `STDIN`

Примерно извикване:

```
$ ./main f - g
```

– извежда съдържанието на файла `f`*, после* `STDIN`*, след това съдържанието на файла* `g`*.*

Зад. 86 2018-SE-01 Напишете програма на C, която да работи подобно на командата `tr`, реализирайки само следната функционалност:

- програмата чете от стандартния вход и пише на стандартния изход

- общ вид на изпълнение: `./main [OPTION] SET1 [SET2]`
- `OPTION` би могъл да бъде или `-d`, или `-s`, или да липсва
- `SET1` и `SET2` са низове, в които знаците нямат специално значение, т.е. всеки знак "*означава*" съответния знак. `SET2`, ако е необходим, трябва да е същата дължина като `SET1`
- ако е подаден като *първи* параметър `-d`, програмата трие от входа всяко срещане на знак μ от `SET1`; `SET2` не е необходим
- ако е подаден като *първи* параметър `-s`, програмата заменя от входа всяка поредица от повторения знак μ от `SET1` с еднократен знак μ; `SET2` не е необходим
- в останалите случаи програмата заменя от входа всеки знак μ от `SET1` със съответстващият му позиционно знак ν от `SET2`.

Примерно извикване:

```
$ echo asdf | ./main -d 'd123' | ./main 'sm' 'fa' | ./main -s 'f'
af
```

Зад. 87 2018-SE-02 Напишете програма на C, която приема два параметъра – имена на файлове:

- примерно извикване: `./main input.bin output.bin`
- файловете `input.bin` и `output.bin` се третират като двоични файлове, състоящи се от `uint32_t` числа
- файлът `input.bin` може да съдържа максимум 4194304 числа
- файлът `output.bin` трябва да бъде създаден от програмата и да съдържа числата от `input.bin`, сортирани във възходящ ред
- *endianness*-ът на машината, създала файла `input.bin` е същият, като на текущата машина
- *ограничения на ресурси:* програмата трябва да работи с употреба на максимум 9 MB RAM и 64 MB дисково пространство.

Зад. 88 2018-SE-03 Напишете програма на C, която да работи подобно на командата `cut`, реализирайки само следната функционалност:

- програмата трябва да чете текст от стандартния вход и да извежда избраната част от всеки ред на стандартния изход;
- ако първият параметър на програмата е низът `-c`, тогава вторият параметър е или едноцифрено число (от 1 до 9), или две едноцифрени числа N и M ($N \leq M$), разделени с тире (напр. `3-5`). В този случай програмата трябва да изведе избраният/избраните символи от реда: или само символа на указаната позиция, или няколко последователни символи на посочените позиции.
- ако първият параметър на програмата е низът `-d`, тогава вторият параметър е низ, от който е важен само първият символ; той се използва като разделител между полета на реда. Третият параметър трябва да бъде низът `-f`, а четвъртият - или едноцифрено число (от 1 до 9), или две едноцифрени числа N и M ($N \leq M$), разделени с тире (напр. `3-5`). В този случай програмата трябва да разглежда реда като съставен от няколко полета (може би празни), разделени от указания символ (първият символ от низа, следващ парметъра `-d`), и да изведе или само указаното поле, или няколко последователни полета на указаните позиции, разделени от същия разделител.
- ако някой ред няма достатъчно символи/полета, за него програмата трябва да изведе каквото (докъдето) е възможно (или дори празен ред).

Зад. 89 2018-SE-04 Напишете програма на C, която приема два параметъра – имена на файлове:

- примерно извикване: `./main input.bin output.bin`
- файловете `input.bin` и `output.bin` се третират като двоични файлове, състоящи се от `uint16_t` числа
- файлът `input.bin` може да съдържа максимум 65535 числа
- файлът `output.bin` трябва да бъде създаден от програмата и да съдържа числата от `input.bin`, сортирани във възходящ ред
- *endianness*-ът на машината, създала файла `input.bin` е същият, като на текущата машина
- *ограничения на ресурси:* програмата трябва да работи с употреба на максимум 256 KB RAM и 2 MB дисково пространство.

Зад. 90 2019-SE-01 В приложната статистика често се използват следните описателни статистики за дадена извадка:

- *средна стойност* $\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$
- *дисперсия* $D = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2$

където $\{x_1, x_2, \ldots, x_N\}$ са стойностите на извадката, а N е техния брой.

Напишете програма на C, която приема задължителен параметър – име на двоичен файл. Файлът съдържа информация за потребители, които са влизали в системата, и се състои от наредени петорки от следните елементи и техните типове:

- *уникален потребителски идентификатор (UID)* `uint32_t`
- *запазено1* `uint16_t` – не се използва в задачата
- *запазено2* `uint16_t` – не се използва в задачата
- *време1* `uint32_t` – момент на започване на сесията (Unix time)
- *време2* `uint32_t` – момент на завършване на сесията (Unix time)

За потребители, които са имали сесии, квадратът на продължителността на които е по-голям от дисперсията D на продължителността на всички сесии във файла, програмата трябва да изведе на `STDOUT` потребителският им идентификатор и продължителността на най-дългата им сесия.

Можете да разчитате на това, че във файла ще има не повече от 16384 записа и че в тях ще се срещат не повече от 2048 различни потребителски идентификатора.

Зад. 91 2020-IN-01 Напишете програма на C, която приема три параметъра – имена на двоични файлове.

Примерно извикване:

```
$ ./main patch.bin f1.bin f2.bin
```

Файловете `patch.bin` и `f1.bin` съществуват, и на тяхна база програмата трябва да създаде `f2.bin`.

Файлът `patch.bin` се състои от две секции – 16 байтов хедър и данни. На базата на хедъра програмата трябва да може да интерпретира съдържанието на файла. Структурата на хедъра е:

- `uint32_t`, *magic* – магическа стойност 0xEFBEADDE, която дефинира, че файлът следва тази спецификация
- `uint8_t`, *header version* – версия на хедъра, с единствена допустима стойност за момента 0x01, която дефинира останалите байтове от хедъра както следва:
  - `uint8_t`, *data version* – версия (описание) на използваните структури в секцията за данни на файла
  - `uint16_t`, *count* – брой записи в секцията за данни
  - `uint32_t`, *reserved 1* – не се използва
  - `uint32_t`, *reserved 2* – не се използва

Възможни структури в секцията за данни на файла спрямо *data version*:

- при версия 0x00
  - `uint16_t`, *offset*
  - `uint8_t`, *original byte*
  - `uint8_t`, *new byte*
- при версия 0x01
  - `uint32_t`, *offset*
  - `uint16_t`, *original word*
  - `uint16_t`, *new word*
- *забележка:* и при двете описани версии *offset* е отместване в брой елементи спрямо началото на файла

Двоичните файлове `f1.bin` и `f2.bin` се третират като състоящи се от елементи спрямо *data version* в `patch.bin`.

Програмата да създава файла `f2.bin` като копие на файла `f1.bin`, но с отразени промени на базата на файла `patch.bin`, при следния алгоритъм:

- за всяка наредена тройка от секцията за данни на `patch.bin`, ако на съответният *offset* в оригиналния файл `f1.bin` е записан елементът *original byte/word*, в изходният файл се записва *new byte/word*. Ако не е записан такъв елемент или той не съществува, програмата да прекратява

изпълнението си по подходящ начин;
- всички останали елементи се копират директно.

Наредените тройки в секцията за данни на файла `patch.bin` да се обработват последователно. Обърнете внимание на обработката за грешки и съобщенията към потребителя – искаме програмата да бъде удобен и валиден инструмент.

Зад. 92 2020-SE-01 Напишете програма на C, която приема шест задължителни позиционни параметъра – имена на файлове. Примерно извикване:

```
$ ./main affix postfix prefix infix suffix crucifixus
```

Всички файлове започват с хедър с фиксирана дължина от 16 байта. Пети и шести (спрямо $\mathbb{Z}^+$) байт от хедъра дефинират `uint16_t` число *count*, което описва броя на елементите във файла. Файловете `affix` и `infix` се състоят от елементи от тип `uint16_t`, файловете `prefix` и `crucifixus` – от елементи от тип `uint8_t`, `postfix` – от `uint32_t`, а `suffix` – от `uint64_t`.

Интервал наричаме наредена двойка числа, която дефинира номер (спрямо $\mathbb{Z}$) на начален елемент и брой елементи от даден файл. *Комплект* наричаме наредена четворка от интервали, които позиционно се отнасят спрямо файловете `{post,pre,in,suf}fix`.

Елементите на файла `affix` дефинират серия от комплекти, на чиято база програмата трябва да генерира изходния файл `crucifixus`.

Зад. 93 2020-SE-02 Инженерите от съседната лабораторя работят с комплекти SCL/SDL файлове, напр. `input.scl`/`input.sdl`.

В SCL файла са записани нива на сигнали (ниско 0 или високо 1), т.е., файлът се третира като състоящ се от битове. В SDL файла са записани данни от тип `uint16_t`, като всеки елемент съответства позиционно на даден бит от SCL файла.

Помогнете на колегите си, като напишете програма на C, която да бъде удобен инструмент за изваждане в нов файл само на тези SDL елементи, които са имали високо ниво в SCL файла, запазвайки наредбата им.

Зад. 94 2021-SE-01 Инженерите от съседната лабораторя ползват специализиран хардуер и софтуер за прехвърляне на данни по радио, но за съжаление имат два проблема:

- в радио частта: дълги поредици битове само 0 или само 1 чупят преноса;
- в софтуерната част: софтуерът, който ползват, може да прехвърля само файлове с четен брой байтове дължина.

