ITKA203 Käyttöjärjestelmät -- tentin yleiskuva -- kevät 2015

Tentin ja kurssin pisteytys

Tentin muoto on tänä vuonna uusittu täysin, painottaen kysymyksiä, joiden oikeat vastaukset ovat yksiselitteisiä ja nopeasti tarkastettavia. Maksimipistemäärä on 24. Kunkin vastauksen maksimiarvo on vähintään 0.5p. Väärä vastaus antaa saman verran miinuspisteitä kuin oikea vastaus antaa pluspisteitä. Tyhjä vastaus ei vaikuta pistemäärään. Tällä tavoin varmistetaan, että odotusarvo täysin satunnaisista vastauksista on nolla tai negatiivinen ja riski saada arvaamalla 50% oikein on hyvin pieni. Ensimmäisen tentin osallistujien arvosanajakauma tarkastetaan. Tarvittaessa pisteytystä skaalataan jälkikäteen toteuman mukaan, mikäli esim. läpäisyprosentti vaikuttaa epätavallisen pieneltä.

Arvosana läpäisyrajalla eli 12 pisteellä on 1 ja täysillä pisteillä 5. Pistealue 12-24 jaetaan tasaväleihin neljällä väliarvolla, jotka pyöristetään lähimpään 0.5 pisteen granulariteetilla ilmaistavaan pistemäärään:

Pisteväli	**Välin pituus**	**Arvosana**
[0.0, 12.0)	12p	hylätty
[12.0, 14.5)	2.5p	1
[14.5, 16.5)	2.0p	2
[16.5, 19.0)	2.5p	3
[19.0, 21.5)	2.5p	4
[21.5, 24.0]	2.5p	5

Jos pistemäärä osuu täsmälleen rajalle, luetaan arvosana rajan paremmalta puolelta, esim. 19.0 pistettä riittää arvosanaan 4.

Vapaaehtoisista demoista saadut bonuspisteet huomioidaan **vain siinä tapauksessa, että tenttivastauksista kertyy vähintään tuo 12 pistettä**. Bonuspisteet ovat voimassa kesän 2015 uusinnoissa. Kurssimerkinnän saaminen edellyttää, että pakolliset demot on tehty ja palautettu. **Osasuoritusten hyväksiluvusta myöhemmillä kurssikerroilla ei ole eikä tule mitään lupauksia**.

Jotta ensimmäiseen tenttipäivään ja uusintapäiviin osallistuvat opiskelijat olisivat arvostelun osalta keskenään samanarvoisessa asemassa, **bonuspisteitä saa vain takarajaan 2.6.2015 mennessä palautetuista vapaaehtoisista demoista**. Arvosanaa ei siis voi hinkuttaa ylöspäin jälkikäteen. "Matti- ja Maijamyöhäset" voivat kuitenkin perustellusta syystä palauttaa *pakollisia* demoja aina kesän viimeiseen uusintaan asti; tämä ei käsittääkseni vaikuta arvosteluun mitenkään eriarvoistavasti. Kesäkuun alusta alkaen palautuksista tulee sopia etukäteen meilillä tai puhelimitse.

Kysymysten aihepiirit ja jakauma

Kysymysten on tarkoitus mitata kurssin alussa määriteltyjen osaamistavoitteiden toteutumista (ks. osaamistavoitelista) mitattavilta osin. Esimerkiksi kykyä tulkita nettifoorumien ohjeita ei voida kirjallisessa tentissä mitata. Tavoitteet on jaettu osioihin *aihepiirien* mukaan ja kunkin aihepiirin sisällä *arvosanatavoitteen* mukaan. Tentissä pyritään noudattamaan jakoa seuraavin periaattein:

- Puolet kysymyksistä (läpäisyyn tarvittava 12 pistettä) liittyvät ydinainekseen eli mittaavat arvosanatavoitteen 1 mukaisia osaamistavoitteita.
- Loput kysymyksistä valitaan korkeammista arvosanatavoitteista tasaisesti siten, että hypoteettinen opiskelija, joka osaa vastata kaikkiin kysymyksiin tiettyyn arvosanatavoitteeseen saakka, jättäen muut kohdat tyhjäksi, saisi arvosanakseen juuri kyseisen arvosanan.
- Mahdollisuuksien mukaan kaikista osa-alueista tulee kysymyksiä, mutta tämän ominaisuuden automatisointi on haastavaa, joten täyttä kattavuutta ei voida täysin luvata. Satunnaisgenerointi saattaa siis painottaa kurssin alku- tai loppupuolen aihekokonaisuuksia.
- Kysymysten järjestys etenee arvosanatavoitteiden mukaisesti, joten esimerkiksi selkeästi arvosanaan 1 tähtäävä opiskelija voi vastata vain alkupuolen kysymyksiin (omalla riskillä väärien vastausten tuomista miinuspisteistä). Mutta eihän meillä toivottavasti ole yhtään tällaista opiskelijaa, vaan jokainen tavoitelee aina viitosta!

Kysymykset valitaan mainittujen periaatteiden mukaan satunnaisesti kysymyspankista, joka on pääosin salainen. Valinta pyritään tekemään koneellisesti (automatisointi tosin puuttuu tätä kirjoitettaessa; riittävän monipuolisen tietomallin ja algoritmin kehittely tentin arpomiseen vaatii näköjään hiukan pohtimista). Koodeihin liittyvissä kysymyksissä on oltava **tarkkana, koska jo yksi merkki voi muuttaa koodin merkityksen** aivan erilaiseksi. Väitekysymysten osalta on oltava **tarkkana, koska jo yksi sana voi vaikuttaa luonnollisella kielellä kirjoitetun väitteen totuusarvoon**.

Kysymykset numeroidaan juoksevasti, ja vastaus kuhunkin kysymykseen on lyhyt (kirjain, symboli, sana, numeroarvo tai "-", mikäli vastaus jätetään tyhjäksi). Vastauspaperille tulee siis kysymysten verran numeroituja rivejä. Mikäli vastataan normaalisti tenttivastauspaperiin, joudut itse kirjoittamaan numerot. Voi olla myös, että vastaukset kirjoitetaan suoraan kysymyspaperiin, joka tulee palauttaa. Vastauspaperin muoto selviää maanantaina 18.5.2015, kun ensimmäisen tenttikerran kysymykset on arvottu ja tulostettu paperille. Maanantaiaamuun mennessä palautetut vapaaehtoisen "Tuunaa tenttisi" -demon tenttikysymykset saattavat ehtiä mukaan kysymyspankkiin jo ennen ensimmäistä tenttikertaa (tosin tämä edellyttänee manuaalista formaattikonversiota, joten ei voi täysin luvata). Osa kysymyksistä toivottavasti edellyttää pohtimista ja suttupaperin käyttöä muistiinpanoille ja ongelmanratkaisulle.

