ՍԵՐԳԵՅ ՎԱՐՏԱՆՈՎ

ՕՊԵՐԱՑԻՈՆ ՀԱՄԱԿԱՐԳԵՐԻ ԿԱՌՈՒՑՄԱՆ ՀԻՄՈՒՆՔՆԵՐԸ

Ուսումնամեթոդական ձեռնարկ

ԵՐԵՎԱՆ - 2026

Օպերացիոն համակարգերի կառուցման հիմունքները

Ա.Ռ. Վարտանով

Բովանդակություն

ՆԵՐԱԾՈՒԹՅՈՒՆ 3
Գլուխ 1. Ունիվերսալ հաշվիչի ճարտարապետություն 7
1.1. Հաշվիչներ և դրանց կառուցվածքը 7
1.2. Հասցեագրում 10
1.3. Ընդհատումներ 11
1.4. Պրոցեսորների քեշ հիշողություն 16
Գլուխ 2. ՕՀ-եր, դրանց ստեղծման պատճառներ և տեսակներ 18
2.1. Բազմախնդրության տրամաբանություն 20
2.2. ՕՀ-երի ծառայություններ 24
2.3. ՕՀ-երի պահանջները 26
2.4. ՕՀ-երի բեռնում 28
2.5. Առաջադրանքներ և ենթաառաջադրանքներ, գործընթացներ և հոսքեր 29
2.6. Ֆոնային և ռեզիդենտ ծրագրեր 30
Գլուխ 3. Առաջադրանքներ, հանձնարարություններ և դրանց կառավարում 31
3.1. Առաջադրանքների և հանձնարարությունների մեկնարկ 31
3.2. Բեռնիչ և առաջադրանքների մեկնարկիչ 32
3.3. Առաջադրանքների կառավարման բլոկներ (հավելվածներ և հոսքեր) 33
3.4. Առաջադրանքների և հոսքերի փոխարկում 38
3.5. Ծրագրային առաջադրանքներ և օպերացիոն համակարգի ռեսուրսների հարցումներ 39
Գլուխ 4. Հիշողություն և դրա կառավարում 40
4.1. Հիշողության ձևավորումը հաշվարկիչներում և հավելվածներում 40
4.2. Օվերլեյներ և սվոփինգ 42
4.3. Վիրտուալ հիշողության կազմակերպում 44
4.4. Վիրտուալ մեքենաներ և վիրտուալ օպերացիոն համակարգեր 46
Գլուխ 5. Օպերացիոն համակարգերի ծրագրերի մշակման առանձնահատկությունները 47
5.1. Ռեենտերական ծրագրեր 47
5.2. Ծրագրային մոդուլների պատճեններ 48
5.3. Մյութեքսներ 48
Գլուխ 6. Ներմուծում-ելք: Ֆայլային համակարգեր: Մուտքի մեթոդներ 50
6.1. Ներմուծման-ելքային սարքերն ու մուտքի մեթոդները 50
6.2. Ֆայլային կառուցվածքներ 52
6.3. Բուֆերացված ներմուծում-ելք 53
Գլուխ 7. Բազմամեքենայական և բազմապրոցեսորային համալիրներ 56
7.1. Բազմամեքենայական և բազմապրոցեսորային համակարգերի ձևավորում 56
7.2. Կառավարման կառուցվածքներ բազմամեքենայական համալիրներում և օպերացիոն համակարգի առաջադրանքներ 59
7.3. Ընդհանուր հիշողության կազմակերպում բազմամեքենայական համալիրներում 62
ԵԶՐԱԿԱՑՈՒԹՅՈՒՆ 64
ԳՐԱԿԱՆՈՒԹՅՈՒՆ. 65
ՀԱՎԵԼՎԱԾ 1. ՏԵՐՄԻՆԱԲԱՆՈՒԹՅՈՒՆ 66