Помогнете на колегите си, като напишете програма на C, която решава тези проблеми, като подготвя файлове за прехвърляне. Програмата трябва да приема два задължителни позиционни аргумента – имена на файлове. Примерно извикване:

```
$ ./main input.bin output.bin
```

Програмата чете данни от `input.bin` и записва резултат след обработка в `output.bin`. Програмата трябва да работи като encoder, който имплементира вариант на Manchester code, т.е.:

- за всеки входен бит 1 извежда битовете 10, и
- за всеки входен бит 0 извежда битовете 01

Например, следните 8 бита вход

```
1011 0110               == 0xB6
```

по описаният алгоритъм дават следните 16 бита изход

```
1001 1010 0110 1001     == 0x9A69
```

Зад. 95 2021-SE-02 Вашите колеги от съседната лаборатория са написали програма на C, която може да обработва подаден входен двоичен файл и на негова база генерира изходен двоичен файл. Програмата работи като encoder, който имплементира вариант на Manchester code, т.е.:

- за всеки входен бит 1 извежда битовете 10, и

- за всеки входен бит `0` извежда битовете `01`

Например следните 8 бита вход

```
1011 0110               == 0xB6
```

по описания алгоритъм дават следните 16 бита изход

```
1001 1010 0110 1001     == 0x9A69
```

Напишете програма на C, която извършва обратния процес, т.е., декодира файлове, създадени от горната програма.

Примерно извикване:

```
$ ./main input.bin output.bin
```

Зад. 96 2021-SE-03

Напишете програма на C, която приема два позиционни параметъра – имена на файлове. Примерно извикване:

```
$ ./main input.bin output.h
```

Файлът `input.bin` е двоичен файл с елементи `uint16_t` числа, създаден на *little-endian* машина.

Вашата програма трябва да генерира C хедър файл, дефиниращ масив с име `arr`, който:

- съдържа всички елементи от входния файл;
- няма указана големина;
- не позволява промяна на данните.

Генерираният хедър файл трябва да:

- съдържа и `uint32_t` променлива `arrN`, която указва големината на масива;
- бъде валиден и да може да се `#include`-ва без проблеми от C файлове, очакващи да "виждат" `arr` и `arrN`.

За да е валиден един входен файл, той трябва да съдържа не повече от 524288 елемента.

Зад. 97 2022-IN-01 Вашите колеги от съседната лаборатория работят със специализирана система Hoge, която съхранява данните си в binary файлове. За съжаление обаче им се налага да работят с две различни версии на системата (стара и нова) и разбира се, те ползват различни файлови формати.

Вашата задача е да напишете програма на C (`./main list.bin data.bin out.bin`), която конвертира файлове с данни от стария (`list.bin` + `data.bin`) в новия (`out.bin`) формат.

Всички файлове се състоят от две секции – 8 байтов хедър и данни. Структурата на хедъра е:

- `uint16_t`, `magic` – магическа стойност `0x5A4D`, която дефинира, че файлът е от системата Hoge и следва тази спецификация;
- `uint16_t`, `filetype` – дефинира какъв тип файл е това, с допустими стойности 1 за `list.bin`, 2 за `data.bin` и 3 за `out.bin`;
- `uint32_t`, `count` – дефинира броя на елементите в секцията с данни на файла.

Секцията за данни на всички файлове съдържа елементи – цели числа без знак, от съответните типове:

- `list.bin` – `uint16_t`;
- `data.bin` – `uint32_t`;
- `out.bin` – `uint64_t`.

Във файлове `data.bin` и `out.bin` елементи, чиято стойност е 0, са валидни елементи, но се игнорират от системата Hoge (тяхното наличие във файла не пречи на работата на системата).

Всеки елемент в секцията за данни на `list.bin` има две характеристики – *позиция* и *стойност*, като *позиция* е отместването на съответния елемент, докато *стойност* е неговата стойност. Тези две числа семантично дефинират отмествания във файловете с данни:

- *позиция* дефинира отместване в `data.bin`;
- *стойност* дефинира отместване в `out.bin`.

На базата на тези числови характеристики елементите на `list.bin` дефинират правила от вида: "елементът на отместване *позиция* в `data.bin` трябва да отиде на отместване *стойност* в `out.bin`".

Забележка: Всички отмествания са спрямо $\mathbb{N}_0$ в брой елементи.

Програмата трябва да интепретира по ред дефинираните правила и на тяхна база да генерира изходния файл.

Обърнете внимание на проверките, обработката за грешки и съобщенията към потребителя – искаме програмата да бъде удобен инструмент.

Забележка: Не е гарантирано, че разполагате с достатъчно памет, в която да заредите всички елементи на който и да от файловете.

Зад. 98 2022-SE-01 Напишете програма на C, която приема два позиционни аргумента – имена на двоични файлове. Примерно извикване: `./main data.bin comparator.bin`

Файлът `data.bin` се състои от две секции – 8 байтов хедър и данни. Структурата на хедъра е:

- `uint32_t, magic` – магическа стойност `0x21796F4A`, която дефинира, че файлът следва тази спецификация;
- `uint32_t, count` – описва броя на елементите в секцията с данни.

Секцията за данни се състои от елементи – `uint64_t` числа.

Файлът `comparator.bin` се състои от две секции – 16 байтов хедър и данни. Структурата на хедъра е:

- `uint32_t, magic1` – магическа стойност `0xAFBC7A37`;
- `uint16_t, magic2` – магическа стойност `0x1C27`;
- комбинацията от горните две magic числа дефинира, че файлът следва тази спецификация;
- `uint16_t, reserved` – не се използва;
- `uint64_t, count` – описва броя на елементите в секциата с данни.

Секцията за данни се състои от елементи – наредени 6-торки:

- `uint16_t, type` – възможни стойности: `0` или `1`;
- 3 бр. `uint16_t`, `reserved` – възможни стойности за всяко: `0`;
- `uint32_t, offset1`;
- `uint32_t, offset2`.

Двете числа `offset` дефинират отместване (спрямо $\mathbb{N}_0$) в брой елементи за `data.bin`; `type` дефинира операция за сравнение:

- `0`: "по-малко";
- `1`: "по-голямо".

Елементите в `comparator.bin` дефинират правила от вида:

- "елементът на `offset1`" трябва да е "по-малък" от "елементът на `offset2`";
- "елементът на `offset1`" трябва да е "по-голям" от "елементът на `offset2`".

Програмата трябва да интепретира по ред дефинираните правила в `comparator.bin` и ако правилото не е изпълнено, да разменя in-place елементите на съответните отмествания. Елементи, които са равни, няма нужда да се разменят.

Зад. 99 2023-IN-01 Вашите колеги от съседната лаборатория са попаднали на вогонски* крипто-вирус и вашата задача е да напишете програма, която възстановява файлове, криптирани от вируса. Примерно извикване: `./main input.encrypted output.decrypted`

Вирусът не бил особенно добре проектиран и след анализ се оказало, че подменя оригиналните файлове с криптирани със следните особености:

- вогоните явно предпочитат да работят с данни от по 128 бита, като всеки такъв елемент ще наричаме *unit* (U). Както входния (оригинален) файл, така и изходния (криптиран) файл се третират като състоящи се от unit-и. Криптираният файл винаги е точен брой unit-и (номерирани спрямо N_0), като при нужда към оригиналният файл преди криптиране се добавят байтове `0x00` (padding), за да стане точен брой unit-и.

*Adams, Douglas. The Ultimate Hitchhiker's Guide to the Galaxy: Five Novels in One Outrageous Volume. Del Rey, 2010.

- unit-ите в криптирания файл могат да се ползват за съхранение на следните типове данни (и техните големини): *header* (4U), *section* (2U), *cryptdata* (1U), *unused data* (1U).
- типовете данни section и cryptdata са обработени с *побитово изключващо или* (XOR) със 128 битов ключ (ключът е голям 1U), като за section ключът се ползва два пъти – за първият и за вторият unit по отделно.
- криптираният файл винаги започва с `header` (4U) и има следната структура:
  - `uint64_t, magic` – магическа стойност `0x0000534f44614c47`, която дефинира, че файлът е криптиран спрямо тази спецификация
  - `uint32_t, cfsb` – размер на криптирания файл в байтове
  - `uint32_t, cfsu` – размер на криптирания файл в брой unit-и
  - `uint32_t, ofsb` – размер на оригиналния файл в байтове
  - `uint32_t, ofsu` – размер на оригиналния файл в брой unit-и
  - `uint32_t, unused1` – не се използва в тази версия на вируса
  - `uint32_t, cksum` – checksum-а на оригиналния файл, получена по следния алгоритъм:
    - оригиналният файл се третира като състоящ се от `uint32_t` елементи
    - при нужда от padding на последния елемент се ползват байтове `0x00`
    - всички елементи се XOR-ват един с друг и полученият резултат е `cksum`
  - 128 бита[†], `sectionkey` – ключ, с който са кодирани section-ите
  - четири "слота", описващи начална позиция в U на section
    - `uint32_t, s0`
    - `uint32_t, s1`
    - `uint32_t, s2`
    - `uint32_t, s3`
    - тъй като unit-и с номера в интервала $[0, 3]$ се ползват за самия header, в горните слотове `0` означава, че този слот не се използва (няма такъв section)
    - unit-ите в изходния (декриптиран) файл са по реда на секциите
- `section` (2U) е кодиран със sectionkey от header-а и след разкодиране има следната структура:
  - `int64_t, relative_offset` – дефинира от кое U започват unit-ите с данни за тази секция. Числото е относително спрямо първият unit на section, например, ако section е записан в U 4/5 и стойността на offset е 2, то първите данни на секцията ще са в U 6.
  - `uint64_t, len` – дефинира брой unit-и в този section
  - 128 бита, `datakey` – ключ, с който са кодирани unit-ите с данни в този section
  - unit-ите в изходния (декриптиран) файл са по реда на unit-ите в секцията
- `cryptdata` (1U) – unit в който има записани данни, кодирани с datakey на секцията, към която е този unit
- `unused data` (1U) – unit, който не се ползва. Колегите ви предполагат, че това е един от начините чрез които вогоните са вкарвали "шум" в криптирания файл, но без да правят декриптирането невъзможно.

Зад. 100 2023-SE-01

Имате файл, съдържащ откъс от поток от съобщения, който може би е повреден на места.

Трябва да напишете програма, която по даден такъв файл, генерира нов файл, съдържащ точно всички последователности от байтове от оригиналния, които са валидни съобщения.

Примерно извикване на програмата: `./main stream.bin messages.bin`, където `stream.bin` е име на съществуващ файл, а `messages.bin` трябва да се създаде (или презапише).

