ЕВРОПЕЙСКИ СЪЮЗ
ЕВРОПЕЙСКИ СОЦИАЛЕН ФОНД

ПОДКРЕПА ЗА ДУАЛНАТА СИСТЕМА НА ОБУЧЕНИЕ

ОПЕРАТИВНА ПРОГРАМА
НАУКА И ОБРАЗОВАНИЕ ЗА ИНТЕЛИГЕНТЕН РАСТЕЖ

УЧЕБНО ПОМАГАЛО
Операционни системи и конкурентно програмиране

за специалност код 4810301 „Приложно програмиране“
професия код 481030 „Приложен програмист“
разработено от авторски екип към Професионална техническа гимназия
„Никола Йонков Вапцаров“, гр. Враца

Авторски екип:
инж. Божидар Пламенов Цветков
инж. Петър Росенов Петров
Редактор: Косена Петрова Савова
Дизайнер: Денислава Яворова Велева
Одобрено от: Емилиян Валентинов Кадийски

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

Съдържание

Част 1 Операционни системи

Част 2 Конкурентно програмиране

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение", финансиран от Оперативна програма „Наука и образование за интелигентен растеж", съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

ЧАСТ 1 ОПЕРАЦИОННИ СИСТЕМИ

ГЛАВА 1 КОМПЮТЪРНИ И ОПЕРАЦИОННИ СИСТЕМИ

- **Какво ще научим?**
- Каква е структурата на една компютърна система?
- Що е то операционна система и за какво служи?
- Защо програмите и потребителите имат нужда от нея?
  Какви са основните компоненти (елементи) на една компютърна система?
- Какви са основните характеристики на един процесор?
- Какво са входно-изходните устройства?
- Видове памети.
  Как да направим добър избор за една компютърна конфигурация?

1. Компютърна система има слоеста структура.

Включва следното:

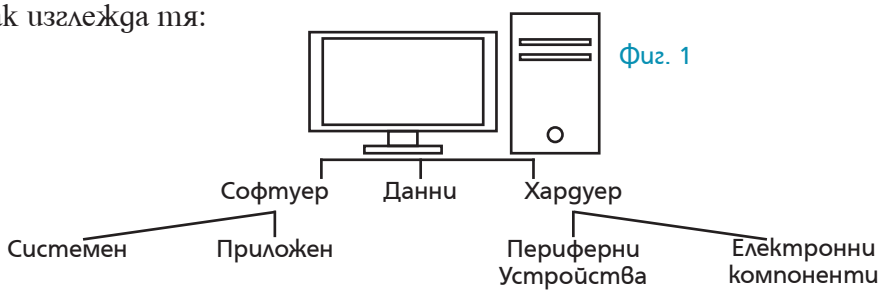
→**Хардуер** – апаратната част на компютърната система.

→**Операционна система** – която включва системен софтуер, програма, която прави връзката между приложните програми и апаратната част.

→**Приложения** – приложни програми, софтуер, който извършва полезна работа.

→**Потребител** – хора или машини, които се възползват от работата на приложния софтуер и услугите които предоставя.

Смисълът е, че всеки слой осигурява функции или действия - операции,чрез които всеки слой комуника с по-горен слой. На фиг. 1 ще разгледаме как изглежда тя:



Фиг. 1

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

Много е важно да знаем, че компютърната система има важни компоненти или т.н.т елементи!

2. Елементите на една компютърна система:

→**Дънна платка** – представлява електронна платка, на която върху слотове се закрепят други основни компоненти: процесор, оперативна памет, видео карта и др. Ролята ѝ е **да осигурява връзка между процесора и другите елементи чрез чипсета и шините за комуникация.**

Пример за производители на дънни платки: Asus, Asrock, Intel, Gigabyte и др.

→**Шина (Bus)** – състои се от проводници и електронни елементи, които осъществяват и контролират връзката между **процесора и оперативната памет**, както и **други входно-изходни** устройства.

→**Захранващ блок (Power supply)** – ролята му е да **преобразува стандартното напрежение до напрежение, необходимо да захрани** дънната платка, процесора и всички останали компоненти. **Мощността** се измерва във ватове **W**. (от 400 до 2000 W)

→**Централен процесор** – това е чип на дънната платка, който представлява капсулирана **силициева пластина с вградени микроелектронни компоненти** (**транзистори**).

Състои се от две основни части: **аритметично-логическо устройство** – то изпълнява всички аритметични и логически функции и **контролно устройство** – то управлява работата на отделните части на компютъра. Двете устройства са свързани с шина, чрез която се осъществява нужната комуникация.

Пример за производители на процесори: Intel, AMD

Основни характеристики на процесора:

→**Тактова честота (работна скорост)** – броя тактове, които се изпълняват от процесора за секунда. Измерва се в мегахерци (MHz) или гигахерци (GHz). Съвременните компютри са с тактова честота до 4 GHz.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

→**Разрядност** – брой битове които могат да се обработват или предават едновременно. Могат да бъдат 32 битови или 64 битови.

→**Кеш памет** – специална работна област за временно съхраняване на данни, които процесорът току-що е използвал. Ускорява обмена на данни между процесора и оперативната памет.

→**Ядра** – многоядрените процесори са изградени от 2 или повече **изчислителни ядра** в един корпус, като всяко ядро може да работи самостоятелно. Съвременните компютри работят с 4 или повече ядра.

След като разгледахме основните компонентни на компютърната система и тяхното предназначение, е време да представим и периферните и устройства, така наречените **входно-изходни устройства**.

3. Периферни устройства:
→**Входни** – клавиатура, мишка, камера, скенер, микрофон...
→**Изходни** – монитор, принтер, тонколони...
→**Входно-изходни** – HDD, SDD, USB-flash памет, Wi-Fi модул...
→**Контролери** – физическо устройство за връзка между периферното устройство и оперативната памет на КС.
→**Физически интерфейси** – USB, PS/2, VGA, HDMI, RS232, RS485, SCSI...
→**Драйвери** – системен софтуер, част от операционната система, който реализира абстракцията между приложен софтуер и физическо устройство.

4. Оперативна памет (RAM):
Тъй като, процесорът почти няма памет, но данните трябва да се пазят някъде! Оперативната памет пази нашите данни!
→Процесорът може да работи директно с оперативната памет.
→Типично оперативната памет е енергозависима, **нуждае се от постоянно електрозахранване (Данните ще се загубят при изключване на електрозахранването)**.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

→Времето за достъп до всяка клетка от паметта е едно и също (random access).
→Свикнали сме да наричаме Оперативната памет – **RAM**.

5. Постоянна памет (Запомнящи устройства):
→Енергонезависима памет – **не се нуждае от постоянно електрозахранване.**
→**Свикнали сме да я наричаме – ROM.**
→Скоростта на достъп е по-ниска от тази на оперативната памет.
→Може да се съхраняват по-големи обеми на по-ниски цени.

Проучете:
1. Какъв е процесът при сглобяване на една компютърна система(**Computer Assembly Steps**)?
2. Имате за цел да изберете конфигурация за компютърна система на **Дизайнер**. Дизайнерът ще използва **Photoshop**. За целта използвайте някой онлайн магазин и конфигурирайте подходящите компютърни компоненти, така че да покриете препоръчителните изисквания на програмата. Направете таблица с компонентите.
3. Каква роля има **софтуерът** в една компютърна система?
4. Каква роля има **хардуерът** в една компютърна система?
5. Намерете името на играта, чрез която можете да създавате(моделирате) Компютърни системи.
6. Направете сравнение между най-новите процесори на **AMD** и **Intel**. Съставете таблица с резултатите от направеното проучване.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

ГЛАВА 2 СТРУКТУРА НА ОПЕРАЦИОННАТА СИСТЕМА

Какво ще научим?

- Определение за операционната система
- Структура на операционната система
- Какви са основните функции на една операционна система?
- Видове операционни системи
- Какво приложение намират различните операционни системи?
- Какво съдържа архитектурата на ОС?
- Как работи една ОС?
- Инсталиране на виртуална машина и операционна система Ubuntu.

В глава първа, разгледахме какво е операционната система и разбрахме, че включва софтуер, програма, която прави връзката между приложните програми и апаратната част.

Да, но това не е всичко, сега ще разгледаме няколко определения, които ще трябва да научим.

В следващите изброени определения ще използваме съкращението **ОС**.

1. Операционна система:

→***ОС*** *е основна част от компютърния софтуер, който управлява и координира ресурсите на хардуера и софтуера и обслужва приложните програми.*

→***ОС*** *е съвкупност от програми, предназначени да организират изчислителния процес и да направят удобно общуването на потребителите с комп. с-ма.*

→***ОС*** *е абстактна (виртуална) машина, която разширява функциите на апаратната част – добавя ниво на абстракция над хардуера.*

→***ОС*** *е разпределител на системните ресурси.*

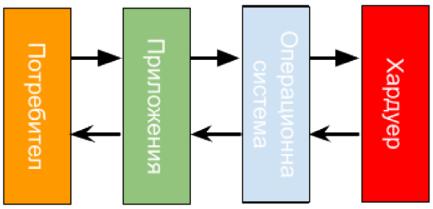
След като вече знаем определенията за ОС и нейното приложение е време да се разгледаме следните функции които извършва тя:

→ОС предоставя начини за взаимодействие на потребителя с нея, чрез

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

командeн интерпретатор и/или графична среда с потребителски интерфейс.
→ОС контролира изпълнението на програмите, като разпределя изчислителните ресурси между отделните процеси.
→Изпълнява входно-изходни операции.
→ОС предоставя възможност на потребителя да чете, пише, създава и изтрива файлове чрез файловата система.
→ОС управлява комуникацията между процесите.
→ОС се грижи за сигурността на ниво потребител, като контролира достъпа до файлове и ресурси. Също така, често част от ОС е и защитна стена, която пази системата от външни за нея атаки.

На фиг. 1 се изобразява как комуникират различни слоеве в операционната система.



Фиг. 1

2. Архитектурата на операционната система

Ядро на ОС – централна част на ОС, обезпечаваща координирания достъп на приложения до ресурсите на компютъра. Според различните архитектури ядрото съдържа в себе си различни инструменти и услуги.

Комадният интерпреатор на ОС може да приема различни команди, свързани с различни нейни функции:
→Създаване и управление на процеси.
→Управление на входно/изходни операции.
→Управление на запомнящи устройства.
→Операции върху файловата система.
→Управление на механизмите за защита.
→Мрежови функции.
→Други.

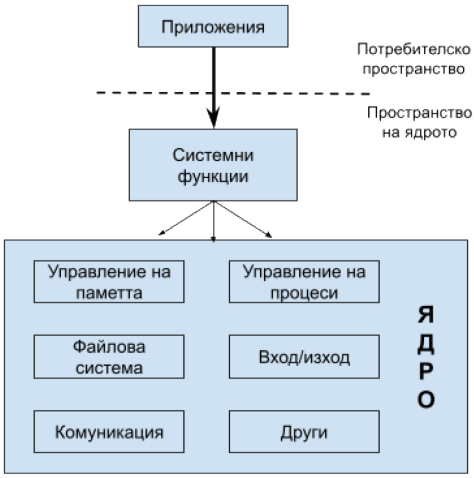
Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

3. Видове архитектури на ядро на ОС

Плюсове	Минуси
Простота	Трудно проследяване на грешките
Бързодействие (не се губи време за комуникация между отделните компоненти)	Необходимост от прекомпилация на цялото ядро при промяна

3.1. Монолитно ядро

Монолитното ядро представлява една монолитна (единна) програма в паметта, която съдържа всички компоненти на ядрото. Отделните компоненти могат лесно и бързо да комуникират помежду си. Недостатък на монолитното ядро е трудната промяна, заради необходимостта от рекомпилация.



Фиг. 2

3.2. Модулно ядро

Модулното ядро е подобно на монолитното ядро. При него имплементацията е с монолитна (единна) програма в паметта. За да се разреши проблемът с необходимостта от рекомпилация, модулното ядро

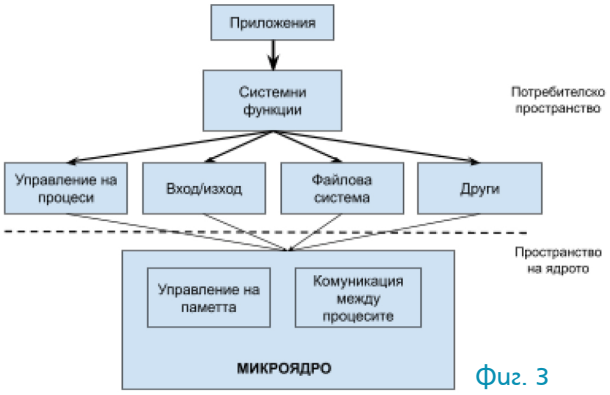
Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

предоставя интерфейс за включване и изключване на отделни модули към него.

3.3. Микроядро

Целта на микроядрото е да бъде възможно най-малко. Поради тази причина то предоставя малък брой услуги, най-вече управление на памет и комуникация между процесите. Микроядрото позволява висока степен на модулност и лесна преносимост и мащабируемост.

При микроядрото се налага повече междупроцесорна комуникация, което води до по-ниска скорост. Услугите, които не могат да бъдат част от микроядрото се заменят с услуги в потребителското пространство. Като следствие от това ОС става по-надеждна, защото по-малко количество код работи в незащитен режим.



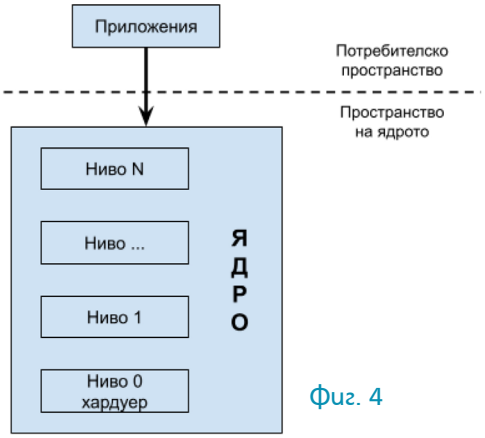
Фиг. 3

3.4. Многослойно ядро

При многослойното ядро компонентите се групират по вида на тяхната функционалност в отделни слоеве. Всеки от слоевете може да използва само предоставеното му от по-долните слоеве. Хардуерът се счита за слой 0, а слой N е потребителският слой, който се състои от потребителския интерфейс и приложенията, които работят в него.

При многослойното ядро дадена заявка за изпълнение на операция минава последователно между слоевете, докато достигне слоя, който е способен да я изпълни. Така се получава защита между слоевете, но се намаля бързодействието заради комуникацията по отделните слоеве.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове



Фиг. 4

3.5. Други видове ядра

→ Наноядро – ядро, което управлява само ресурсите.
→ Екуядро – наноядро с координация на процесите.
→ Хибридни ядра.

4. Как работи една Операционна система?

→ Работата на ОС се управлява от прекъсванията.
→ Софтуерните грешки и заявките за операции на ОС генерират софтуерни прекъсвания.
→ ОС работи в два режима – режим на ядрото и режим на потребителя – режимът на работата на процесора позволява да се прави разлика между тях.
→ Извикването на системна функция води до промяна на режима на работа на процесора от потребителски режим към режим на ядрото.
→ Когато изпълнението на функцията приключи, режимът се превключва обратно към потребителски.

5. Файлова структура в Linux

5.1. Кое какво е от фиг.5?

Корен /

→ Всеки файл и директория се намират под кореновата директория – т.е. явяват се нейни наследници.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

→Само root потребителите имат право да пишат в тази директория.
→Важно е да се има предвид, че / **root** е домашната директория за потребителя **root**, което не е същото като /.

/bin – user binaries

→Съдържа изпълними файлове в двоичен вид.
→Програмите, които съответстват на често използваните команди, се намират в тази директория. Например: **ps**, **ls**, **ping**, **grep**, **cp**.

/sbin – system binaries

→Подобно на /bin, тук се съдържат изпълними файлове в двоичен вид.
→Съдържа програми, съответстващи на системни команди, които обикновено се изпълняват от системни администратори.
→Например: **iptables**, **reboot**, **fdisk**, **ifconfig**, **swapon**.

Фиг. 5

/etc – конфигурационни файлове

→Съдържа конфигурационни файлове, които се използват от програмите.
→Съдържа и скриптове за стартиране и изключване, които се използват, за да пускат/спират отделни програми.
→Например: /etc/resolv.conf, /etc/logrotate.conf .

/dev – устройства (devices)

→Името на директорията идва от devices, а не от developer.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

→Съдържа файлове на устройствата.
→Устройствата могат да бъдат различни видове – терминални устройства, usb и др. Например: /dev/tty1, /dev/usbmon0, /dev/sda, /dev/null.

/proc – информация за процесите

→Съдържа информация за системния процес.
→Това е псевдо файлова система, която съдържа информация за процесите.
→Например: /proc/{pid} съдържа информация за процеса с конкретен PID номер. Всеки процес има собствен уникален PID.
→В /proc се съдържа и виртуална файлова система с информация за системните ресурси, например: /proc/uptime.

/tmp – временни файлове

→Директория, която съдържа временни файлове, създадени от системата и потребителите.
→Файловете от тази директория се изтриват при рестартиране.

/var – променливи файлове

→Съдържанието на файловете в директорията се очаква да расте.
Често тук се съдържат:
→системни логове (/var/log);
→файлове на пакети и бази данни (/var/lib);
→мейли (/var/mail);
→файлове на опашка за принтиране (/var/spool);
→заключващи файлове (/var/lock);
→временни файлове, необходими между рестратиранията (/var/tmp).