ՆԵՐԱԾՈՒԹՅՈՒՆ

Այս ուղեցույցը նվիրված է օպերացիոն համակարգերի (ՕՀ) դասընթացի նկարագրությանը, որը սահմանում է ՕՀ-ի խնդիրները, ՕՀ կառուցման սկզբունքները և հաշվողական համալիրների ռեսուրսների կառավարման խնդիրների լուծման մեթոդները:

Դասընթացը սկսվում է հաշվողական տեխնիկայի ընկալման ի հայտ գալու և զարգացման նախապատմությամբ, հաշվարկների և հաշվիչների տարբեր մոդելների ստեղծման տեսական և գործնական հիմքերի ձևավորմամբ:

Հաշվարկները և հաշվողական գիտությունները դարավոր պատմություն ունեն, եթե ելնենք նպատակային և լիարժեք հետազոտությունների սկզբից: Նախևառաջ, բազմաթիվ ուշագրավ մաթեմատիկոսների կոլեկտիվ քննարկումներից «հաշվողական ալգորիթմ» իմաստալից հասկացության և դրա բնորոշ հատկությունների մասին:

Որպես արդյունք, ձևակերպվեց այսպես կոչված Չորչի թեզիսը (երբեմն՝ Չորչի-Թյուրինգի թեզիս), որն ընդլայնվեց Ս.Քլինիի կողմից այն պնդման համար, որ ալգորիթմների միջոցով հաշվարկված բոլոր մասնակի ֆունկցիաները համընկնում են մասնակի ռեկուրսիվ ֆունկցիաների դասի հետ, որը համարժեք է բոլոր Թյուրինգի մեքենաների դասին:

Չորչի թեզիսը մինչ օրս հիմք է հանդիսանում հաշվարկների և հաշվողական տեխնիկայի ոլորտում ըմբռնման և հետազոտության համար:

1941 թ.-ին գերմանացի ինժեներ Կոնրադ Զուզեն (Zuse) ստեղծեց աշխարհում առաջին իսկապես աշխատող համակարգիչը՝ Z3 (Z-ն, հավանաբար, ազգանվան առաջին տառից է, չնայած դա գովազդված չէր), իսկ մի փոքր ավելի ուշ՝ պատմության մեջ առաջին բարձր մակարդակի լեզուն՝ Plankalkül (Z4 հաշվիչի համար), որի անունը կարելի է մեկնաբանել որպես «պլանավորված հաշվարկ»:

1940-ականներին սկսվեց արագ զարգացումը ինչպես կազմակերպության տեսական մոդելների, ալգորիթմացման և հաշվարկների ծրագրավորման, այնպես էլ հիմնականում լաբորատոր հաշվարկների ստեղծման համար: Չնայած, հենց Z4-ն է դարձել աշխարհում առաջին կոմերցիոն համակարգիչը, որը վաճառվել է 1950թ.-ին: Այսպես թե այնպես, հաշվիչները մնացին բավականին եզակի սարքեր:

Հետազոտություններն ու մշակումները հիմնականում իրականացվել են հիմնականում նոր տարրական բազայի ստեղծման, ինչպես նաև հաշվարկների տեսական մոդելների իրականացման ուղղությամբ, ինչպիսիք են՝

- հաջորդական հաշվողական մեքենաներ (այսպես կոչված «ֆոն Նոյմանի մոդել»),
- վեկտորային հաշվարկներ,
- վեկտորային-կոնվեյերային հաշվարկներ,
- ասոցիատիվ հաշվարկներ,
- մատրիցային հաշվարկներ,
- հոսքային հաշվարկներ,

որոնք ստացան մեկ այլ բաժանում ըստ մոտեցումների և ճարտարապետության՝ հաջորդական հաշվարկներ և հաշվիչներ, և զուգահեռ հաշվարկներ և հաշվիչներ:

Գրեթե բոլոր թվարկված մոդելները սկսեցին ներառել բազմամշակման տարրեր, կամ գոնե ներդրվեցին մասնագիտացված ենթամշակիչներ, որոնք օգտագործվում էին առանձին գործառույթների համար (սովորաբար մուտքային-ելքային գործողություններ իրականացնելու համար):