Валидно съобщение с дължина N байта има вида:

- байт 1 — `0x55` – указва начало на съобщение
- байт 2 — числото N – представено като 8-битово число без знак
- байтове от 3 до $(N-1)$ — произволни данни (съдържанието на съобщението)
- байт N — checksum на байтове от 1 до $(N-1)$

Checksum на поредица от байтове е резултатът от прилагане операцията "побитово изключващо или" (xor) върху тях. Възползвайте се от него, за да проверявате валидността на съобщенията.

Зад. 101 2023-SE-02

Имате файл, съдържащ английски тълковен речник, представен във формат, описан по-долу. Напи-

[†] Ако искате да работите с 128 бита наведнъж удобно по познатия начин, ползвайте `#define uint128_t __uint128_t`

шете програма, която по подадена дума изписва нейната дефиниция на stdout (ако думата я няма в речника, извежда подходящо съобщение).

Програмата приема два аргумента: търсена дума и файл с речник (`./main respect english.dic`)

Речникът се състои от записи, а всеки запис се състои от:

- Нулев байт
- Дума, представена като последователност от байтове (между 1 и 63), **не** съдържаща символи за нов ред и нулеви байтове
- Символ за нов ред
- Описание на думата (произволно дълга последователност от байтове), **не** съдържащо нулеви байтове

Записите са сортирани лексикографски по думите.

Тази задача изисква решение, прочитащо най-много $O(M \log(N))$ байта от речника, където N е размерът на речника а M е максималната дължина на запис. Решения с по-лоша сложност се оценяват с най-много 8 точки. (*Подсказка:* двоично търсене.)

За улеснение, програмата може да приеме като трети аргумент индексен файл, състоящ се от 32-битови числа без знак (`./main respect english.dic english.idx`). Тези числа са позициите на всички нулеви байтове в речника. Ако програмата ви работи без този аргумент, ще получите бонус точки.

Зад. 102 2024-IN-01

Вашите колеги от съседната лаборатория използват специализирана система за съхранение на данни (СССД), която работи *основно* с два вида *обекти* – *volume*-и (текущо използвана версия на блоково устройство) и *snapshot*-и (стара версия на даден обект). Всеки обект има уникален идентификатор $id \in \mathbb{N}_0$, като стойност 0 е запазена. Текущото състояние на СССД е описано в binary файл, който се състои от три секции в следния ред:

- секция *header* – състои се от четири `uint16_t` променливи:
  - `magic` – всички служебни файлове на СССД имат магическа стойност `0x6963`;
  - `ver` – версия и формат на файла, в тази задача разглеждаме файлове с формат `0x6e73`;
  - `cp` – брой на елементите в preambule секцията;
  - `co` – брой на елементите в objects секцията;
- секция *preamble* - съдържа *елементи* със следната структура:
  - `v1`, `uint16_t`;
  - `v2`, `uint16_t`;
  - `v3`, `uint32_t`;
  - preamble елементите не се използват пряко в задачата;
- секция *objects* – съдържа *елементи* със следната структура, които описват обекти от СССД:
  - `ctime`, `uint32_t` – време на създаване на обекта (Unix/Epoch time);
  - `opt`, `uint16_t` – съдържа флагове, описващи различни характеристики на обекта; двата най-старши бита описват вида на обекта, като `00` се използват за volume, а `10` – за snapshot;
  - `parent_id`, `uint16_t` – идентификатор на предхождащ обект (родител, предишната версия на обекта), като ако обектът няма родител, стойността на `parent_id` е 0;
  - `size`, `uint32_t` – размер на обекта в байтове;
  - `ssize`, `uint32_t` – действително използвани байтове за съхранение на обекта, $ssize \in [0, size]$.

Забележки:

- съотношението на `ssize` към `size` на даден обект ще наричаме *коефициент на запълване*;
- отместването на елемент от *objects* секцията спрямо началото ѝ (в брой елементи) дефинира `id` на обекта, описан от елемента.

Съществува система за управление на СССД, която спрямо дефинирани критерии автоматично създава дневни snapshot-и на някои volume-и. Дефиницията за *дневен* snapshot е такъв snapshot, който е на време-разстояние един ден плюс-минус 10 минути от родителя си. Например, един volume и три негови snapshot-а изглеждат по следния начин:

Напишете програма, която приема параметър – име на файл в описания формат. Програмата трябва да пресмята и извежда на STDOUT колко е претеглената по размер на обекта средна стойност на коефициента на запълване за дневните snapshot-и. Колегите ви искат програмата да минимизира броя

	id	parent_id	ctime
volume 420	420	13	1704212400
snapshot C	13	8	1704471480
snapshot B	8	3	1704385182
snapshot A	3	0	1704298880

на операциите с плаваща запетая (floating-point operations), за да не се натрупва грешка.

Забележка: При непразна наредена n-торка от стойности $(x_1, x_2, \dots, x_n)$ и съответстващите им тегла $(w_1, w_2, \dots, w_n)$, претегленото аритметично средно $\bar{x}$ се изчислява по формулата

$$\bar{x} = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i} \tag{1}$$

Зад. 103 2024-SE-01

Имате файл, съдържащ записи, които са организирани в свързан списък. Някои от записите във файла вече не се използват (били са премахнати от списъка и вече не са част от него). За съжаление, по GDPR вие нямате право да пазите дълго време потребителски данни, които не се използват – затова трябва да напишете програма, която цензурира неизползваните записи.

Програмата се извиква с един аргумент (име на файл): `./main tasks.db`

Файлът се състои от последователни 512-байтови записи, всеки от които описва възел и има следния вид:

- `next`: 8 байта, представящи 64-битово число без знак – индекс на следващия възел (в брой записи спрямо началото на файла), или `0`, ако след този възел няма следващ
- `user_data`: 504 байта потребителски данни

Първият възел от списъка е винаги първи запис във файла (с индекс 0). Всеки следващ възел може да е на произволно място във файла.

Програмата трябва да промени така файла, че всеки запис, който не е част от списъка (не е достижим, обхождайки целия списък), да се замени с 512 байта нули.

Забележки:

- Забележете, че последният *възел от списъка* (този, който има `next == 0`) най-вероятно **няма да е** последният *запис във файла*.
- Може да приемете, че възлите, които са в списъка, няма да образуват цикъл.
- Максималният брой записи е 2^{40} - може да няма начин да съберем в паметта информация за всички записи. Ако искате да създадете временен файл, вижте `mkstemp(3)`.

Зад. 104 2024-SE-02

Компресирането е начин да представим дадени данни, използвайки по-малко място от оригиналните данни. Разглеждаме минимален алгоритъм за компресиране, който премахва повтарящите се байтове.

Напишете програма, която по подаден компресиран файл, възстановява оригиналния файл. Програмата приема два аргумента - име на компресиран файл и име на резултатен файл, в който да бъдат записани оригиналните данни.

Компресираният файл започва с хедър, който има следния вид:

- `magic`: 4 байта, представящи 32-битово число без знак, което винаги има стойността 0x21494D46 и обозначава, че файлът следва това описание;
- `packet_count`: 4 байта, представящи 32-битово число без знак – брой на *пакетите* в компресирания файл;
- `original_size`: 8 байта, представящи 64-битово число без знак – размер на оригиналния файл, в байтове;

След хедъра има поредица от *пакети*, които описват какви данни трябва да има в резултатния файл. *Пакетите* имат променлива дължина и винаги започват с един байт, който определя вида и размера

им. Този първи байт на *пакет* в двоична бройна система има вида: `tsss ssss`, където старшият бит `t` определя *вида* на *пакета* (`0` или `1`), а останалите 7 бита образуват 8-битово число без знак от вида `0sss ssss`, което ще наречем `N`.

Съществуват два *вида* пакети:

- Пакетите от *вид* `0` наричаме *обикновени пакети*. Такъв пакет записва следващите `N+1` байта от компресирания файл в резултатния файл.
- Пакетите от *вид* `1` наричаме *повтарящи се пакети*. Такъв пакет записва следващия байт от компресирания файл `N+1` пъти в резултатния файл.

Примери (всички числа са дадени в шестнайсетична бройна система):

- Пакетът `02 12 34 56` записва `12 34 56` в резултатния файл (този пакет е от вид `0` и за него `N=2`);
- Пакетът `84 78` записва `78 78 78 78 78` (вид `1`, `N=4`);
- Трите пакета `00 13 82 37 01 73 33` записват `13 37 37 37 73 33`

Програмата трябва да изчете пакетите от компресирания файл последователно и да запише тяхното съдържание в резултатния файл, за да получи оригиналните данни.

2.2 Процеси, pipe-ове, вход и изход (теми 6,7,8)

Зад. 105 2016-SE-01 Напишете програма на C, която по подадено име на (текстови) файл като параметър, извежда съдържанието на файла сортирано, чрез употреба на външните програми cat и sort през pipe().

Зад. 106 2016-SE-02 Напишете програма на C, която реализира simple command prompt. Тя изпълнява в цикъл следната поредица действия:

1. Извежда промпт на стандартния изход.
2. Прочита име на команда.
3. Изпълнява без параметри прочетената команда.

Командите се търсят в директорията /bin. За край на програмата се смята въвеждането на exit.

Зад. 107 2017-IN-01 Напишете програма на C, която използвайки външни shell команди през `pipe()` да извежда статистика за броя на използване на различните shell-ове от потребителите, дефинирани в системата. Изходът да бъде сортиран във възходящ ред според брой използвания на shell-овете.

Примерно извикване и изход:

```
$ ./main
      1 /bin/sync
      3 /bin/bash
      7 /bin/false
     17 /usr/sbin/nologin
```

Зад. 108 2017-IN-02 Напишете програма на C, която приема незадължителен параметър – име на команда. Ако не е зададена команда като параметър, да се ползва командата `echo`. Максималната допустима дължина на командата е 4 знака.

Програмата чете низове (с максимална дължина 4 знака) от стандартния си вход, разделени с интервали (`0x20`) или знак за нов ред (`0x0A`). Ако някой низ е с дължина по-голяма от 4 знака, то програмата да терминира със съобщение за грешка.

