OS (2. dio) → kolokvij

1. Koji je princip izrade programa od izvornog koda do izvršne datoteke?

Princip je sljedeći:

1. Piše se izvorni (source) kod u nekom editoru (može biti više fileova, npr. glavni.c, zbroji.c, zaglavlje.h);

→ u datoteci sa .h nastavkom se nalaze prototipi funkcija, ali ne i definicije dok se u fileu s .c nastavkom može napisati i definicija funkcije (uholiko već ne postoji u nekom predefiniranom fileu).

2. Izvorni kod se prevodi (compile) u binarni (→ objektni) i kreira se tablica simbola.

→ kompajler prvo provjerava je li sve ispravno napisano i ide po kodovima gledajući ispravnost opira funkcija (postoje li)

Sada može prevesti izvorni u binarni kod.
Kreira se tablica simbola. Binarna datoteka sadrži adrese i binarni kod programa koji se nalazi na tim adresama.
Tablica simbola se sastoji od imena funkcije, value (→ adresa fun.), class, type, size, line i section. Name je ime funkcije, class može biti T (definirana, tj. postoji, odnosno dohvaćen je kod) i U (undefined), odnosno kod te funkcije još nije dohvaćen.
Primjer: main T type: FUNC, veličina: 000000 2C...
Tablica je zapisana u binarnoj (objektnoj) datoteci glavne funkcije (zajedno s kodom).
Dakle, tablica simbola se sastoji od zapisa svih funkcija (→ glavnog programa) i globalnih varijabli.
Postupak se ponavlja i s drugim datotekama (kao u pr. sa zbrojiv.c)
Predefinirane funkcije (kao što su printf ili scanf) su već kompajlirane i nalaze se u standardnoj biblioteci lib.

3. Slijedi postupak povezivanja (link-anje) svih komponenti u jednu cjelinu

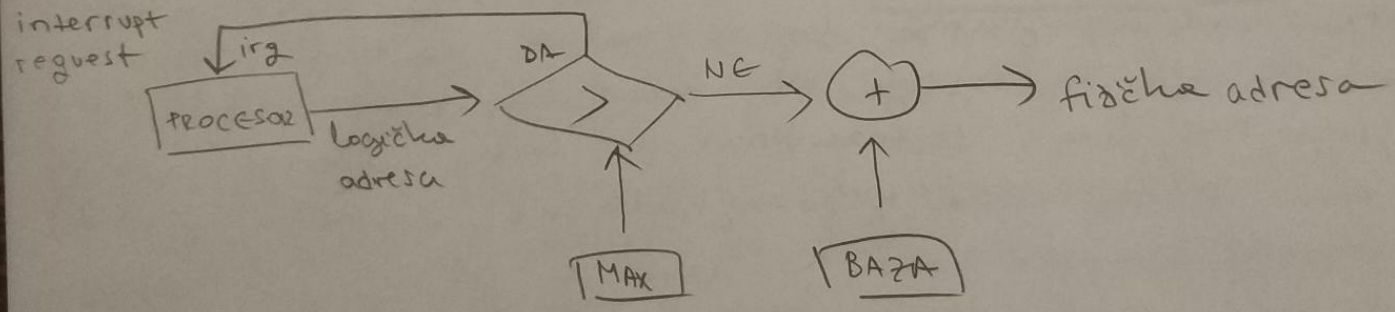
U memoriji se upiše glavni program (kod) te pozivi njegovih funkcija. Definicije funkcija se traže u libu ili ostalim datotekama i kada se nađe, kopira se kod funkcije u memoriju, a pointer za poziv se ažurira da pokazuje na početak koda.

Po istom principu se traže i sve ostale funkcije.
Tablicom simbola evidentiramo što tražimo.
Postoje dvije vrste linkanja.
Dinamičko – sve se povezuje još u izvođenju, odnosno tijekom izvođenja. Primjer je C#, prednost je što ne zauzimamo puno memorije jer inicijalno nemamo ne funkcije, ali je sporije

Statičko – Sve se poveže pa se tek onda izvodi. Primjer je C, puno je brže.

Na početku svega imamo kod programa → logički adresni prostor od neke početne (0) do neke krajnje (MAX) adrese. U njemu sami kod u memoriji su podaci (globalne varijable), heap (dinamičke varijable) te stack (stog, lokalne varijable).
→ kod funkcija

2. Kako se vrši generiranje fizičke adrese?
Fizička adresa se dobije kao zbroj baze i logičke adrese.
Generiranje vrši Memory Managment Unit sklop.