Միևնույն ժամանակ, հաշվողական կայանքները սկսեցին բաժանվել ըստ կազմակերպության սկզբունքների՝ միապրոցեսորային, բազմապրոցեսորային և բազմամեքենայական հաշվողական համալիրներ:

1960-ականների երկրորդ կեսին ձևավորվեց զուգահեռ հաշվարկների նկարագրության հիմնական տեսական հիմքը՝ սկսած Կարպի և Միլլերի աշխատանքից [2], որը խթան հանդիսացավ զուգահեռ հաշվարկման մոդելների տարբեր էվոլյուցիաների զարգացման համար: Միևնույն ժամանակ, Ֆլինը (Flynn) առաջարկեց [3], որը դարձավ դասական, հաշվիչների և հաշվարկների դասակարգում, որի չորս դասերից միայն մեկը վերաբերում էր այն ժամանակ ավանդական հաջորդական հաշվարկներին:

60-ականների նշանակալի իրադարձությունը IBM-360 [4] ճարտարապետության ի հայտ գալն էր, որը, ըստ էության, դարձավ միապրոցեսորային համակարգիչների ստանդարտը: Հետագա տարիներին 360/370 [5] համակարգերը սկսեցին ձեռք բերել ծայրամասային սարքերի լայն տեսականի, որոնք ապահովում էին տեղեկատվության մուտքագրում, ելքագրում և մուտքագրում-ելքում: Դրանց թվում էին ցուցադրման կայաններ (սկզբում ավելացվեցին տառատեսակային, ավելի ուշ՝ գրաֆիկական), բոլոր տեսակի տպիչներ, պլոտտերներ, սկաներներ և այլն: Սկավառակային սարքերի հնարավորությունները աճեցին՝ աստիճանաբար հետ մղելով մագնիսական ժապավենները և այլ թղթերի և այլ կրիչների վրա:

Ավելացվել են նաև հեռահաղորդակցության միջոցներ, որոնք թույլ են տալիս հեռավոր հեռավորությունների վրա շփվել հաշվիչների հետ: Միաժամանակ սկսեցին հայտնվել հաշվիչների բազմապրոցեսորային մոդելներ:

Ըստ էության, հենց IBM-360/370-ի [6] հայտնվելը ճարտարապետության առումով, միջմեքենաների և միջպրոցեսորների փոխգործակցության տարբեր տարբերակներ, բազմաբնույթ առաջադրանքների լուծումների կազմակերպում, լուծումների ստանդարտացում և այլն, ներառյալ մշակումների մշակույթի և սկզբունքների ստեղծումը ինչպես էլեկտրոնիկայի, այնպես էլ ծրագրային ապահովման առումով, դարձել են հղում: Այստեղ ձևակերպվել են նաև օպերացիոն համակարգերի (ՕՀ) [7, 11] հիմնական պահանջները, ինչպես նաև դրանց կառուցման սկզբունքները:

Այս դասընթացում մենք չենք կենտրոնանում որոշակի օպերացիոն համակարգերի առանձնահատկությունների վրա, այլ նկարագրում ենք որոշակի բաղադրիչների համար ամենատարածված լուծումները:

Դասընթացի ընթացքում կներկայացվեն.

- հաշվիչների կառուցման սկզբունքները,
- պրոցեսորների աշխատանքի սկզբունքները, մեքենայական կոդերի մշակումը երկհասցե, եռհասցե և բազմահասցե մաթեմատիկական մեքենաների հիման վրա,
- ընդհատման համակարգերի կազմակերպման կարևորությունն ու սկզբունքները, ինչպես նաև սինխրոն և ասինխրոն իրադարձությունների հետ աշխատելու հնարավորությունները ընդհանուր առմամբ,
- օպերացիոն համակարգերի ստեղծման հիմնավորումը, դրանց ընդհանուր կառուցվածքը, ֆունկցիոնալ խնդիրները,
- ՕՀ միջուկի առաջադրանքները և դրանց ծառայությունները,
- ՕՀ-ի բեռնման սխեմաներ,
- ՕՀ-ի համար առաջադրանքների հերթեր ձևավորելու ընտրանքներ,
- առաջադրանքների բեռնման և դրանց մեկնարկի սխեմաներ,