Mallitentti

Jokaisen numeroidun kohdan (1-48) pisteytys: Oikea vastaus +0.5p, väärä vastaus -0.5p, tyhjä eli viiva (-) 0.0p.

1. Väite: Prosessilla on aina vähintään kaksi säiettä (A=kyllä, B=ei)

Yhdistä lauseen loppua vastaava kirjain numeroitujen lauseenalkujen perään siten, että lauseet ovat totta käyttöjärjestelmistä puhuttaessa. (A="... on prosessikohtainen", B="... liittyy useiden eri prosessien käsittelyyn", C="... ei liity prosesseihin")

2. Prosessitaulu ...
3. Prosessielementti (PCB) ...
4. Sivutaulu ...
5. Ready-jono ...

Järjestelmässä A kellokeskeytys tapahtuu kiinteästi 1000 kertaa sekunnissa. Järjestelmässä B kellokeskeytys tapahtuu kiinteästi 50 kertaa sekunnissa. Ajatellaan tilannetta, jossa molemmissa suoritetaan yhtäaikaa noin 10 intensiivisesti laskevaa sovellusta; vuoronnusmenettelyksi on molemmissa valittu priorisoimaton kiertojono.

6. Kumpi järjestelmä (A vai B) painottaa enemmän prosessorin käyttöastetta (engl. *utilization*)?
7. Kumpi järjestelmä (A vai B) painottaa enemmän yksittäisen prosessin läpimenoaikaa (engl. *turnaround*)?

Yhdistä lauseen loppua vastaava kirjain numeroitujen lauseenalkujen perään siten, että lauseet ovat totta käyttöjärjestelmien osajärjestelmistä puhuttaessa. (A="... tekee toimenpiteitä jokaisen kellokeskeytyksen yhteydessä"; B="... tarjoaa palvelut mm. kriittisen alueen keskinäiseen poissulkuun"; C="tekee toimenpiteitä sivukeskeytyksen (engl. *page fault*) yhteydessä"; D="tarvitaan ainoastaan, kun suoritetaan virtuaalikoneita"; E="ei ole välttämätön osa nykyaikaista käyttöjärjestelmää")

8. Virtuaalimuistin hallinta (*virtual memory management*) ...
9. IPC (*inter-process communication*) ...
10. Ikkunointijärjestelmä (engl. *window manager*) ...
11. Vuorontaja (engl. *scheduler*) ...

Järjestä seuraavat muistikomponentit niiden nopeuden mukaan: A=keskusmuisti, B=kovalevy, C=rekisteri, D=välimuisti.

12. vastauksessasi neljä järjestettyä kirjainta: nopein ensin, hitain viimeisenä

Käsitellään laitteistorajapintaa ja syöttö-/tulostusjärjestelmää. Pitävätkö seuraavat väittämät paikkansa?

13. Väite: Ns. laitteistoriippumattoman I/O-ohjelmiston tehtävänä on aloittaa laitteelta saapuvan keskeytyksen käsittely (A=kyllä, B=ei)
14. Väite: Laitteistoriippumaton I/O-ohjelmisto määrittelee järjestelmään kytketyille laitteille yksilölliset nimet (A=kyllä, B=ei)
15. Väite: Kokonaisuuden suorituskyvyn kannalta on hyödyllistä, että muut prosessit odottavat blocked-tilassa sillä välin, kun I/O-operaatio tehdään alusta loppuun (A=kyllä, B=ei)

Minkä osia seuraavat asiat ovat? (A="käyttöjärjestelmätilassa toimivan ohjelmiston"; B="käyttäjätilassa toimivan ohjelmiston"; C="laitteiston")

16. Keskeytyskäsittelijä
17. Muistinhallintayksikkö (MMU)
18. Käyttöjärjestelmäkutsun tekeminen (esim. AMD64:n assembler-käsky "syscall")
19. käskyosoiterekisteri (esim. RIP)

Yhdistä lauseen loppua vastaavat kirjaimet (mahdollisesti useita) lauseenalkun perään siten, että yhdistelmälause vastaa todellisuutta. Kaikki sopivat loput on löydyttävä vastauksesta. (A="mahdollistaa monen ohjelman yhdenaikaisen suorittamisen"; B="toimii vain moniydinprosessorissa"; C="ei voi aiheutua käyttäjätilassa toimivan prosessin toimenpiteiden johdosta"; D="voi tapahtua minkä tahansa konekielikäskyn jälkeen, riippumatta prosessorin hetkellisestä tilasta".

20. Prosessorin keskeytyskäsittely ... (mahdollisesti useita vaihtoehtoja)

Seuraava sessioesimerkki näyttää bash-shellissä tehtyjä komentoja ja niiden tulosteita (merkistökoodaus on UTF-8):

```
[nieminen@localhost esimerkki]$ whoami
nieminen
[nieminen@localhost esimerkki]$ ls -l
total 576
-rwxrw-r--. 1 nieminen nieminen 586431 May 14 20:32 kayttojarjestelmat.tex
[nieminen@localhost esimerkki]$ echo Heippa > kayttojarjestelmat.tex
[nieminen@localhost esimerkki]$
```

21. Väite: komentojen jälkeen tiedoston `kayttojarjestelmat.tex` pituus on pienempi kuin 586431 tavua. (A="kyllä"; B="ei")
22. Väite: komentojen jälkeen komento `./kayttojarjestelmat.tex` tulostaa konsoliin "Heippa"

Olkoon meillä seuraava POSIX-yhteensopiva shell-komentorivi:

```
who | grep "kortex.jyu.fi" | sort -f -r | uniq > kortexilaiset
```

23. Montako komentoa rivillä on yhteensä?
24. Montako argumenttia rivillä on yhteensä?

Yhdistä lauseen loppua vastaava kirjain lauseenalun perään siten, että lause on totta käyttöjärjestelmien suunnittelusta puhuttaessa. (A="käyttöjärjestelmän ja apujärjestelmän ohjelmakoodin kokoon"; B="käyttöjärjestelmän palveluiden toteuttamiseen käyttäjätilassa toimivina prosesseina")

25. Monoliittisen ja microkernel -käyttöjärjestelmän suurin ero liittyy ...

26. Kurssin esimerkeissä ja demoissa on tutkittu Linux-käyttöjärjestelmälle ja x86-64 -prosessoriarkkitehtuurille käännettyjä konekielisiä ohjelmia. Missä järjestyksessä seuraavat alueet/segmentit sijaitsevat muistiosoitteiden suuruusjärjestyksessä nykyään käytössä olevan ABI:n mukaan? Luettele pienimmästä osoitealueesta suurimpaan: A=pino; B=koodi; C=keko; D=data.
27. Väite: Tietoturvan kannalta on hyvä, että muistin pinoalueella on suoritusoikeus (A=kyllä; B=ei)