Подадените на стандартния вход низове програмата трябва да третира като множество от параметри за дефинираната команда. Програмата ви трябва да изпълни командата колкото пъти е необходимо с *максимум два* низа като параметри, като изчаква изпълнението да приключи, преди да започне ново изпълнение.

Примерни вход, извиквания и изходи:

```
$ cat f1
a1
$ cat f2
a2
$ cat f3
```

```
a3
$ echo -e "f1\nf2 f3" | ./main cat
a1
a2
a3
$ echo -e "f1\nf2 f3" | ./main
f1 f2
f3
```

Зад. 109 2018-SE-01 Напишете програма на C, която приема параметър – име на директория. Програмата трябва да извежда името на най-скоро променения (по съдържание) файл в тази директория и нейните под-директории, чрез употреба на външни шел команди през `pipe()`.

Зад. 110 2019-SE-01 Напишете програма-наблюдател P, която изпълнява друга програма Q и я рестартира, когато Q завърши изпълнението си. На командния ред на P се подават следните параметри:

- *праг за продължителност* в секунди – едноцифрено число от 1 до 9
- Q
- незадължителни *параметри на* Q

P работи по следния алгоритъм:

- стартира Q с подадените параметри
- изчаква я да завърши изпълнението си
- записва в текстов файл `run.log` един ред с три полета - цели числа (разделени с интервал):
  - момент на стартиране на Q (Unix time)
  - момент на завършване на Q (Unix time)
  - код за грешка, с който Q е завършила (exit code)
- проверява дали е изпълнено *условието за спиране* и ако не е, преминава отново към стартирането на Q

Условие за спиране: Ако наблюдателят P установи, че при две последователни изпълнения на Q са били изпълнени и двете условия:

1. кодът за грешка на Q е бил различен от 0;
2. разликата между момента на завършване и момента на стартиране на Q е била по-малка от подадения като първи параметър на P праг;

то P спира цикъла от изпълняване на Q и сам завършва изпълнението си.

Текущото време във формат Unix time (секунди от 1 януари 1970 г.) можете да вземете с извикване на системната функция `time()` с параметър `NULL`; функцията е дефинирана в `time.h`. Ако изпълнената програма е била прекъсната от подаден сигнал, това се приема за завършване с код за грешка 129.

Зад. 111 2020-SE-01 Напишете две програми на C (`foo` и `bar`), които си комуникират през *наименована тръба*. Програмата `foo` приема параметър - име на файл, програмата `bar` приема параметър - команда като абсолютен път до изпълним файл.

Примерни извиквания и ред на изпълнение (в отделни терминали):

```
./foo a.txt
./bar /usr/bin/sort
```

Програмата `foo` трябва да изпълнява външна команда `cat` с аргумент името на подадения файл, така че съдържанието му да се прехвърли през тръбата към програмата `bar`, която от своя страна трябва да изпълни подадената и като аргумент команда (без параметри; `/usr/bin/sort` в примера), която да обработи получените през тръбата данни, четейки от стандартен вход. Еквивалент на горния пример би било следното изпълнение:

```
cat a.txt | /usr/bin/sort
```

Зад. 112 2020-SE-02 При изграждане на система за пренасяне на сериен асинхронен сигнал върху радиопреносна мрежа се оказало, че големи поредици от битове само нули или само единици смущават сигнала, поради нестабилно ниво. Инженерите решили проблема, като:

- в моментите, в които няма сигнал от серийният порт, вкарвали изкуствено байт `0x55` в потока;
- реалните байтове `0x00`, `0xFF`, `0x55` и `0x7D` се кодирали посредством XOR-ване (побитова обработка с *изключващо-или*) с `0x20`, като полученият байт се изпращал през потока, предхождан от `0x7D`, който играе ролята на *escape character*.

Разполагате със запис от такъв поток. Напишете програма на C, която приема два параметъра - имена на файлове. Примерно извикване:

```
$ ./main input.lfld output.bin
```

Програмата трябва да обработва записа и да генерира `output.bin`, който да съдържа оригиналните данни. Четенето на входните данни трябва да става посредством изпълнение на външна shell команда.

Зад. 113 2020-SE-03 Напишете програма на C, която приема един задължителен позиционен параметър - име на файл. Файлът се състои от не повече от 8 наредени тройки елементи:

- *име на файл* – точно 8 байта, последният от които задължително е `0x00`. Ако името е по-късо от 7 знака, излишните байтове са `0x00`;
- *offset* – `uint32_t`, който дава пореден номер на елемент (спрямо N_0) във файла;
- *length* – `uint32_t`, който дава брой елементи.

За всяка наредена тройка програмата трябва да пусне child процес, който да XOR-не (обработи с *изключващо-или*) елементите (`uint16_t`) от съответния файл един със друг, и да върне резултата на parent процеса, който от своя страна трябва да XOR-не всички получените резултати и да изведе полученото число в следния формат:
`result: 573B`

Забележка: С пълен брой точки се оценяват решения, в които child процесите работят паралелно.

Зад. 114 2021-SE-01 Ваши колеги - асистенти по ОС се оплакват, че студентите упорито ползват командата `sudo`, въпреки че им е обяснено да не я ползват. Вашата задача е да подготвите фалшиво `sudo`, което:

- добавя ред в текстови файл `foo.log` с командата, която потребителят се е опитал да изпълни във формат `ДАТА ПОТРЕБИТЕЛ ДАННИ`, където:
  - `ДАТА` е във формат `YYYY-MM-DD HH:MM:SS.UUUUUU` (`UUUUUU` са микросекунди)
  - `ПОТРЕБИТЕЛ` е login name (username) на потребителя
  - `ДАННИ` е всичко, което потребителят е написал, например `./main ls -l /root`
  - горните полета са разделени с интервал
  - пример: `2021-05-10 14:38:17.584344 s99999 ./main ls -l /root`
- заключва акаунта, който е изпълнил програмата
- приключва изпълнението на всички процеси, стартирани от този потребител

За да може да изпълни тези неща, фалшивото `sudo` ще има нужда от root права, а в същото време трябва да знае кой акаунт изпълнява командата. Традиционно това може да се реши със SUID, което обаче не важи за shell скриптове. За да заобиколите този проблем, напишете програма на C (`main`), която за заключването на акаунта и терминирането на процесите извиква външни shell команди през `pipe()`.

Полезни man страници:

- `gettimeofday(2)`
- `localtime(3)`
- `strftime(3)`
- `getuid(2)`
- `getpwuid(3)`
- `passwd(1)`

Опишете в `doc.txt` как бихте настроили компилираната вече програма така, че студентите да ползват нея, вместо оригиналното `sudo`.

Естествено, по време на разработка на програмата можете да я тествате само с вашият акаунт и нямате `root` права. За да може все пак да тествате, подменете в кода финалните команди, които програмата изпълнява (тези, които наистина изискват `root` права), като им добавите едно `echo`. Например, ако трябва да изпълните командата

```
foo -m pesho
```

вместо това в кода ви трябва да е дефинирано изпълнение на

```
echo foo -m pesho
```

Зад. 115 2022-IN-01 Ваши колеги - асистенти по ОС имат нужда от демострационна програма на C, която да служи като пример за конкурентност и синхронизация на процеси. Напишете такава програма, приемаща два задължителни позиционни параметъра – едноцифрени числа. Примерно извикване: `./main N D`

Общ алгоритъм на програмата:

- началният (родителски) процес създава процес-наследник
- N на брой пъти се изпълнява:
  - родителският процес извежда на stdout низа "`DING `"
  - процесът-наследник извежда на stdout низа "`DONG`"
  - родителският процес изчаква D секунди

Гарантирайте, че:

- процесът-наследник винаги извежда "`DONG`" след като родителския процес е извел "`DING `"
- родителският процес винаги започва изчакването след като процеса-наследник е извел "`DONG`"

Забележка: За изчакването погледнете sleep(3).

Зад. 116 2023-IN-01

Ваши колеги – асистенти по ОС – имат нужда от демострационна програма на C, която да служи като пример за конкурентност и синхронизация на процеси.

Дефиниран е *нареден* списък с три думи $L = (\texttt{"}tic\texttt{ "}, \texttt{"}tac\texttt{ "}, \texttt{"}toe\backslash n\texttt{"})$, като всяка дума е с дължина четири знака. Напишете програма, приемаща две числа като аргументи (`./main NC WC`), като NC е ограничено в интервала $[1, 7]$, а WC – в интервала $[1, 35]$. Програмата трябва *задължително* да работи по следния общ алгоритъм:

- началният (родителски) процес създава NC на брой процеси-наследници;
- групата процеси извеждат на stdout общо WC на брой думи от горния списък при следните правила:
  - ако броят на думите за извеждане WC е по-голям от общия брой думи в L, след изчерпване на думите в L програмата започва списъка отначало колкото пъти е нужно;
  - всеки процес извежда само една дума;
  - първата дума се извежда от родителския процес;
  - ако броят на думите за извеждане WC е по-голям от *общия* брой процеси, след изчерпване на процесите програмата започва да ги ползва от начало. Например при родителски процес P и два процеса-наследници C_1 и C_2 редът на използване на процесите ще бъде P, C_1, C_2, P, C_1, C_2, P, C_1, ...
  - изведените думи *гарантирано* трябва да са по реда, дефиниран в L;
  - всеки процес *гарантирано* започва да извежда на stdout поредната дума, след като предходния процес е приключил с извеждането на предишната.

Забележка: За конвертиране на аргументите погледнете `strtol(3)`.

Зад. 117 2023-SE-01

Напишете програма на C, която приема като аргумент име на директория и генерира хешове на файловете в нея, работейки паралелно.

При изпълнение върху директория `my-dir`, програмата:

- За всеки файл в директорията и нейните поддиректории (напр. `my-dir/foo/my-file`), трябва да създаде файл `my-dir/foo/my-file.hash`, съдържащ хеш-сумата на `my-file`
- Може да извиква външните команди `find` (за откриване на файлове) и `md5sum` (за пресмятане на хеш-сума)
- Не трябва да обработва файлове, чиито имена вече завършват на `.hash`
- Трябва да позволява `md5sum`-процесите да работят паралелно един спрямо друг

Зад. 118 2023-SE-02

Напишете програма, която изпълнява списък от команди в паралелни процеси, но щом някой от про-

цесите изведе ред с текста “found it!”, прекратява изпълнението на всички процеси.