/usr – потребителски програми

→Съдържа двоични файлове, библиотеки, документация и сорс код на програми. В /usr/bin се съдържат двоичните файлове на потребителските програми. Ако не откривате дадена програма в /bin, проверете в /usr/bin.
→Примерни програми тук: **at**, **awk**, **cc**, **less**, **scp.**

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

/usr/sbin съдържа двоично файлове за системни администратори. Ако не можете да намерите файл в /sbin, потърсете го в **/usr/sbin**.

→Примерни програми тук: **atd, cron, sshd, useradd, userdel**.

/usr/lib съдържа нужните библиотеки за програмите от /usr/bin и /usr/sbin.

/usr/local съдържа потребителските програми, които инсталирате от източник, например, когато инсталирате apache от източник, той ще отиде в **/usr/local/apache2**.

/home – домашни директории

→Домашни директории, в които потребителите съхраняват личните си файлове. Например: **/home/ivan**, **/home/maria.**

/boot – Boot Loader Files

→Съдържа файлове свързани с boot loader-а. (системата за зареждане на Линукс). Съдържат се initrd, vmlinux, grub файлове – част от ядрото.

→Пример: **initrd.img-2.6.32-24-generic, vmlinuz-2.6.32-24-generic**.

/lib – Системни библиотеки

→Съдържа библиотечни файлове, които са необходими на програмите в /bin и /sbin Имената на библиотеките са във формат: ld* или lib*.so.*

→Пример: **ld-2.11.1.so, libncurses.so.5.7**.

/opt – Допълнителни приложения (optional)

→Съдържа допълнителни (add-on) приложения. Приложенията трябва да се инсталират в /opt или в поддиректория на /opt.

/mnt – Mount

→Директория за временно монтиране, където могат да се монтират файлови системи.

/media – Removable media devices

→В тази директория могат да бъдат намерени устройства като флашки, CD и DVD дискове, SD карти и други.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

/srv

→ Директорията /srv съдържа данннннни свързани с услуги (services).
→ Например, /srv/cvs съдържа данни за CVS.

Видове системни функции?

→ Предоставят интерфейс между потребителската програма и ОС.
→ Достъпни са за програми не езици за системно програмиране (C/C++) и асемблер.

Ще разгледаме и някои системни функции.

→ Запазват се в област на паметта, която се предава в някой от регистрите на процесора.
→ Слагат се в стека на програмата и се вадят от стека на ОС.

Ще разгледаме и някои системни функции:

→ Функции за управление на процеси – **fork(), exec(), wait(), getpid()** и др.
→ Функции за работа с файлове – **open(), close(), read(), write()** и др.
→ Функции за работа с файловата система – **mkdir(), rmdir(), chmod()** и др.

Проучете:

1. Проучете защо има /bin и /usr/bin?
2. Каква е разликата между тях? А /sbin и /usr/sbin?
3. Каква е разликата между виртуална машина и операционна система?
4. Какви операционни системи познавате?
5. Каква ОС използват програмистите?
6. Каква операционна система използват обикновените потребители?

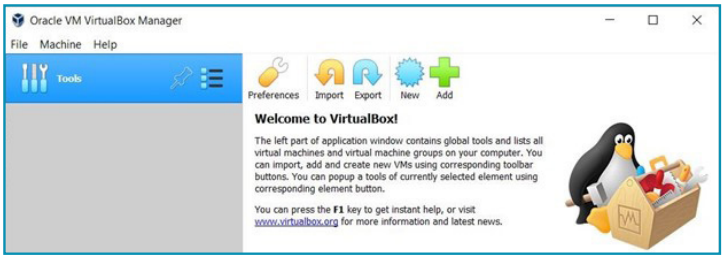
Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

ПРАКТИКУМ: СЪЗДАВАНЕ НА ВИРТУАЛНА МАШИНА ПОСРЕДСТВОМ VIRTUALBOX И ИНСТАЛИРАНЕ НА ОПЕРАЦИОННА СИСТЕМА UBUNTU.

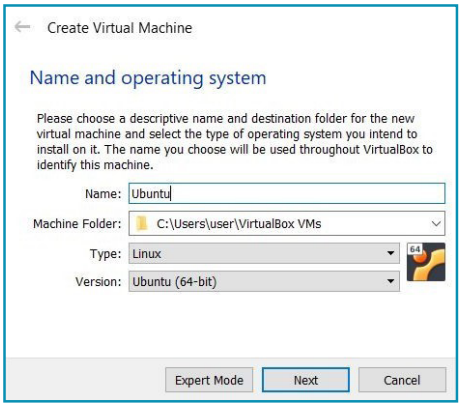
1. Свалете си VirtualBox и го инсталирайте.
→Посетете сайта https://www.virtualbox.org/wiki/Downloads
→В случай, че сегашната ви операционна система е Windows, изберете: **Windows hosts**
→Свалете файла и пуснете инсталацията.
→Инсталацията е от типа на Next -> Next -> Finish

2. Изтеглете си **Ubuntu 20.04 LTS** от сайта на Ubuntu: https://ubuntu.com/download/desktop

3. Отворете VirtualBox и създайте нова виртуална машина (бутон New):

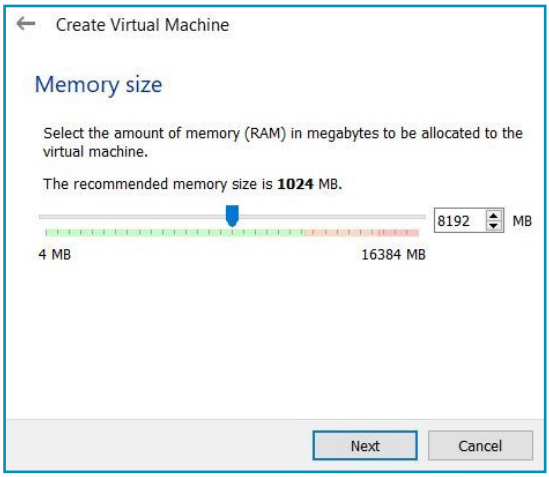


4. Изберете Linux и Ubuntu:



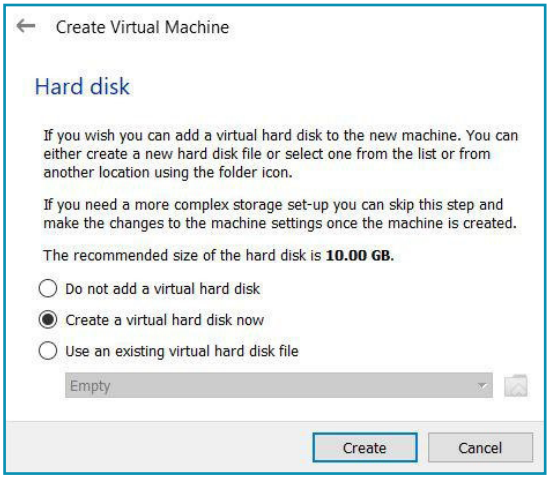
Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

5. Изберете подходящо количество памет за машината според възможностите. Препоръчително е да изберете поне 4 гб:



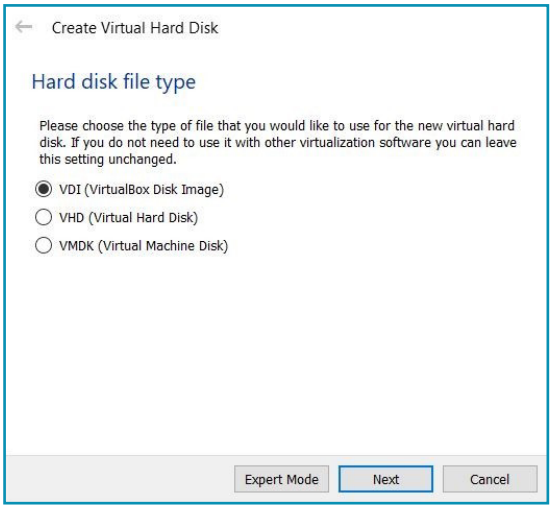
В случай, че вашата машина е по-слаба, може да изтеглите lightweight дистрибуция от типа на Lubuntu от този сайт и да ѝ дадете по-малко RAM памет: https://lubuntu.net/downloads/

6. Създайте виртуален хард диск:

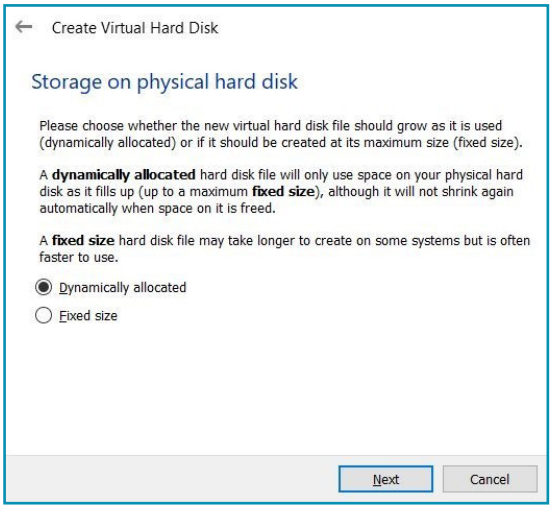


Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

7. Изберете VDI:



8. Изберете dynamically allocated:

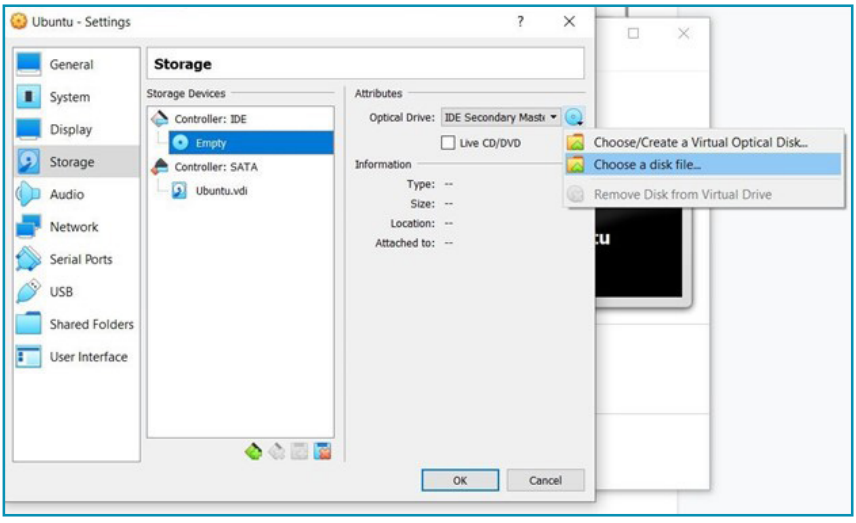


Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

9. Задайте размер на хард диска (препоръчително е да дадете поне 25 GB) и продължете.

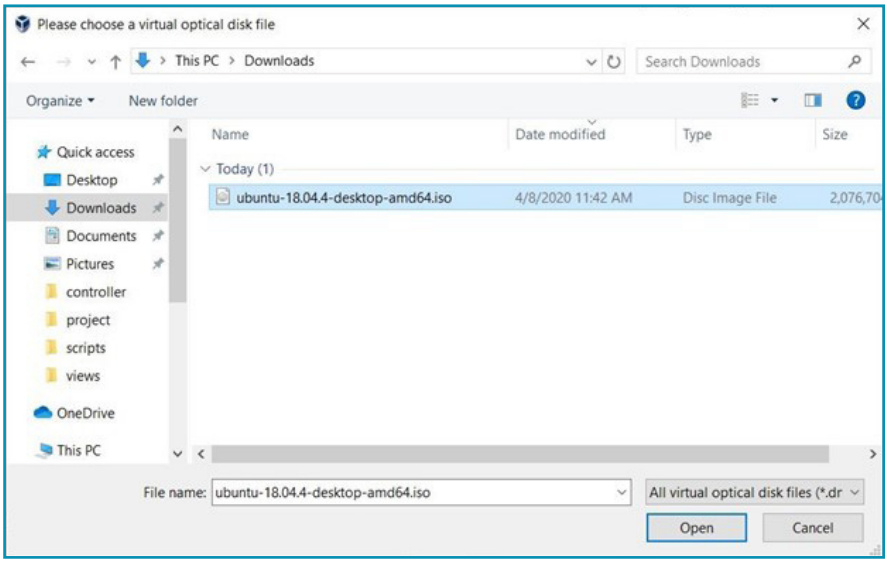
10. Вашата машина е почти готова.

11. Влезте в настройките и посочете изтегления ISO файл:

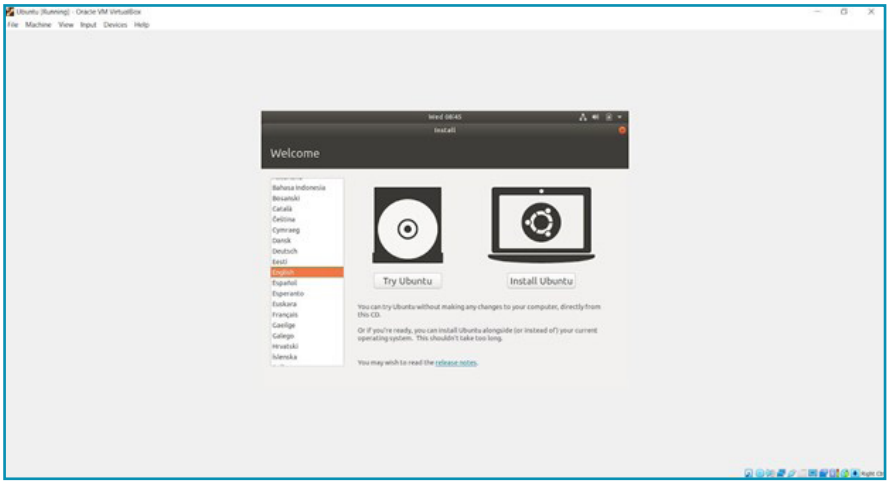


Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

12. Намерете и изберете файла iso файла:

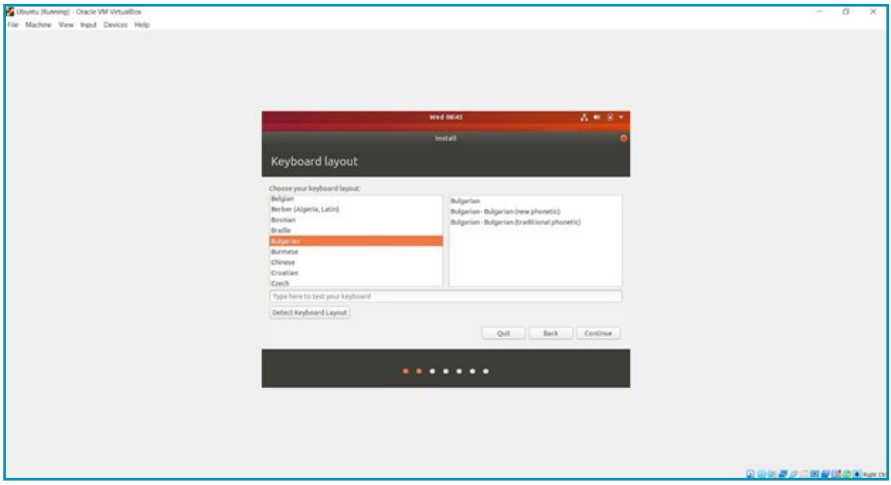


13. Стартирайте машината и започнете инсталацията.

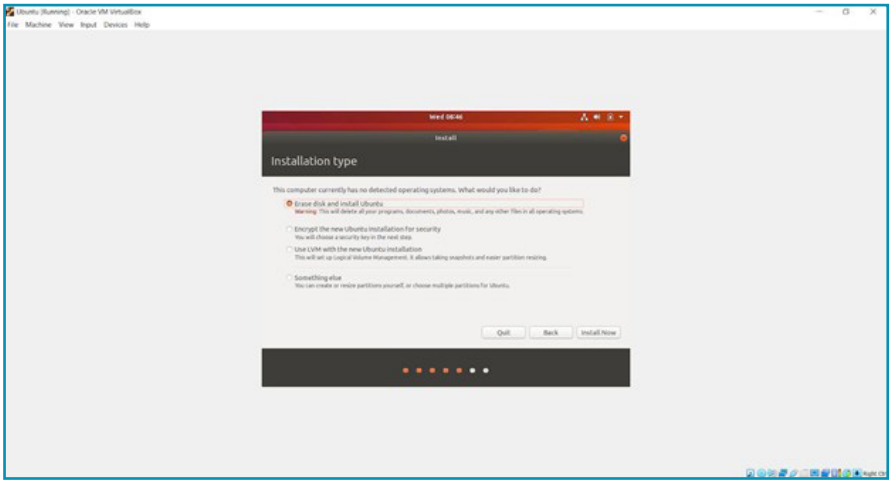


Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

14. Не забравяйте да изберете и българска клавиатурна подредба:

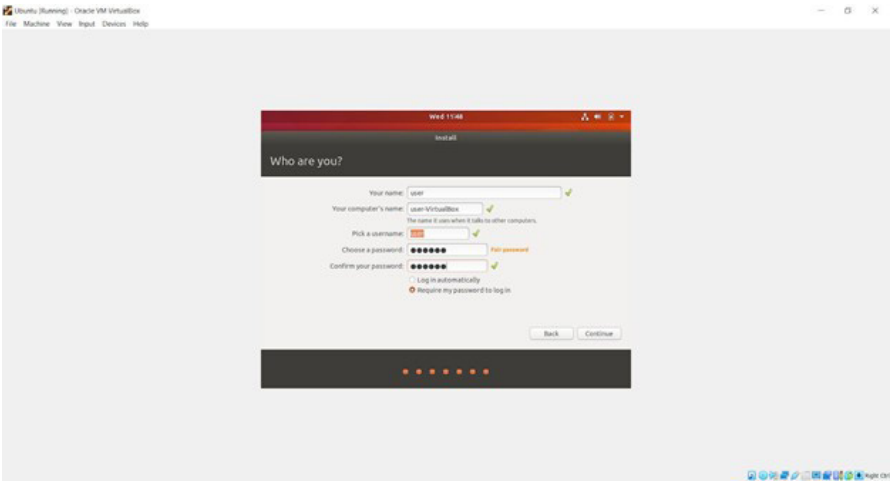


15. В случай, че правите чиста инсталация върху виртуалната машина изберете „Erase disk“ опцията. Ако инсталирате Ubuntu като втора ОС на вашата система, ще имате опция за инсталацията на Ubuntu съвместно с Windows.



Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

16. Задайте потребителските си настройки.



17. Инсталаторът ще качи файловете и ще завърши инсталацията. След това ще може да използвате вашата нова ОС.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

ГЛАВА 3 ПАКЕТНИ СИСТЕМИ

Какво ще научим?

- Какво са пакетните системи - мениджъри на пакети rpm, dpkg, apt, yum, pkgtool, pacman?
- Къде най-вече се прилагат пакетните системи?
- Инсталиране на софтуер в ОС

1. Пакетни системи

Повечето модерни Unix-подобни операционни системи предлагат централизиран механизъм за намиране и инсталиране на софтуер. Софтуерът обикновено се разпространява под формата на **пакети** , съхранявани в **хранилища** . Работата с пакети е известна като **управление на пакети**, като пакетите предоставят основните компоненти на операционната система, заедно със споделени библиотеки, приложения, услуги и документация.

Система за управление на пакети прави много повече от еднократна инсталация на софтуер. Също така предоставя инструменти за надграждане на вече инсталирани пакети т.е. хранилищата помагат да се гарантира, че кодът е проверен за използване във Вашата система и че инсталираните версии на софтуера са одобрени от разработчици и поддържащи пакети.

2. Пакети

Пакетите са компресирани файлове, които:

→ Съдържат всички файлове на дадено приложение за инсталация.
→ Съдържат инструкции за инсталация за ОС.
→ Съдържат информация за зависимости (други пакети, които трябва да се инсталират).

3. Какво са пакетните мениджъри?

Пакетните мениджъри са средство за управление на пакети, които позволяват:

→ инсталиране на пакети от софтуер;

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

→ обновяване на пакети от софтуер;
→ изтриване на пакети от софтуер.

Пакетните мениджъри се грижат автоматично за удовлетворяването на зависимостите от други пакети. Популярни пакетни мениджъри са: apt, rpm, yum и други.

Когато конфигурирате сървъри или среди за разработка, често е необходимо да ползвате пакети, които се намират извън официалните хранилища. Пакетите в стабилното издание на дистрибуция може да са остарели, особено когато става въпрос за нов или бързо променящ се софтуер. Независимо от това, управлението на пакети е жизненоважно умение за системните администратори и разработчиците.

3.1. Хранилища

→ Пакети.
→ Позволяват изтегляне и инсталиране на пакети.
→ Позволяват търсене на пакети по име и ключови думи.

Работа с **APT**

APT е инструментът за работа с пакети в Ubuntu. Нека да разгледаме някои команди:

→ Инсталация на пакет
sudo apt install <име на пакет>
→ Деинсталация на пакет
sudo apt remove <име на пакет>
→ Деинсталация на пакет с премахване на конфигурационните файлове
sudo apt purge <име на пакет>
→ Към командите може да се подава и повече от един пакет - имената на отделните пакети се разделят с интервал
→ Синхронизиране на информацията за наличните пакети в хранилището:
sudo apt update
→ Обновяване на версиите на пакетите и ядрото:
sudo apt upgrade

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

В заключение, се запознахме с основните операции, които могат да бъдат полезни при работа с различните операционни системи и използването на пакетната система **APT**.

Проучете:

1. Проучете следните пакетни системи: rpm, yum?
2. Къде се използва пакетният мениджър rpm?
3. Защо трябва да използваме пакетни мениджъри?
4. Разгледайте рейтинга на различните пакетни мениджъри?

Практикум: Проучете *как става инсталирането и деинсталирането на приложен софтуер в операционна система Windows 10.*

Практикум: Проучете *как става инсталирането и деинсталирането на приложен софтуер в операционна система Ubuntu.*

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

ГЛАВА 4 ПРОЦЕСИ И ПАМЕТ

Какво ще научим?

- Какво е физическа и логическа памет?
- Какво са процесите?
- Какви състояния имат процесите на една операционна система?
- Как ОС обработва процесите?

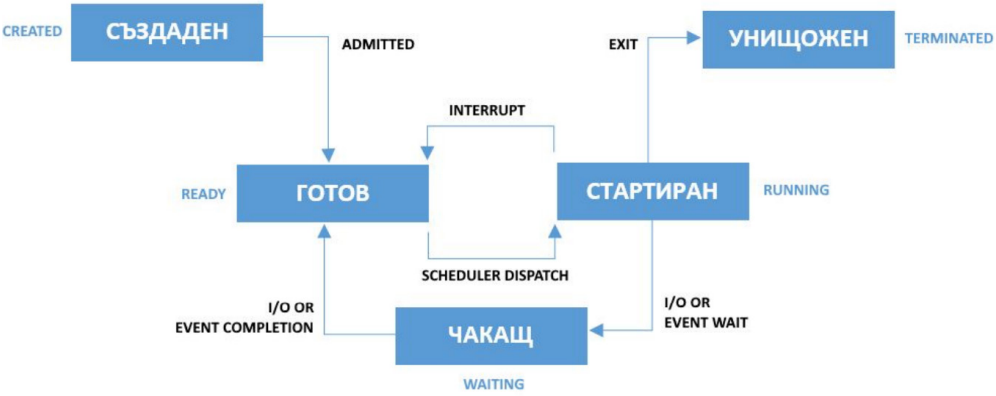
1. Процеси

→ Процесът извършва задачи в рамките на Операционната система.
→ Процес - програма в режим на изпълнение
→ Всеки процес има уникален идентификационен номер (pid)
→ Процесите се състоят от данни (data), ресурси (resources), състояние (state) и виртуализиран компютър (virtualized computer).

2. Състояние на процеси

По време на изпълнението процесът може да промени състоянието си при определени обстоятелства. Процесът може да има едно от следните състояния:

→ Running
→ Waiting
→ Stopped
→ Zombie



Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

→ Логическа памет – как програмата вижда паметта
→ Физическа памет – как процесорът вижда паметта

3. Обработка на процесите от машина

→ Съвременните компютри и ОС поддържат огромно количество процеси едновременно.
→ Съвременните компютърни системи разполагат с многоядрени процесори или дори няколко процесора.
→ Процесите трябва да се обработват бързо, а изчислителният ресурс да се използва оптимално.

4. Process Control Block структура

PCB - структура от данни, която съдържа информация за процеса, която е необходима за оптималната работа на всеки един процес. Сред данните са:
→ състояние на процеса;
→ номер на процеса;
→ брояч на инструкциите; регистри;
→ списък на отворени файлове;
→ информация за планировчика на процесите и др.

5. Контекст на процес

При изпълнение за процесът се пази информация:
→ Регистри
→ Брояч на инструкциите
→ Памет използвана от процеса и др.

Контекстът на процес наричаме нужната информация, която е необходима за възобновяване на процеса, в случай на прекъсване.

Превключване на контекста

Често процесите се изпълняват с прекъсвания – тогава достъпът до изчислителен ресурс на даден процес се прекъсва временно за сметка на друг процес.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

При смяна на процесите се извършва и превключване на контекста:
→ Запазваме контекста на текущия процес.
→ Избира се нов процес, чието състояние минава в състояние на изпълнение, заменяйки стария процес.
→ Връщаме стария процес, зареждайки го в процесора, използвайки PCB и стойностите за регистрите.

6. Планиране на процеси
→ Планирането на процеси позволява на ОС да задели интервал от време на изпълнение от процесора за всеки процес.
→ По този начин се постига висока производителност на процеса и минимално време за отговор на програмите.

Сътрудничещи (cooperating) процеси
→ Процеси, които могат да въздействат на други процеси или върху които се въздейства от други процеси.
→ Например: споделяне на общи данни, комуникация между процесите.

Причини за сътрудничество
→ Модулярност – разпределяне на обща задача между по-малки задачи, които се изпълняват от различни процеси, с цел ефективност. Споделяне на информация – например, достъп до един и същ файл едновременно.
→ Удобство.
→ Забързване на изчислението.

Методи за сътрудничество
→ Споделяне на общи файлове, променливи, БД и др. Важно е да осигурим адекватен достъп, който да предотврати загуби на информация и липса на синхронизация.
→ Комуникация между процесите – чрез изпращане на съобщения. Тук може да възникне deadlock (мъртва хватка), ако един процес чака съобщение от друг, а другият също чака съобщение от него. Възможно е да се предизвика и starvation (процесът никога не получава съобщение).

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

7. Памет – физическа и логическа памет

→ Логическа памет наричаме концепцията, която описва как програмата вижда и работи с паметта.
→ Физическа памет наричаме концепцията, която описва как процесорът вижда и работи с паметта.

Логическо адресно пространство

Защо ни е логическо адресно пространство, а не ползваме физическото?
→ Абстракция – така не е нужно приложението да познава физическата памет.
→ Изолация и защита – всяко приложение има собствено логическо адресно пространство.

Виртуална памет – техника, при която цялата ОП се контролира от ОС. В състава на виртуалната памет може да влиза:
→ RAM памет;
→ Дискова памет.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

Проучете:

1. Коя RAM памет е най-бърза?
2. Коя ROM памет е най-бърза?
3. Какво е поколението на ROM паметите през годините от създаването им?
4. Какво е поколението на RAM паметите от създаването им?
5. Защо е добре да използваме виртуална памет?

ПРАКТИКУМ: УПРАВЛЕНИЕ НА ПРОЦЕСИТЕ С КОМАНДА PS.

Първо ще научим нещо много важно, какво е **shell**?

Shell е потребителски интерфейс за достъп до услуги на съответната операционна система. Като цяло, операционните системи използват или интерфейс с команден ред, или графичен потребителски интерфейс.

Така изглежда командният прозорец на операционната система **Linux**.

```
petar@petar-Precision-M4800:~$ ps a
    PID TTY      STAT   TIME COMMAND
   1992 tty2     Ssl+   0:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME
_
   1996 tty2     Sl+   65:21 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/use
r
   2049 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --syst
e
 162025 pts/0    Ss     0:00 bash
 162036 pts/0    R+     0:00 ps a
```

Фиг. 1

За да покажете всички процеси, които вървят от текущия shell, използвайте ***ps***. Ако не е пуснато нищо друго, то върната информация ще изглежда подобно:

```
ps
PID  TTY          TIME CMD
5763 pts/3    00:00:00 zsh
8534 pts/3    00:00:00 ps
```

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

Резултатът съдържа 4 колони информация:
- PID – идентификационния номер на процеса;
- TTY – името на конзолата, с която е логнат потребителя;
- TIME – количеството процесорно време, което процесът е изразходил;
- CMD – командата, свързана с процеса.

Как да изведем всички процеси?

Вариантът е чрез ***ps -e*** опцията.

```
ps -e
PID TTY          TIME CMD
  1 ?        00:00:01 systemd
  2 ?        00:00:00 kthreadd
  3 ?        00:00:00 ksoftirqd/0
....
```

Може да комбинираме тази опция с -f и -F опциите, за да изведем повече информация за процесите. Опцията -f извежда пълната информация:

```
ps -f
UID          PID  PPID  C STIME TTY          TIME CMD
root           1     0  0 19:58 ?        00:00:01 /sbin/init
root           2     0  0 19:58 ?        00:00:00 [kthreadd]
root           3     2  0 19:58 ?        00:00:00 [ksoftirqd/0]
...
```

```
ps -F
UID          PID  PPID  C    SZ   RSS PSR STIME TTY          TIME CMD
root           1     0  0 13250  6460   1 19:58 ?        00:00:01 /sbin/init
root           2     0  0     0     0   1 19:58 ?        00:00:00 [kthreadd]
root           3     2  0     0     0   0 19:58 ?        00:00:00
[ksoftirqd/0]
...
```

Друг предпочитан и често използван начин е: ***ps aux.***

Как да изведем процесите на даден потребител?

За да изведете всички процеси на даден потребител, използвайте опцията -u. Тази опция поддържа идентификационния номер или потреби-

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

телското име:
ps -u pesho

Как да изведем процесите на дадена група?
За да изведете всички процеси на дадена група, използвайте опцията -g. Тази опция поддържа идентификационния номер или името на групата:
ps -g students

→ Процеси по номер/а на процесите:
ps -p 12608 3995

→ Процеси по име на изпълнимия файл:
ps -C apache2

→ Йерархия на процесите
ps -eH
ps -e --forest

→ Вземане на PID:
ps -ef | grep firefox
pgrep firefox

→ Търсене за процес, който да убием:
pkill firefox

→ Сортиране на процес по употреба на памет:
ps aux --sort=-pcpu,+pmem

→ Извеждане на дъщерните процеси на даден процес:
ps -o pid,uname,comm -C apache2
https://shapeshed.com/unix-ps/
https://www.binarytides.com/linux-ps-command

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

ГЛАВА 5 УСЛУГИ

Какво ще научим?

- Базови услуги services в операционната система
- Какво е системата за инициализацията?
- Използване на команди за стартиране на услуги
- Използване на команди за спиране на услуги по график
- Различните интернет протоколи
- Практикум за инсталиране на SSH сървър

1. Услуга

Услуга (service) се нарича приложение, което работи във фонов режим. Потребителите не си взаимодействат пряко с такъв вид приложения. Поради тази причина обичайно услугите нямат графичен потребителски интерфейс. Услугите изпълняват важни системи функционалности. В Линукс услугите се наричат още и демони (daemons, Disk And Execution MONitor).

2. Система за инициализация

→ Управлява централизирано услугите.
→ Стартира услугите в определен момент.
→ Следи и контролира работата на услугите.
→ systemd – системата за инициализация в Ubuntu Linux.
→ Инициализира ядрото на ОС и стартира нужните услуги на ОС след това.
→ Systemd следи за аварийни ситуации в услугите и се опитва да ги рестартира при възникване на такива.

3. Управление на услуги (services)

Управлението на услуги в systemd се случва с инструмента systemctl.
Той помага за:

→ стартиране на услуги;
→ спиране на услуги;
→ рестартиране на услуги;

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

→ проверка на състоянието на услуга;
→ анализ на ефективността на услуга.

4. Команди на systemctl

→ Стартиране на услуга
sudo systemctl start услуга
→ Спиране на услуга
sudo systemctl stop услуга
→ Рестартиране на услуга
sudo systemctl restart услуга
→ Презареждане на услуга (без спиране на нормалната ѝ работа):
sudo systemctl reload услуга
→ Пускане при стартиране
sudo systemctl enable услуга

→ Изключване на пускането при стартиране
sudo systemctl disable услуга

→ За да изкарате всички активни компоненти на systemd:
systemctl list-units

→ Списък с всички аквитни и неактивни компоненти на systemd, които е направен опит да бъдат заредени:
systemctl list-units --all

→ Списък с всички компоненти на systemd:
systemctl list-unit-files

→ Логове свързани със systemd:
journalctl

→ Справка за статуса на услуга:
systemctl status услуги

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

5. Стартиране на услуги по график

Стартиране на задачи по график

Linux разполага с механизми за стартиране на задачи, които стартират задачи през определен интервал от време или в точно определен момент

Планиране на задачи с cron

→Cron помага да стартираме задачи на определен времеви интервал (всяка минута, всеки час, всеки ден…).
→Управлява се от crontab файла.
→За да видите текущия списък от планирани задачи: crontab -l.
→За да видите текущия списък от планирани задачи за потребител: sudo crontab -u потребител -l.

Редактиране на crontab

→За да редактирате crontab, използвайте командата crontab -e.
→Всеки ред в crontab дефинира задача, форматът е следният: <minute><hours><day_of_month><month><day_of_week><command_to_run>
→Възможни стойности:
1. minute: 0 to 59
2. hours: 0 to 23
3. day of the month: 1 to 31
4. month: 1 to 12
5. day of the week: 0 (Sunday) to 6 (Saturday)

Ако използвате * всяка възможна стойност ще бъде използвана. Можете да генерирате подходящ запис за crontab с помощта на: https://crontab-generator.org/

at

В Ubuntu се инсталира допълнително

Проучете: Разгледайте повече информация за at на адрес: https://www.geeksforgeeks.org/at-command-inlinux-with-examples

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

Базови услуги (services в OS): ssh (keys), ftp

SSH

Мрежовият протокол Secure Shell (SSH) предоставя на потребителите сигурна и криптирана комуникация между отдалечени хостове чрез несигурни мрежи като интернет. Той предлага силно удостоверяване и сигурен криптиран канал, за да обменя данни с поверителност и цялост и да изпълнява безопасно отдалечени команди. SSH протоколът се използва главно в Linux и Unix базирани системи. SSH е създаден, за да бъде сигурна алтернатива на Telnet.