Բացի այդ, դասընթացը ուսումնասիրում է բազմախնդրության կազմակերպման հիմունքները, դրա կազմակերպման տարբեր տարբերակները (ինչպիսիք են իրական զուգահեռ համակարգերը, որոնք հիմնված են բազմամշակման վրա, ժամանակի քվանտացված համակարգեր, I/O գործողությունների և իրադարձությունների հիման վրա զուգահեռացման հնարավորություններ և այլն):

Դասընթացում ներկայացված են հիշողությունը ՕՀ-ի հիշողության և կիրառական հիշողության բաժանման եղանակները, ՕՀ-ի և հավելվածների կողմից պահանջվող հիշողության գրանցման ձևերը:

Դիտարկվում են փոփոխականների միավորումների (կանչի կույտեր, ծրագրերի դինամիկ հիշողություն և առանձին ենթաառաջադրանքներ), օպերատիվ հիշողության դինամիկ պահանջվող տարրերի ձևավորման հնարավորությունները՝ առաջադրանքների և ենթաառաջադրանքների բեռնման, մեկնարկի և կատարման պահերին:

ՕՀ-երի ծրագրային մոդուլների առկայության համատեքստում, որոնք սպասարկում են բազմաթիվ հաշվարկային գործընթացներ, դիտարկվում են ծրագրերի մշակման գործընթացում վերաբաշխման խնդիրները, ծրագրային մոդուլների պատճենների դինամիկ բեռնման եղանակները, ներառյալ դինամիկ բեռնվող ծրագրային գրադարանները:

Դասընթացը ներկայացնում է I/O սարքերի հետ փոխազդեցությունների կազմակերպման տարբեր եղանակներ ճարտարապետություններում, որոնք թույլ են տալիս.

- տվյալների ուղղակի տեղափոխում RAM և CPU-ի վրա հիմնված I/O ճարտարապետություններ,
- I/O բուֆերացմամբ և առանց դրա,
- I/O ընդհատման համակարգի բազմազանություն,
- հաշվողական ցանցերի հետ փոխգործակցության սկզբունքները:

Դասընթացի վերջին փուլը բազմամեքենայական համակարգերի, մասնավորապես կլաստերների ներկայացումն է: Հաշվի են առնվում միջմեքենայական կապերի մեթոդները, ինչպես նաև բազմամեքենայական համակարգերում կառավարման խնդիրները:

Դասընթացի անցկացման համար ստեղծվել է ցուցադրական ծրագիր՝ վերը թվարկված բոլոր թեմաները ցուցադրելու հնարավորություններով և պրոցեսորների անհատական սխեմաների իմիտացիայով, ընդհատումների մշակմամբ, բազմախնդրության տարբերակներով և այլն:

*) Քանի որ պատկերները ցուցադրական ծրագրի էկրանի նկարներ են, որը նմանակում է համակարգիչների և օպերացիոն համակարգի վարքագիծը: Ծրագիրը հեղինակի կողմից գրվել է Caper զուգահեռ ծրագրավորման լեզվով:

Գլուխ 1. Ունիվերսալ հաշվիչի ճարտարապետություն

Որպես գլխի նախաբան, մենք ձևակերպում ենք մի քանի հիմնարար թեզիսներ:

Հաշվիչի ճարտարապետությունը կանխորոշում է դրա օպերացիոն համակարգի ճարտարապետությունը:

Հաշվիչի ճարտարապետությունը կանխորոշում է դրա ծրագրավորման հայեցակարգը:

Ժամանակակից ունիվերսալ համակարգիչների ճարտարապետությունները տարբեր աստիճաններով ժառանգում են այսպես կոչված ֆոն Նեյմանի կառուցվածքը, որը ենթադրում է սարքում երեք բաղադրիչների առկայություն՝ պրոցեսոր, օպերատիվ հիշողություն և տեղեկատվության I/O սարք: Այս դեպքում առավել հաճախ հիշատակվում է ծրագրային կոդերը մեքենայական լեզվով RAM-ում, ինչպես նաև տվյալները տեղադրելու սկզբունքը: Այս առումով նշենք, որ ծրագրային կոդը կարող է փոփոխվել հաշվարկների ընթացքում՝ ինչպես դրսից, այնպես էլ ծրագրի կողմից (տե՛ս ստորև՝ ծրագրերի փոփոխման և ինքնափոփոխման մասին): Այնուամենայնիվ, որոշակի ճարտարապետություններում կա պաշտպանված բաժանում ծրագրային հիշողության և տվյալների հիշողության:

Ժամանակակից համակարգիչը, ցանկացած հաշվողական կայանք, տարբեր ռեսուրսների ամբողջություն է, որոնք պետք է կառավարվեն:

Համակարգիչների վրա կատարվող ծրագրերի համար կան ռեսուրսներին մուտք գործելու և դրանք օգտագործելու խնդիրներ, որոնք պահանջում են կանոնավոր ծրագրավորում: Միևնույն ժամանակ, հաշվողական միջավայրում բազմաթիվ հաշվարկներ կարող են միաժամանակ իրականացվել և կատարվել: Հետևաբար, ռեսուրսների համար մրցակցության խնդիրներ են առաջանում, երբ դրանք հավաքագրվում, օգտագործվում և ազատվում են:

Օպերացիոն համակարգերը ծրագրային և ապարատային համալիրներ են, որոնք նախատեսված են օգտագործողի (կիրառական) ծրագրերը բեռնելու և բեռնաթափելու, դրանց կատարումը սկսելու, հաշվարկային կայանքում տեղի ունեցող իրադարձությունների ամբողջ շրջանակը վերահսկելու, ինչպես նաև հաշվիչների, այդ թվում՝ պրոցեսորների ռեսուրսները վերահսկելու և կառավարելու համար:

Օպերացիոն համակարգը և դրա հիմնական բաղադրիչները ռեսուրսների հետ աշխատում են Մոնիտորի սկզբունքով, այսինքն՝ ցանկացած ռեսուրսի մուտքը և մանիպուլյացիան իրականացվում է համապատասխան ՕՀ-ի բաղադրիչների միջոցով՝ առանց ռեսուրսին անկախ և անմիջական մուտքի:

[1]) Համատեղ ռեսուրսների կառավարման մեթոդների ընդունված հայեցակարգային բաժանումը՝ սեմաֆորներ, մոնիտորներ, սենտինելներ (ըստ [1]-ի):

1.1. Հաշվիչներ և դրանց կառուցվածքը

Յուրաքանչյուր հաշվողական համալիր, սկսած առանձին համակարգչից և վերջացրած բարդ բազմապրոցեսորային և բազմամեքենայական հաշվիչներով, հանդիսանում է.