За улеснение, командите нямат свои аргументи и се подават като аргументи на вашата програма.

Примерно извикване: `./main ./tests/foo ./tests/bar ./tests/baz`

Прекратяването на процес става, като му изпратим SIGTERM и след това го изчакаме да приключи.

Изходът на командите (извън това дали `stdout` съдържа търсения текст) и техните статуси не ни интересуват. Програмата ви не трябва да извежда нищо, а нейният статус трябва да е:

- 0, ако някой процес е извел текста “found it!”
- 1, ако всички процеси са завършили и никой процес не е извел текста
- 26 при друга грешка

Зад. 119 2024-SE-01

Напишете програма *fuzzer*, търсеща входове, при които друга дадена програма крашва. Fuzzer-ът приема следните 3 аргумента:

1. име (път) на друга програма /*тестваната програма*/
2. цяло число $N \in [0, 2^8)$ - брой тестове
3. име на резултатен файл

Fuzzer-ът трябва да стартира програмата многократно: до завършване на N на брой изпълнения или до достигане на краш. Приемаме, че програмата е крашнала, ако вместо да завърши нормално, нейният процес е убит от сигнал.

При всяко изпълнение на програмата, fuzzer-ът избира случайно число $S \in [0, 2^{16})$ и подава на `stdin` на програмата S на брой байта, взети от файла `/dev/urandom`. Този виртуален файл съдържа безкраен поток от случайни данни - използвайте ги и за генериране на самото S. Също така, fuzzer-ът трябва да се погрижи всякакъв изход от тестваната програма да не стига до терминала - той не ни интересува.

След завършване на N на брой итерации без краш, резултатният файл трябва да е празен, а статусът на завършване на fuzzer-а - успешен. При намерен краш, резултатният файл трябва да съдържа входа, подаден на програмата при съответното изпълнение, а статусът на завършване на fuzzer-а да е 42.

Зад. 120 2024-SE-02

Напишете програма наблюдател, която няколкократно изпълнява други програми до тяхното успешно изпълнение.

Програмата получава между 1 и 10 аргумента – имена на наблюдавани програми:

`./main <наблюдавана програма 1> <наблюдавана програма 2> ... <наблюдавана програма N>`

Програмата наблюдател трябва да изпълни всяка от подадените ѝ наблюдавани програми без аргументи. Програмите трябва да се изпълняват в паралелно работещи процеси под следните условия:

- Когато някой процес завърши успешно (с нулев статус), съответстващата му програма се счита за приключила и вече не се използва от вашата програма наблюдател.
- Когато някой процес завърши неуспешно (с ненулев статус), съответстващата му програма трябва да се изпълни наново, като за новия процес важат същите правила.
- Ако някой процес бъде убит преди да е завършил (напр. при краш), програмата наблюдател трябва да:
  1. Прекрати изпълнението на всички други текущо изпълняващи се процеси. Прекратяването на процес става, като му изпратим сигнал `SIGTERM` и след това го изчакаме да приключи.
  2. Завърши със статус, съответстващ на номера на програмата, чийто процес е бил убит. Например, ако процес, изпълняващ наблюдавана програма 2, бъде убит, вашата програма наблюдател трябва да завърши със статус 2.

С други думи, всяка наблюдавана програма трябва да бъде изпълнявана многократно до първи успех, а ако което и да е изпълнение на която и да е програма бъде убито, програмата наблюдател спира работа.

Решения, в които наблюдаваните програми не се изпълняват паралелно, ще получат най-много 6 точки. В тази ситуация, очакваме последователно изпълнение на наблюдаваните програми (първо програма 1 до първи успех, после програма 2 и т.н.), а условието за прекратяване на всички процеси при убит процес отпада.

Зад. 121 2025-IN-01

В желанието си да разберем как светът работи, често се опитваме да изграждаме модели, които биха могли да предскажат или симулират системи в хода на тяхното движение. Процесорите и програмите, които те изпълняват, са включени в това изследователско пътешествие.

Вашото безкрайно желание да вникнете дълбоко в това как инструментите, с които всеки ден боравим, работят, ви амбицира да си измислите задача, която ви позволява по-добре да разберете сложния материал от света на КАРХ.

Както би трябвало да знаем, всеки процесор работи с набор от регистри, които чете/пише по време на изпълнението на инструкциите на процесът късметлия, който в този момент е бил избран от task scheduler-а на ОС. Освен тях, процесорът има достъп и до RAM паметта, която действа като удължение на пространството, с което процесът разполага.

В задачата си, вие решавате да си напишете **симулатор на паралелно изпълняващи се процесори**. В този симулатор, всеки процесор разбира един и същ набор от инструкции, описана в таблицата по-долу. Разликата между процесорите, обаче, е в броя на техните регистри и големината на RAM паметта, с която те разполагат. Разбира се, програмите, които те ще изпълняват, също **би трябвало** да бъдат съобразени с ограничението на конкретния процесор, върху който работят.

Като вход програмата ви получава един файл input.bin – описание на различните процесори. Файлът се състои от наредени тройки елементи:

- *ram_size* – `uint16_t`, който описва големина на RAM памет за конкретния процесор. **Ограничено до 512**.
- *register_count* – `uint16_t`, който описва брой регистри за конкретния процесор. **Ограничено до 32**.
- *filename* – точно 8 байта, последният от които задължително е `0x00`. Ако името е по-късо от 7 знака, излишните байтове са `0x00`

За всяка наредена тройка, файлът, посочен от *filename*, е от следната структура:

- *секция register values – register_count* на брой байта, които задават началните състояния на регистрите за процесора.
- *секция ram init – ram_size* на брой байта, които задават началното състояние на RAM паметта, с която процесорът ще работи.
- *секция instructions* – последна секция, която продължава до края на файла. Съдържа инструкциите(програмата), които съответният процесор ще изпълнява.

В процесорите, които симулираме, **всеки регистър е с големина един байт**, а **всяка инструкция е с големина точно 4 байта и разполага с 3 операнда**. Байтовете във всяка инстуркция имат следното значение:

- *opcode* – идентификатор на инструкцията.
- *op1* – първи операнд на инструкцията.
- *op2* – втори операнд на инструкцията.
- *op3* – трети операнд на инструкцията.

За всяка наредена тройка от входния файл, вашата програма трябва да стартира нов процес, който да действа като симулатор за конкретния процесор. Симулацията се изразява в следните стъпки:

- инициализиране на регистрите с байтовете от *секция register values*.
- инициализиране на RAM паметта с байтовете от *секция ram init*.
- изпълнение на инструцкиите от *скеция instructions*.

След приключване изпълнението на инструкциите, всеки от процесите-симулатори трябва да презапише секциите *register values* и *ram init* във файлът, с който работи, с новото състояние на регистрите и RAM паметта.

Таблица 1: Инструкциите на нашият имагинерен процесор и тяхното действие

Name	Opcode	Effect
AND	0	registers[op1] = registers[op2] & registers[op3]
OR	1	registers[op1] = registers[op2] | registers[op3]
ADD	2	registers[op1] = registers[op2] + registers[op3]
MULTIPLY	3	registers[op1] = registers[op2] * registers[op3]
XOR	4	registers[op1] = registers[op2] ^ registers[op3]
PRINT	5	записва registers[op1] към стандартния изход
SLEEP	6	процесорът бива приспан за registers[op1] секунди
LOAD	7	registers[op1] = RAM[registers[op2]]
STORE	8	RAM[registers[op2]] = registers[op1]
JNE	9	ако registers[op1] != registers[op2], то процесорът трябва да продължи изпълнението си от op3-тата инструкция
LOADI	10	registers[op1] = op2
STOREI	11	RAM[registers[op1]] = op2

Забележка: Навсякъде, където се среща registers[opN], се има предвид регистъра с индекс opN. Навсякъде, където се среща RAM[N], се има предвид байтът с индекс N от RAM паметта.

Забележка: Възможно е инструкция да не използва всеки операнд, въпреки че има общо 3 на разположение. PRINT и SLEEP са пример за това. В такъв случай, стойностите в допълнителните операнди не са от значние за инструкцията.

Пример: За инструкция (ADD 11 7 2), стойностите в регистър 7 и регистър 2 трябва да бъдат сумирани и записани в регистър 11. Тук op1 е със стойност 11, op2 е със стойност 7, а op3 е със стойност 2.

Забележка: Решения, в които процесите не работят паралелно, ще получат по-малко точки.

3 Теоретични задачи

Упътвания за семафорите:

- Семафорът е обект за синхронизация, локалните му данни са брояч cnt и списък на приспаните процеси L. Конструкторът му `init(n)` присвоява начална стойност на брояча ($cnt = n$), списъкът се инициалира празен.
  Семафорът има два метода – `wait()` и `signal()`.
  Методът `wait()` намаля с единица брояча cnt и ако стойността на брояча стане отрицателна, добавя в списъка L информация за текущия процес и го спира временно (процесът бива приспан, блокиран).
  Методът `signal()` увеличава cnt и ако стойността на брояча преди увеличението е отрицателна, изважда процес от списъка L и го събужда. Ако от L се вади най-рано приспания процес, наричаме семафора силен. Всяка друга стратегия на събуждане реализира слаб семафор.
- Приемете, че инициализацията на семафорите се прави от процес, който поражда процесите, обсъждани в условието на задачата, преди тяхното стартиране.

3.1 Единични процеси

Зад. 122 2016-SE-01 Всеки от процесите P и Q изпълнява поредица от три инструкции:

```
process P          process Q
  p_1                q_1
  p_2                q_2
  p_3                q_3
```

Осигурете чрез семафори синхронизация на P и Q така, че инструкция p_1 да се изпълни преди q_2, а q_2 да се изпълни преди p_3.