SSH използва криптография с публичен ключ за удостоверяване на отдалечените хостове и се използва широко за влизане в отдалечени системи и за изпълнение на отдалечени команди. Чрез използването на протокола SSH могат да бъдат предотвратени злонамерени атаки като подслушване, отвличане на съобщения за модифициране на прехвърлянето на данни, атаки „човек в средата“ и пренасочване на връзки към фалшиви сървъри, тъй като използва криптирана връзка за транзит на данни.

FTP

Протокол от тип клиент-сървър за обмен на файлове между машини, свързани в локална мрежа или чрез Интернет.

Връзката към FTP работи чрез:

- Потребителско име.
- Парола.
- Адрес на сървър + порт на сървър (по подразбиране е 21).

Мрежови протоколи

SCP

Протоколът за защитено копиране (SCP) сигурно и лесно копира файлове между отдалечените компютри в мрежата и използва SSH защитена връзка за прехвърляне на файловете. Той също така предлага същата сигурност като криптирания SSH. Той се предлага най-вече в системи Unix и Linux, но има различни графични интерфейси, като е

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

достъпен за всички операционни системи.

SCP е комбинация от RCP и SSH протоколи. RCP осъществява трансфера на файлове между два компютъра и SSH протоколът осигурява удостоверяване и криптиране с помощта на криптография с публичен ключ за SCP.

Каква е разликата между SSH и SCP?

→ Както SSH, така и SCP се използват за сигурен обмен на данни между компютрите в мрежата въз основа на криптиране с публичен ключ.

→ SSH протоколът е за създаване на защитен криптиран канал между двойка отдалечени устройства, докато SCP протоколът е за безопасно прехвърляне на файлове между двойка хостове. Тъй като SCP използва SSH връзка за своята работа, SSH и SCP протоколите са еднакви, но има някои ключови разлики.

→ SSH протоколът се използва широко за влизане в отдалечени системи и за управление на отдалечени системи, докато протоколът SCP се използва за прехвърляне на файлове между отдалечени компютри в мрежа.

DNS (Domain Name System**)** е система за имена в интернет, чрез която домейните се транслират в IP адреси. Видимата част на DNS системата са домейните, а целта им е да се предостави начин за наименуване на ресурсите/услугите в интернет. Почти всички услуги в интернет ползват DNS като уеб, имейл услугата, услугите за трансфер на файлове и други. Благодарение на DNS отваряме уеб сайтовете в уеб браузъра, като само изпишем името на домейна им. Без тази система, вместо домейн ще трябва да въвеждаме IP адреса на сървъра, на който реално се намира съдържанието на сайта.

DHCP, абревиатура от Dynamic Host Configuration Protocol е мрежов протокол за динамично конфигуриране на хостове. С неговата помощ всяко устройство, оперирацо на трети слой от OSI модела (компютър, мобилен телефон, маршрутизатор и пр.), може да се сдобие автоматично с default gateway адрес, IP адрес, събнет маска и адрес на DNS сървър.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

Ясно е, че идеята при използване на DHCP протокола е да се улесни управлението на мрежата. Повечето безжични рутери използват DHCP за раздаване на адреси.

Проучете:

1. Какви протоколи и мрежови услуги се използват от един уеб сървър?

2. За какво служи системата VNC и протоколът RFB?

3. Как може да правите автоматичен бекъп на дадена директория с помощта на crontab?

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

ПРАКТИКУМ: ИНСТАЛИРАНЕ НА SSH СЪРВЪР.

В рамките на този практикум ще инсталираме SSH сървър, който ще достъпим после чрез SSH клиентът за Windows - PuTTY

1. Инсталиране на сървър Въведете следната команда в терминала, за да инсталирате софтуера: sudo apt install openssh-server

2. Разрешаване на service Въведете следната команда в терминала, за да разрешите ssh service-а (услугата). Това е необходимо, за да може ssh сървъра да е готов за комуникация с външния свят по всяко едно време. Нека да припомним, че услугата (service) е процес, който върви във фонов режим, готов да бъде използван. Командата, за да разрешим на нашата услуга да работи е: sudo systemctl enable ssh

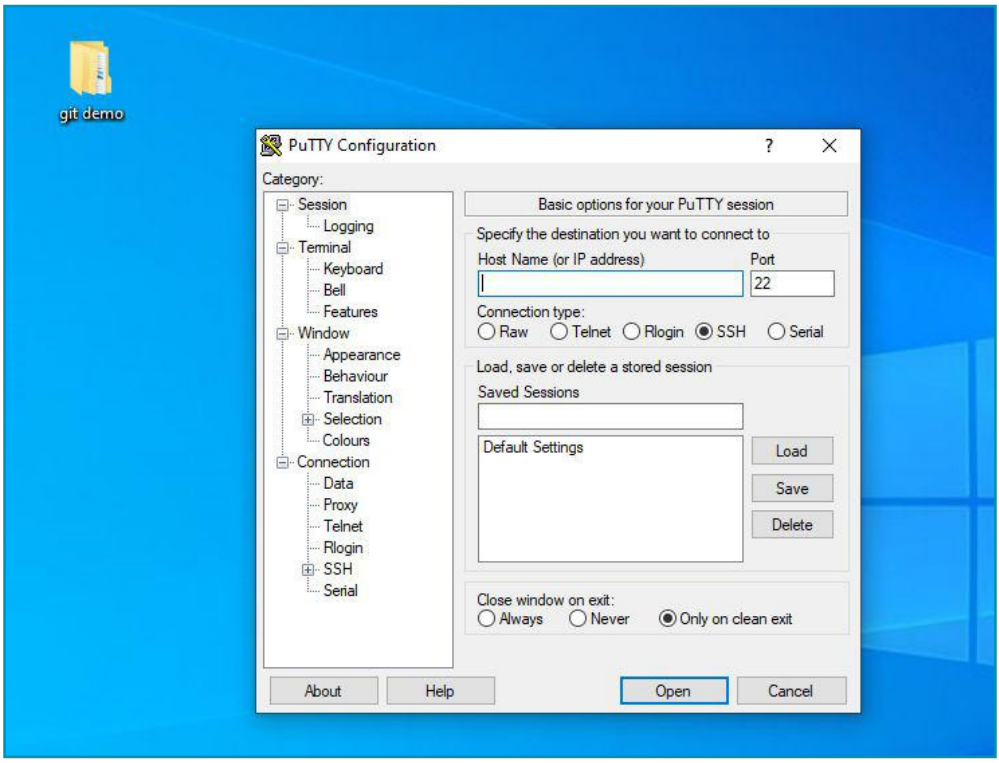
3. Стартиране на service-а: sudo systemctl start ssh След като сме извършили това, трябва да имаме работещ SSH сървър. За да потвърдим, че сървърът работи, трябва да го достъпим чрез клиент. За да разберем какъв е IP адресът на сървъра, първо трябва да изпълним командата ifconfig. Тя ще върне няколко групи от записи. В случая, тъй като Linux е на виртуална машина, се опитайте да използвате записа, който е маркиран като virbr0 (възможно е цифрата накрая да е различна).

Забележка: При различна инсталация и обстоятелства е възможно да има разлика между резултата, който трябва да гледате. Ако не успеете, пробвайте с някой от другите. IP адресът е записан след inet секцията в резултата, който получавате, както е показано по-долу:

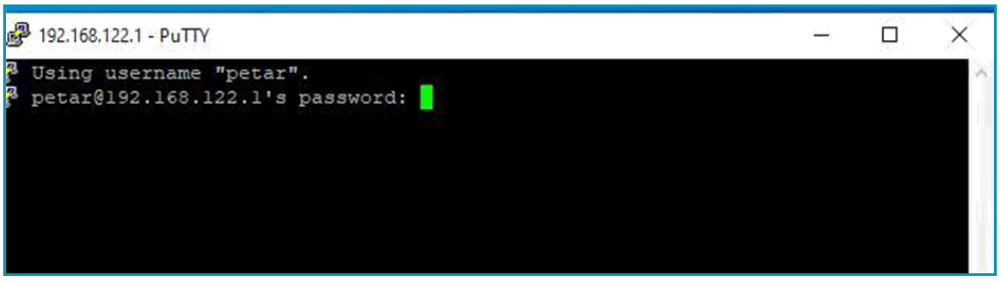
```
virbr0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 192.168.122.1  netmask 255.255.255.0  broadcast 192.168.122.255
        ether 52:54:00:d8:44:05  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Сега инсталирайте PuTTY на Windows-скатa си машина. Изтеглете го оттук: https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html Изберете putty-0.73-installer.msi Инсталирайте клиента.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

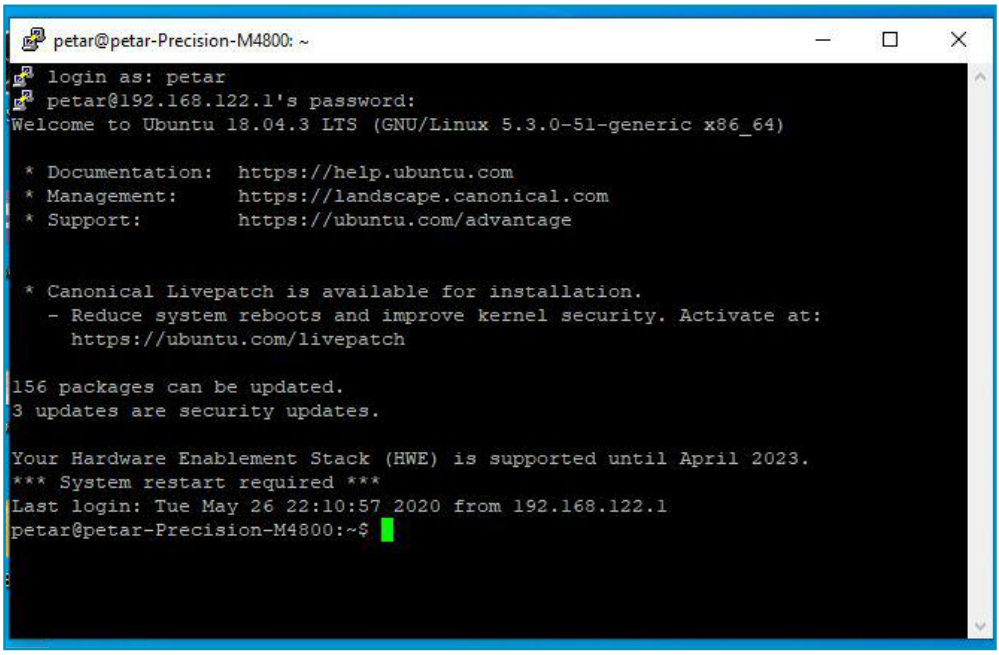


В полето Host Name (or IP address) напишете името на потребителя си в Линукс@IP адреса, примерно: gosho@192.168.23.1 - не забравяйте да смените с вашия IP адрес и името на вашия потребител в Линукс. Ако не сте сигурни как се казва потребителя ви, използвайте командата whoami в Линукс, за да разберете. При успешна връзка ще бъдете попитани за паролата на потребителя:



Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

Въведете я и натиснете Enter. Ако всичко е успешно и работи нормално, ще стигнете до етап, където може да пускате команди, все едно сте в реалния линукс терминал на машината:



Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

ГЛАВА 6 ФАЙЛОВИ СИСТЕМИ

Какво ще научим?

- Какво е **файловата система(ФС)**?
- Какво е **логическа** и **физическа** организация на ФС?
- Как е организиран **хард дискът**?
- Какво е **MBR**?
- Какво е дял и разделяне на **дялове(partitioning)**?
- Монтиране и демонтиране на файлови системи?
- Практикум форматиране на usb флаш драйв(usb flash drive) и избиране на подходяща файлова система

1. Какво е файлова система?

Операционната система(ОС) дава абстрактен слой, скриващ особеностите на запомнящите устройства. Този слой предоставя логическа единица за съхранение на данни – файл. Файлът е съвкупност от свързани данни и програмите могат да се съхраняват във файлове. Файловете се групират в директории и имат права, които показват кой и какво може да прави с файла (четене, писане и изпълнение).

2. Файловата система отговаря за:

→Създаване/изтриване на файлове и директории.
→Поддръжка на основните операции – четене, писане, преместване, преименуване и др.
→Устойчиво съхранение на файловете и директориите.
→Контролира организацията, достъпа и съхранението на данни.

3. Основни функции:

1. Определя къде се намира файлът, как се казва, какъв е неговият формат и размер;
2. Определя параметрите на файла;
3. Определя кои участъци от логическия раздел са свободни и кои не;
4. Определя максимален брой файлове във физически раздел.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

4. Какво е логическа и физическа организация на ФС?

→Логическа организация на ФС

1. Представяне на данните от файловата система под формата на файлове и директории

2. Възможност за изпълнение на команди върху файлова система

3. Предоставя API за останалите приложения за реализация на операции OPEN, CLOSE, READ, WRITE и др.

→Физическа организация на ФС

Борави с данните върху самото устройство (диск)

5. Как е организиран дискът?

1. Хард дискът е разделен на сектори (sectors).

2. Определено количество сектори формират дял(partition).

3. Всички сектори в един дял трябва да са съседни.

6. Какво е MBR?

→MBR - Master Boot Record.

→Първият сектор от диска, който се използва за първоначално зареждане (boot) и стартиране.

→MBR обичайно е 512 байта, което съответства на един сектор
https://www.ionos.com/digitalguide/server/configuration/what-is-mbr/

→Постепенно се замества от GPT
https://www.diskpart.com/gpt-mbr/mbr-vs-gpt-1004.html

7. Какво е дял и разделяне на дялове (partitioning)?

→Дискът може да бъде разделен на дялове, с цел подпомагане на организацията на файловете и повишаване на сигурността.

8. Какво е монтирането и демонтирането на файлови системи?

→Монтиране – процес, при който файловете на дадено устройство стават достъпни за останалата част от системата.

→Демонтиране – процес, при който се прекъсва достъпа до файловете, информацията, която е трябвало да се запише се записва и се позволява безопасното премахване на устройството.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

9. За какво служи fsck?

→ fsck (file system check) е инструмент в командния ред, който позволява проверка на консистентността на файловата система.
→ Fsck помага за поправяне на проблеми във файловата система.
→ Fsck работи с различни файлови системи.
→ Fsck е възможно спасение, когато файловата система не иска да се монтира или не се стартира.

10. За какво служи parted?

→ Parted е известен инструмент за лесно управление на дялове.
→ С негова помощ може да се създаде, изтрие или преоразмери дял, като да се избере и файлова система за конкретния дял.

12. Ext файлови системи

→ Вид файлови системи (ext, ext2, ext3, ext4)
→ Използват се сериозно в Линукс света
→ https://opensource.com/article/17/5/introduction-ext4- filesystem

13. Xfs файлова система

→ Още една популярна в Линукс света файлова система.
→ Тя е оптимизирана за висока производителност
→ https://www.electronicdesign.com/industrial-automatio n/article/21804944/whats-the-difference-between-linu x-ext-xfs-and-btrfs-filesystems

14. Ntfs файлова система

→ Файлова система, разработена от Microsoft за Windows
→ https://www.datto.com/library/what-is-ntfs-and-how-do es-it-work

15. Fat файлови системи

→ Остаряла файлова система
→ Използва се от някои флашки, дискети, дискове и др.
→ https://www.guidingtech.com/11205/difference-betwee n-ntfs-and-fat-32-file-systems/

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

ПРАКТИКУМ: КАК СЕ ФОРМАТИРА ФЛАШ ПАМЕТ (USB FLASH DRIVE) ПОД ОПЕРАЦИОННА СИСТЕМА WINDOWS 8.1 И MAC.

Флашката е енергонезависима компютърна памет, която служи за бърза и лесна връзка с компютъра с помощта на **USB** (universal serial bus) шина. Тя е малка по размер и може да се използва за многократно записване на информация, което я прави много удобна. Чрез форматиране от флашката се изтрива всякаква записана информация. По този начин тя се изчиства напълно и става готова за записване на нови неща. Това означава, че чрез форматирането ще бъдат **премахнати и всякакъв тип вируси** от флашката, ако има такива. При форматиране също може да се зададе и файловата система на флашката, за да стане съвместима с някои по-специфични устройства или операционни системи.

Как се прави?

Как се форматира флашка под Windows?

1. Включете флашката в някой от **USB** портовете на компютъра или лаптопа.

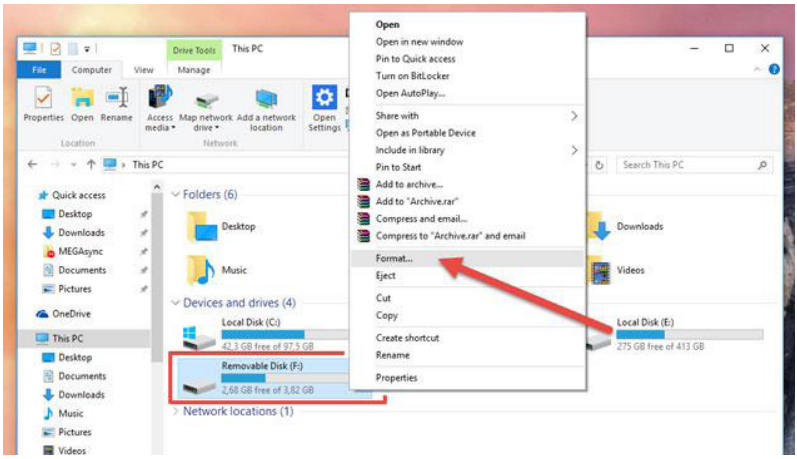
2. Изчакайте 1-2 секунди компютърът да разпознае флашката. Долу вдясно трябва да ви излезе малко прозорче, което казва, че флашката е разпозната и готова за използване.

3. Отидете в ***My Computer*** (Моят компютър) като се навигирате от Старт менюто или с клавишна комбинация ***Win+E***.

→При ***Windows 8.1, 10*** отидете на Старт менюто или натиснете ***Win***, напишете ***this pc*** и натиснете ***Enter***.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

4. С десен клик щракнете върху иконата на флашката, която вече трябва да се е появила под **„Devices with Removable Storage“.**

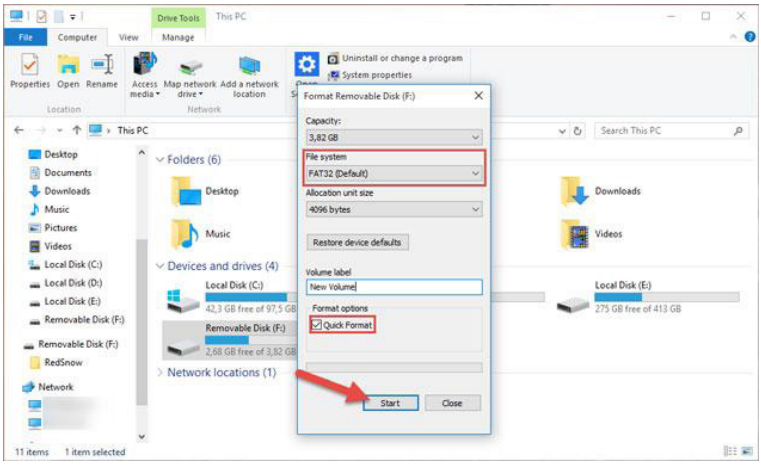


5. В появилото се прозорче с ляв клик щракнете върху „***Format…***“. Ще се отвори прозорец с настройки за формат.

→Оставете „***Capacity***“ както си е. Това е капацитетът на флашката. Компютърът ви автоматично го засича.

→Щъкнете тикчето на „***Quick Format***“ (Бърз формат), ако искате да направите формата по-бързо.

→Оставете „***Allocation unit size***“, както си е по подразбиране.



Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

6. Избере файловата система, в която да бъде форматирана флашката. Имайте някои неща предвид, когато правите избора на файлова система.

А) FAT – това е най-простата система за запазване на файлове. Тя е по-бавна, но поради простотата си рискът от несъвместимост с други устройства е много малък. Въпреки това много от съвременните устройства вече не я използват по подразбиране.

Б) FAT32(препоръчително) – чрез нея постигате максимално дисково пространство. Тя е идеална, ако ще използвате флашката на **Windows** или **Mac**. Този тип файлова система обаче не позволява качването на файлове, по-големи от 4GB, както и не се поддържа от флашки, по-големи от 32GB.

В) exFAT32 – позволява записването на файлове по-големи от 4GB. Разпознава се както от Windows, така и от Mac OS X. Може да срещнете трудности при по-стари системи и устройства.

Г) NTFS – най-бързата файлова система. Компютри с Windows могат да четат и да записват файлове, но Mac OS X компютри могат само да четат от NTFS, не и да записват файлове.

7. Започнете форматирането, като натиснете бутона „***Start***“.

Натиснете бутона „***OK***“ в появилия се прозорец.

Когато процесът завърши, цялата информация от флашката ще се изтрие и тя ще бъде готова за ползване, форматирана според зададените по-рано опции.

Как се форматира флашка под Mac

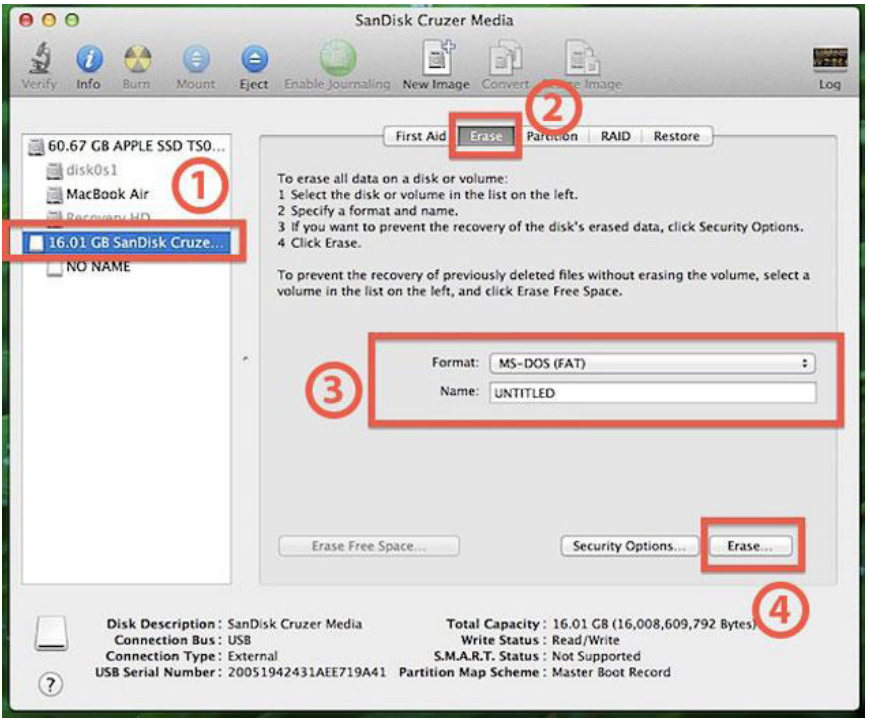
1. Свържете флашката с компютъра.

2. Стартирайте ***„Disk Utility“***, което се намира в ***„Aplications“*** > ***„Utilities“***.

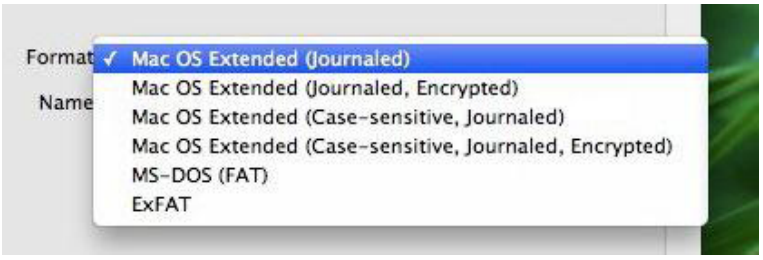
3. Щракнете върху името на флашката, което би трябвало да се вижда от лявата страна на излезлия прозорец.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

4. Щракнете на таба „***Erase***“, намиращ се горе в дясната част.



5. В полето „***Format***“ изберете файловата система, в която да бъде форматирана флашката – ***„Mac OS Extended (Journaled)“***.



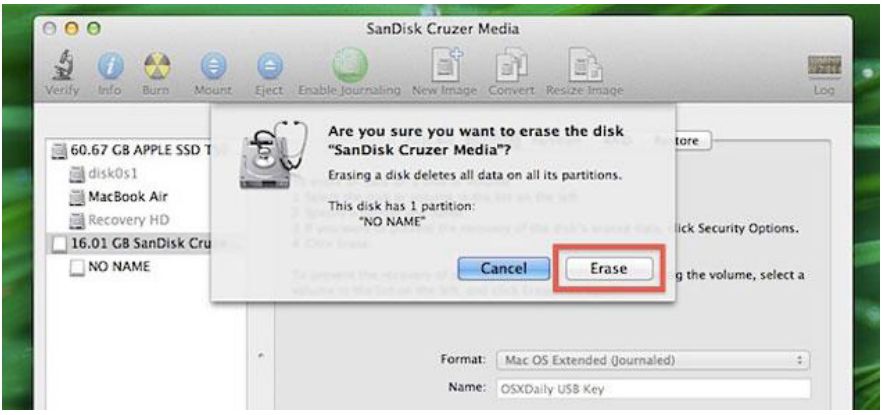
A) Mac OS X Extended (Journaled) – това е най-добрият вариант за форматиране, ако ще използвате флашката на Mac OS X. Ще ви даде максимална скорост при четене и запис на данни при OS X.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

Б) exFAT (FAT64) – позволява записването на файлове по-големи от 4GB. Разпознава се както от Windows, така и от Mac OS X. Може да срещнете трудности при по-стари системи и устройства.

В) Windows NT File System (NTFS) – доста бърза файлова система. Компютри с Windows могат да четат и да записват файлове, но Mac OS X компютри могат само да четат от NTFS, не и да записват файлове.

6. В полето „***Name***“ въведете желано от вас име на флашката.
7. Щракнете върху „***Erase***“ бутона и потвърдете.
8. В изскочилия прозорец потвърдете пак с „***Erase***“.



Това е всичко, сега флашката ще започне форматиране и цялата информация на нея ще бъде изтрита.

Проучете:
1. Какво представлява външното запомнящо устройство?
2. Какво представлява физическият достъп до данните в едно запомнящо устройство?
3. Какво представлява логическият достъп до данните в едно запомнящо устройство?
4. Какво е SSD(Solid-State Drive)?

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

ГЛАВА 7 ПИСАНЕ НА СКРИПТОВЕ

Какво ще научим?

- Какво е Shell?
- Как да създадем и изпълним Shell скрипт?
- Какво е PATH променливата?
- Как да боравим с променливи и изрази в Shell?
- Как да боравим с условни конструкции и цикли в Shell?
- Как да работим с потоци от данни и насочвания (pipes)?
- Как да работим с масиви в Shell?

1. Какво е Shell?

В множество Unix и Linux – базирани операционни системи съществува специална програма, която се нарича **Shell** (обвивка). По същество тя представлява команден интерпретатор. Тя позволява въвеждането на команди, които се изпълняват от ОС. Съществуват различни програми shell-ове. Такива, например са bash, zsh, tcsh, ksh, Fish и др.

2. Какво е Shell скриптирането?

Shell, освен команден интерпретатор, поддържа употреба и като специализиран програмен език. В този език съществуват променливи, условни и циклични конструкции, масиви, функции и др. Shell скриптирането е мощно средство за автоматизация.

3. Как се създават Shell скриптове?

Както много други файлове с програмен код shell скриптовете също са текстови файлове. Тяхното обичайно разширение е **.sh**. За да може обаче те да се изпълнят от командния интерпретатор, трябва да са налице няколко много важни момента:

→ Файлът трябва да има права за изпълнение (executable bit), това става с подходяща chmod команда от типа на **chmod +x filename.sh**.

→ Файлът трябва да стартира с **#!/bin/bash** или друго подходящо указание кой интерпретатор ще се използва.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

Опитайте сами: *Създайте и изпълнете Shell скрипт, който да извежда вашето име на екрана. Използвайте командата echo и проучете нейната употреба.*

4. Какво е Shebang(#!)?

Shell скриптовете започват с #!. Тази последователност от символи се нарича шибанг (shebang). Символите са последвани от програмата-интерпретатор, която ще се използва. Освен това в рамките на реда могат да бъдат зададени и допълнителни опции за интерпретатора.

5. Какво е PATH променливата?

В редица операционни системи PATH променливата има важно значение. Тя указва в кои директории Shell програмата може да търси изпълними файлове, които съответстват на команди. Всъщност немалка част от командите представляват изпълними файлове (програми), които Shell може да извика. По същество PATH променливата е променлива на средата (environment variable). Те могат да бъдат използвани от Shell за различни цели. Съществуват и други environment variables в Линукс - HOME, HOSTNAME, PATH, LANG, DISPLAY и др. Можете и сами да задавате и редактирате такива променливи.

6. Как трябва да се казва Shell скриптът?

Задавайте смислени имена на скриптовете. Избягвайте да кръщавате скриптовете с имена на вече съществуващи команди. Може да използвате командата type, за да разберете дали дадено име се ползва за нещо или не:

```
petar@petar-Precision-M4800:~$ type ping
ping is /bin/ping
petar@petar-Precision-M4800:~$ type pong
bash: type: pong: not found
petar@petar-Precision-M4800:~$
```

7. Как се създават променливи в Shell скриптирането?

Променливите в Shell скриптирането се създават по сходен начин с променливите от програмирането. Особено прилича на създаването на

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

променливи в Python. Синтаксисът е:

```
име=стойност
```

ВАЖНО: Не бива да поставяте интервали между името и стойността! Ако стойността на променливата съдържа интервал, то трябва да я оградите с кавички.

Пример:
x=10
message="Welcome back!"

Ако искате да разберете каква е стойността на дадена променлива, трябва да поставите $ пред името ѝ. Често се използва заедно с командата echo или в друго присвояване.

Пример:
message="Hello, $USER"
echo $message

Променливите в Shell се именуват по сходни правила с тези от други програмни езици:
→ Разрешени за ползване са само букви, числа и долна черта.
→ Първият символ в името трябва да е буква или долна черта.
→ Има разлика между главни и малки букви (case-sensitive).
→ Избягвайте имена, съставени изцяло от главни букви – обикновено с главни букви са означени променливите на средата.
→ Добър навик е да използвате малки букви за именуването на вашите променливи при писане на Shell скриптове.

В Shell е добре да използвайте къдрави скоби, особено ако желаете да направите конкатенация, например: ${foo}bar. Това ще изкара стойността на променливата foo последваната от фразата „bar". От друга страна $foobar би дало стойността на променливата foobar. Използвайте променливи на средата, когато това е възможно. Например $HOME вместо ~.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение", финансиран от Оперативна програма „Наука и образование за интелигентен растеж", съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

8. Как се въвежда стойност от клавиатурата за променлива?

Въвеждането от клавиатурата на стойност става чрез read. Може да се изведе и подканващо съобщение за потребителя, за да знае какво трябва да въведе. Пример:

→read var
→read -p „Enter your name: “ name

9. Какво да правим, ако имаме грешки в Shell скрипта?

Понякога в Shell скриптовете могат да възникнат доста неприятни ситуации, породени от неправилно изписване, грешно извикване или други грешки. Shell не обича да показва грешки и не е особено помагащ за отстраняването им в общия случай. За щастие Shell предлага дебъгване. За да използвате дебъгване на всеки ред от Shell скрипта, в първия ред задайте -x опцията: #!/bin/bash -x. Това ще активира дебъгване в целия скрипт. А ако желаете дебъгването да е само в част от скрипта, то го активирайте с командата set -x в началото на областта, която желаете да дебъгнете и го изключете с командата set +x.

```
#!/bin/bash
#... код, който не желаете да дебъгвате
#...
set -x
#код, който ще бъде дебъгнат
set +x
#още код, който НЯМА да бъде дебъгнат
```

Опитайте сами: *Създайте и изпълнете Shell скрипт, който да въвежда две числа и да изведе на екрана тяхната сума.*

10. Как се реализира условна конструкция в Shell скриптирането?

Условната конструкция в Shell се реализира по подобен начин на останалите програмни езици.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

Съществува кратка форма, която изглежда както е показано по-долу:

```
if condition; then
    #do something
fi
```

Има и пълна форма, която изглежда както е показано:

```
if condition; then
    #do something
else
    #do something else!
fi
```

11. Как се пишат логическите изрази?

Изразите се ограждат в двойки квадратни скоби:

Израз	Значение
[[$var]]	Проверява дали var не е празна.
[[$var = „hello“]]	Проверява дали var има стойност „hello“.
[[$var=„hello“]]	Присвоява стойността, връща **true** при успешно присвояване (в повечето случаи присвояването е успешно)! **Забележете как в единия случай около оператора има интервали, а в другия не!**
[[-e $filename]]	Проверява дали съществува файл с име съвпадащо със стойността на filename

12. Как се сравняват числа?

Формат: [[arg1 OP arg2]], където OP е:

→ -eq: проверка за равенство
→ -lt: проверка за по-малко
→ -gt: роверка за по-голямо
→ -le: проверка за по-малко или равно

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

→ -ge: проверка за по-голямо или равно
→ -ne: проверка за различие

Забележка: В някои имплементации работят и традиционните оператори =, <, > и т.н., но не винаги се поддържат!

13. Могат ли условните конструкции да се влагат?

Условните конструкции могат да се влагат по аналогичен начин с други програмни езици.

Пример:

```
if [[ ! -d $bindir ]]; then
    #if not: create bin directory
    if mkdir "$bindir"; then
        echo "created ${bindir}"
    else
        echo "Could not create ${bindir}"
        exit 1
    fi
fi
```

Опитайте сами: *Създайте и изпълнете Shell скрипт, който да въвежда име на директория от клавиатурата. Скриптът да провери дали директорията съществува. Ако не съществува да я създаде. Използвайте за ориентир примера по-горе.*

14. Могат ли да се проверяват поредица от условия?

Аналогично на други програмни езици, shell поддържа проверка на поредица от условия:

```
if [[ $1 = "cat" ]]; then
    echo "meow"
elif [[ $1 = "dog" ]]; then
    echo "woof"
elif [[ $1 = "cow" ]]; then
    echo "mooo"
else
    echo "unknown animal"
fi
```

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

15. Поддържат ли се операции като логическо „и“, „или“ и други?

→Може да реализирате проверка с and (и) по следния начин:
[[condition1 && condition2]]
→Може да реализирате проверка с or (или) по следния начин:
[[condition1 || condition2]]
→Може да направите условие базирано на отрицание с !:
[[! condition]]

16. Какви са потоците от данни?

Shell работи с потоци от данни. Има няколко основни вида потоци:
→Входен (stdin) – означение: 0, /dev/stdin
→Изходен (stdout) – означение: 1, /dev/stdout
→За грешки (stderr) – означение: 2, /dev/stderr
→/dev/null - унищожава всякаква информация, насочена към него

17. Можем ли да приемаме данни от друг поток, а не от stdin?

Може да приемате входни данни от друго място, а не от stdin, чрез оператор <.

Пример: grep pesho < names.txt

Забележка: grep командата се използва за търсене на съвпадение в набор от данни/файлове.

Проучете:
1. Потърсете повече информация за grep командата? Можете ли да я използвате с регулярни изрази (regular expressions)?

18. Как да насочим изхода от дадена команда към друго място?

Обичайно изходът от командите се насочва към stdout. Може да изпращате изходни данни към друго място, а не към stdout, чрез оператор >.

Пример: ls ~ > filelist.txt

Може да използвате и оператор >>.

→> операторът презаписва файла, ако вече съществува или го създава, в противен случай.
→>> операторът записва към края на файла, ако той съществува.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

19. Какви други насочвания могат да се използват?

Можете да насочите конкретен поток към дадено място с помощта на **N>**:

→ команда 2> $HOME/error.log

Можете да направите насочване и с >&N:

→ >&2 (насочва изхода към stderr, може да се запише и 1>&2)

→ 2>&1 насочва stderr към stdout

Можете да насочите едновременно stdout и stderr към един файл:

→ команда > logfile 2>&1

където logfile е път/име на файл, в който да се запазват данните

20. Може ли изходът от една команда да се ползва като вход на друга?

Командите в Линукс поддържат т.нар. piping. Това позволява изходът от една команда да се ползва като вход на друга. Пример: ls | grep file.txt. В този пример ще се изведе списък от файлове и директории, сред който ще се потърси споменаване на file.txt от grep командата.

21. Какви циклични оператори съществуват в Shell?

→ Shell предлага няколко циклични оператора:

→ while, който повтаря код, докато дадено условие **е вярно**.

→ until, който повтаря код, докато дадено условие **НЕ Е вярно**.

→ for, който обхожда колекции.

C-style for, който е цикъл с променлива-брояч.

Примери:

```
while condition; do
    #код, който се повтаря
done
```

```
for i in collection; do
    #код, който се повтаря
done
```

```
until condition; do
    #код, който се повтаря
done
```

```
for (( INIT; CONDITION; UPDATE )) do
    # повтарящ се код
done
```

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

В циклите могат да се използват аналогично на други програмни езици ключовите думи break и continue.

22. Как се работи с масиви в Shell?

В Shell се поддържат масиви по сходен начин с познатите до момента от програмирането масиви. Масивът е структура, която съдържа множество данни. Данните се съхраняват и извличат чрез индекси, започващи от 0.

Пример:

→ x[0]="hello"
→ x[1]="world"
→ Извличане: ${x[0]}
→ Извличане на всички елементи: ${x[*]}

Инициализиране на масив:

ar=(1 3 2 5 6 9)

→ Брой на елементи в масив/дължина на низ:
${#array[@]}
→ Списък с индексите в масива:
${!array[@]}

Възможно е да има липсващи индекси - не е задължително индексите да са последователни! В по-новите си версии bash поддържа асоциативни масиви.

Проучете:

1. Shell скриптовете могат да приемат параметри при извикването си, които са под формата на масив. Те могат да се използват като се напише $ последван от реда им: $0, $1, $2 и т.н. Потърсете информация как се работи с тях.

2. Разберете как може да приемате допълнителни опции с помощта на getopts.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение", финансиран от Оперативна програма „Наука и образование за интелигентен растеж", съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

ГЛАВА 8 ВИРТУАЛИЗАЦИЯ И КОНТЕЙНЕРИ

Какво ще научим?

- Какво е виртуализация?
- Какво е контейнеризация?
- Какво е Docker?
- Как да работим с Docker контейнери и да контейнеризираме приложения?

1. Какво е виртуализацията?

Виртуализацията позволява върху една и съща физическа машина да имаме инсталирани множество операционни системи. Виртуалната машина може да се третира по същия начин като физическата машина и в този смисъл позволява инсталирането на ОС и управлението на ресурси. Виртуалните машини се управляват от софтуер, наречен хипервайзър (hypervisor), който управлява ресурсите и контролира работата на виртуалните машини.

2. Какво е контейнеризацията?

Контейнеризацията е концепция за виртуализация на ниво ОС. При нея, вместо да се инсталира отделна ОС, се използва ядрото на основната ОС на машината, но се създава самостоятелна изолирана версия на потребителското пространство на ОС. Тази изолирана версия на потребителското пространство наричаме *контейнер*. Това позволява в рамките на контейнера да бъдат инсталирани конкретни версии на споделени библиотеки и софтуер, които са нужни за изпълнението на даден софтуер.

3. Какъв проблем решава контейнеризацията?

Контейнеризацията решава проблемите, свързани с необходимостта на определени приложения от конкретни версии на дадени библиотеки и съвместимостта им с операционната система. Това става като се посочи какви версии са необходими за стартирането на даден контейнер. Също така това позволява на една и съща физическа маши-

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

на да работят множество контейнери, тъй като те са изолирани един от друг. Друго предимство е, че по този начин инсталацията на необходимите за контейнера компоненти може да се автоматизира, като по този начин се пести време и се дава възможност приложенията да се стартират по-лесно.

4. Какво е Docker?

Docker е платформа, която дава възможност за разработка, доставяне и изпълнение на софтуерни приложения. Docker позволява отделянето на приложенията от инфраструктурата, като по този начин програмистите не е нужно да се тревожат за наличието на конкретни версии и библиотеки на софтуерни компоненти, необходими за приложението им. Платформата дава възможност за управление на приложенията, предоставяйки възможност за лесно стартиране, спиране и тестване, а дори и автоматизиране на софтуерни приложения.

5. Как да използваме Docker?

Може да използвате Docker като го инсталирате на машината си. Вижте повече на сайта на Docker (https://docs.docker.com/desktop/). Също така е възможно да изпробвате Docker онлайн на https://labs.play-with-docker.com/.

6. Как се контейнеризира приложение за Docker?

За да изпълним приложение в контейнер, трябва да го контейнеризираме. Контейнеризираното приложение съдържа кода и инструкции за неговите зависимости и как то да бъде изпълнено. Точната технология на приложението няма значение. Инструкциите и всичко необходимо за изпълнението от Docker се описват в Dockerfile.

Проучете:

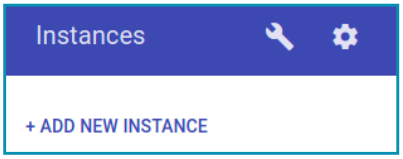
1. Потърсете информация в Интернет за docker командния интерфейс. Какви команди съществуват и с каква цел се използват?

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

ПРАКТИКУМ: ИЗПОЛЗВАЙКИ DOCKER ИЛИ PLAYWITHDOCKER, ОТВОРЕТЕ И СТАРТИРАЙТЕ ПРИЛОЖЕНИЕТО ОТ СЛЕДНОТО ХРАНИЛИЩЕ: HTTPS://GITHUB.COM/PESHOPBS2/ELECTRO-CONTAINER.

1. Влезте в **PlayWithDocker**, използвайки своя акаунт. Ако нямате такъв, създайте си.

2. Стартирайте нова инстанция.



3. Клонирайте хранилището:

```
[node1] (local) root@192.168.0.8 ~
$ git clone https://github.com/peshopbs2/electro-container
Cloning into 'electro-container'...
remote: Enumerating objects: 28, done.
remote: Counting objects: 100% (28/28), done.
remote: Compressing objects: 100% (20/20), done.
remote: Total 28 (delta 6), reused 27 (delta 5), pack-reused 0
Receiving objects: 100% (28/28), 21.06 KiB | 7.02 MiB/s, done.
Resolving deltas: 100% (6/6), done.
```

4. Влезте в директорията и създайте изображение:

```
[node1] (local) root@192.168.0.8 ~
$ cd electro-container
[node1] (local) root@192.168.0.8 ~/electro-container
$ docker image build -t peshopbs2/electro-container .
Sending build context to Docker daemon  134.7kB
Step 1/7 : FROM node:current-alpine
current-alpine: Pulling from library/node
4e9f2cdf4387: Pull complete
793e63d627b9: Pull complete
bcd7e3c33e7f: Pull complete
fbe7e7dfd913: Pull complete
Digest: sha256:3da1c08529fef7007d57d2133a0feb0fa8c60fdd4ad6691978f9dfcb0365b430
Status: Downloaded newer image for node:current-alpine
```

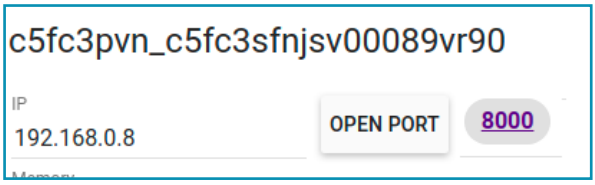
Командата по създаване на изображението ще свали, инсталира и конфигурира необходимия софтуер посочен в Dockerfile.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

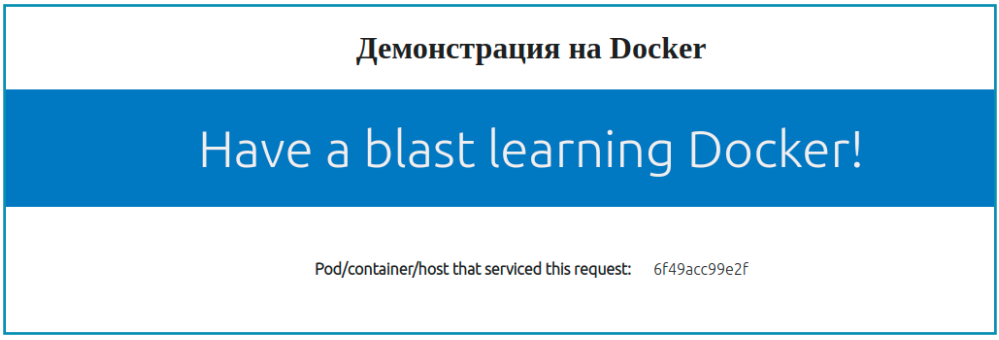
5. Стартирайте контейнера:

```
$ docker container run -d --name electro -p 8000:8080 peshopbs2/electro-container
6f49acc99e2f4cf546d1eb859360bbb79e39ef08d72a09f333379e5cee074819
```

При стартиране, отгоре ще се появи бутонче с номера на посочения порт:



6. Кликнете върху бутона с номера на порта и ще се отвори приложението:



Опитайте сами: *Създайте уеб приложение на технология по ваш избор. Проучете и създайте Dockerfile за неговата контейнеризация. Стартирайте го чрез Docker.*

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

ЧАСТ 2 КОНКУРЕНТНО ПРОГРАМИРАНЕ

ГЛАВА 1 КОНКУРЕНТНОСТ. ИЗПЪЛНЕНИЕ НА ПРОГРАМАТА. ПРОЦЕС.

Какво ще научим?
- Какво е конкурентност?
- Защо програмите (и потребителите) имат нужда от конкурентност?
- Как работят програмите?
- Какво е процес?

1. Конкурентността (concurrency) се явява важен инструмент в създаването на качествени и добри програми. В днешно време потребителите не обичат да чакат. Ако един потребител не успее да зареди уеб сайта, който иска да достъпи, в рамките на 2-3 секунди, то той често ще се откаже и ще посети конкурентен уебсайт. Въпросът стои не по-различно с приложенията – ако накараме потребителя да чака, то той често ще се откаже от употребата на нашето приложение, а ние бихме могли да го загубим като потребител и клиент.

2. А защо програмите „забиват"?

Често, когато една програма е „забила", тя просто извършва дейност, свързана с изчакване на нещо, което отнема повече време. Например прочитане на голям файл, достъп на информация от уеб сървър или просто тежка изчислителна операция. Това забиване се случва, защото стандартно програмите могат да извършват по едно действие в даден момент. Такива са традиционните програми, които сме свикнали да пишем.

Обсъдете: ***Дайте пример за програми, които сте използвали и забиват. Кога се случва това? Какво означава това за Вас като потребител?***

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение", финансиран от Оперативна програма „Наука и образование за интелигентен растеж", съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

3. Какво ако нашата програма започне да прави няколко неща едновременно?

Оказва се, възможно е една и съща програма да прави няколко неща едновременно или поне привидно едновременно. Това се случва благодарение на конкурентността. Въпреки че конкурентността в програмирането съществува от дълго време, тя постоянно продължава да създава неприятности на програмистите. В днешно време все пак е възможно лесно да се пишат програми със средствата на конкурентното програмиране благодарение на библиотеки и съвременни средства в езиците за програмиране. В следващите няколко урока ще демонстрираме средствата за конкурентно програмиране в .NET и C#.

Определение: Конкурентност – Изпълнение наведнъж на няколко действия в едно и също време.

Обсъдете: ***Съществува ли конкурентност в реалния живот? Конкурентност ли е да чакаме на опашка в супермаркет? А да чакаме поръчката си на маса в ресторант?***

4. Как се изпълняват програмите?

В нормалния случай програмите се изпълняват операция след операция. Това означава, че ако например имаме следния програмен фрагмент:

```
int number = int.Parse(Console.ReadLine());
if(number > 0) {
   Console.WriteLine(„Positive”);
} else {
   Console.WriteLine(„Negative or zero”);
}
```

то редът на изпълнение е следният:

1. Въвеждаме стойност за числото.
2. Извършваме проверка.
3. Извеждаме един от двата текста според резултата от проверката.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

Опитайте сами: Разгледате изпълнението на тази и други програми, които сте създавали чрез дебъгера във Visual Studio в постъпков режим (F10).

5. Какви форми на конкурентност съществуват?

Оказва се, че съществуват различни форми на конкурентност. Една популярна такава е многопоточността (многонишковост). Това е форма на конкурентност, при която програмата се изпълнява чрез няколко потока, които могат да извършват различни действия. Употребата на тази форма е с цел оползотворяване на наличните ядра на процесора.

Определение: Многопоточност (Многонишковост) – форма на конкурентност, която използва няколко програмни потока на изпълнение

Определение: Паралелна обработка (паралелно програмиране) – изпълнение на голям обем задачи чрез разпределение върху няколко потока (нишки), които се изпълняват едновременно.

Друга съвременна форма на конкурентността е асинхронното програмиране. При него действията, които трябва да се извършат се представят като операции, които предстои да бъдат завършени в бъдещ момент и се представят като специални обекти (обещания). В миналото този вид конкурентност е бил труден за реализиране чрез код. В последните години употребата му е изключително улеснена чрез подходящ синтаксис на езиците за програмиране. В C# се използват ключовите думи async и await. Аналогични средства съществуват в езици като JavaScript, Python и др.

Определение: Асинхронно програмиране – разновидност на конкурентността, която използва обещания (promises) или обратни извиквания (callbacks) с цел предотвратяване на създаването на излишни потоци.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

Определение: Обещание (promise) – тип, който представлява операция, която ще бъде завършена в бъдещето. В .NET това са Task и Task<TResult>. Аналози има и в други програмни езици.

Определение: Обратно извикване (callback) – функция/метод, който се изпълнява при приключването на дадена операция. Характерни са за някои по-стари имплементации на асинхронно програмиране и/или за някои технологии. Широко разпространени в миналото са в JavaScript света.

Определение: Асинхронна операция (asynchronous operation) – стартирана операция, която ще приключи след някакво време.

Обсъдете: Какви примери за асинхронност сте срещали в реалния живот? Асинхронност ли е да оставим колата си за ремонт и да получим известие, когато е готова, за да отидем да си я вземем?

Проучете повече:
1. Потърсете повече информация за шаблоните за дизайн (Design patterns) в програмирането. Поинтересувайте се от Observer шаблона.
2. Съществуват ли структури, които поддържат конкурентна обработка в .NET? Поддържа ли паралелизъм LINQ библиотеката в .NET?

6. Какво се случва с програмата, когато тя се стартира?

При стартиране програмата се зарежда в оперативната памет (ОП). Тогава операционната система (ОС) създава абстракция за изпълнението на тази програма, която наричаме процес. Процесът изпълнява всички инструкции, зададени в програмата. Всеки процес има 4 части: стекова памет, динамична памет (heap memory), данни, текст.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

Стекова памет	Съдържа временни данни като извиквания на методи/функции и техните параметри, адрес на връщане и локални променливи.
Динамична памет	Съдържа динамично заделената памет за процеса, в т.ч. обекти, масиви и др.
Текст	Съдържа текущата дейност на програмата, представена от брояча на инструкциите и съдържанието на регистрите в процесора
Данни	Съдържа глобални и статични променливи.

Проучете:

1. Какъв е жизненият цикъл на един процес?
2. Какви състояния може да има процесът?
3. Различни ли са те в различните операционни системи (Windows/Linux/Android/MacOS)?

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

ГЛАВА 2 БЛОКИРАЩИ ОПЕРАЦИИ. ВИДОВЕ БЛОКИРАЩИ ОПЕРАЦИИ.

Какво ще научим?
- Какво е блокираща операция?
- Какви видове блокиращи операции съществуват?
- Как можем да направим, така че програмата да не „забива"?

Понякога изпълнението на даден алгоритъм зависи от определени ресурси (данни, файлове, приключване на изпълнението на друг алгоритъм, информация от отдалечен сървър и други). От своя страна ресурсите, от които зависи това изпълнение може да се окажат липсващи към момента, когато възникне необходимост от тях. Тогава програмата ги „изчаква", а самата операция довела до това се оказва блокираща операция.

Пример за това е изчакването на входни данни от потребителя. Без потребителя да е въвел данни, програмата ще остане в състояние на изчакване на вход.

Определение: Блокираща операция наричаме такава операция, която блокира продължението на изпълнение на програмата до приключване на съответната операция.

1. Какви други видове блокиращи операции съществуват?

Други видове блокиращи операции съществуват при:

→**Прочитане/писане на данни от/във файл**

При опит да се прочете наведнъж голям файл или да се запише наведнъж голямо количество информация, потокът от команди на програмата е блокиран. Това може да доведе до привидно забила програма от гледна точка на потребителя.

→**Извършване на тежки или продължителни изчислителни операции**

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение", финансиран от Оперативна програма „Наука и образование за интелигентен растеж", съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

Някои видове софтуер изискват извършването на тежки изчислителни операции, които би отнело известно време да приключат. По време на извършването на тези операции програмата отново може да изглежда привидно забила, но всъщност тя просто изчаква резултата от изчисленията.

→Достъп до мрежови ресурси

Някои приложения изпращат заявки чрез различни мрежови протоколи, например HTTP до други приложения и използват получения отговор. Например, смартфоните изпращат заявки до сървър, който им дава информация за метеорологичните устройства, за да могат да я покажат на потребителя си.

По време на комуникацията със сървърите често са необходими няколко секунди за отговор от страна на сървъра. **Възможно е и ако сървърът е натоварен, той изобщо да не върне отговор на заявката.** По време на мрежовата комуникация отново може да се случи „забиване“ на програмата от гледна точка на потребителя.

Възможно е при дадени обстоятелства да се предизвика умишлено блокираща операция, например когато по преценка на програмиста програмата трябва да изчака определено време или изпълнението на друг блок от код.

2. Как да направим така че програмата да не забива?

Често реализирането на блокиращи операции е необходимо за нашите програми. Въпреки това това не означава, че трябва да оставим нашата програма да „забива“ и да създава впечатление у потребителя, че тя не работи правилно. Това става като прехвърлим дейността на програмата на отделни потоци от команди (нишки). Съществуват и други начини за справяне с проблемите, пред които се изправяме при употребата на блокиращи операции.

В следващите теми подробно ще се запознаем с понятието „нишка“ и „асинхронна операция“.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

Обсъдете:

1. По какъв начин приложението може да сигнализира потребителя, ако трябва да изчака изпълнението на блокираща операция?

2. Възможно ли е приложението да извърши определени операции във фонов режим? Дайте пример за софтуерни приложения, които работят и извършват операции на фонов режим.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

ГЛАВА 3 НИШКИ. СЪЗДАВАНЕ И УПРАВЛЕНИЕ НА НИШКИ.

Какво ще научим?

- Какво е нишка?
- Какъв е жизненият цикъл на нишките?
- Как можем да създадем и изпълним нишка?
- Какви видове нишки има?
- Каква е връзката между процес и нишка?

Определение: Нишка (thread) – самостоятелна редица от програмни инструкции, които се изпълняват последователно.

1. Какво са нишките?

Нишките се явяват един от най-простите способи за реализиране на конкурентност в една програма. Създаването и употребата на нишки позволява оползотворяването на отделните ядра на съвременните процесори. Всяко едно от тези ядра може да изпълнява различни инструкции под формата на различни нишки в даден момент.

Всяка програма използва поне една нишка, която наричаме главна. Разбира се, бихме могли да създадем и използваме допълнително нишки. Важно е да уточним, че точният ред на изпълнение на отделните нишките зависи от определени фактори като например тяхната конфигурация, разпределението на задачите, което се осъществява от операционната система, натовареността на процесора и др.

2. Каква е връзката между процес и нишка?

Всеки един процес може да притежава многобройни нишки. Тези нишки обаче споделят общите ресурси на процеса. Това означава, че те споделят и обща памет. Програма, която може да изпълнява едновременно няколко редици от програмни инструкции (нишки, threads) наричаме многонишкова програма (multithreaded program).

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

3. Какъв е жизненият цикъл на нишките?

Нишките могат да имат различен статус, в зависимост от етапа, на който се намират в жизнения си цикъл. При създаването си една нишка получава статус **„незапочната“ (Unstarted)**. Следващият етап е преминаването ѝ в статус **„подлежаща на изпълнение“ (Runnable)**. Този статус е особен, понеже по време на него нишката все още не е стартирана, но очаква операционната система да я придвижи за изпълнение. Когато нишката премине в режим на изпълнение, нейният статус е **„в изпълнение“ (Running)**. По време на изпълнение нишката е възможно да бъде временно спряна и да премине в статус **„временно спряна“ (suspended, not running)**. В даден момент нишката може да бъде продължена и да мине отново в статус на **„изпълнение“ (Running)**. Нишката е възможно да бъде и убита, тогава тя има статус **„мъртва“ (Dead)**.

4. Как можем да създадем нишка?

Нишките в C# представляват обекти на класа **Thread**. За да използваме този клас, е нужно да добавим библиотеката за нишки:

```
using System.Threading;
```

След това създаването на нишка е лесно като създаването на всеки друг обект:

```
Thread thread = new Thread(threadMethod);
```

Изискуем параметър за конструктора се явява метод, който реално съдържа кода на нишката. Тук бихме могли да посочим и анонимен метод.

За стартиране на изпълнението на нишката, т.е. да прехвърлим нейния статус към **„подлежащ на изпълнение“ (Runnable)**, трябва да извикаме метод Start на съответния обект:

```
thread.Start();
```

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

5. Как да направим нишка за метод от друг клас?

Нишка може да бъде създадена и за изпълнението на метод от друг клас:

```
public class ExampleThread {
   public void print() {
      for (int i = 0; i < 3; i++) {
         Console.WriteLine(„Other Thread”);
      }
   }
}
```

В Main метода или друг метод създаваме нишката по сходен начин:

```
ExampleThread obj = new ExampleThread();

Thread thread = new Thread(new ThreadStart(obj.print));
thread.Start();
```

6. А какво следва, ако трябва да подадем параметри в метода?

Ако искаме да подаваме параметри на метода, който желаем да се изпълни на отделна нишка, то можем да използваме:

- Ламбда изрази
- Делегати
- ParameterizedThreadStart

Лесен, удобен и ефективен начин е употребата на ламбда изрази, която може да се случи по сходен начин:

```
//нека методът изглежда така:
public void MyMethod(string param1, int param2) {
   //do stuff
}
//можем да стартираме нишката по следния начин:
Thread thread = new Thread(() => MyMethod(„param1”,5));
thread.Start();
```

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

7. Какви нишки има в C#?

→Главна нишка (main thread)

C# разполага винаги с **главна нишка (main thread)**. Това е нишката, с която главно се стартира и изпълнява програмата.

→Нишка на преден план (foreground thread)

Нишката на преден план остава активна до приключването на работата си дори главната нишка вече да е приключила.

→Нишка на заден план (background thread)

Нишката на заден план е свързана с главната. Ако главната нишка приключи работата си, това ще доведе до терминирането и на нишката на заден план.

По подразбиране нишките, които създаваме, са нишки на преден план. Ако желаем нишката да е на заден план, трябва да променим нейното свойство **IsBackground**:

```
thread.IsBackground = true;
```

Опитайте сами:

1. Създайте две нишки, които представляват цикъл, който брои до 10. Двете нишки една след друга ли се изпълняват?

2. Към двете нишки, които вече добавихте добавете изчакване от по 1 секунда след извеждане на числото. За тази цел използвайте следната команда:

```
System.Threading.Thread.Sleep(1000);
```

3. Изпълнете програмата поне няколко пъти. Всеки път изходът ли е един и същ? А какво, ако използвате случайно число за параметър на командата за изчакване?

4. Създайте приложение с няколко нишки, които трябва да съберат над 100 точки. На всяка итерация добавяйте случайно число от 1 до 10 – брой на точките, които се печелят в дадения рунд. Пуснете програмата няколко пъти. Една и съща нишка ли „печели състезанието“?

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

ГЛАВА 4 ПУЛ ОТ НИШКИ (THREADPOOL). ВИДОВЕ ПРОБЛЕМИ ПРИ УПРАВЛЕНИЕ НА НИШКИ.

Какво ще научим?

- Как да работим ефективно с нишки?
- Какво е **ThreadPool** и защо е за предпочитане?
- Какви видове проблеми могат да възникнат при употребата на нишки?

При създаването на практически софтуерни решения употребата на множество нишки често се явява сериозна необходимост. Създаването и управлението на нишки е сравнително тежка операция по отношение на ресурси, особено на процесора. В този смисъл постоянното създаване на отделни нишки може да доведе до драстичен спад в производителността на приложението. Струва си да се отбележи, че в практиката често една нишка след създаването си остава в спящ режим и реално не е нужна през по-голямата част от времето.

1. Как се решава проблемът с производителността, ако имаме нужда от повече нишки?

В .NET проблемът с производителността на нишките е решен като се поддържа т.нар. **ThreadPool (пул от нишки)**. Пулът от нишки представлява набор от предварително създадени нишки от самата среда на .NET, които се управляват от средата. По този начин се елиминира необходимостта от постоянното създаване и освобождаване на нишки. Вместо това се заема нишка от пула, докато се извърши съответната задача, след което нишката се освобождава и тя отново е налична за извършване на други задачи.

Аналогични решения се използват и в други езици, които поддържат многонишковост.

Нишките от **ThreadPool** се използват и от редица готови обекти в .NET. Например, обектите от клас **Task** използват нишки от **ThreadPool**.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

Жизнен цикъл на нишка в C#

Вземане на обекта на нишката от пула

2. Как да използваме ThreadPool?

За да използваме **ThreadPool**, е нужно да добавим библиотеката за нишки:

```
using System.Threading;
```

След това с помощта на ThreadPool.QueueUserWorkItem можем да изпълним даден метод чрез нишка от пула от нишки:

```
ThreadPool.QueueUserWorkItem(new WaitCallback(ThreadMethod));
```

Примерна реализация на метода **ThreadMethod()**:

```
public static void ThreadMethod(object obj) {
    Thread thread = Thread.CurrentThread;
    string message = $"Background: {thread.IsBackground}, Thread Pool:
{thread.IsThreadPoolThread}, Thread ID: {thread.ManagedThreadId}";
    Console.WriteLine(message);
}
```

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

3. Как да вземем информация за текущата нишка?

С помощта на **Thread.CurrentThread** можем да вземем информация за самия обект на нишката. Някои по-интересни свойства са дадени в таблицата по-долу:

Свойство	Предназначение
IsBackground	Показва дали нишката е на фонов режим. Има стойност true, ако нишката работи на фонов режим и false, в противен случай.
IsThreadPoolThread	Има стойност true, ако нишката е част от пул с нишки и false, в противен случай.
Name	Задава или показва името на нишката.
ManagedThreadId	Уникален идентификатор на нишката.
Priority	Задава или показва приоритета на нишката. Приоритетът е един от следните: **Highest** (най-висок) **AboveNormal** (над нормалния) **Normal** (нормален) **BelowNormal** (под нормалното) **Lowest** (най-нисък)
ThreadState	Състояние на нишката

Проучете:

1. Какво значение има стойността на Priority свойството за изпълнението на нишките?
2. Какви са възможните стойности на ThreadState? Как се постигат тези състояния на нишката?

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

4. Винаги ли да използваме ThreadPool?

ThreadPool несъмнено предлага доста предимства и удобства и на практика се използва през повечето време, но съществуват специфични ситуации, в които е по-добре да използваме самостоятелно създадени нишки. Такива ситуации изискват програмистът да има повече контрол над изпълнението. Например, **ThreadPool** създава само нишки на заден план. В случай на нужда от нишка на преден план, тя трябва да се създаде ръчно чрез **Thread** класа.

Припомнете си от предната тема каква е разликата между нишките на преден и заден план?

Проучете и потърсете отговор:

1. Възможно ли е да управляваме приоритета на нишки от ThreadPool?
2. Възможно ли е да прекратим изпълнението на дадена нишка?
3. Какви други начини съществуват за използване на нишки от пула освен ThreadPool класа?
4. Какво е Task Parallel Library и как тя помага за работа с нишки?

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

ГЛАВА 5 ВИДОВЕ ПРОБЛЕМИ ПРИ УПРАВЛЕНИЕ НА НИШКИ. СИНХРОНИЗАЦИЯ

Какво ще научим?

- Кога има нужда от синхронизация на нишките?
- Какви видове проблеми могат да възникнат при употребата на нишки?
- Как да синхронизираме нишки?

При употребата на нишки могат да възникнат множество особени ситуации. Това е така поради природата на многонишковото програмиране и на изпълнението на нишките от процесора. Напълно възможно е при едно стартиране на програмата, например, нишка А да приключи по-бързо от друга нишка Б. При последващо стартиране е възможно тък нишка А да приключи по-бавно от другата нишка. Това може да породи множество различни ситуации, в които ако нишките не се синхронизират, според изискванията и очакванията на програмиста, програмата може да има съвсем различно поведение от очакваното.

1. Какви проблеми могат да възникнат при употребата на нишки?

При употребата на нишки могат да възникнат редица различни проблеми. Проблемите често са свързани с достъп до общи ресурси или с необходимостта от изчакване на приключването на дадена нишка преди да започне друга. Понякога проблемите са свързани и с прекомерна употреба на ресурси.

По-често срещаните проблеми са:

→ Race condition (състояние на гонка);
→ Мъртва хватка (dead lock);
→ Жива хватка (live lock);
→ Гладуване (starvation).

2. Какво е Race condition (състояние на гонка)?

Нека да разгледаме следния клас **Counter**, който има метод **Increment()** за увеличаване на броя и метод **Decrement()** за намаляване

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

(отляво) и метод за употреба на обект от класа (отдясно):

```
class Counter
{
  public int Count { get; private set; }

  public void Increment()
  {
    Count++;
  }

  public void Decrement()
  {
    Count--;
  }
}
```

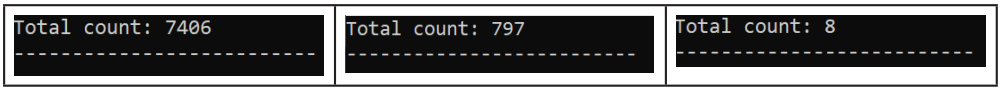
```
static void TestCounter(Counter c)
{
  for (int i = 0; i < 10000; i++)
  {
    c.Increment();
    c.Decrement();
  }
}
```

Създаваме няколко нишки, които изпълняват **TestCounter()** метода:

```
Counter c = new Counter();

Thread thread1 = new Thread(() => TestCounter(c));
Thread thread2 = new Thread(() => TestCounter(c));
Thread thread3 = new Thread(() => TestCounter(c));
thread1.Start();
thread2.Start();

thread3.Start();
thread1.Join();
thread2.Join();
thread3.Join();

Console.WriteLine($“Total count: {c.Count}“);
```

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

Ако стартираме програмата, ще забележим, че в различни изпълнения получаваме **различни** резултати:



Обяснение:

Това се случва, защото използваме три нишки, които достъпват брояча на един и същ обект. При изпълнението увеличаването и намаляването се случват последователно в цикъл, но това може да доведе до недетерминистични (неопределени) резултати. В идеалния случай би трябвало да се получи 0, но в повечето случаи ще се получават различни числа.

Това се случва, защото класът **Counter** не е **безопасен (thread-safe)**. Ако няколко нишки достъпят брояча едновременно и първата получи, че стойността на брояча към момента е 10, тя ще го увеличи на 11. Тогава, втора нишка получава, че стойността на брояча е 11 и я увеличава на 12. Първата нишка през това време получава новата стойност на брояча – 12, но преди да успее да я намали, втората нишка е взела също стойността на брояча като 12. Тогава, първата нишка намаля брояча на 11 и го запазва, а втората през това време прави същото! Така имаме две увеличавания на стойността и само едно намаляване, което не би трябвало да е правилно. Такава ситуация се нарича състояние на гонка (race condition) и е често срещан проблем при употреба на нишки. Възникването на ситуацията може да има произволен характер и зависи от реда на изпълнение на нишките, който се случва в конкретното изпълнение на програмата.

3. Как да решим race condition проблем?

Решението на race condition проблемите е да се използва специален заключващ обект с помощта на конструкцията **lock** в C#. При достъп на една нишка до заключващия обект, тя го заключва за достъп от всички останали нишки. Ако през това време някоя нишка се опита да го достъпи, то тя ще бъде поставена в режим на изчакване. Нека да

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

разгледаме thread-safe вариант на класа от по-горе:

```
class LockedCounter
{
  private readonly object _syncRoot = new
Object();

  public int Count { get; private set; }

  public override void Increment()
  {
    lock (_syncRoot)
    {
      Count++;
    }
  }

  public override void Decrement()
  {
    lock (_syncRoot)
    {
      Count--;
    }
  }
}
```