28. Väite: Käyttäjätilassa toimiva ohjelma ei pysty millään keinolla varmistamaan, että sen kaksi peräkkäistä konekielikäskyä varmasti seuraisivat toisiaan prosessorin suoritussyklissä (A=kyllä; B=ei)
29. Väite: Keskeytykseen siirtymisen yhteydessä prosessori tallentaa muistiin automaattisesti meneillään olevan prosessin tiedostodeskriptorit (A=kyllä; B=ei)

Tutkitaan seuraavaa luentoesimerkeistä tuttua, mahdollisesti osittain muokattua, C-kielistä ohjelmanpätkää (POSIXin pthread-kirjastoa hyödyntävä):

```
#define N 1000
pthread_mutex_t mymutex = PTHREAD_MUTEX_INITIALIZER;
uint64_t summa = 0;
void * saikeen_koodi(void *v) {
    int i;
    for (i = 1; i <= N; i++){
        summa++;
    }
    return NULL;
}

int main(int argc, char *argv[]) {
  pthread_t saieA, saieB;
  pthread_create(&saieA, NULL, saikeen_koodi, NULL);
  pthread_create(&saieB, NULL, saikeen_koodi, NULL);
  pthread_join(saieA, NULL);
```

```
  pthread_join(saieB, NULL);
  if (summa==2000){
    return 0; // homma toimi
  } else {
    return 1; // homma ei toiminut
  }
}
```

30. Väite: Yllä esitetty main() palauttaa aina 0:n (A=kyllä; B=ei)
31. Väite: Yllä esitetty ohjelma ratkaisee kuluttaja-tuottaja -ongelman kahdella säikeellä (A=kyllä; B=ei)
32. Väite: Yllä esitetty ohjelma voi aiheuttaa lukkiutumistilanteen (A=kyllä; B=ei)

Seuraavassa on C-mäisenä pseudokoodina ehdotus rengaspuskuria käyttävän Kuluttaja-tuottaja -ongelman ratkaisemiseksi semaforipalvelua käyttäen.:

```
tuottaja(){
  while(true) { // tuotetaan loputtomiin
    tuota();    // tehdään yksi datapätkä
    wait(EMPTY);
    wait(MUTEX);
    siirra_puskuriin();
    signal(MUTEX);
    signal(FULL);
  }
}

kuluttaja(){
  while(true) { // kulutetaan loputtomiin
    wait(FULL);
    wait(MUTEX);
    lue_puskurista();
    signal(MUTEX);
    signal(EMPTY);
    kuluta();   // käytetään yksi datapätkä
  }
}

main(){
  EMPTY=tee_semafori( PUSKURIN_KOKO );
  FULL=tee_semafori( 0 );
  MUTEX=tee_semafori( 1 );
  kaynnista_saie(tuottaja);
  kaynnista_saie(kuluttaja);
}
```

33. Väite: Esitetty koodi ratkaisee kuluttaja-tuottaja -ongelman oikeellisesti (A=kyllä; B=ei)

34. [edellinen tehtävä on arvoltaan 1.0p; tämä kohta jätetään tyhjäksi]

Seuraavassa on luennolla ja monisteessa esitetty minimalistinen shell-ohjelma, tai ainakin läheisesti sitä vastaava (kommentit poistettu, rivit numeroitu, näytetty vain olennainen toimintopätkä):

```
1   while(true){
2     luekomento(komento, argumentit);
3     pid = fork();
4     if (pid > 0) {
5       status = wait();
6     } else if (pid == -1) {
7       ilmoita("fork() epäonnistui!");
8       exit(1);
9     } else {
10      exec(komento, argumentit);
11      ilmoita("Komentoa ei voitu suorittaa!");
12      exit(1);
13    }
14  }
```

35. Väite: Rivin 10 exec() -kutsun tuloksena syntyy uusi prosessielementti (A=kyllä; B=ei)

36. Väite: Kun edellä esitetty minimalistinen shell on käynnistänyt uuden prosessin, se jättää prosessin suorittamaan tehtäväänsä ja odottaa välittömästi uuden komennon antamista (A=kyllä; B=ei)

37. Väite: Rivin 12 suorittaminen lopettaa minimalistisen shellin suorituksen. (A=kyllä; B=ei)

Kuvitellaan, että meillä on käytössä yksinkertainen tietokone, jonka muistiosoitteissa on 20 bittiä, joista ensimmäiset 8 ilmoittavat sivunumeron ja loput 12 tavun indeksin sivun sisällä. Keskusmuistiin mahtuu vain 8 sivua muistia. Käyttöjärjestelmän täytyy heittovaihtaa loput kovalevylle. Tietorakenteiden tilanne on seuraava:

```
Prosessin (PID=2) sivutaulu:
rivi# virt.   fyys.  muist.  dirty   diskIndex
      sivu    sivu

0     0x02    0x03   1         0      0x0010
1     0x03    0x04   0         0      0x0022
2     0x7f    0x05   1         1      0x00bc

Prosessin (PID=7) sivutaulu:
rivi# virt.   fyys.  muist.  dirty   diskIndex
      sivu    sivu

0     0x02    0x02   0         0      0x0004
1     0x03    0x04   1         0      0x0008
```

```
2     0x04   0x01   0          0       0x00f2
3     0x2e   0x06   1          1       0x006c
4     0x7f   0x00   1          1       0x0007
```

Järjestelmän kehystaulu:

fyys. sivu	omistajan PID	omistajan rivi#	aikayksiköt edellisen käyttökerran jälkeen
0x00	7	4	12
0x01	234	9	2
0x02	567	6	5
0x03	2	0	195
0x04	7	1	7
0x05	2	2	155
0x06	7	3	12
0x07	876	1	4

38. Mihin fyysiseen muistiosoitteeseen kohdistuisi prosessin 2 tekemä kirjoitus virtuaalimuistiosoitteeseen 0x7f123?

39. Mihin fyysiseen muistiosoitteeseen kohdistuisi prosessin 7 tekemä luku virtuaalimuistiosoitteesta 0x7f123?

40. Prosessi 2 suorittaa hyppykäskyn aliohjelmaan muistiosoitteessa 0x03200. Tapahtuuko prosessorissa sivunvaihtokeskeytys? (A=kyllä; B=ei)

41. Prosessissa 7 aiheutuu sivunvaihtokeskeytys. Korvausalgoritmi on LRU. Minkä fyysisen sivun sisältö tallennetaan levylle?

