**1. SJF algoritmus:**

SJF = Shortest Job First

Musíme vedieť dĺžku ďalšej požiadavky na dávku CPU pre každý proces. Vyberá sa proces s najkratšou požiadavkou na CPU. SJF je optimálny algoritmus (dáva minimálnu priemernú dobu čakania).

NEPREEMPTIVNI ALGORITMUS : je to algoritmus bez predbiehania - ako náhle sa CPU poskytne vybranému procesu, tento proces nemôže byť predbehnutý žiadnym iným procesom, pokiaľ pridelenú dávku CPU nedokončí.

PREEMPTIVNI ALGORITMUS : ako náhle sa vo fronte pripravených procesov objaví proces s kratšou dĺžkou dávky CPU ako je ostávajúca doba k dokončeniu dávky práve bežiaceho procesu, je práve bežiaci proces vo využívaní CPU predbehnutý novým procesom. Nazýva sa aj SRTF - Shortest Remaining Time First.

**Nepreemtivní algoritmus:** P1 vznik v čase 0, trvaní 7

P2 vznik v čase 2, trvaní 4

P3 vznik v čase 4, trvaní 1

P4 vznik v čase 5, trvaní 4

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | T9 | T10 |
| P1 | P1 | P1 | P1 | P1 | P1 | P1 | P3 | P2 | P2 |
|  | P2 – T4 |  | P3 – T1 | P4 – T4 |  |  |  |  |  |

Ktorý proces pobeží v čase 10? ­> P2

Nepreemptivní SJF - bez preskakovania

P1,P3,P2,P4 -> 0-7,7-8,8-12,12-16

**Preemptivní algoritmus:** P1 vznik v čase 0, trvaní 7

P2 vznik v čase 2, trvaní 4

P3 vznik v čase 4, trvaní 1

P4 vznik v čase 5, trvaní 4

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | T9 | T10 |
| P1 | P1 | P2 | P2 | P3 | P2 | P2 | P4 | P4 | P4 |
|  |  | P2 – T4 | P3 – T1 | P5 – T4 |  |  |  |  |  |