3. Što je defragmentacija?
To je postupak prilikom kojeg se programu koji se ne može zapisati u bloku, ali ima dovoljno memorije u dijelovima, mora osloboditi dovoljno mjesta na način da se drugi programi premještaju kako bi se dobio kontinuirani blok. Budući da traži dosta procesorskog vremena, nije povoljno izvršenje. Stoga će se koristiti podjela programa na dijelove.

(4) Objasniti dodjelu memorije po stranicama i preslikavanje iz logičkog u fizički adresni prostor.

Neka neki program ima određen broj lokacija za kod (na disku). To je logički adresni prostor. Memorija mora imati isto određeni broj lokacija. To je fizički adresni prostor. Te lokacije su kodirane s određenim brojem bita (npr. za 14 lokacija je potrebno 4 bita → 2 lista nadopuna).

logički adresni prostor ćemo podijeliti na stranice veličine 2^N (blokove → stranice). Ukoliko je potrebno, program se proširi do pune zadnje stranice. (br. lokacija)

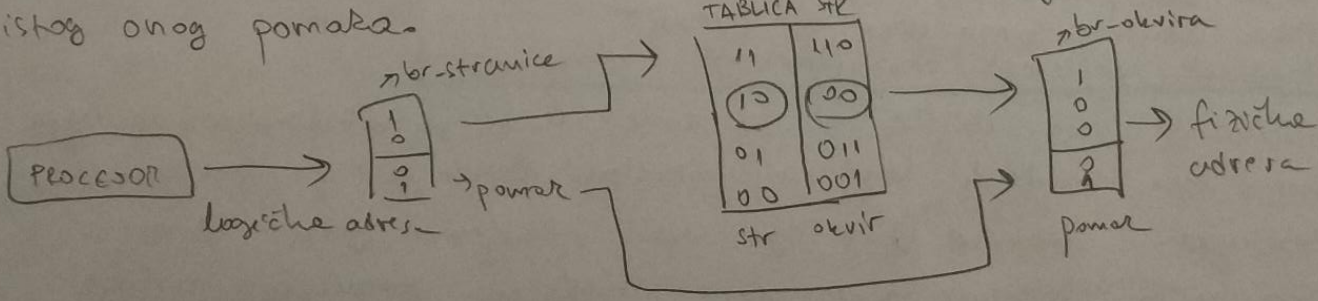
Fizički adresni prostor ćemo podijeliti na okvire veličine 2^N (blokove → okvire).

Svaka dio bitova (koji definiraju lokaciju) predstavlja broj stranice (okvira), a dio pomak od početka stranice (okvira).

Svaka se stranica logičkog adresnog prostora preslikava u proizvoljan okvir fizičkog adresnog prostora.

Tablica stranica pridružuje skup parova stranica-okvir, odnosno u koji okvir se preslikala koja stranica.

Preslikavanje iz logičkog u fizički adresni prostor:
Logička adresa se sastoji od broja stranice i pomaka. Pomak ostane isti, dok se na osnovu broja stranice u tablici stranica nađe odgovarajući okvir i preslika u konačnu fizičku adresu koja se sastoji od broja okvira i istog onog pomaka.



Primjer:
Program 16 lokacija → potrebno 4 bita: 0000 → 1111
Memorija 32 lokacije → 5 bita → 00000 → 11111
Blokovi veličine 4 lokacije
Program: prva dva bita su broj stranice, a druga 2 pomak.
Memorija: prva tri bita su broj okvira, a druga 2 pomak.
(gledamo veličine blokova: koliko nam treba za pomak → 2 bita).

5. Objasniti TLB

TLB (Translation Lookup Buffer) je nastao kao Intelovo rješenje sklopa za preslikavanje logičke u fizičku adresu, ukoliko je broj stranica prevelik (samim time i tablice stranica), odnosno nastao je zbog različite veličine programa (veliki raspon). Problem je bio u tome što bi procesor morao imati različit broj registara za tablicu stranica.
Tablica stranica je zapisana u memoriji u PCB-u (čitava).
Preslikavanje ide ovako: postavi se tablica stranica odredene veličine (random stranica se odabere) i kada je potrebno preslikavanje (br. stranice se traži u toj tablici. Kada se nađe odgovarajući okvir, generira se fizička adresa s identičnim pomakom kao u logičkoj adresi. Kada se nađe odgovarajući okvir, to se zove pogodak (jer se nalazi u toj random tablici stranica).