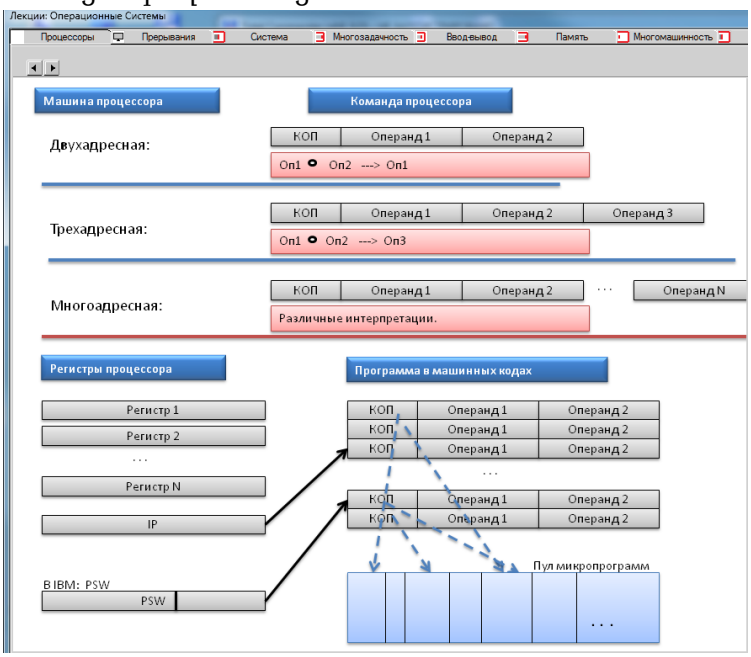
- համակարգ, որը կառուցված է հաշվարկների ընկալման և դրանց իրականացման տեսական մոտեցման հիման վրա,
- համակարգ, որը պարունակում է էլեկտրոնային և ծրագրային բաղադրիչներ, որոնք կազմում են ռեսուրսների ամբողջություն,
- համակարգ, որի կառուցվածքում կան հիմնական տարրերը՝ պրոցեսորներ, օպերատիվ հիշողության (RAM) կրիչներ և հիմնական կապի գործիքներ (համակարգային շինաներ և հատուկ շինաներ պրոցեսորների և RAM-ի հետ միանալու և փոխազդելու համար), ինչպես նաև երկրորդական գործիքներ, որոնք հաճախ անվանում են ծայրամասային սարքեր՝ տեղեկատվության մուտքագրում և ելք, դրա պահպանում և փոխադրում:

Համակարգերի և դրանց առանձին տարրերի կառուցման տեսական սկզբունքները կոչվում են ճարտարապետություններ (պրոցեսորներ, օպերատիվ հիշողություն, ծայրամասային սարքեր և այլն):

Որոշ ճարտարապետություններում պրոցեսորները, օպերատիվ հիշողությունը, մուտքային/ելքային կառավարման տարրերը կարող են առանձնացվել ըստ տեսակի, նպատակի, առաջադրանքների: Օրինակ, DSP պրոցեսորները [10] (ազդանշանների մշակման պրոցեսորներ) ունեն ստատիկ և դինամիկ հիշողություն, իսկ պրոցեսորն ինքնին մասնագիտացված ենթապրոցեսորների ամբողջության ինտեգրատոր է:

Ունիվերսալ պրոցեսորները, որոնք ծրագրերի կատարման հիմնական սարքերն են, կոչվում են կենտրոնական պրոցեսորներ (անգլալեզու տերմինաբանության մեջ՝ CPU - Կենտրոնական մշակման միավոր):

Նկ. 1-ում ներկայացված են պրոցեսորային մեքենաների տարբերակները՝ երկհասցե, ամենատարածվածը, եռհասցե և բազմահասցե:



Նկար 1. Պրոցեսորի հրահանգների ճարտարապետություն՝ երկու, երեք և բազմահասցե հրահանգներ

Երկհասցե պրոցեսորներն աշխատում են հետևյալ սխեմայի համաձայն.

1. Պրոցեսորի ներքին հիշողության մեջ ձևավորվում են արժեքներ և (կամ) ցուցումներ առաջին և երկրորդ օպերանդների համար:
2. ԿՕՊ (գործողության կոդ) – որպես կանոն, գործողությունը կատարող միկրոծրագրին միկրոծրագրերով հիշողության մեջ բաժնի համապատասխան կոդի նշում:
3. Միկրոծրագրի գործողության արդյունքը սովորաբար տեղադրվում է առաջին օպերանդում:

Եռահասցե պրոցեսորները գործում են գրեթե նույն կերպ, ինչ երկհասցե պրոցեսորները, բայց գործողության արդյունքը տեղադրվում է երրորդ օպերանդի կողմից նշված տեղում:

Բազմահասցե պրոցեսորները կարող են գործել տարբեր ձևերով՝ կախված հաշվողական ճարտարապետության ընդհանուր մոտեցումից: Մասնավորապես, բազմակի նշանակման հնարավորություններով ճարտարապետության դեպքում, երրորդ և հաջորդ օպերանդները կարող են պարունակել ընթացիկ գործողության արդյունքը: *)