Зад. 123 2017-SE-01 Всеки от процесите P, Q и R изпълнява поредица от три инструкции:

```
process P          process Q          process R
  p_1                q_1                r_1
  p_2                q_2                r_2
  p_3                q_3                r_3
```

Осигурете чрез семафори синхронизация на P, Q и R така, че инструкция p_1 да се изпълни преди q_2 и r_2.

Забележка: Решения на задачата с повече от един семафор не носят пълен брой точки.

Зад. 124 2017-SE-04 Всеки от процесите P и Q изпълнява поредица от две инструкции:

```
process P          process Q
  p_1                q_1
  p_2                q_2
```

Осигурете чрез семафори синхронизация на P и Q така, че инструкция p_1 да се изпълни преди q_2, а q_1 да се изпълни преди p_2.

Зад. 125 2017-SE-05 Всеки от процесите P и Q изпълнява поредица от три инструкции:

```
process P          process Q
  p_1                q_1
  p_2                q_2
  p_3                q_3
```

Осигурете чрез два семафора синхронизация на P и Q така, че отделните инструкции да се изпълнят в следния времеви ред: p_1, q_1, p_2, q_2, p_3, q_3.

Зад. 126 2018-CS-01 Всеки от процесите P, Q и R изпълнява поредица от три инструкции:

```
process P               process Q               process R
  p_1                     q_1                     r_1
  p_2                     q_2                     r_2
  p_3                     q_3                     r_3
```

Осигурете чрез семафори синхронизация на `P`, `Q` и `R` така, че да се изпълнят едновременно следните изисквания:

- инструкция `p_1` да се изпълни преди `q_2` и `r_2`
- инструкция `r_2` да се изпълни преди `p_3`

Забележка: Решения на задачата с повече от два семафора не носят пълен брой точки.

Зад. 127 2018-CS-02 Всеки от процесите `P`, `Q` и `R` изпълнява поредица от три инструкции:

```
process P               process Q               process R
  p_1                     q_1                     r_1
  p_2                     q_2                     r_2
  p_3                     q_3                     r_3
```

Осигурете чрез семафори синхронизация на `P`, `Q` и `R` така, че да се изпълнят едновременно следните изисквания:

- Инструкция `p_1` да се изпълни преди `q_2`.
- Инструкция `q_1` да се изпълни преди `r_2`.
- Инструкция `r_1` да се изпълни преди `p_2`.
- Инструкция `r_3` да се изпълни след `p_2` и `q_2`.

Зад. 128 2018-SE-01 Всеки от процесите `P`, `Q` и `R` изпълнява поредица от три инструкции:

```
process P               process Q               process R
  p_1                     q_1                     r_1
  p_2                     q_2                     r_2
  p_3                     q_3                     r_3
```

Осигурете чрез семафори синхронизация на `P`, `Q` и `R` така, че да се изпълнят едновременно следните изисквания:

- някоя от инструкциите `p_2` и `q_2` да се изпълни преди `r_2`;
- ако инструкция `p_2` се изпълни преди `r_2`, то `q_2` да се изпълни след `r_2`;
- ако инструкция `q_2` се изпълни преди `r_2`, то `p_2` да се изпълни след `r_2`;

Забележка: Решения на задачата с повече от два семафора не носят пълен брой точки.

Забележка: Решения на задачата при които `p_2` или `q_2` винаги изчакват `r_2` носят 0 точки.

Зад. 129 2019-CS-01 Всеки от процесите `P`, `Q` и `R` изпълнява поредица от три инструкции:

```
process P               process Q               process R
  p_1                     q_1                     r_1
  p_2                     q_2                     r_2
  p_3                     q_3                     r_3
```

Осигурете чрез семафори синхронизация на `P`, `Q` и `R` така, че да се изпълнят едновременно следните изисквания:

- инструкция `p_1` да се изпълни преди `q_2` и `r_2`
- инструкция `p_3` да се изпълни след `q_2` и `r_2`

Зад. 130 2019-CS-02 Всеки от процесите `P`, `Q` и `R` изпълнява поредица от две инструкции:

```
process P         process Q         process R
  p_1               q_1               r_1
  p_2               q_2               r_2
```

Осигурете чрез три семафора синхронизация на `P`, `Q` и `R` така, че отделните инструкции да се изпълнят в следния времеви ред: `p_1`, `q_1`, `r_1`, `p_2`, `q_2`, `r_2`.

Зад. 131 2019-SE-01 Всеки от процесите `P` и `Q` изпълнява поредица от три инструкции:

```
process P               process Q
  p_1                     q_1
  p_2                     q_2
  p_3                     q_3
```

Осигурете чрез два семафора синхронизация на `P` и `Q` така, че да са изпълнени едновременно следните времеви зависимости:

1. инструкция `p_1` да се изпълни преди `q_2`
2. инструкция `q_2` да се изпълни преди `p_3`
3. инструкция `q_1` да се изпълни преди `p_2`
4. инструкция `p_2` да се изпълни преди `q_3`

Забележка: Решения на задачата с повече от два семафора не носят пълен брой точки.

Зад. 132 2020-CS-01 Процесите `P`, `Q` и `R` изпълняват поредица от две инструкции:

```
process P             process Q             process R
  p_1                   q_1                   r_1
  p_2                   q_2                   r_2
```

Осигурете чрез семафори синхронизация на `P`, `Q` и `R` така, че да се изпълнят едновременно следните изисквания:

- инструкцията `p_1` да се изпълни преди `q_2` и `r_2`
- инструкцията `q_1` да се изпълни преди `p_2` и `r_2`
- инструкцията `r_1` да се изпълни преди `p_2` и `q_2`

Зад. 133 2020-SE-03 Всеки от процесите `P`, `Q` и `R` изпълнява поредица от три инструкции:

```
process P               process Q               process R
  p_1                     q_1                     r_1
  p_2                     q_2                     r_2
  p_3                     q_3                     r_3
```

Осигурете чрез семафори синхронизация на `P`, `Q` и `R` така, че да се изпълнят едновременно следните изисквания:

- инструкция `p_1` да се изпълни преди `q_2` и `r_2`;
- ако `q_2` се изпълни преди `r_2`, то и `q_3` да се изпълни преди `r_2`;
- ако `r_2` се изпълни преди `q_2`, то и `r_3` да се изпълни преди `q_2`.

Обосновете отговора си.

Зад. 134 2025-SE-01

Дадена е следната стандартна имплементация на стек с краен буфер от N елемента:

```
struct Stack(N):
  buf: Item[N]
  idx: Uint

init(s: Stack(N)):
  s.idx := 0

peek(s: Stack(N)) -> Item:
  if s.idx == 0:
    return error
  return s.buf[s.idx - 1]
```

```
push(s: Stack(N), x: Item):
  if s.idx >= N:
    return error
  s.buf[s.idx] := x
  s.idx += 1

pop(s: Stack(N)) -> Item:
  if s.idx == 0:
    return error
  result := s.buf[s.idx - 1]
  s.idx -= 1
  return result
```

Променете структурата от данни, така че да са изпълнени следните условия:

- Новата имплементация да бъде безопасна за конкурентно използване от различни процеси, без възможност за race condition.
- Случаите, които в старата имплементация водят до грешка, в новата имплементация да водят до блокиране до момент, в който операцията вече може да бъде изпълнена.
- **Не е** задължително операциите `peek()` да могат да се изпълняват едновременно. Ако все пак решението ви позволява паралелно изпълнение на `peek()`, ще получите 20 точки вместо 10.

Упътване: Ако не имплементирате бонуса, състоянието на всички полета на `Stack` би следвало да е еднакво преди и след операцията `peek()`

3.2 Процеси с много копия

Зад. 135 2017-SE-02 Преди стартиране на процеси `P` и `Q` са инициализирани два семафора и брояч:

```
semaphore e, m
e.init(1); m.init(1)
int cnt = 0
```

Паралелно работещи няколко копия на всеки от процесите `P` и `Q` изпълняват поредица от инструкции:

```
process P                       process Q
  m.wait()                        e.wait()
    cnt=cnt+1                       q_section
    if cnt=1 e.wait()             e.signal()
  m.signal()

  p_section

  m.wait()
    cnt=cnt-1
    if cnt=0 e.signal()
  m.signal()
```

Дайте обоснован отговор на следните въпроси:

а) Могат ли едновременно да се изпълняват инструкциите `p_section` и `q_section`?
б) Могат ли едновременно да се изпълняват няколко инструкции `p_section`?
в) Могат ли едновременно да се изпълняват няколко инструкции `q_section`?
г) Има ли условия за deadlock или starvation за някой от процесите?

Упътване:

- Ще казваме, че `P` е в критична секция, когато изпълнява инструкцията си `p_section`. Същото за `Q`, когато изпълнява `q_section`.
- Изяснете смисъла на брояча `cnt` и какви процеси могат да бъдат приспани в опашките на двата семафора.
- Покажете, че в опашката на семафора `e` има най-много едно копие на `P` и произволен брой копия на `Q`.
- Покажете, че в момента на изпълнение на `e.signal()` в кой да е от процесите, никой процес не е в критичната си секция.

Зад. 136 2017-SE-03

а) Няколко копия на процеса `P` изпълняват поредица от три инструкции:

```
process P
  p_1
  p_2
  p_3
```

Осигурете чрез семафор синхронизация на копията така, че най-много един процес да изпълнява инструкция p_2 във всеки един момент.

б) Опишете разликата при реализация на слаб и силен семафор.

в) Възможно ли е в зависимост от начина на реализация на семафора в подусловие а) да настъпят условия за deadlock или starvation? Ако да, опишете сценарий за поява на неприятната ситуация.

Зад. 137 2017-SE-08 Множество паралелно работещи копия на всеки от процесите P и Q изпълняват поредица от две инструкции:

```
process P               process Q
  p_1                     q_1
  p_2                     q_2
```

Осигурете чрез семафори синхронизация на работещите копия така, че да са изпълнени едновременно следните условия:

- в произволен момент от времето да работи най-много едно от копията;
- работещите копия да се редуват във времето – след изпълнение на копие на P да следва изпълнение на копие на Q и обратно;
- първоначално е разрешено да се изпълни копие на P.