Ako se ne nađe odgovarajući okvir, to se zove promašaj i tablica stranica se ažurira s novim parovima sve dok ne dođe do pogotka. (Napomena: ta privremena tablica je u registrima).
Skica ko prije samo više sri parovi*

6. Objasnit prednost modularstva programa!

Aplikaciji se dijele u module te se u memoriju upisuju samo oni moduli koji su potrebni. Ostali se po potrebi dopisuju. Stranice (kažemo da upisujemo samo dio stranica u memoriju, moramo tablicu stranica proširiti sa bitom prisustva, kako bi os znao da treba pripirati stranicu sa diska.

7. Objasniti postupak dopisivanja stranice koja nedostaje u memoriju, s diska (kada dođe do promašaja stranice).

Postupak je sljedeći:

1. OS sprema stanje aktivnog procesa u PCB;
2. aktivni proces se prebacuje u stanje čeka-disk;
3. OS pronalazi slobodan okvir u memoriji;
4. pronalazi stranicu na disku (iz logičke adrese);
5. pokreće transfer stranice u memoriju (10ms-10ms, sporo);
6. OS pokreće prvi proces iz reda pripravnih;
7. kada je stranica prebačena, OS prebacuje proces u red pripravnih, ažurira tablicu stranica i bit postojanja ide 1.

8. Koje su prednosti i mane virtualne memorije?

Prednost: Ne moramo imati cijeli program u memoriji čime se štedi vrijeme (brže je pokretanje) i memorija.

Nedostatci: Promašaji znatno umanjuju brzine izvođenja što znači da se mora osigurati da ne dogadaju što manje.

9. Opiši sloj prema UIF uređajima kod komunikacije između aplikacije i UI uređaja (Programsko ostvarenje UI operacije)

Sloj se sastoji od registara: data (podaci), control i status.
Ti su registri potrebni kako bi se omogućio pristup hardveru na unificiran način (koji uređaj, koja operacija i podaci).
Kada dođe neki podatak, očito će i pronaći u registre CPU-a i konačno u memoriju. U ovom procesu će koristiti međuspremnike.
Da bi procesor znao da je došao podatak, stalno ispituje status, sve dok ne bude 1, kada pokreće prijenos podataka.
Pseudokod bi bio:

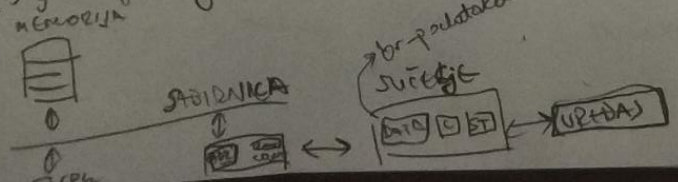
PROCESOR:
početak
  pročitaj zastavicu
  ako je zastavica=0 tada
    pročitaj zastavicu
  pročitaj podatak iz međuspremnika;
  inače
kraj

UIF uređaj
početak
  pročitaj zastavicu
  ako je zastavica=1 tada
    pročitaj zastavicu,
  upiši podatak u međusprij
  inače
kraj

Ova je programsko obavljanje UIT operacija, jer program provjerava status, ali mana pristupa je što CPU troši puno procesorskog vremena provjeravajući status i stoga je bolje rješenje sklop koji hardverskim signalom javlja status 1.

10. Opiši obavljanje UIT operacija pomoću zahtjeva za prekidom
Kada se izvodi neki program, može doći do signala koji označava zahtjev za prekidom (IRQ - Interrupt signal) zbog neke UIT operacije. Tada se u prvom programu tekuća naredba mora završiti do kraja, a PC (trenutni) programa spremiti na stog. Tada te procedura grana u proceduru za obradu prekida. Dakle, PC je pushan na stog (a na vrh stoga pokazuje unij. registra SP - Stack Pointer), pokreće te funkcija za prijenos podataka, na koju sada pokazuje PC registar (automatski se upiše). Ova funkcija prenosi podatak u registar pa u memoriju. One vrijednosti registara CPU-a koje će biti pregažene ovom funkcijom će se pushati na stog kako bi se mogle obnoviti. Kada te podatak prenese do kraja u memoriji, sa stoga se popuju sve vrijednosti (obnove te registri i PC prethodnog, prekinutog programa) i program se nastavi (rti naredbom unutar funkcije, return from interrupt) dobijemo adresu programa u PC-u). U memoriji te na početnoj adresi nalazi i reset vektor (100) koji pokreće osnovni program nakon restarta, na adresi 100.
Međutim i ovaj način ima manu; podatak se ne prenosi izravno u memoriju. Stoga možemo napraviti sklop kojim se ostvaruje izravan pristup memoriji.