Tutkitaan seuraavaa GNU assemblerilla kirjoitettua ohjelmanpätkää:

```
_start:
        movq    $4,%rax  # luku 4 rekisteriin RAX
        movq    $0,%rdi  # luku 0 rekisteriin RDI
silimu:
        inc     %rdi     # lisää 1 rekisterin RDI arvoon
        dec     %rax     # vähennä 1 rekisterin RAX arvosta
        jnz     silimu   # hyppää, jos edellisen käskyn tulos ei ollut 0
```

42. Kuinka monta konekielikäskyä ohjelmanpätkän jälki sisältää, ts. kuinka monta käskyä sen suorittaminen alusta loppuun vaatii?

43. Mikä on rekisterin RAX sisältö ohjelmanpätkän suorituksen lopuksi?

44. Mikä on rekisterin RDI sisältö ohjelmanpätkän suorituksen lopuksi?

45. Väite: Unixin i-solmuihin (engl. *inode*) perustuvassa tiedostojärjestelmässä tiedoston nimi sisältyy tiedoston i-solmuun (A=kyllä; B=ei)

46. Väite: Unixin i-solmuihin (engl. *inode*) perustuvassa tiedostojärjestelmässä tiedoston viimeisin muutosaika sisältyy tiedoston i-solmuun (A=kyllä; B=ei)

Kurssin esimerkeistä tuttua Linuxille käännettyä C-ohjelmaa ajetaan x86-64 -arkkitehtuurilla ja debuggerilla on nähtävissä seuraavat hetkelliset tiedot:

```
Rekistereitä:
RIP (käsky)  0x0000000000400504
RSP (huippu) 0x00007fffffffdcc0
RBP (kanta)  0x00007fffffffdcf0

Muistin sisältöä:
                 0x7fffffffdd08:      0x0000000100000000
                 0x7fffffffdd00:      0x00007fffffffde38
                 0x7fffffffdcf8:      0x0000000000400647
       kanta --> 0x7fffffffdcf0:      0x00007fffffffdd50
                 0x7fffffffdce8:      0x0000000000000001
                 0x7fffffffdce0:      0x0000000000000000
                 0x7fffffffdcd8:      0x0000000000000000
                 0x7fffffffdcd0:      0x0000000000000000
                 0x7fffffffdcc8:      0x0000000000000000
      huippu --> 0x7fffffffdcc0:      0x0000000000000000
```

47. Mikä on kutsupinossa nykyistä aktivaatiota *edeltävän* aktivaation pinokehyksen kannan osoite? Ilmoita heksalukuna.

48. Montako tavua nykyisellä aktivaatiolla on käytössään paikallisille muuttujille - ilmoita kymmenjärjestelmän lukuna.

Mallivastaukset perusteluineen

1. B (ei tietenkään kahta vaan yksi tai "ei yhtään 'säiettä sinänsä'". Joka tapauksessa ei missään nimessä kahta. Lämmittelykysymys...)

2. B (taulussa on kaikkien prosessien elementit)

3. A (yhden prosessin tiedot omassa elementissään)

4. A (kunkin prosessin muistikartta omassa sivutaulussaan)

5. B (jonossa on kaikki suoritusvalmiit prosessit; liittyy ilman muuta prosesseihin, mutta ei yksittäiseen)

Vaihtoehto C oli tietoinen hämäys kohdissa 2-5; jos ei tiedä, mikä on ready-jono tai sivutaulu, niin voisi riskillä veikata ettei liity prosesseihin, kun nimessä ei sitä sanota suoraan. Onneksi kurssin jälkeen tiedät, että sivutaulu ja ready-jono liittyvät mitä suurimmassa määrin käyttöjärjestelmän tärkeimpään

abstraktioon eli prosesseihin. Varo lisähämäystä: voisihan tässä olla myös jotain, mikä ei liitykään prosesseihin... vaikea tietysti keksiä, mikä sellainen asia yksikäsitteisesti ajatellen olisi. Ehkä vaikkapa "prosessorin suoritussykli" ... sehän alkaa aina tietokoneen käynnistyessä, vaikka yhtään ns. prosessia ei ole olemassa ennen kuin käyttöjärjestelmän alkulataaja on päässyt luomaan ensimmäisen prosessin pitkän aikaa käynnistyksen jälkeen(?).

6. B (mitä harvemmin keskeytetään, sen pidempiä pätkiä voidaan käyttää laskentaan)
7. B (mikäli kuorma on pääasiassa laskentaa, vuoronnusmenettely on kiinnitetty kiertojonoksi ja prosesseja on vähän, pidemmät aikaikkunat lyhentävät myös yksittäisen prosessin läpimenoaikaa)

(Tehtävänantoa kohtiin 6-7 piti täsmentää mallivastausta miettiessä, koska muuten olisi tilannesidonnaista: esim. satoja yhdenaikaisia prosesseja kiertojonossa johtaisi 50 Hz kellolla sekuntien mittaisiin vasteaikoihin, mikä ei kyllä kuulosta hyvältä läpimenoaikojenkaan suhteen prosesseille, jotka eivät välttämättä tarvitsisi edes koko 0.02s aikaikkunaansa... **Jos varsinaiseen tenttiin tulee vahingossa moniselitteisiä kysymyksiä**, tee paras arvaus tai jätä tyhjäksi ja **kommentoi vastauspaperiin, mistä havaitsemasi moniselitteisyys johtuu** - otetaan havainnot huomioon pisteyksessä esim. poistamalla huono kysymys arvostelusta. Korjattuna kysymys on käsittääkseni OK; ilmoita heti, jos olet eri mieltä..)

8. C
9. B (IPC:n kommunikointipalvelut ovat syy synkronoinnin tarpeelle, ja sama moduuli tarjoaa myös nämä synkronointipalvelut)
10. E (Esim. POSIX ei ota mitään kantaa graafisiin käyttöliittymiin)
11. A

Tehtävissä 8-11 vaihtoehto D oli tietoinen harhautus "onnenonkijoille", jotka eivät ole tutustuneet asiaan yhtään; silloin voisi vaikka mennä sekaisiin "virtuaalikone" ja "virtuaalimuisti". Asiaan tutustuneena toki tiedät, että virtuaalimuisti on ihan oma käsitteensä, joka on mukana kaikissa nykyisissä prosessoreissa ja käyttöjärjestelmissä, olivatpa ne virtuaalisia tai eivät.