```
static void TestCounter(LockedCounter c)
{
  for (int i = 0; i < 10000; i++)
  {
    c.Increment();
    c.Decrement();
  }
}
```

Кодът, който трябва да се изпълнява само от една нишка в даден момент се поставя в блок след **lock()** конструкцията, а в кръглите скоби се посочва заключващият обект.

Заключващият обект не може да бъде от примитивен тип и ако се налага да работим с някаква стойност от примитивен тип, то създаваме допълнителен обект от тип object.

Опитайте сами: C# може да бъде застaвен да извършва операции от горепосочения тип като атомарни. Това става с помощта на Interlocked. Проучете как работи той и се опитайте да реализирате аналогичен пример на дадения по-горе с помощта на Interlocked.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

4. Какво е deadlock (мъртва хватка)?

Друг често срещан проблем, който може да доведе до проблеми в работата с програмата се нарича **deadlock (мъртва хватка)**. Мъртвата хватка може да възникне по следния начин:

→ Нишка 1 заема ресурс A;
→ Нишка 2 заема ресурс B;
→ Нишка 1 опитва да заеме ресурс B;
→ Нишка 2 опитва да заеме ресурс A.

Тъй като ресурсите A и Б вече се използват съответно от нишка 1 и нишка 2, при опит на другата нишка да заеме ресурс, то тя трябва да изчака. Това води до взаимно изчакване и на двете нишки, което може да продължи безкрайно. Мъртва хватка е възможно да възникне и между по-голям брой ресурси.

Опитайте сами: *предизвикайте мъртва хватка, използвайки два заключващи обекта.*

Проучете: Как да разрешите проблема на мъртвата хватка с помощта на класа Monitor? Има ли други начини за разрешаване на този проблем?

5. Какво е livelock (жива хватка)?

Състоянието на livelock (жива хватка) е самоизключващо се блокиране. Такова състояние може да възникне, ако нишка А засече, че нишка Б се опитва да заеме ресурс и се опита да го освободи. От своя страна при получаване на ресурса нишка Б също се опитва да го освободи отново на нишка А. Това може да продължи до безкрайност, но за разлика от мъртвата хватка, тук състоянието на нишките е работещо, а не изчакващо.

6. Какво е starvation (гладуване)?

Състоянието на гладуване на нишка е свързано с ресурсите, които получават нишките. Нека имаме нишка А, която използва общ ресурс

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

с друга нишка, но често го задържа за себе си за по-дълго време, което я прави ***greedy (лакома)***. Нишка Б получава ресурса на цената на дълго чакане, което я превръща в ролята на ***гладуваща (starving)***.

Проучете: Съществуват редица други начини за синхронизация като Mutex, Semaphore, Barrier и др. в C#. Потърсете информация как и в какви ситуации се използват.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

ГЛАВА 6 АСИНХРОННИ ОПЕРАЦИИ. ЗАДАЧИ/ОБЕЩАНИЯ (TASK) И ОБРАТНО ИЗВИКВАНЕ (CALLBACK).

Какво ще научим?

- Какво е асинхронна операция?
- Какво е класът Task?
- Как се реализират асинхронни операции?
- Употреба на ключовите думи async и await

1. Какво е асинхронна операция?

Асинхронните операции са форма на конкурентното програмиране, която е широко употребявана в съвременните софтуерни приложения. За приложенията с графичен потребителски интерфейс е изключително важно да не забиват и винаги да отговарят на потребителските действия, като не карат потребителя да чака, без да знае какво се случва. Асинхронните операции са важни и при сървърните приложения (бекенд на уеб приложенията), тъй като това позволява обслужването на повече заявки от потребители, като се намалява времето за изчакване.

2. Как работят асинхронните методи?

При асинхронните методи последователността от операции извършват цялата или по-голямата част от своята работа **след** връщане на управлението към мястото, където методът е бил повикан. По това асинхронните методи се различават от традиционните синхронни методи, които водят до блокиране на викащия ги метод до приключване на изпълнението им.

Асинхронното програмиране е вид конкурентност, защото операциите извършвани от асинхронните методи се извършват конкурентно с функционалността на метода, който е повикал асинхронния метод.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

3. Какво представлява клас Task?

Средата .NET предоставя за употреба класа **Task<TResult>**, част от **System.Threading.Tasks**. Той описва операция, която продължава в бъдещето. Един начин за изпълнение на такива операции е чрез **Task.Run()** метода. Методът инструктира средата CLR на .NET да изпълни операцията паралелно на отделна нишка, докато методът, в който сме поставили извикването към **Task.Run()** продължава своята работа. Имайте предвид, че класът **Task** е шаблонен и неговият тип се определя от стойността на връщане на метода, който се вика в делегата-параметър на **Task.Run()**.

Примерно повикване на **ExampleCalculation()** метод, който връща стойност от тип int:

```
Task<int> ExampleCalculationAsync()
{
   return Task.Run(() => ExampleCalculation());
}
```

Проучете: Разгледайте повече за класа Task в документацията на Microsoft.

Методът **ExampleCalculationAsync()** е асинхронен по същество, т.е. веднага връща контрола върху управлението към извикалия го метод, но в същото време самият той продължава да се изпълнява конкурентно.

Често възниква необходимост викащият метод да получи резултата от изпълнението на асинхронния метод, когато резултатът е наличен. Това става чрез метода **GetAwaiter()**, който позволява на викащия метод да добави продължение. Продължението се явява код, който се изпълнява при успешно завършване.

Пример за употребата на **GetAwaiter()**:

```
Task<string> task = GetSomeDataAsync();

var awaiter = task.GetAwaiter();
awaiter.OnCompleted( () => //продължение при успешно завършване
{
   string result = awaiter.GetResult(); //получаване на обекта със самия резултат
   Console.WriteLine(result); //извеждане на резултата на екрана
});
```

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

Проучете: Възможно е да се постави продължение и при други ситуации, например, ако задачата претърпи неуспех. Как се случва това?

4. Как се използват ключовите думи async и await в C#?

Ключовата дума **await** е *синтактична захар*. Компилаторът я замества с GetAwaiter() парче от код.

```
var result = await израз;
команди();
```

```
var awaiter = expression.GetAwaiter();
awaiter.OnCompleted(() =>
{
var result = awaiter.GetResult();
команди();
});
```

5. Какво е значението на ключовата дума async?

Методи, в които се използва ключовата дума **await** трябва да бъдат маркирани като асинхронни. Това се случва с ключовата дума **async**, в противен случай методът няма да бъде успешно компилиран.

Примерна употреба на **async** и **await**:

```
async void ExampleMethod()
{
   int result = await ExampleCalculationAsync();
}
```

Проучете:

1. Методи, маркирани с async, могат да връщат стойности и от други типове. Какви са те?
2. Прочетете повече за TAP шаблона в .NET за асинхронно програмиране.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

ГЛАВА 7 АСИНХРОННИ ОПЕРАЦИИ И „КЛИЕНТ-СЪРВЪР“ ПРИЛОЖЕНИЯ. РАБОТА С RESTFUL API И КОНСУМИРАНЕ НА RESTFUL API.

Какво ще научим?

- В рамките на този практикум ще разработим система, която взема информация за времето в дадена локация, без да позволяваме на системата да „забие“.
- За целта ще използваме услугата на Open Meteo API.

Дискутирайте и проучете:

1. Какво е API? Какво е Web API услуга? Какво е характерно за Restful API услугите?
2. Какви формати се използват за предаване на информацията в Web API и Restful API услугите?
3. Проучете: Прочетете документацията за ползване на Open Meteo API на адрес https://open-meteo.com/en/docs

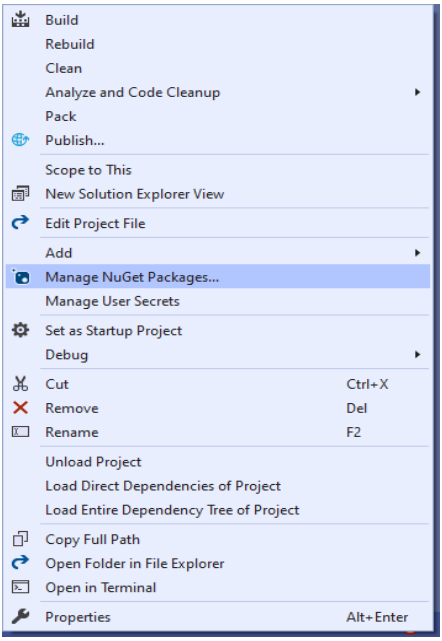
1. Ще използваме API адрес, който да ни даде информация за текущите метеорологични условия в района на Бургас:
https://api.open-meteo.com/v1/forecast?latitude=42.45&longitude=27.41&current_weather=true

Опитайте сами: *Проучете как и променете URL адреса, така че да дава информация за текущите метеорологични условия във Вашето местоположение.*

2. Създайте проект за конзолно приложение. Използвайте .NET 5.

3. Добавете NuGet пакета **Microsoft.AspNet.WebApi.Client**. Това става, като кликне върху проекта с десен бутон и се избере Manage NuGet packages.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове



4. Създайте клас **WeatherInfo**, който да описва върнатия JSON обект. **Забележете, че за именуването трябва да използвате сходни имена като тези в JSON обекта**:

```
public class WeatherInfo
{
      public double Elevation { get; set; }
      public double Latitude { get; set; }
      public CurrentWeatherInfo Current_Weather { get; set; }

      public class CurrentWeatherInfo
      {
         public int WeatherCode { get; set; }
         public double Temperature { get; set; }
      }
}
```

5. Създайте метод, който да извежда информацията за обекта.

```
static void PrintWeatherInfo(WeatherInfo info)
{
   Console.Clear();
   Console.WriteLine($"Current weather at {info.Latitude} {info.Elevation}: „);
   Console.WriteLine($"Temperature: {info.Current_Weather.Temperature}");
}
```

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

6. Създайте обект за **HttpClient** в класа Program.

```
private static HttpClient client;
```

7. Създайте асинхронен метод **GetCurrentWeatherAsync()**, който изпраща заявка до сървъра.

```
static async Task<WeatherInfo> GetCurrentWeatherAsync()
{
        WeatherInfo weatherInfo = null;
        HttpResponseMessage response = await client.GetAsync(„https://api.open-
meteo.com/v1/forecast?latitude=42.45&longitude=27.41&current_weather=true”);
        if (response.IsSuccessStatusCode)
        {
            weatherInfo = await response.Content.ReadAsAsync<WeatherInfo>();
        }
        return weatherInfo;
}
```

8. Използвайте методите в **Main()** метода:

```
static void Main(string[] args)
{
   Console.WriteLine(„Please wait...”);
   client = new HttpClient();
   var info = GetCurrentWeatherAsync()
      .GetAwaiter()
      .GetResult();
   PrintWeatherInfo(info);
}
```

Опитайте сами:

1. Добавете останалите данни от JSON в WeatherInfo класа. Променете метода за извеждане на обекта.

2. Опитайте да реализирате извиквания към други функционалности на Open Meteo API.

3. Потърсете и други публични API услуги, които може да използвате. Реализирайте приложение-клиент за избрани от вас API услуги.

4. Подобрете структурата на примерния код. Опитайте се да реализирате отделен клас за клиент.

5. Направете възможно в програмата да се въвежда име на локация и програмата сама да определя координатите, които са необходими за повикване на Open Meteo API.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

ГЛАВА 8 РАБОТА С ПРИЛОЖЕНИЯ С ГРАФИЧЕН ПОТРЕБИТЕЛСКИ ИНТЕРФЕЙС,ИЗПОЛЗВАЩИ АСИНХРОННИ ОПЕРАЦИИ

Какво ще научим?

- В рамките на този практикум ще създадем просто приложение с графичен потребителски интерфейс, което ще ни служи за конвертиране на пари от една валута в друга. Приложението ще използва Currency API достъпно тук: https://github.com/fawazahmed0/currency-api#readme

1. Какво може приложението?

Функционалността на приложението е проста. То трябва да предоставя възможност за избор на входна и целева валута. Изборът на валути се осъществява от падащо меню, което е попълнено с възможностите за избор.

Опитайте сами: *Намерете информация в документацията на Currency API как да разберете кои са поддържаните валути и направете така, че вашата програма да сваля динамично тази информация при стартиране.*

2. Каква технология ще бъде използвана за потребителски интерфейс?

За целите на демонстрацията тук ще използваме Windows Forms. Въпреки това, препоръчваме да разгледате актуалната технология WinUI 3.0 и UWP и да се опитате да пресъздадете примера, използвайки WinUI 3.0.

1. Създайте Windows Forms приложение.

2. Създайте клас Currency, който ще съхранява обект на валута.

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

```
public class Currency
{
    public string Code { get; private set; }
    public string Name { get; private set; }

    public Currency(string code, string name)
    {
        Code = code;
        Name = name;
    }

    public override string ToString()
    {
        return Name;
    }
}
```

3. Създайте клас, който комуникира с API-то. Комуникацията е аналогична на тази в предния практикум.

```
public class CurrencyService
{
    private static HttpClient client;
    public CurrencyService()
    {
        client = new HttpClient();
    }

    public async Task<Dictionary<string, string>> GetCurrenciesList()
    {
        Dictionary<string, string> currencies = new Dictionary<string, string>();
        HttpResponseMessage response = await client.GetAsync(„https://cdn.jsdelivr.
net/gh/fawazahmed0/currency-api@1/latest/currencies.min.json”);
        if (response.IsSuccessStatusCode)
        {
            currencies = JsonSerializer.Deserialize<Dictionary<string, string>>(await
response.Content.ReadAsStringAsync());

        }
        return currencies.OrderBy(item => item.Value).ToDictionary(item => item.Key,
item => item.Value);
    }

    public async Task<double> GetExchangeRate(string currencyFrom, string
currencyTo)
    {
        string url = $“https://cdn.jsdelivr.net/gh/fawazahmed0/currency-api@1/latest/
currencies/{currencyFrom}/{currencyTo}.json”;
        Dictionary<string, object> exchangeInfo = new Dictionary<string, object>();
        double rate = 0.0;
        HttpResponseMessage response = await client.GetAsync(url);
        if (response.IsSuccessStatusCode)
        {
            exchangeInfo = JsonSerializer.Deserialize<Dictionary<string,
object>>(await response.Content.ReadAsStringAsync());
        }

        JsonElement element = (JsonElement) exchangeInfo[currencyTo];

        return element.GetDouble();

    }
}
```

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

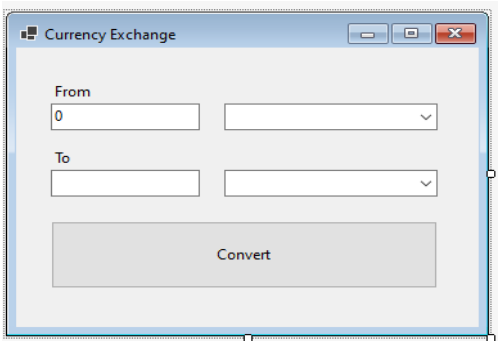
В класа ще имаме метод **GetCurrenciesList()**, който е асинхронен и изпраща заявка, с която да вземе всички валути, които се поддържат от API-то. Валутите се връщат във вид на речник с ключ низ (кода на валутата) и стойност низ (името на валутата).

Методът **GetExchangeRate()** ще се грижи за вземането на обменния курс между две валути, които се приемат като параметри на метода. **Забележете, че редът на подаване има значение!**

4. Инсталирайте отново NuGet пакетите **Microsoft.AspNet.WebApi.Client** и **System.Text.Json.**

5. Създайте графичен потребителски интерфейс.

Примерен външен вид на приложението:



6. Добавете код, който се изпълнява при зареждане на формата, за да се попълват падащите избори с валути.

```
private async void ExchangeForm_Load(object sender, EventArgs e)
{
        Dictionary<string, string> currencies =
            await currencyService.GetCurrenciesList();

        LoadCurrencies(cmbFrom, currencies);
        LoadCurrencies(cmbTo, currencies);
}

private void LoadCurrencies(ComboBox cmb, Dictionary<string, string>
currencies)
{
        foreach (var item in currencies)
        {
            cmb.Items.Add(
                new Currency(item.Key, item.Value)
            );
        }
}
```

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове

7. Създайте поле за обекта от CurrencyService в ExchangeForm класа и го инициализирайте в конструктора:

```
private CurrencyService currencyService;

public ExchangeForm()
{
    InitializeComponent();

    currencyService = new CurrencyService();
}
```

8. Добавете код, който се изпълнява при кликване на бутона.

```
private async void btnConvert_Click(object sender, EventArgs e)
{
        if(cmbFrom.SelectedItem == null || cmbTo.SelectedItem == null)
        {
            MessageBox.Show(„Please choose your source and target currency!”,
„Error”, MessageBoxButtons.OK, MessageBoxIcon.Error);
            return;
        }
        string currencyFrom = ((Currency)cmbFrom.SelectedItem).Code;
        string currencyTo = ((Currency)cmbTo.SelectedItem).Code;
        double rate = await currencyService.GetExchangeRate(currencyFrom,
currencyTo);

        txtTo.Text = Math.Round(rate * double.Parse(txtFrom.Text), 2) + „”;
}
```

Опитайте сами: Реализирайте приложението с WinUI 3.0

Учебното помагало е разработено в рамките на проект BG05M2OP001-2.014-0001 „Подкрепа за дуалната система на обучение“, финансиран от Оперативна програма „Наука и образование за интелигентен растеж“, съфинансирана от Европейския съюз чрез Европейските структурни и инвестиционни фондове