Ktorý proces pobeží v čase 10? > P4

Preemptivní SJF - s preskakovaním

P1,P2,P3,P2,P4,P1 -> 0-2,2-4,4-5,5-7,7-11,11-16

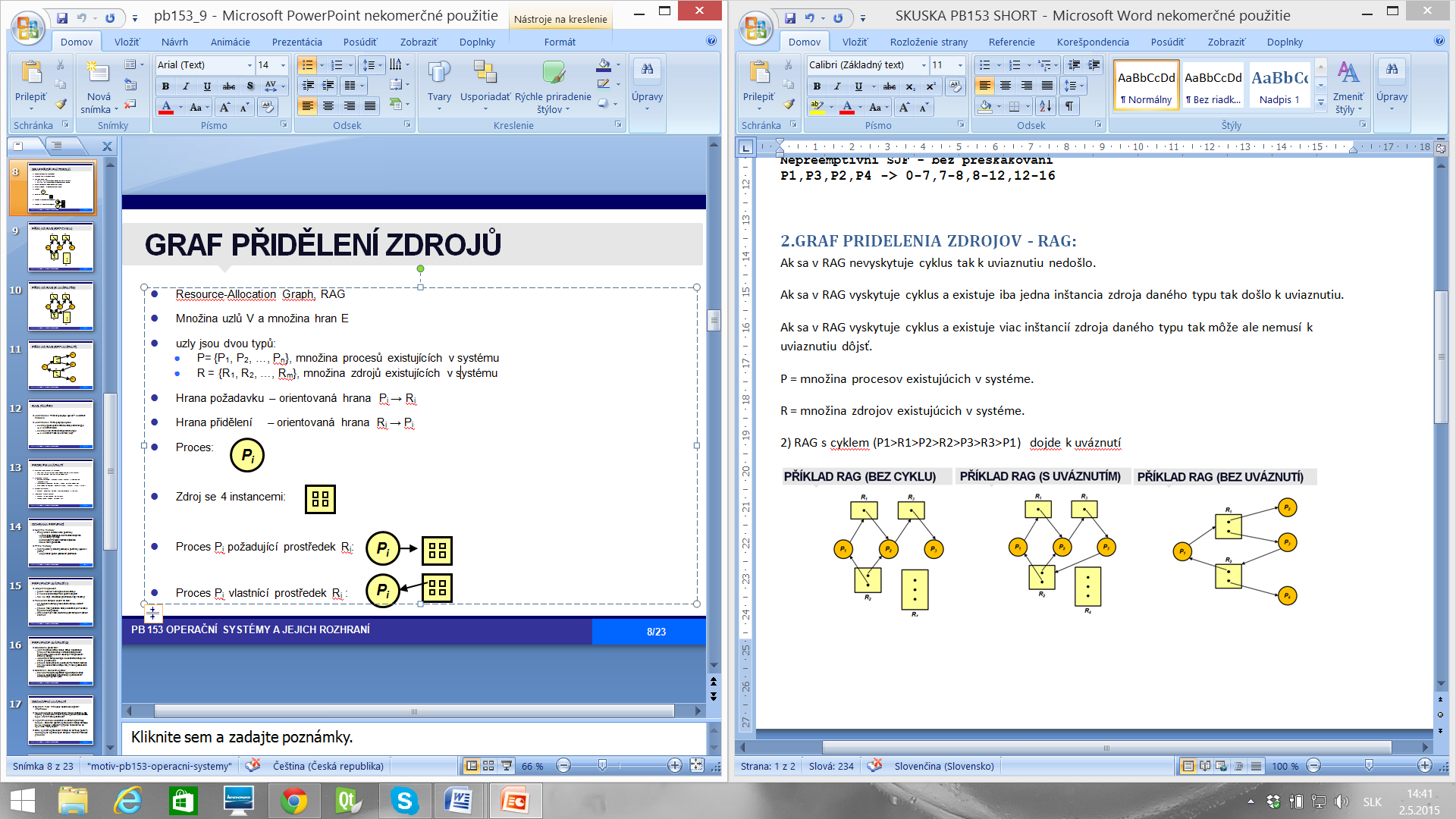
**2.GRAF PRIDELENIA ZDROJOV - RAG:**

Ak sa v RAG nevyskytuje cyklus tak k uviaznutiu nedošlo.