Зад. 138 2018-SE-03 Множество паралелно работещи копия на всеки от процесите P и Q изпълняват поредица от три инструкции:

```
process P               process Q
  p_1                     q_1
  p_2                     q_2
  p_3                     q_3
```

Осигурете чрез семафори синхронизация на работещите копия, така че три инструкции – p_1, q_2 и p_3 се редуват циклично:

- първа се изпълнява инструкция p_1 на някое от работещите копия на процес P;
- след завършването ѝ се изпълнява инструкция q_2 на някое копие на Q;
- след нея – p_3 на някое копие на P;
- с това едно минаване през цикъла завършва и отново може да се изпълни инструкция p_1 на някое от работещите копия на процес P.

Зад. 139 2020-CS-01 Множество паралелно работещи копия на всеки от процесите P, Q и R изпълняват поредица от две инструкции:

```
process P           process Q           process R
  p_1                 q_1                 r_1
  p_2                 q_2                 r_2
```

Осигурете чрез семафори синхронизация на P, Q и R така, че да се изпълнят едновременно следните изисквания:

- в произволен момент от времето да работи най-много едно от копията.
- работещите копия да се редуват във времето – след изпълнение на копие на P да се изпълни копие на Q, после копие на R. Следва ново изпълнение на P и т.н.
- първоначално е разрешено да се изпълни копие на P.

Зад. 140 2020-SE-02 Множество паралелно работещи копия на всеки от процесите P, Q, R и W изпълняват поредица от две инструкции:

```
process P         process Q         process R         process W
  p_1               q_1               r_1               w_1
  p_2               q_2               r_2               w_2
```

Осигурете чрез семафори синхронизация на работещите копия така, че да са изпълнени едновременно следните условия:

а) В произволен момент от времето да работи най-много едно от копията (един-единствен процес глобално).
б) Работещите копия да се редуват във времето – първо се изпълнява копие на P или Q. След това трябва да се изпълни копие на R или W. Следва ново изпълнение на копие на P или Q и т.н.

Обосновете отговора си.

Зад. 141 2021-SE-01 Всеки от процесите P, Q и R изпълнява поредица от инструкции:

```
process P                process Q                process R
  p_1                      q_1                      r_1
  p_2                      q_2                      r_2
```

Процесите P и Q са единични, процесът R се изпълнява в много копия.

Осигурете чрез семафори синхронизация на P, Q и R така, че да се изпълнят едновременно следните изисквания:

- всички инструкции на P и Q да се изпълнят преди инструкция r_1 на всяко копие на R;
- процесите P и Q да се изпълнят ефикасно, т.е. да е възможно паралелното им изпълнение, без да се изчакват.

Обосновете отговора си.

Зад. 142 2023-CS-01

Паралелно работещи копия на всеки от процесите P и Q изпълняват поредица от две инструкции:

```
process P                process Q
  p_1                      q_1
  p_2                      q_2
```

Осигурете чрез семафори синхронизация на работещите копия така, че:

- в началото само едно копие на процеса P изпълнява инструкциите си, всички останали процеси изчакват;
- след като първото копие на P изпълни кода си, останалите процеси могат да изпълняват своите инструкции.

Пълен брой точки ще получат решения, при които различните копия на P и Q могат да се изпълняват паралелно и няма излишни изчаквания след приключването на първото копие на P.

Зад. 143 2023-SE-01

Паралелно работещи копия на всеки от процесите P, Q и R изпълняват поредица от две инструкции:

```
process P                process Q                process R
  p_1                      q_1                      r_1
  p_2                      q_2                      r_2
```

Осигурете чрез семафори синхронизация на работещите копия така, че да се изпълнят едновременно следните изисквания:

- в произволен момент от времето да работи най-много едно от копията;
- работещите копия да се редуват във времето – след изпълнение на копие на P се изпълнява копие на Q, после копие на P, следва копие на R и цикълът се повтаря – процесите се редуват PQPRPQPR...
- първоначално се изпълнява копие на P.

Зад. 144 2023-SE-02

Паралелно работещи копия на процеса P изпълняват поредица от две инструкции:

```
process P
  p_1
  p_2
```

Осигурете чрез семафори синхронизация на работещите копия така, че да са изпълнени едновременно следните изисквания:

- в началото само едно копие на процеса P изпълнява инструкциите си, всички останали процеси изчакват;
- след като първото копие на P изпълни кода си, останалите процеси могат да изпълняват своите инструкции.

Забележка:

Пълен брой точки ще получат решения, при които различните копия на P (след първото) могат да се изпълняват паралелно и няма излишни изчаквания (след приключването на първото копие на P).

Зад. 145 2024-SE-01

Разглеждаме програма, първоначално изпълняваща единствен процес P, състоящ се от следната поредица от инструкции:

```
process P
  p_1
  repeat N times:
    start_Q
  p_2
```

Всяко изпълнение на инструкцията `start_Q` създава нов процес Q, изпълняващ следната поредица от инструкции:

```
process Q
  q_1
  q_2
```

Осигурете чрез семафори синхронизация на процесите така че инструкцията p_2 да се изпълни след като всички (общо N на брой) процеси Q са изпълнили своята инструкция q_2.

Задължително е:

- Различните копия на Q да могат да се изпълняват паралелно.
- Константата N, както и текущият брой процеси Q да **не се** използват извън процеса P

3.3 Ползване на брояч

Зад. 146 2019-SE-01 Множество паралелно работещи копия на процеса P изпълняват поредица от две инструкции:

```
process P
  p_1
  p_2
```

Осигурете чрез семафори синхронизация на работещите копия така, че да се изпълнят едновременно следните изисквания:

- инструкцията p_2 на всяко от работещите копия да се изпълни след като инструкция p_1 е завършила изпълнението си в поне 3 работещи копия.

Упътване: Освен семафори, ползвайте и брояч.

Зад. 147 2020-SE-01 Множество паралелно работещи копия на всеки от процесите P и Q изпълняват поредица от три инструкции:

```
process P                process Q
  p_1                      q_1
  p_2                      q_2
  p_3                      q_3
```

Осигурете чрез семафори синхронизация на P и Q така, че поне една инструкция p_1 да се изпълни преди всички q_2, и поне една инструкция q_1 да се изпълни преди всички p_2.

Обосновете отговора си.

Зад. 148 2022-CS-01 Паралелно работещи копия на всеки от процесите P и Q изпълняват поредица от две инструкции:

```
process P                process Q
  p_1                      q_1
  p_2                      q_2
```

Осигурете чрез семафори синхронизация на работещите копия така, че да се изпълнят едновременно следните изисквания:

- в произволен момент от времето да работи най-много едно от копията;
- работещите копия да се редуват във времето – след изпълнение на копие на P, два пъти се изпълнява копие на Q, и цикълът се повтаря – веднъж P, два пъти Q и т.н.;
- първоначално е разрешено да се изпълни копие на P.

Упътване: Освен семафори, ползвайте и брояч.

Зад. 149 2022-IN-01

Ваксинационен център работи на следния принцип:

- множество доктори поставят ваксини;
- има маса, на която може да има максимум M на брой ампули с ваксина;
- всеки доктор взима една ампула от масата (ако на масата има ампули) и поставя ваксината на пациент;
- ако на масата няма ампули, докторът казва на медицинската сестра да донесе нова партида ампули и тя "презарежда" масата с нови M на брой ампули;
- работният ден започва с "пълна" маса с M на брой ампули.

Множество паралело работещи копия на процеса D (лекари) изпълняват следния код:

```
while true:
    getVaccineFromTable()
    administerVaccineToPatient()
```

Единичен процес N (мед. сестра) изпълнява следния код:

```
while true:
    reloadTableWithVaccines(M)
```

Ограничения:

- процес D не може да изпълни `getVaccineFromTable()`, ако на масата няма ампули с ваксина;
- процес N може да изпълни `reloadTableWithVaccines(M)` само ако на масата няма ампули с ваксина.

Осигурете чрез семафори синхронизация на процесите така, че да са изпълнени синхронизационните ограничения. Обосновете решението си.

Упътване: Освен семафори, ползвайте и брояч.

Зад. 150 2022-IN-02 Паралелно работещи копия на всеки от процесите P и Q изпълняват поредица от две инструкции:

```
process P                process Q
  p_1                      q_1
  p_2                      q_2
```

Осигурете чрез семафори синхронизация на работещите копия така, че да се изпълнят едновременно следните изисквания:

- в произволен момент от времето да работи най-много едно от копията;
- работещите копия да се редуват във времето – след изпълнение на две копия на P, три пъти се изпълнява копие на Q, и цикълът се повтаря – два пъти P, три пъти Q и т.н.;
- първоначално е разрешено да се изпълни копие на P.

Упътване: Освен семафори, ползвайте и брояч(и).

Зад. 151 2023-IN-01 Множество от N паралелно работещи копия на процеса P изпълняват следният цикъл с две инструкции:

```
WHILE true
  p_1
  p_2
ENDWHILE
```

Осигурете чрез семафори и броячи синхронизация на работещите копия така, че на всяка итерация на цикъла всички процеси се изчакват взаимно, а именно: инструкцията p_2 се изпълнява, след като всички копия са изпълнили p_1 (на същата итерация).

Зад. 152 2023-SE-01

Всеки от процесите P, Q и R изпълнява поредица от инструкции:

```
process P          process Q          process R
  p_1                q_1                r_1
  p_2                q_2                r_2
```

Процесите P и Q са единични, процесът R се изпълнява в много копия.

Осигурете чрез семафори синхронизация на P, Q и R така, че да се изпълнят едновременно следните изисквания:

- всички инструкции на някой от процесите P и Q да се изпълнят преди инструкция r_1 на всяко копие на R;
- един от процесите P и Q да се изпълни след като поне 3 копия на R са изпълнили кода си.

Забележки:

- решения, в които единият от процесите P и Q чака другия да завърши, няма да получат точки;
- решения, в които копията на R се изпълняват последователно, ще получат по-малко точки.