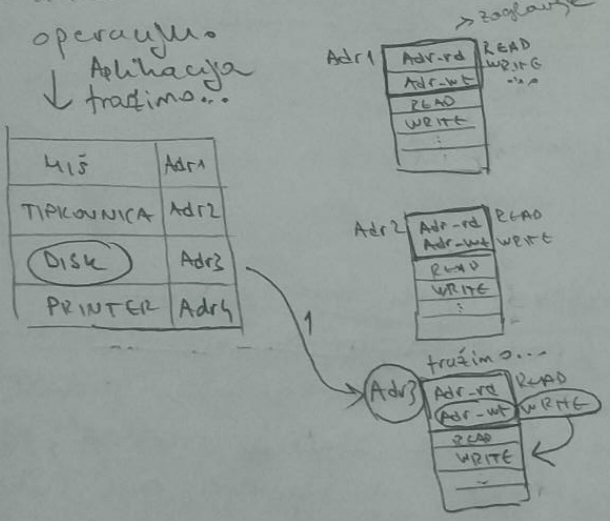
11. Opiši DMA sklop. *vidi stranicu 8 (sklop)
DMA (Direct Memory Access) sklop je sklop koji se koristi direktnim pristupom memoriji kad UIT operacija. Sadrži inteligentni DMA kontroler. On sadrži registar adrese, registar stanja podataka te brojilo podataka. Da bi se podatak prenio u memoriju, UIT međusklop pošalje zahtjev za prijenosom. Potom DMA pošalje procesoru zahtjev za sabirnicom (DMARQ) i kada može, CPU dozvoli korištenje sabirnice (DMAACK). Potom se podatak direktno prebaci u memoriju na odredenu adresu i DMA na kraju šalje zahtjev za prekidom procesoru → IRQ signal.



Napomena: Ako je više podataka, zahtjev za prekidom će doći kad brojilo podataka bude 0.

12. Opiši kako rade programska podrška modula (drivera) U IZ uređaja?

Aplikacija šalje OS-u podatke o U/I operaciji; uređaj, operaciju i podatke (npr. disk, write, neki podatak → adresa), a OS ih mora transformirati kako bi podatke razumio U IZ uređaj.
Prvo trebamo dobiti adresu modula koji treba obaviti U IZ operacije. Stoga postoji tablica priključenih uređaja koja ujedno sadrži i pokazivače na tablice funkcija uređaja. Te tablice funkcija u zaglavlju sadrže adrese pojedinih modula (funkcija) (npr. read, write). Tako kada nađemo uređaj koji trebamo preko pokazivača i tablice u zaglavlju dobijemo željene funkcije za operacije.