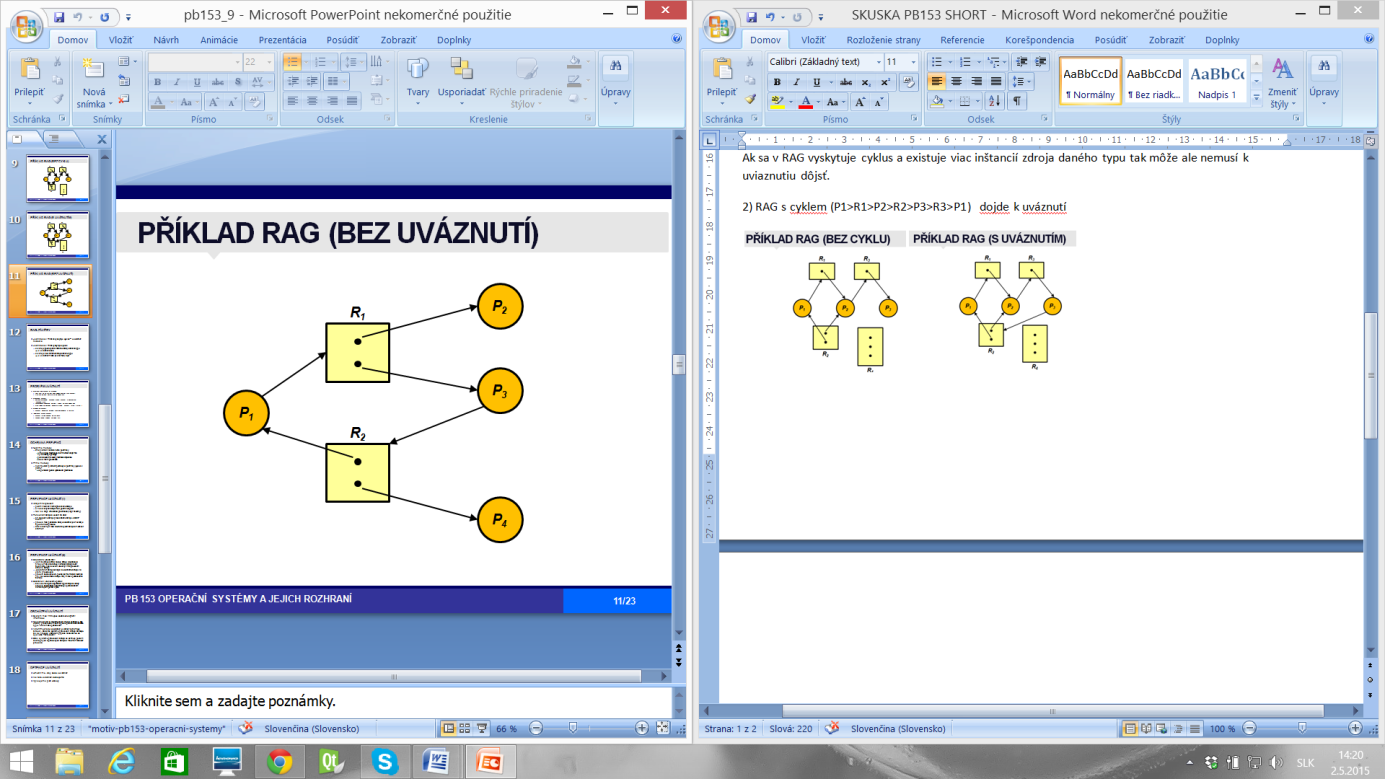
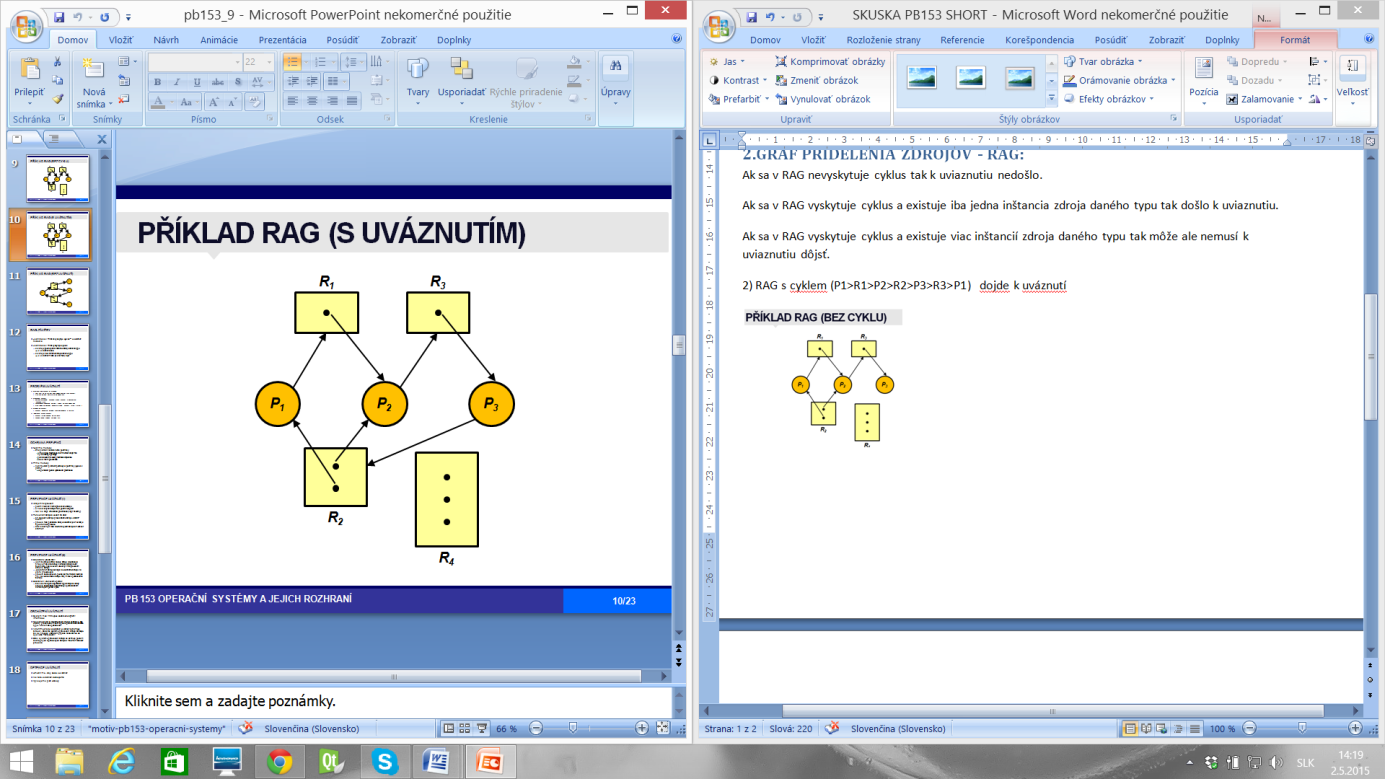
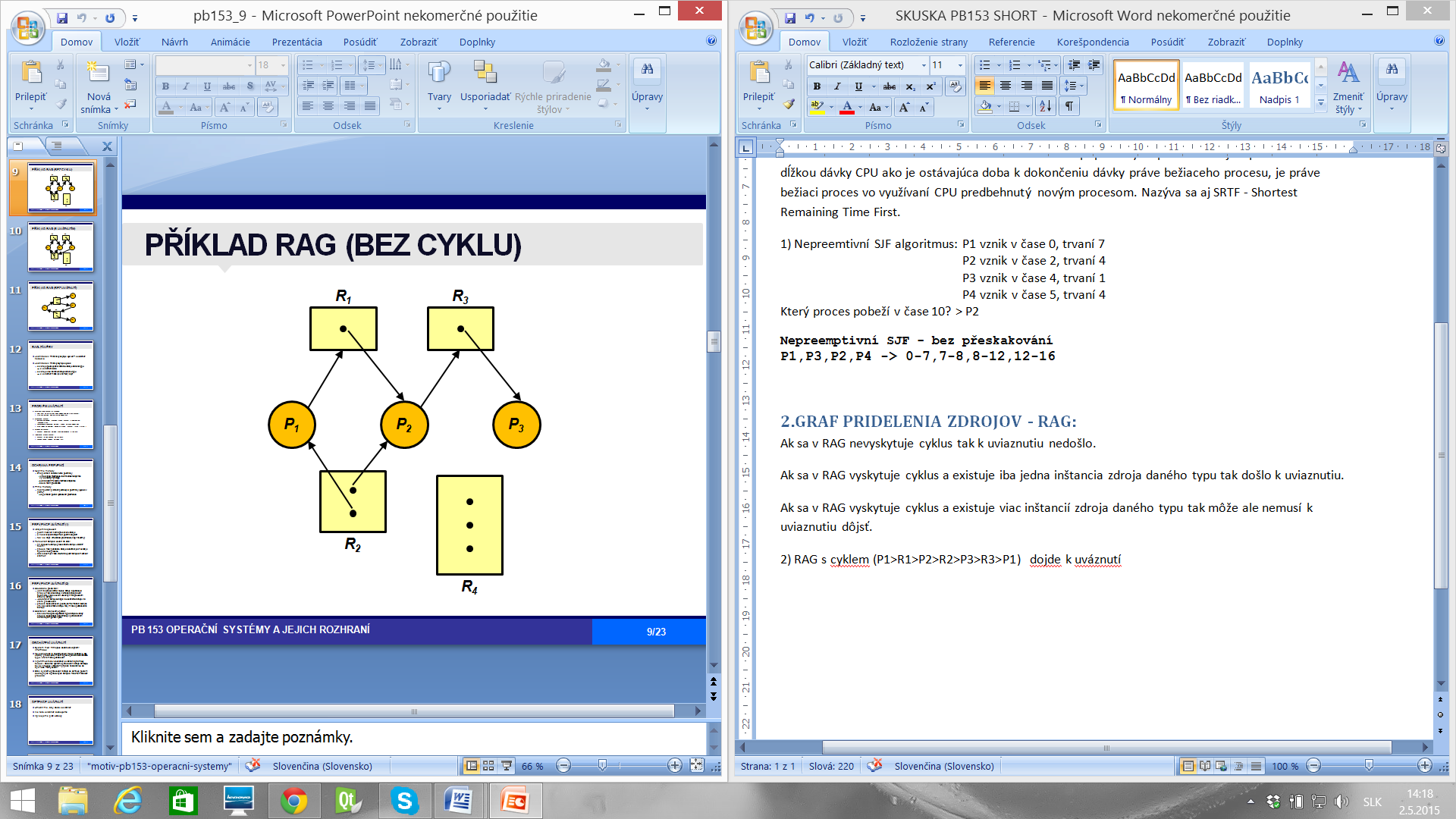
Ak sa v RAG vyskytuje cyklus a existuje iba jedna inštancia zdroja daného typu tak došlo k uviaznutiu.

Ak sa v RAG vyskytuje cyklus a existuje viac inštancií zdroja daného typu tak môže ale nemusí k uviaznutiu dôjsť.

P = množina procesov existujúcich v systéme. R = množina zdrojov existujúcich v systéme.



2) RAG s cyklem (P1­>R1­>P2­>R2­>P3­>R3­>P1) ­ dôjde k uviaznutiu



P1->R1->P2->R3->P3 ==> bez cyklu bez uviaznutia

P1->R1->P2->R3->P3->R2.1->P1 && R2.2->P2 ==> s cyklom s uviaznutím

P1->R1.1->P2&&R1.2->P3->R2.1->P1&&R2.2->P4 ==> s cyklom bez uviaznutia

P1->R1->P2->R2->P3->R3->P1 ==> s cyklom s uviaznutím

**3. InterProcess Communication**

IPC = medziprocesová komunikácia - signály (asynchrónne udalosti), rúry, správy, semafory, zdieľaná pamäť.

**4. Mikrojadro**

Malé jadro OS plniace iba niekoľko málo potrebných funkcií - primitívna správa pamäte, komunikácia medzi procesmi. Väčšina funkcií z jadra sa presúva do užívateľskej oblasti - ovládače HW zariadení, služby súborového systému, virtualizácia pamäte, medzi užívateľskými procesmi sa komunikuje posielaním správ.

Výhody mikrojádra

ľahká prenositeľnosť OS, jadro je malé, je stabilnejšie

vyššia spoľahlivosť (moduly majú jasné API a sú ľahšie testovateľné )

vyššia bezpečnosť (menej kódu OS beží v režimu jadra)

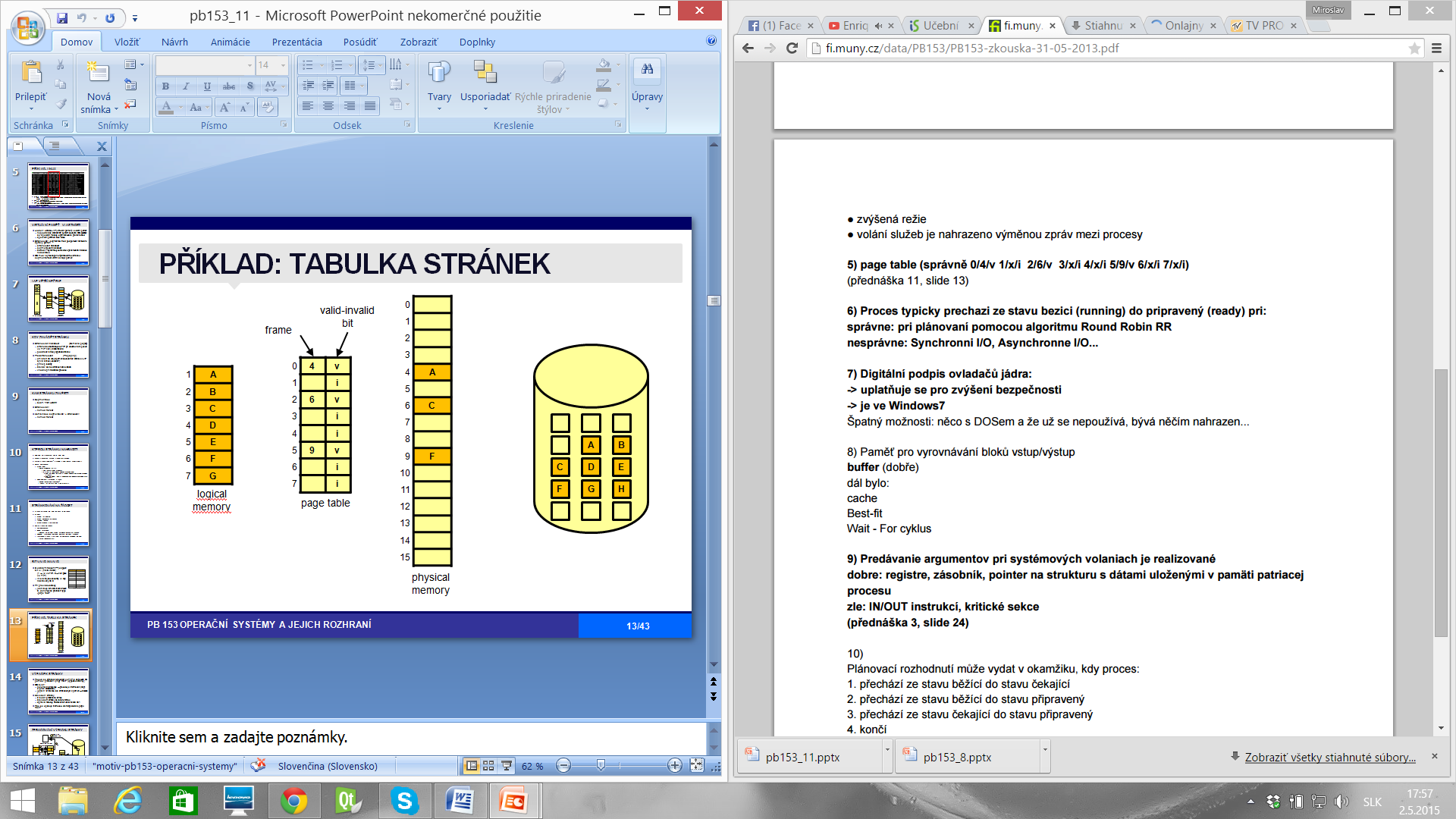
flexibilita (jednoduchšia modifikácia, pridanie, odberanie modulov)

všetky služby sú poskytované jednotne (výmenou správ)

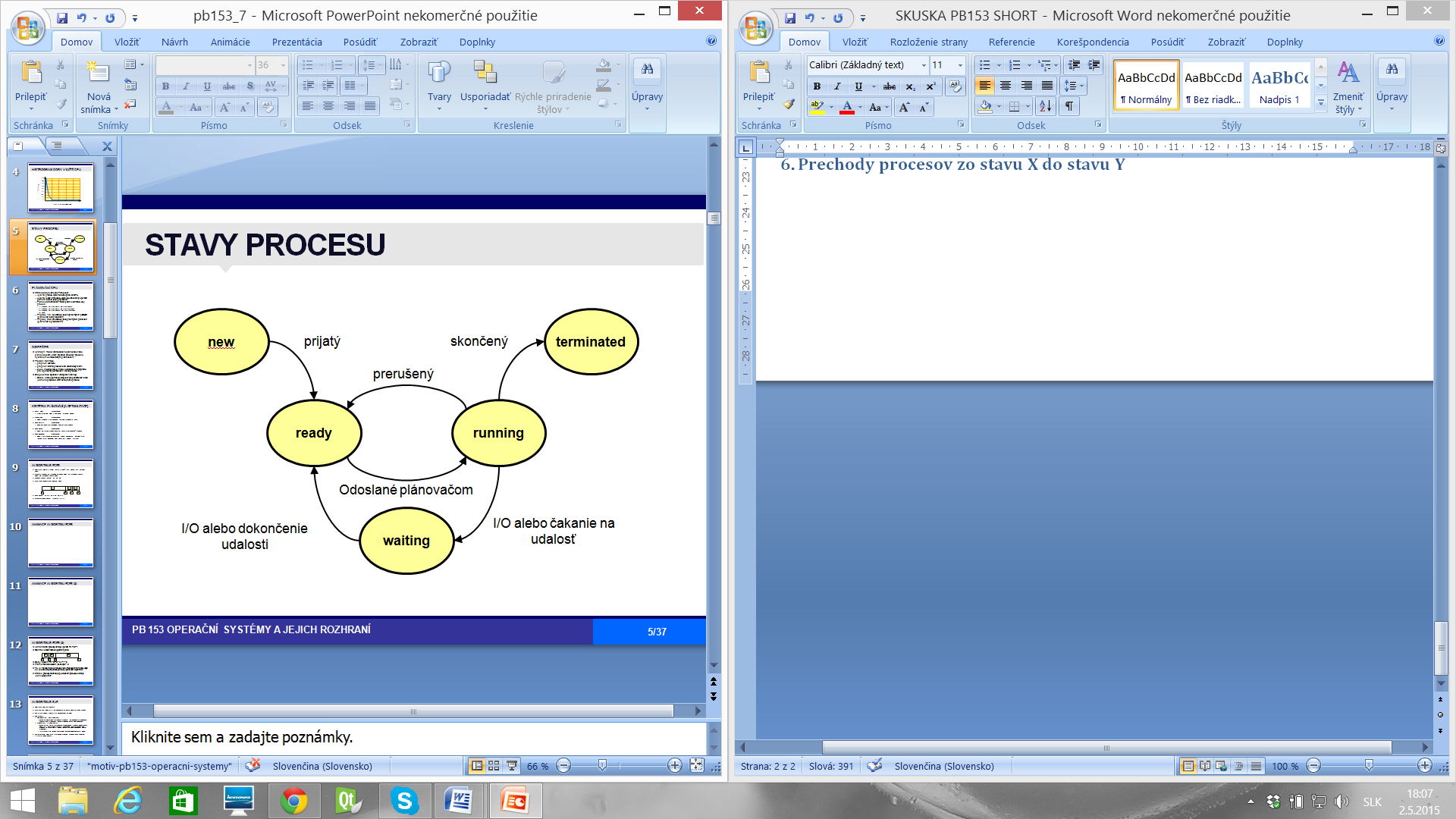
Nevýhoda mikrojádra

zvýšená réžia - volania služieb je nahradené výmenou správ medzi procesmi

**5. Page Table**



**6. Prechody procesov zo stavu X do stavu Y**



Proces typicky prechádza zo stavu bežiaci (**running**) do čakajúci (**waiting**) pre **synchrónne I/O**.

Proces typicky prechádza zo stavu bežiaci (**running**) do pripravený (**ready**) pri **plánovaní pomocí RR** (**odstavení od procesoru algoritmom RR**).

Proces sa môže nachádzať v jednom zo stavov:

- nový (new): práve vytvorený proces

- bežiaci (running): niektorý procesor práve vykonáva inštrukcie procesu

- čakajúci (waiting): čaká na určitú udalosť

- pripravený (ready): čaká na pridelenie času procesoru

- ukončený (terminated): ukončil svoje prevedenie

**7. Správa procesov**   
Proces je spustený program. Jeden program môže byť zároveň spustený viacerými užívateľmi. Proces je jednotkou práce systému. Pre svoju činnosť potrebuje zdroje. Typické aktivity OS v oblasti správy procesov: vytváranie a ukončovanie užívateľských a systémových procesov, potláčanie a obnovovanie procesov, mechanizmy pre synchronizáciu a komunikáciu medzi procesmi a mechanizmy pre detekciu a riešenie uviaznutia (**deadlock**).

**8. Plánovacie rozhodnutie sa môže vydať v okamihu, kedy proces:**

1. prechádza zo stavu bežiaci (running) do stavu čakajúcí (waiting)

2. prechádza zo stavu bežiaci (running) do stavu pripravený (ready)

3. prechádza zo stavu čakajúci (waiting) do stavu pripravený (ready)

4. končí

**Prípady 1 a 4 sa označujú ako nepreemptivné plánovanie (plánovanie bez predbiehaním).**

**Prípady 2 a 3 sa označujú ako preemptivné plánovanie (plánovanie s predbiehaním).**

**9. Plánovací algoritmy na Linuxe**  
Deadline

Nope  
Anticipatory  
CFQ

**10. Digitálny podpis ovládačov jadra:**

U Windows nejde pri podpisovaní o čistotu jadra ani o licencie. Ide predovšetkým o spoľahlivosť systému a s tým súvisiacu bezpečnosť jadra. Je vo WIN7. Linux moduly zvyčajné nepodpisuje ale všíma si licencie. Podpisovanie modulov v Linuxe - je k dispozícii od 2004 ale nevyužíva sa často.

**11.** Pamäť pre vyrovnávanie blokov vstupu/výstupu **je:** buffer.

**12. Vlákna:**

Vlákno je objekt ktorý vzniká v rámci procesu, je viditeľný iba vo vnútri procesu a je charakterizovaný svojím stavom. Model - procesy a vlákna - proces = jednotka vlastniaca zdroje, vlákno = jednotka plánovania činnosti. Linux-Unix-Posix - Knihovna „pthreads“.

Každé vlákno si udržuje svoj vlastný zásobník, program counter, registre a TCB (thread context block).

Prečo použiť vlákna - využitie multiprocesorových strojov, jednoduchšie programovanie.

Vlákno môže pristupovať k pamäti a zvyšným zdrojom svojho procesu. Zdroje procesu zdieľajú všetky vlákna jedného procesu. Súbor otvorený jedným vláknom majú k dispozícii všetky ostatné vlákna.

**Vlákna zdieľajú: pamäť (adresový priestor), číslo procesu, číslo vlákna**

Windows CreateThread() - Implementuje vlákna na úrovni jadra OS (implementace je zdařilá, umožňuje mimo iné paralelní beh vlákien jedného procesu na rôznych procesoroch).

Prečo využiť vlákna:

- využitie multiprocesorových strojov (vlákna jedného procesu môžu behať na rôznych CPU)

- jednoduchšie programovanie

- typický príklad: jedno vlákno prevádza užívateľom požadovaný úkol a druhé vlákno prekresľuje obrazovku

Výhody

vlákno sa vytvorí rýchlejšie než proces

vlákno sa ukončí rýchlejšie než proces

medzi vláknami sa rýchlejšie prepína než medzi procesy

jednoduchšie programovanie (jednoduchšia štruktúra programu)

u multiprocesorových systémoch môže na rôznych procesoch bežať viacero vlákien jedného procesu súčasne

- prepnutie vlákna pri plánovaní CPU je rýchlejšie ako prepnutie procesu

- aktuálna verzia Linuxu podporuje kernel-level threads

- Každé vlákno si udržuje svoj zásobník

- Vlákno vo Windows: fiber

- Vlákno v Linux, vytvorenie: clone()

**13. Fork:**

Vytvorí nový proces (v Linuxu) ako kópiu rodiča, jeho ekvivalent je createProcess()- vo Win API. Volání fork() implementované ako copy-on-write (tj. pokiaľ pamäť nie je menená je zdieľaná a až pri pokuse o modifikáciu je vytvorená kópia).

**14. Sysenter**

slúži k rýchlemu systémovému volaniu

**15. Predávanie argumentov pri systémových volaniach je realizované:**

registrami, zásobníkom, blokom v pamäti & pointer – (pointer na štruktúru s dátami uloženými v pamäti patričného procesu)

**16. Systémové volania**

Tvoria rozhranie medzi užívateľským procesom a OS. Typicky sú popísané ako inštrukcie assembleru a sú uvedené v programátorskom manuálu k OS. Nič menej existujú určité štandardy, ktoré uľahčujú prenositeľnosť programového kódu - v oblasti UNIXu: POSIX, v oblasti Windows: Win32 (Windows API).

- príkladom sú LINUXové volania open, close, exit

- slúžia ako rozhranie medzi aplikáciami a jadrom OS

- možné implementovať pomocou prerušenia

- systémové volania MS-DOS: int 0x21

- systémové volania Linux štandardne cez 0x80, novšie (synseter/syscall)

**17. POSIX**(skratka z Portable Operating System Interface) je prenosné rozhranie pre operačné systémy, štandardizované ako IEEE 1003 a ISO/IEC 9945. Vychádza zo systémov UNIX a určuje, ako majú POSIX-kompatibilné systémy vyzerať, čo majú vedieť, čo sa má ako robiť a podobne.

**18. FREEBSD**   
Systémové volanie cez VOLANIE FUNKCIE, alebo cez volanie brány call 7:0, ktorá obsahuje int 0x80. Číslo systémového volania je v registry eax. Viaže sa na zásobník. Výsledok sa nachádza v registry eax. Volajúci proces musí vyčistiť zásobník (parametre). Parametre systémového volania na zásobníku( 1 parameter uložený ako posledný).

**19. MS-DOS**   
Neumožňuje spúšťať procesy paralelne (single-tasking systém), Popisovač procesu (Program Segment Prefix)

**20. RES**

rezidentní pamäť, ktorá nebola od swapovaná (rezidentná čase, neodswapovaná fyzická pamäť, kterú proces použil)

**21. Real time systémy**   
sú určené pre špeciálne aplikácie typu riadenie strojov.

**22. Paralelný VS distribuovaný systém – Rozdiel je v zdieľaní/ nezdieľaný pamäti**

Paralelní systémy – úzko zviazané systémy - **niekoľko vzájomné komunikujúcich CPU zdieľajúcich jednu pamäť** a hodinový signál. Výhodou je vyššia priepustnosť systému a ekonomické využitie PC. –

-SMP - symetrický multiprocesorový systém, procesory sú si rovne, na všetkých beží rovnaká kopia OS, dnes podporuje OS, Linux, Windows, FreeBDS

- AMP asymetrický multiprocesorový systém, každý procesor rieši špecifickú úlohu, napr. jeden procesor plánuje ostatným prácu alebo určité typy bežia na jednotlivých procesoch

Distribuované systémy - **každý CPU ma vlastnú pamäť**, komunikujú pomocou komunikačných spojov. Výhodou je zdieľanie zdrojov a vyššia spoľahlivosť. Architektúry Klient-server ,Peer-to-peer.

**23. Multitasking - Timesharing**

CPU je multiplikovaný, v skutočnosti vždy beží len jedna úloha, medzi týmito úlohami sa však CPU prepína, takže užívatelia získajú dojem, že úlohy sú spracovávané paralelne. Multitaskingový systém umožňuje rade užívateľom počítačový systém zdieľať.

Oproti multiprogramovániu znižuje dobu odozvy (response time) interaktívnych procesov

Multitaskingové systémy sú už značne komplexné

správa a ochrana pamäti, virtuálna pamäť

synchronizácia a komunikácia procesov

CPU plánovacie algoritmy, súborové systémy

Procesy a mutiprogramovanie

prekladaním behu procesu maximalizujeme využití procesoru a minimalizujeme dobu odpovedi, procesu sú prideľovaný zdroje systému, berieme v úvahu priority a vzájomnú výlučnosť operácii, musíme zabrániť „uviaznutiu“ procesov (deadlock)

Multiprogramovanie – zvyšuje využitie CPU, ide o efektívnosť využitia CPU

**24. Nutná podmienka uviaznutia?**Obsadenie zdroja a čakanie na pridelenie ďalšieho

K uviaznutiu dôjde, keď začnú súčasne platiť 4 nasledujúce podmienky:

- **vzájomné vylúčenie (mutual exclusion)** / zdieľaný zdroj môže v jednom okamžiku používať len jeden proces

- **ponechanie si zdroja a čakanie na ďalší (hold and wait)** / proces vlastníctva aspoň zdroj čaká na získanie ďalšieho zdroje, dosiaľ vlastneného iným procesmi

- **bez predbiehania (no preemption)** / zdroj je možné uvoľniť len procesom, ktorý ho vlastní, dobrovoľne po tom, čo daný proces zdroj ďalej nepotrebuje

- **kruhové čakanie (circular wait)**

**25. Strategický plánovač** **– Dlhodobý**

Vyberá proces ktorý je možné zaradiť medzi pripravené procesy. Určuje stupeň multiprogramovania. Nie je spúšťaný veľmi často preto nemusí byť rýchly. Typicky pri ukončení jedného procesu rozhodne, ktorú úlohu ďalej vybrať k zavedeniu do pamäti a spustenie.

**26.Krátkodobý plánovač**

Prideľuje procesor pripraveným procesom, je spúšťaný často a musí byť rýchly.

**27. Strednodobý plánovač** **– Taktický**   
Vyberá proces ktorý môže byť zaradený medzi odložené procesy a naopak, čiže vyberá proces ktorý môže byť zaradený medzi pripravené procesy.

**28. K čomu slúži strace:**Strace is a diagnostic, debugging and instructional userspace utility for Linux. It is used to monitor interactions between processes and the Linux kernel, which include system calls, signal deliveries, and changes of process state. The operation of strace is made possible by the kernel feature known as ptrace. Some Unix-like systems provide other diagnostic tools similar to strace, such as truss.

Monitoruje systémové volania a signály pomocou špeciálneho syscallu a ptrace, ktorý slúži pre trasovanie bežiaceho kódu programu, ptrace je rozsiahla a veľmi silná záležitosť, dokáže sa napojiť na proces krokovať ho, zasahovať do jeho pamäti, odchytávať signály a pod., taktiež využívané GNU, debugerom gdb

**29. pthread\_create()?**

POSIX Threads, obvykle označováný ako Pthreads, je POSIXový standard pro vlákno (thread). Standard, POSIX.1c, Threads rozšírenie (IEEE Std 1003.1c-1995), definuje API pre vytváranie a manipuláciu s vláknami.

CreateThread() – vytvorenie vlákana, Win32

Implementácia API je možná na mnohých Unix operačných systémoch podporujúcich POSIX ako napr. FreeBSD, NetBSD, OpenBSD, GNU/Linux, Mac OS X, Solaris, DR-DOS a implementácia pro Microsoft Windows taktiež existuje za použitia podsystému SFU/SUA, ktorý poskytuje nativnu implementáciu niekoľkými POSIX API, poprípade taktiež pri využití balíčku tretej strany pthreads-w32, ktorý implementuje pthreads na vrcholu stávajících Windows API.

**30. Linux** – modulárne monolitické jadro

**31. Plánovanie fibers vo windows** – app si musí plánovať sama

**32. Zaistenie MUTAL EXCLUSION** – pasívna verzia

/na test máte 40 min, otázok bolo 17, max. bodov 40, potrebných na prechod 20, výbery z otázok a – e, viacero správnych odpovedí alebo žiadna, zvyčajne na jednu otázku nie je žiadna správna odpoveď, odpovede sú bodované od 1 – 3 body, vždycky je tam jeden RAG – pozrite si hlavne šípky ako smerujú, alebo PAGE TABLE, v každom teste je tiež nepreemtivný alebo preemptivní algoritmus, kľudne si nakreslite tabuľku uľahčí Vám to prácu/