Зад. 153 2024-IN-01 Паралелно работещи копия на процеса P изпълняват поредица от инструкции:

```
process P
  p_1
  p_2
  p_3
```

Осигурете чрез семафори синхронизация на работещите копия на P така, че да се изпълнят едновременно следните изисквания:

- първото копие на процеса P изпълнява всичките си инструкции;
- останалите копия не изпълняват инструкцията си p_1, но изчакват тя да е приключила в първото копие, след което изпълняват инструкциите си p_2 и p_3.

Забележка: Точки ще получат решения, при които различните копия на P могат да се изпълняват паралелно и няма излишни изчаквания след приключване на инструкция p_1 на първото копие.

Зад. 154 2024-SE-01

Всеки от процесите P и Q изпълнява поредица от инструкции:

```
process P          process Q
  p_1                q_1
  p_2                q_2
```

Процесите P и Q се изпълняват в много копия.

Осигурете чрез семафори синхронизация на P и Q така, че да се изпълнят едновременно следните изисквания:

- инструкция p_1 на някое копие на P да се изпълни преди всяко копие на Q;
- инструкция p_2 на всички копия на P да се изпълни след като поне 3 копия на Q са изпълнили кода си.

Забележка: Решения, в които копията на P или Q се изпълняват последователно, ще получат по-малко точки.

3.4 Анализ на race condition

Зад. 155 2019-SE-03 Процесите P и Q се изпълняват паралелно. Споделената променлива A има начална стойност 4. Променливата R е локална за двата процеса.

```
process P                process Q
  R=A                      R=A
  R=R+3                    R=R+2
  A=R                      A=R
```

Каква е стойността на A след изпълнението на процесите? Дайте обоснован отговор.

3.5 Програми на C

Зад. 156 2018-CS-01 Дадена е програма за ОС Linux, написана на езика C:

```
#include <unistd.h>
#include <string.h>
int main(void)
{
  char* buff = "Hello world!\n";
  int p;
  if (fork()==0) write(1,buff,strlen(buff));
  p=fork();
  write(1,buff,strlen(buff));
}
```

- Колко пъти ще се отпечата текста `Hello world!` при изпълнението на програмата? Обосновете отговора си.
- Нарисувайте кореновото дърво с върхове процесите, които ще се стартират в резултат от изпълнението на програмата и ребра двойките родител-наследник.

Зад. 157 2018-SE-02 Дадена е програма за ОС Linux, написана на езика C:

```
#include <unistd.h>
#include <stdio.h>
int main(void)
{
  int p1, p2;
  p1=fork();
  p2=fork();
  printf("Hello world!\n");
}
```

а) Колко пъти ще се отпечата текста `"Hello world!"` при изпълнението на програмата? Обосновете отговора си.

б) Как работи системното извикване `fork()`?

в) Нарисувайте кореновото дърво с върхове процесите, които ще се стартират в резултат от изпълнението на програмата и ребра двойките родител-наследник.

Зад. 158 2019-CS-02 Дадена е програма за ОС Linux, написана на езика C:

```
#include <unistd.h>
#include <stdio.h>
int main(void)
{
  int p1, p2, p3;
  p1=fork();
  if (p1==0) {
    p2=fork();
    if (p2>0) p3=fork();
  }
  printf("Hello world!\n");
}
```

а) Колко пъти ще се отпечата текста `Hello world!` при изпълнението на програмата? Обосновете отговора си.
б) Как работи системното извикване `fork()`?
в) Нарисувайте кореновото дърво с върхове процесите, които ще се стартират в резултат от изпълнението на програмата и ребра двойките родител-наследник.

Зад. 159 2022-IN-01

Дадена е следната програма на C за GNU/Linux операционна система:

```
 1 #include <sys/types.h>
 2 #include <unistd.h>
 3 #include <stdlib.h>
 4 int
 5 main(void)
 6 {
 7     pid_t p1,p2,p3;
 8     write(1,"A",1);
 9     p1 = fork();
10     if (p1 == 0)
11     {
12         write(2,"BB",2);
13         p2 = fork();
14         if (p2 > 0)
15         {
16             write(1,"C",1);
17             p3 = fork();
18             write(2,"DD",2);
19             if (p3 > 0) exit(0);
20         }
21     }
22     write(1,"A",1);
23     exit(0);
24 }
```

Забележка: Номерата на редовете са за илюстративни цели и не са част от кода на програмата.

При компилиране и стартиране на програмата се създава процес с PID 42. Изберете примерни стойности за PID на всеки създаден процес-наследник.

а) Опишете обосновано изпълнението на програмата, като включите следната информация:
- процес `<PID>`, ред `<n>`, създава процес `<PID>`
- процес `<PID>`, ред `<n>`, извежда...

- процес `<PID>`, ред `<n>`, терминира (прекратява изпълнението си)

б) Нарисувайте диаграма, която визуализира взаимоотношенията между процесите. Например, може да нарисувате кореново дърво с върхове процесите, които ще се стартират в резултат от изпълнението на програмата и ребра двойките родител-наследник. Маркирайте върховете със съответния PID.

в) Какъв ще бъде изхода на `stdout` след приключване на процесите? Обосновете отговора си.

Упътване: Не е гарантирано, че системното извикване `fork()` приключва успешно.

3.6 "Разказвателни" задачи

Зад. 160 2016-SE-02 Опишете накратко основните процедури и структури данни, необходими за реализация на семафор.

Каква е разликата между слаб и силен семафор?

Опишете максимално несправедлива ситуация, която може да се получи в избирателна секция, ако на входа на секцията пазач – член на изборната комисия пуска гласоподавателите вътре така:

- във всеки момент в секцията може да има най-много двама гласоподаватели.
- пазачът работи като слаб семафор.

Зад. 161 2017-SE-06 Да приемем, че в съвременната операционна система процесът има 4 състояния:

- R – работещ (*running*, използва CPU)
- A – активен (*ready*, очаква CPU)
- S – блокиран (*sleeping*, очаква вход/изход)
- T – изчакващ време (*sleeping*, очаква времеви момент)

Нарисувайте диаграма на състоянията и преходите между тях. Диаграмата е ориентиран граф с върхове отделните състояния и ребра – възможните преходи.

Опишете накратко събитията, предизвикващи преход по всяко ребро на графа.

Зад. 162 2017-SE-07 Процесът `P` създава тръба (pipe) с извикване на функцията `pipe(int pipefd[2])` в ОС GNU/Linux.

а) Кои процеси не могат да ползват тръбата?

б) Опишете друг метод за изграждане на комуникационен канал, който дава възможност на произволни процеси да изградят и ползват канала. Допълнително искаме новоизградения канал да е достъпен само за процесите, които са го създали.

Упътване: Прочетете man-страницата за функцията `pipe()`.

Зад. 163 2018-CS-03 При споделено ползване на памет от няколко процеса е възможно да настъпи надпревара за ресурси (race condition).

а) (2k точки) Дефинирайте понятието race condition.

б) (k точки) Възможно ли е да настъпи race condition в еднопроцесорна система? Ако да, при какви условия.

в) (3k точки) Какви инструменти ползваме, за да избегнем race condition?

Зад. 164 2018-CS-04 Една от класическите задачи за синхронизация се нарича *Задача за читателите и писателите* (*Readers-writers problem*).

а) (k точки) Опишете условието на задачата.

б) (2k точки) Опишете решение, използващо семафори.

Зад. 165 2018-CS-05 Какви са възможните състояния на процес.

Нарисувайте диаграма на състоянията и преходите между тях.

Опишете накратко ситуациите, предизвикващи преходи между състояния.

Зад. 166 2018-SE-04 Опишете накратко кои системни извиквания изграждат стандартните комуникационни канали в UNIX – неименувана тръба (pipe), връзка процес-файл, двустранна връзка процес-процес (connection).

Зад. 167 2018-SE-05 Опишете какви изисквания удовлетворява съвременна файлова система, реализирана върху блоково устройство (*block device*). Опишете накратко реализацията и целта на следните инстру-

менти:

а) отлагане на записа, алгоритъм на асансьора;
б) поддържане на журнал на файловата система.

Зад. 168 2019-CS-01 При реализация на файлова система върху твърд диск файловете и директориите се записват върху сектори от диска. Времето за достъп до секторите зависи от текущото положение на механичните компоненти на диска – над коя пътечка е главата за четене/запис и каква е позицията ѝ над пътечката.

Защо се прави разместване във времето на операциите по четене и запис върху диска?

Опишете накратко реализацията и целта на алгоритъма на асансьора.

Зад. 169 2019-CS-03 Опишете как се изгражда комуникационен канал (connection) между процес-сървер и процес-клиент със следните системни извиквания в стандарта POSIX:

socket(), bind(), connect(), listen(), accept()

Зад. 170 2019-CS-04 Опишете накратко основните комуникационни канали в ОС Linux. Кои канали използват пространството на имената и кои не го правят?

Зад. 171 2019-SE-04 Опишете разликата между синхронни и асинхронни входно-изходни операции. Дайте примери за програми, при които се налага използването на асинхронен вход-изход.

Зад. 172 2019-SE-06 Опишете реализацията на комуникационна тръба (pipe) чрез семафори. Предполагаме, че тръбата може да съхранява до n байта, подредени в обикновена опашка. Тръбата се ползва от няколко паралелно работещи изпращачи/получатели на байтове. Процесите изпращачи слагат байтове в края на опашката, получателите четат байтове от началото на опашката.

Упътване: В теорията на конкурентното програмиране задачата е известна като “producer-consumer problem”.

Зад. 173 2020-CS-01 Определете понятията *взаимно блокиране* (deadlock) и *гладуване* (livelock, resource starvation). Дайте примери.

Зад. 174 2020-IN-01 Кои системни извиквания работят с файлов дескриптор, създаден с `open(2)` на обикновен файл, но винаги ще завършат с грешка за дескриптор, създаден с `pipe(2)`?

Опишете накратко функционалността ѝм.

Зад. 175 2021-CS-01 Каква задача решава инструментът spinlock (активно изчакване)?

Опишете хардуерните инструменти, необходими за реализацията на spinlock.

В кои ситуации не бива да се ползва spinlock?

Зад. 176 2024-SE-01 Какъв проблем решава журналната файлова система?

Опишете накратко реализацията ѝ.