12. C,D,A,B
13. B (eihän toki; laitteistoriippumaton ohjelmisto on I/O-ohjelmistokerroksen yläpuolella, ja sen alla on laiteajurit ja keskeytyskäsittelijät, joista viimeiseksi mainittu aloittaa laitteelta saapuvan keskeytyksen)
14. A (kyllä, nimien määritys on yksi I/O-ohjelmiston yläkerroksen tärkeistä tehtävistä)
15. B (ei missään nimessä; päinvastoin, on syytä suorittaa mahdollisimman paljon muita prosesseja sillä välin kun laite hoitaa mahdollisesti hyvin pitkään kestävän fyysisen I/O-toimenpiteen)
16. A (vähintään keskeytyskäsittelijän ensimmäinen käsky tapahtuu aina käyttöjärjestelmätilassa, johtuen prosessorin laitteistotasolla tapahtuvasta FLIH-toiminnosta)
17. C (MMU on osa prosessorilaitteistoa)

18. B (käyttöjärjestelmäkutsu eli käyttöjärjestelmän koodiin siirtyminen on tarkoitettu sovellusohjelman käyttöön, siis itse käsky on aina käyttäjätilassa toimivan ohjelmiston osa)

19. C (tietenkin prosessorilaitteen komponentti)

20. A,D (Vaihtoehdot B,C olivat tässä tietoinen hämäys: Keskeytyskäsittely toimii tietysti yksittäisessäkin prosessorissa. Se aiheutuu käyttäjätilan prosessin syscall-käskyn tai minkä tahansa laittoman operaation tuloksena, siis esim. väärä käsky tai muistiviittaus kartoittamattomaan osoitteeseen tai osoitteeseen, jonka sisältö ei ole keskusmuistissa heittovaihdon takia. Vaihtoehto D vaatii pohtimista ja *täytyypä miettiä, onko liian paha kompa oikean tentin kysymyspankkiin*: prosessorin suoritussyklin mukaan laite- tai kellokeskeytystä ei voi tulla, mikäli keskeytykset eivät ole sallittuina, esim. aina keskeytyskäsittelijän koodin alussa. *Mutta* jos käyttöjärjestelmäkoodissa on laiton käsky, niin kyllä tällainen "poikkeus-/virhetilanteen" aiheuttama keskeytys tapahtuu, ja sitten on NMI:t, joita kurssilla ei edes käsitelty... pahus. *Toistetaan:* **jos oikeaan tenttiin tulee vahingossa jotakin moniselitteistä, laita siitä kommentti vastauspaperiin** - otetaan huomioon arvostelussa koko tenttikerran osalta.)

21. A (tiedoston pituus tulee olemaan 7 tavua, koska sisältönä tasan ASCII-merkit H, e, i, p, p, a ja unix-rivinvaihto. Vaikka rivinvaihto olisi kahden tavun yhdistelmä, olisi tiedoston pituus 8 tavua eli paljon pienempi kuin alkuperäinen 586431.)

22. B (ei tietenkään; missään ei luvata että näin käy.)

Miten tuossa kohdassa 22 varsinaisesti käy, niin mahdollisesti järjestelmä yrittää tulkita tiedostoa sh-skriptinä, vaikkei alussa olekaan "#!" - silloin se yrittää suorittaa komennon nimeltä Heippa. Ei nyt lähdetä olettamaan, että tällaista komentoa olisi asennettu järjestelmään ja että se tulostaisi Heippa! Ei myöskään saivarrella siten, että tulostuisi virheilmoitus "Heippa: unknown command", joka sisältää myös merkkijonon "Heippa". Tehtävän voisi korjata yksikäsitteisemmäksi laittamalla komennoksi `echo ./Heippa > kayttojarjestelmat.tex`, jolloin on `ls -l` -tulosteesta selvää, että komentoa `./Heippa` ainakaan ei ole olemassa. Pitäisi myös periaatteessa vielä sopia, että ennen komentoyritystä `./kayttojarjestelmat.tex` kukaan ei ole kirjoittanut toisesta prosessista mitään muuta tiedoston päälle!

Jospa nyt vaan **yksinkertaisuuden vuoksi sovitaan, että tällaisten shell-esimerkkejä sisältävien tehtävien tapauksessa järjestelmässä ei tapahdu samaan aikaan mitään, mitä tehtävässä ei ole nähtävillä.**

Huomaa, että tästä saadaan generoitua pienin muutoksin hyvin erilaisia tehtävävariaatioita. Esim. voisi olla `echo Heippa >> kayttojarjestelmat.tex` (silloin kysymyksen 22 vastausta ei oikeastaan voi tietää tuntematta tiedoston `kayttojarjestelmat.tex` sisällön alkua; tilanne tosin muuttuu yksikäsitteiseksi, jos tiedostolla ei ole suoritusoikeutta nähtävissä `ls -l`:n tulosteessa; silloin komento `./kayttojarjestelmat.tex` ei varmasti onnistu). Toinen variaatio olisi esim. `cat ./kayttojarjestelmat.tex`, joka nykyisessä esimerkissä tulostaisi "Heippa". Kolmas variaatio voisi olla `echo ./kayttojarjestelmat.tex` tai jopa `./kayttojarjestelmat.tex > output` tai `cat ./kayttojarjestelmat.tex > output` (jotka tulostaisivat tiedostoon `output` eikä varmasti konsoliin ollenkaan; hämäyksenä tiedoston nimi voisi olla myös `console` - ei kuitenkaan `konsoli` koska muuttuisi ikävästi kompaksi taas: tulostuisi "konsoliin" eli tiedostoon, jonka nimi on "konsoli")

Melkein joka luennolla nähdyt komennot `cat` ja `echo` sekä operaattorit `>`, `>>` ja `|` täytyy tuntea, joten em. variaatioihin on helppo vastata. **Tarkkaavainen pitää olla syntaksin suhteen**. Tällaisella on

kuitenkin helppo hämätä kylmiltään tenttiin tulevia onnenonkijoita tai pakolliset shell-demot heikolla keskittymisellä tehneitä.

23. 4 (komennot ovat operaattoreilla eroteltujen pätkien ensimmäiset merkkijonot eli who, grep, sort ja uniq)
24. 3 (komennoissa on välilyönneillä erotellut argumentit "kortex.jyu.fi", "-f", "-r"; sen sijaan lopussa oleva "kortexilaiset" on tietysti tiedoston nimi, koska se tulee uudelleenohjausoperaattorin `>` jälkeen

Huom: Tehtävien 23-24 esimerkkinä olevan komentorivin kaltaisia on helppo tehdä erilaisia, joten tentissä joudut tulkitsemaan jotakin mahdollisesti ennen näkemätöntä riviä siten, kuin shell sen tekee. Komentojen ei myöskään tarvitse olla luennolta tuttuja eikä järkeviä. Yhtä hyvin olisi voinut olla:

```
xyzzy | viutsivautsi "tupu hupu lupu" | ankka echo käki | kissa > cat
```

Tämä olis hiukan pahempi hämäys, koska siinä on argumentin roolissa "tutun näköinen" merkkijono `echo`, tiedostonimen roolissa tutun näköinen `cat`, ja pitäisi myös muistaa, että lainausmerkit sijoittavat välilyöntejä sisältävän merkkijonon samaan argumenttiin. Eli tässä olisi komennot xyzzy, viutsivautsi, ankka ja kissa (4 kpl) ja argumentit "tupu hupu lupu", "echo", "käki" (3 kpl). Viimeinen "cat" olisi sen tiedoston nimi, johon tuloste tulisi, jos nuo käytetyt komennot olisivat olemassa ja tekisivät jotakin järkevää.

Kun olet miettinyt shellin perussyntaksin kunnolla läpi, *ei tässä pitäisi olla mitään epäselvää*, olipa esimerkki miten kiero tahansa - oikeasti pahoja kierouksia, eli oikeasti tilanneriippuvia syntakseja tai muuta, *ei* tietenkään tule kysymykseen.

25. B
26. B,D,C,A (näinhän se on piirretty kaikkiin muistiavaruuskuviin, selitetty teksteissä ja nähty debugger-esimerkeissä)
27. B (ei sitten missään nimessä; pikemminkin vaarallista olisi se!)
28. A (totta kyllä; käyttäjätilan ohjelma ei voi estää keskeytyksiä, joten ei voi varmistaa kahden käskynsä todellista peräkkäisyyttä.)

    Kysymyksessä 28 on vaara kuvitella, että esim. keskinäinen poissulku mutexilla aikaansaisi oikeasti atomista käyttäjäkoodin suorittamista. Atomisesti suoritetaan kuitenkin vain käyttöjärjestelmäkutsun minimaalisesti tarvittavat osat. Itse kriittisen alueen keskinäinen poissulku perustuu siihen, että nimenomaan *samaa lukkoa odottavat prosessit* eivät pääse suoritukseen ennen lukon vapauttamista. Muihin järjestelmässä toimiviin prosesseihin tai kello- ym. keskeytyksiin tämäkään ei vaikuta!
29. B (tiedostodeskriptorit eivät liity FLIH-toimintaan mitenkään!)

Kysymyksessä 29 olisi voinut olla mikä tahansa asia, joka ei tallennu prosessorin atomisessa FLIH-käsittelyssä. Milloin vastaus olisi ollut A eli kyllä? Jos järjettömän ehdotuksen "meneillään olevan prosessin tiedostodeskriptorit" sijasta olisi ollut esimerkiksi "meneillään olevan prosessin kontekstin olennaisimmat osat", "rekisterin RIP sisällön", "keskeytyvän prosessin jatkamiseksi tarvittavat tiedot".

30. B (no ei. lupaavasti ohjelman alussa määritellään `mymutex` niminen lukko, mutta *eihän sitä missään vaiheessa pyydetä varaamaan eikä vapauttamaan.* Muuttujaan `summa` kohdistuvan kilpa-ajon vuoksi ei missään nimessä voida olla varmoja, että lopuksi summa olisi kaikissa olosuhteissa oikeellinen eli tässä tapauksessa 2000.)

31. B (ei mitään tekemistä kuluttaja-tuottaja -ongelman kanssa; hyvä, että tiesit tämän saman tien, kuin apteekin hyllyltä :))

32. B (ei lukkoja, niin ei lukkiutumisia, tietenkään)

Tehtävien 30-32 koodin olisi voinut rikkoa myös eri tavoin: Lukko voitaisiin esim. varata ennen summausta, mutta ei vapauttaa. Silloin ensimmäinen lukitseva säie pääsisi suorittamaan, mutta toinen ei koskaan (jäisi odottamaan lukittua mutexia). Pääohjelma odottaisi molempia säikeitä, joista ainakaan toinen ei ikinä päättyisi. Eli pääohjelman säie odottaa säikeen toimenpidettä (loppuminen), joka odottaa toisen säikeen toimenpidettä (lukon vapauttaminen). Näin ollen vastaukset kysymykseen olisivat 30: B (eli sama mutta eri syystä; nyt pääohjelma ei koskaan palauta mitään), 31: B (edelleen ei tekemistä eri asian kanssa), 32: A (voi aiheuttaa, koska se varmasti aiheuttaa lukittumisen).

Koodi olisi voinut olla myös kunnossa, eli lukko sekä varataan ja vapautetaan säikeessä. Vaihtoehtoisia sinänsä toimivia paikkoja käskyille voisi olla useita ja jatkokysymys voisi olla esim:

```
/* Vaihtoehto A */
void * saikeen_koodi(void *v) {
    int i;
    for (i = 1; i <= N; i++){
        pthread_mutex_lock(&mymutex);
        summa++;
        pthread_mutex_unlock(&mymutex);
    }
    return NULL;
}

/* Vaihtoehto B */
void * saikeen_koodi(void *v) {
    pthread_mutex_lock(&mymutex);
    int i;
    for (i = 1; i <= N; i++){
        summa++;
    }
    pthread_mutex_unlock(&mymutex);
    return NULL;
}
```

Kummassa vaihtoehdossa (A vai B) käytetään aika tehokkaammin hyödylliseen laskentaan? (selvästi B; vrt. luentoesimerkki, jossa tätä käytiin - lähes kaikki aika menee lukitusten hoitamissen vaihtoehdossa A).

Huomaa, että koodi on mahdollista rikkoa myös esimerkiksi seuraavalla tavalla (ei varmasti toimi, mutta menisi liian kompaksi siinä mielessä, että varaamattoman lukon vapauttamisen vaikutus pitäisi tarkistaa standardista):

```
void * saikeen_koodi(void *v) {
    int i;
    pthread_mutex_lock(&mymutex);
    for (i = 1; i <= N; i++){
        summa++;
        pthread_mutex_unlock(&mymutex);
    }
    return NULL;
}
```

Tämäkin aiheuttaisi lukkiutumisen, eikä olisi edes mitenkään kompa:

```
void * saikeen_koodi(void *v) {
    int i;
    pthread_mutex_lock(&mymutex);
    for (i = 1; i <= N; i++){
        summa++;
    }
    return NULL;
    pthread_mutex_unlock(&mymutex);
}
```

Pittäis vaan ymmärtää rakenteisen kielen peräkkäissuoritus, eli se, että returnin jälkeen tulevaa koodia ei suoriteta (mikä on olennaisesti Ohjelmointi 1:n esitieto ja kaikilla hyvin hallussa).

Vastaavan tyyppisiä synkronointiin liittyviä kysymyksiä voi tulla myös kuluttaja-tuottaja -tilanteesta sekä "ruokailevat nörtit" -kuvaelmasta. Toimenpiteiden järjestysvaihtoehtoja kannattanee piirrellä suttupaperille, jos se helpottaa vastauksen löytämistä.