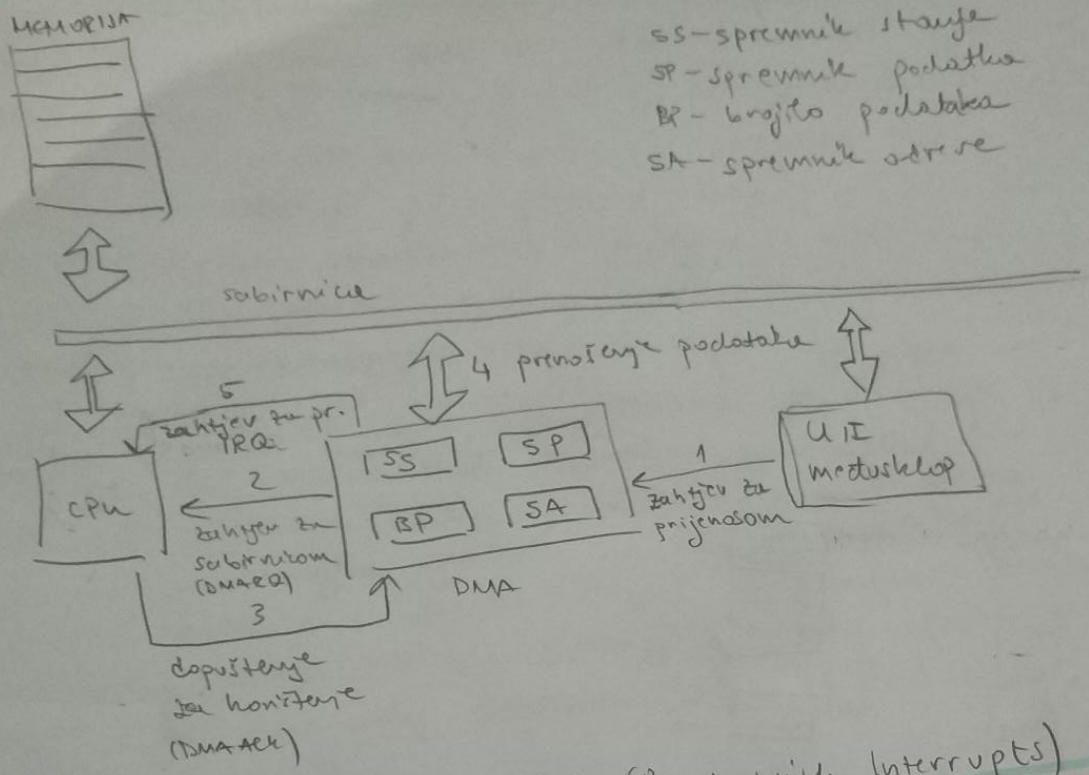


13. Kako izgleda programski: pristupanje aplikacije U IZ uređaju?

Aplikacija pristupa U IZ uređaju na jedinstven način bez obzira na karakteristike uređaja.

```
#include <stdio.h>
void main()
{ FILE *fp;          → adresa modula
  fp = fopen(name, mode);   → uređaj → operacija
  fscanf(fp, "format string", var-list); // fp - adresa za čitanje
  fprintf(fp, "format string", var-list); // fp - adresa za pisanje
  fclose(fp); // više ne koristimo uređaj;
}
```

11. Sklop DMA ✱



15. Opiši čitanje pomoću prekida (Read with interrupts)

Aplikacije šalje OS-u triplet podataka: uređaj, operacija i adrese podataka. Sučelje UI se sastoji od registara podaci (data), command i status. Svaki se ispituje status registar, odnosno je li uređaj spojen. U command registar se upisuje read (ukoliko je podatak došao. U tablici uređaja se traži uređaj koji je definirala aplikacija (jer su naredbe, funkcije unificirane po razinama) treba odrediti uređaj). Sada je podatak stigao u data registar, Interrupt handler gleda tko je napravio prekid, podatak se šalje Device handleru koji će obaviti funkciju za aplikaciju nad uređajem i prenijeti podatak aplikaciji. Postupak je završen.

14. Opiši što je File Managment sustav i načine alokacije datoteka.

To je sustav koji omogućuje preslikavanje fizičke organizacije podataka (ove na disku) u logičku (kako je vidi korisnik). Svaka datoteka ima svoj deskriptor koja sadrži podatke o toj datoteci i podatke o smještaju u blokove.
Block Managment se brine kako dodijeliti blokove na disku.
Postoje tri strategije:

1. Kontinuirana alokacija - datoteka se zapisuje u blokove na disku u kontinuitetu. Pritom za svaku datoteku postoji podatak o početnom bloku i duljini. Problem pristupa je što se u slučaju manjka prostora sve mora premještati kako bi ta datoteka stala u kontinuitetu, ali je prednost brzina pristupa (i brzina pristupa nekom byteu) jer se lako izračuna). Stoga se koristi za read-only datoteke koje se ne mijenjaju često, kao što su slike.

2. Vezane liste - svaki blok datoteke pokazuje na svog sljedbenika i prethodnika (pokazivači). Stoga s datoteka lako povećava i smanjuje, ali je mana brzine pristupa zbog premještavanja (kao i pristup određenom byteu).

3. Indeksna alokacija - postoji tablica s parovima blok→blok_u_memoriji (npr. blok 1 u memoriji je blok 58). Tako definiramo preslikavanje. Sama tablica se također nalazi u nekom bloku. Prednost pristupa je direktan pristup pa time i lako čitanje bajtova, ali je mana što kod prevelikih datoteka ne može cijela tablica stat u 1 blok.

Rješenje UNIXA:
inode - tablica koja sadrži podatke o datoteci i podatke o smještaju (15 pokazivača na blokove). Prvih 12 pokazivača su direktni → Direct 0 → Direct 11
pokazivači na blokove; sljedeći je single indirect koji pokazuje na blok sa 255 pokazivača na blokove podataka, potom double indirect koji pokazuje blok sa 255 pokazivača od kojih svaki tej pokazuje na blok od 255 pokazivača, a svaki od tih pokazivača na jedan blok podataka, ukupno 65 025 blokova, potom triple indirect koji pokazuje na blok sa 255 pokazivača od kojih svaki pokazuje na blok od 255 pokazivača, a opet svaki na blok od 255 pokazivača, dobiti pokazivači pok na blokove podataka.

Ukupno je to 255^3 pokazivača. Ovakvim pristupom može pokazivati na $12+255+255^2+255^3 \approx 16 \cdot 10^6$ blokova, a ako svaki blok ima 1kB, onda datoteka može imati 16GB.

inode: