Kende Viktor Domonkos

ENQMZK

1.) a.)

	E_1	E_2	E_3
P_1	67-53 **14**	15-14 **1**	5-4 **1**
P_2	13-0 **13**	5-5 **0**	3-1 **2**
P_3	107-46 **61**	27-17 **10**	5-0 **5**
P_4	132-127 **5**	25-0 **25**	4-1 **3**

b.) E_1: 67+13 + 107 + 132 = 319 (80, 187)

219 - 53 - 0 - 46 - 127 = **93** (266, 220)

E_2: 15+5+27+25 = 72

72 - 14 - 5 - 17 - 0 = **46** (58, 53)

E_3: 5+3+5+4 = 17

17 - 4 - 1 - 0 - 1 = **11**

c.)

2.) Lapozásos virtuális memória kezelés

A program adatai ~~lapokban memóriába~~ lapokra van felosztva és a memóriában létrehozott lapokban fogják őket tárolni. Ezek a lapok egységnyi méretűek. A virtuális címkezelés azt jelenti, hogy a program fordításakor a memóriacímek csak virtuális címek, nem valós fizikai címek. Valós címeket az OS memória menedzselő komponense fog nekik adni, futás időben.

Előnye, hogy a külső töredezettség kicsi, a belső töredezettség pedig szinte ~~elhanyag~~ semennyi. Illetve a fizikai memóriánál nagyobb programok is futtathatóak.
Hátránya, hogy a lapkezelés is erőforrásigényes, előfordulhat a laphiba.

Laphiba: ha a program futásához éppen szükséges lap nincs a memóriában.
Nem váratlan/szokatlan esemény, de cél az előfordulásának minimalizálása, mert lekezelése és a lap betöltése időigényes.

3.) Processz kontextus:

Minden az adott processz futásához szükséges adat. PID, processz kódrészei, process counter, erőforrások, amiket igényelhet, adatok amiket használ.
Kontextus váltás akkor történik, amikor az adott processztől elveszik a CPU-t, ~~de~~ például ütemezés vagy interrupt miatt és egy másik processz kapja azt meg. Ilyenkor az előző processz kontextusa betöltődik és az új processzé kerül betöltésre.
Ennek eredményeképp egy processz mindig csak a saját kontextusát látja és számára nem érzékelhető az ütemezés, olyan, mintha mindig is zavartalanul futott volna és a CPU végig az övé lett volna.

4., Kernel implementációs struktúrája

Struktúra szerint kétféle kernel implementációt különböztetünk meg.
Egyik a monolitikus kernel, amikor az egész kernel egyszerre töltődik be, mint egy egység. Egyszerű megvalósítás és gyors, viszont hátránya, hogy az is betöltődik, ami nem szükséges.
Ellenben a rétegekre osztott kernel az logikailag összetartozó rétegekre van osztva és csak az töltődik be, ami éppen szükséges.
Lassabb, de helytakarékosabb, illetve a rétegek absztrakt módon elfedik a részleteket, absztraktabb interfészt nyújtva.

Kernel alapvető interfészei:
- memória menedzselő komponensek
- I/O menedzselő komponensek
- hardver driver menedzselés
- ütemezés

5., Lapozásos virtuális memória kezelés

Lap: a memória ~~[illegible]~~ egyforma méretű (2-4KB) lapokra van osztva.
Virtuális: a program készítésekor csak virtuális memóriacímek jönnek létre, azokhoz valós fizikai címet futás időben a memória menedzselő komponens biztosít.
Allokálás az a folyamat, amikor egy processz lapjaihoz a memóriában lapokat foglalunk le, hogy oda töltődhessenek. Ez lehet fizikai memória, de szükség esetén háttértár is.
A szabad és foglalt lapokat nyilván kell tartani, hogy tudjuk, hova tölthetünk meg be, illetve hogy szükség szerint ~~[illegible]~~ kilapozást használjunk.
Laphiba: ha a processz futásához szükséges lap nincs a memóriába betöltve. Ilyenkor azt be kell tölteni, mielőtt a processz folytatódhatna.

Kilapozási stratégiák:
- FIFO: legrégebben betöltött lap megy ki. Egyszerű megvalósítás, de a régen betöltött, mégis sokat használt lap fölöslegesen megy ki.
- Második esélyes FIFO: hasonló az előzőhöz, csak minden lapnak két esélye van, amit egy bit reprezentál. Ha egy lap kimenne, bitje 0 lesz. Ha használják, újra 1.
- Legrégebben használt lap kilapozása
- Időarányosan a használatok számának figyelembe vétele