33. A

34. Järkevää jatkokysymystä tuottaja-kuluttajasta en toistaiseksi keksinyt, ja osaamistavoitteenkin muoto oli yksittäinen "osaa verifioida, toimiiko annettu esimerkki vai ei". Pisteytyksen osalta poikkeuksellisesti 1.0, mutta kysymyskohtia mieluiten tasaluku 48, joten tämä kohta jää ainakin toistaiseksi tyhjäksi.

Tässä oli kopioituna ja lyhennettynä monisteen pseudokoodiesimerkki. Toinen vaihtoehto olisi pätkä luentoesimerkkinä olleesta ja toimivaksi näytetystä C-kielisestä toteutuksesta.

Tällainen on helppo rikkoa muuttamalla semaforien wait() ja signal() -kutsujen järjestystä tai parametrina annettavien semaforien järjestystä niin, että kuluttaja tai tuottaja tai molemmat menevät sekaisin. Silloin oikea vastaus tietenkin on, että ei ratkaise ongelmaa.

Toimenpiteiden järjestysvaihtoehdoista on uskoakseni syytä varmastua suttupaperilla, koska muuten on vaikeata ymmärtää.

35. B (ei tietenkään; fork() on ainoa tapa luoda uusi prosessielementti; exec() on jo seuraavaa vaihetta..)

36. B (ei, koska tämä esimerkki on sama kuin monisteessa, sisältäen kutsun wait(), joka odottaa lapsiprosessin loppumista; **tarkkana:** mielellään laitan kysymysarvontaan variaatioon, jossa wait() ei tapahdu. Silloin lapsiprosessi *jää* taustalle ja uutta komentoa lähdetään lukemaan saman tien!)

37. B (ei, koska se tapahtuu lapsiprosessissa, ei alkuperäisen shellin prosessissa, jonka on tarkoitus jatkaa soolona forkin epäonnistuttua ja siksi tämä exit tässä tehdäänkin)

38. 0x05123 (luetaan prosessin 2 sivutaulusta, mitä fyysistä sivua muistiosoitteen alkuosa vastaa; loppuosa eli 4k-sivun sisäinen indeksi pysyy samana.)

39. 0x00123 (luetaan prosessin 7 sivutaulusta, mitä fyysistä sivua muistiosoitteen alkuosa vastaa; loppuosa eli 4k-sivun sisäinen indeksi pysyy samana.)

40. A (sivutaulun virtuaalista sivua 0x03 vastaavalla rivillä on muistibitti 0, mistä tiedetään, että sivu ei ole tällä hetkellä fyysisessä muistissa; prosessori tekee tällöin sivuvirhekeskeytyksen)

41. HUPS! Tämäkin kysymys oli viallinen. Oikeasti ei tarvitse tallentaa minkään sivun tietoja, koska tässä korvattaisiin fyysinen sivu 0x03, mutta sen sisältöä ei tarvitsisi tallentaa - dirty-bitti on 0, eli sivun sisältö on nykyisessä muodossaan tallessa jo ennestään.

Oikeellinen kysymys 41 olisi ollut "Minkä fyysisen sivun sisältö korvataan lukemalla levyltä?" jolloin yksikäsitteinen vastaus olisi ollut alunperin tarkoittamani 0x03, koska kyseistä sivua on käytetty kauimman aikaa sitten. Toinen tapa saada vastauksesta yksikäsitteinen olisi ollut varmistaa, että sivutaulussa olisi ollut vastaava "dirty-bitti" ykkönen!

Oikeassa tentissä yritän viimeiseen asti välttää moniselitteisiä kysymyksiä tai sellaisia, joihin ei ole yksikäsitteistä vastausta! *Toistetaan vielä kerran*: **Jos oikeassa tentissä on moniselitteisiä kysymyksiä, jätä vaikka tyhjäksi ja kommentoi, miksi kysymys ei ole validi ja yksikäsitteinen.** Hyvä puoli on se, että jos asian on ymmärtänyt, niin myös moniselitteisyys käy ilmi kysymyksestä!

Kysymysten 38-41 esimerkki oli nyt suoraan luentomonisteesta. Haave on, että tällainen generoitaisiin erikseen jokaiseen tenttiin, jossa kysymys esiintyy, ja kysymysvaihtoehtojakin olisi enemmän kuin tässä (ks. esimerkiksi monisteen esimerkkikysymykset), joten vastausten antaminen edellyttäisi muistinhallintaluvun asian osaamista luvussa olevan leluesimerkin tasolla. Pelkkä ulkoaopettelu ei siis auttaisi. Automatisointi pystyisi myös varmistamaan, että moniselitteisiä vastausvaihtoehtoja ei voisi tulla vahingossa. Ongelma automatisoinnissa on, että aika käy vähiin ennen kuin ensimmäisen tentin kysymysten täytyy olla kopioitavana toimistossa, eikä automaattia vielä ole olemassa...

Muita kysymysmalleja olisivat esimerkiksi: tallennetaanko fyysisen sivun 0x03 tiedot levylle? **Tarkkana tässä taulukoiden ja algoritmin kanssa:** Jos tosiaan vanhin sivu ei ole likainen, eli "dirty-bitti" on 0, sivua *ei* tallenneta. Tai esimerkiksi: tallennetaanko fyysisen sivun 0x01 tiedot levylle? LRU-menetelmässä ei muutella sivun tilannetta, jos kyseessä ei ole kauimman aikaa käyttämättä ollut. Eli tässä (osin virheellisessä) esimerkissä olisi käsittely kohdistunut vain fyysiseen sivuun 0x03, jos johonkin.

42. 14 (johtuen silmukasta, joka päättyy siihen, että RAX on ykkösen vähentävän dec-käskyn jälkeen nolla.)

Huom: En usko, että tällainen olisi mahdollista tehdä oikein ilman kynää ja suttupaperia, esim. seuraavalla tavoin (mahdollisesti kevyemmällä notaatiolla, jonka itse vaan jotenkin ymmärrät):

```
_start:
        movq    $4,%rax  # luku 4 rekisteriin RAX (jäljen 1. käsky)
        movq    $0,%rdi  # luku 0 rekisteriin RDI (jäljen 2. käsky)
silimu:
        inc     %rdi     # lisää 1   (RDI=1)      (jäljen 3. käsky)
        dec     %rax     # vähennä 1 (RAX=3)      (jäljen 4. käsky)
        jnz     silimu   # hyppää, jos tulos ei 0 (jäljen 5. käsky)
silimu:
        inc     %rdi     # lisää 1   (RDI=2)      (jäljen 6. käsky)
        dec     %rax     # vähennä 1 (RAX=2)      (jäljen 7. käsky)
        jnz     silimu   # hyppää, jos tulos ei 0 (jäljen 8. käsky)
silimu:
        inc     %rdi     # lisää 1   (RDI=3)      (jäljen 9. käsky)
        dec     %rax     # vähennä 1 (RAX=1)      (jäljen 10. käsky)
        jnz     silimu   # hyppää, jos tulos ei 0 (jäljen 11. käsky)
silimu:
        inc     %rdi     # lisää 1   (RDI=4)      (jäljen 12. käsky)
        dec     %rax     # vähennä 1 (RAX=0)      (jäljen 13. käsky)
        jnz     silimu   # hyppää, jos tulos ei 0 (jäljen 14. käsky)

# Sitten ei tapahdu hyppyä, koska vähennyksen tulos oli 0. Pätkän
# jälki on valmis
```

43. 0
44. 4

Helppoja modauksia tämän tehtävän mallikoodiin olisi esimerkiksi lisätä loppuun vielä käsky:

```
mov rax,rdi # siirrä rekisteristä RAX rekisteriin RDI
```

Silloin jäljessä olisi 15 käskyä ja lopputuloksena RAX==0 ja RDI==0 myös.

Tehtävän esimerkki voi tietysti olla muukin kaikkein yksinkertaisimmista AMD64-käskyistä koostuva max. 10 riviä pitkä koodi, joka on kommentoitu vastaavalla tavalla rivi riviltä. Myös muistiin kohdistuvia operaatioita voi olla, jolloin syntaksi selitetään täysin auki kommentissa. Silloin voidaan kysyä muistipaikan sisältöä pätkän lopuksi.

Kynä ja paperi auttavat!

45. B (i-solmu sisältää metatietoa, mutta ei tiedoston nimeä, joka on hakemistotiedoston sisältöä. Linkkien vuoksi samaan tiedostoon voi olla useita eri nimiä (engl. *hard link*) eli ikäänkuin nimettyjä viitteitä muista hakemistorakenteen kohdista. Tämä on aivan normaalia, sillä esim. jokainen alihakemisto viittaa omaan ylähakemistoonsa nimellä `..` ; ks. tiedostojärjestelmiä käsittelevä luento...)
46. A (kyllä; i-solmu pitää yllä nimeä vaille kaikesta metatiedosta, mm. aikaleimoista, tiedoston koosta sekä sisällön varsinaisesta sijainnista esim. kovalevyn pinnassa)

47. 0x7fffffffdd50 (sillä nyt ei niin väliä, onko kaikki äffät kirjoitettu tai onko alussa ylimääräisiä nollia, mutta vastauksesta pitää ilmetä, että olet ymmärtänyt, että edellisen kannan osoite löytyy nykyisen kantaosoitteen ilmoittamasta muistipaikasta, joten alku pitää olla "0x7f" loppu pitää olla tasan "dd50" eikä mitään muuta.

48. 48

HUOM: Alkuperäisessä mallivastauksessa oli itselläni ajatusvirhe tässä:

Tehtävässä 48 pitää muistaa, että pinon huippuosoitteessa *sijaitsee viimeisin pinon käyttöön varattu tila.* Näinhän sen täytyy toimia, jotta uuden datan työntäminen onnistuu minkä kokoiselle objektille tahansa! Sinnehän voitaisiin seuraavaksi työntää yksi tavu tai useampia, vaikka tämän kurssin esimerkkejä on helpotettu siten, että kaikki luvut ovat olleet 64-bittisiä eli aina 8 tavun mittaisia.

Näin ollen myös osoitteesta 0x7fffffffdcc0 alkavat 8 tavua ovat nyt aliohjelman käytössä.

Lisäksi pitää muistaa, että ensimmäinen paikallisille muuttujille varattu paikka on edellisessä tehtävässä kysytyn aiemman aktivaation kantaosoitteen jälkeen, ts. tässä tapauksessa muistipaikan 0x7fffffffdcf0 jälkeen. Vähennyslaskulla saa kaikille paikallisille varatun tilan eli 0x7fffffffdcf0 - 0x7fffffffdcc0 = 0x30. Muunto kymmenjärjestelmään: 3 * 16 = 48.

Voi myös katsoa tehtävänannon tulostetta ja laskea, että kannan kohdalla olevan 8-tavuisen luvun ja huipun osoitteen väliin jää tilaa 6 kappaleelle 8-tavuisia lukuja. Siis 6 * 8 = 48 tavua. (Eli omakin ajatusvirheeni tässä oli, että pinon huippuosoittimen kohdalla ei olisi käytössä olevaa dataa, mutta *onhan siellä tietysti* tosiaan viimeisimmäksi varattu kohde tai tila.)

Tämän tehtävän tarkoitus on mitata aktivaatiokehyksen kokonaisosaamista, joten sen variaatioissa voidaan kysyä myös esim. aiemman aktivaation RIP:tä sen jälkeen kun tästä aktivaatiosta tullaan palaamaan käskyllä `ret`.

Enemmän tarkkaavaisuutta vaativassa variaatiossa voisi kysyä, mikä tulee RIP:n arvoksi, jos välittömästi seuraava suoritettava käsky olisi `ret` - vastaus pitäisi napata pinon huipun alta. Mallitentin esimerkki ei silloin olisi kovin hyvä, koska pinon päällä olisi 0x0, joka olisi esimerkkinä olleessa ABI:ssa laiton muistiviittaus (tarkoituksella kartoittamaton osoite). Nyt sitten pitäisi lukea tosi tarkasti speksit, että muuttuuko RIP tosiaan nollaksi *ennen* kuin järjestelmä yrittää hakea osoitteesta käskyä. Voihan se olla niinkin, mutta näitä nyansseja ei varmasti ole voitu käsitellä tällä kurssilla, joten kysymystä ei voisi oikein tuolla tapaa asettaa. Jos huipulla olisi vaikkapa nykyinen RIP (0x0000000000400504) tai periaatteessa mikä tahansa hiukan yli 0x400000 (varmasti kartoitettu koodialueelle), niin vastaukseksi tulisi sitten yksikäsitteisesti juuri se! Esimerkiksi olisi ollut:

```
Muistin sisältöä:

                0x7fffffffdcd0:      0x0000000000000000
                0x7fffffffdcc8:      0x0000000000000000
      huippu --> 0x7fffffffdcc0:      0x0000000000400567
```

Sitten jos olisi käsky `ret` niin yksikäsitteisesti seuraava RIP:n arvo olisi 0x400567. Soveltuva jatkokysymys olisi tälle, että mikä on RSP:n arvo kyseisen käskyn `ret` jälkeen, niin tietysti `0x7fffffffdcc8`, koska pinon huipulta poksautettiin pois 64-bittinen muistiosoite.