Մեկ այլ տարբերակ կարող է լինել Ֆլինի դասակարգման [2] համաձայն SIMD (Single Instruction Multiple Data) սխեմայի շրջանակներում կատարումը, երբ մեկ գործողություն կիրառվում է օպերանդների ամբողջ բազմության վրա:

Կան ճարտարապետություններ, որոնք հնարավորություն ունեն գրելու բազմաթիվ հրամաններ, որոնք կարող են կատարվել մեկ պրոցեսորային ցիկլում: Առավել հայտնի են

գերսկալյար և VLIW ճարտարապետությունները, որոնք վերաբերում են զուգահեռ հաշվիչների և հաշվարկների դասընթացին:

Այստեղ, օպերացիոն համակարգերի կառուցվածքները և դրանց գործառույթները ցուցադրելու նպատակով, մենք կսահմանափակվենք ավանդական միապրոցեսորային և բազմապրոցեսորային ֆոն Նոյմանի ճարտարապետություններով:

Վաղ (մինչև այսպես կոչված քեշերի ի հայտ գալը) պրոցեսորները կատարում են հետևյալ գործողությունները յուրաքանչյուր տարրական քայլում (որը կարող է բաղկացած լինել մի քանի ցիկլերից).

1. Ֆիքսված երկարության տվյալների մեկ կամ ավելի նմուշառում կատարվում է RAM - ից (հետագայում՝ պրոցեսորի ներքին պահոցներից) դեպի պրոցեսորի ռեգիստր: Դա տվյալների ընտրության ֆիքսված երկարությունն է (ուղղակիորեն կախված է պրոցեսորն ու հիշողությունը միացնող ավտոբուսի վրա փոխանցվող բիթերի քանակից), որը մեկ քայլով դարձավ ծրագրավորման լեզուներում "բառի" (word) տիպի նախատիպ (հիշեցնեմ, որ բառը - int նշանով է): Միևնույն ժամանակ, IBM Հաշվիչների վաղ տարբերակներում, 4 բայթ բառի երկարությամբ, կային հրամաններ ինչպես 6, այնպես էլ 8 բայթ, ինչը պահանջում էր անցկացնել առնվազն երկու նմուշ:
2. Պրոցեսորային ճարտարապետությունների մեծ մասն ունի ռեգիստր (Instruction Pointer – IP, IBM-ում այն PSW կառավարման ռեգիստրի մաս է կազմում), որը պահպանում է հաջորդ հրամանի հասցեն, որը ձևավորվում է գործողության կոդի և համապատասխան օպերանդների երկարությունների հիման վրա: Այսինքն՝ ընթացիկ հրամանի հասցեն նրա երկարությամբ մեծացնելով՝ ձևավորվում է հաջորդ հրամանի հասցեն:
3. Հաջորդը, պրոցեսորն իրականացնում է օպերանդների պատրաստում : Միևնույն ժամանակ, տվյալներն արդեն կարող են տեղակայվել պրոցեսորի գրանցամատյաններում, այնուհետև ամեն ինչ պարզեցվում է, կամ տեղակայվել RAM-ում, ինչը պրոցեսորից կպահանջի դրանք մղել ներքին ռեգիստրներ: Եվ կրկին, երկարությունը նմուշային բառի չափն է:
4. Միայն տվյալների նախապատրաստումից և անհրաժեշտ տեղադրումից հետո է իրականացվում գործողության կոդին համապատասխանող միկրոծրագրի մեկնարկը: Դրա ավարտից հետո արդյունքը, որպես կանոն, տեղադրվում է երկհասցե մեքենաների համար առաջին օպերանդի հասցեում:
5. Անցում է կատարվում IP բովանդակության հաջորդ գործողությանը:

Նշենք, որ բոլոր ունիվերսալ պրոցեսորներին բնորոշ իտերացիայի կազմակերպման գործիքները (պարզ ասած ՝ հրամանի հասցեին պայմանական կամ անվերապահ անցման հրամանները) մեկնաբանվում են որպես հաջորդ հրամանի նոր հասցեի տեղադրում IP-ում կամ PSW-ում:

Ըստ էության, պրոցեսորը ընտրված հրամանների թարգմանիչն է (տե՛ս նկ. 1-ը համապատասխան միկրոծրագրային գործողության կոդերին մատնանշող սլաքների համար):

Ընդգծենք, որ համակարգիչների նախագծման մոտեցումների հիմքը բուլյան հանրահաշիվն է, իսկ միկրոծրագրերը կազմվում են արդեն ձևավորված ավանդույթի համաձայն՝ OR, AND, NOT և ավելացված XOR գործողությունների հիման վրա:

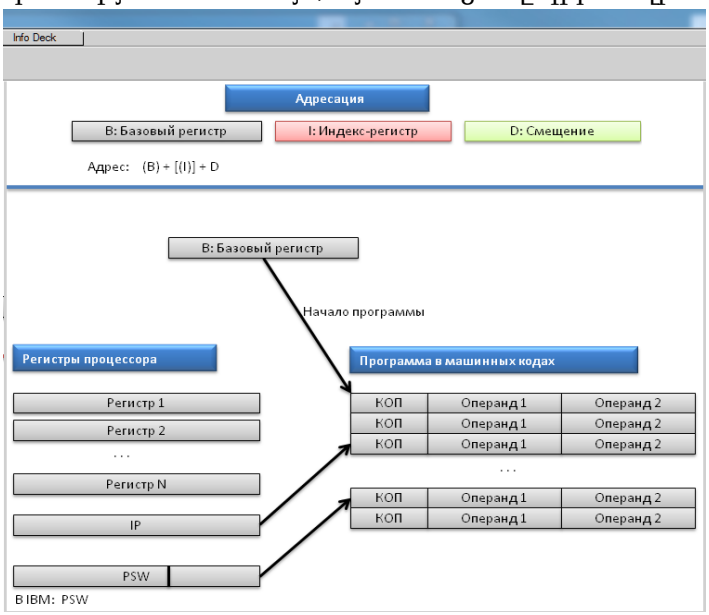
*) PSW - Ծրագրի կարգավիճակի բառ, որն ընդունվել է IBM-360-ում և առկա է ժամանակակից ESA/390-ում (IBM-390): Առավել ուշագրավն այն է, որ IBM-ը պահպանել է իր մեքենաների ճարտարապետական առանձնահատկությունները՝ ըստ էության, կրկնապատկելով միայն ընդհանուր օգտագործման գրանցամատյանների և կառավարման գրանցամատյանների չափերը (ինչը խոսում է ճարտարապետության խորը մտածվածության մասին դեռևս 1960-ականներին՝ փոխելով միայն տարրական և քանակական հիմքերը մշակման ընթացքում): Այսպիսով, եթե IBM-360-ը հաջորդ հրամանի հասցեի համար օգտագործել է ութից վերջին երեք բայթերը, ապա ժամանակակից մոդելներն ունեն 8 ընդհանուր գրանցամատյան (64 բիթ), իսկ PSW-ն z/Architecture ռեժիմի համար՝ 16 բայթ (128 բիթ): Հաջորդ հրահանգի հասցեն զբաղեցնում է վերջին 4

բայթերը: Ամեն դեպքում, լինի դա առանձին գրանցամատյան, թե պրոցեսորի մեկ այլ ճարտարապետական տարրի մաս, մենք հաջորդ հրահանգի ցուցիչը կնշանակենք որպես IP:

*) Դասական ծրագրային սխեմատոլոգիայում բազմակի վերագրումը ներկայացված է $y_1, y_2, \ldots, y_n := f(x_1, x_2, \ldots, x_m)$ արտահայտությամբ, այսինքն՝ f-ի արդյունքը m պարամետրերի նկատմամբ տեղադրվում է n փոփոխականներում (տեղերում) մեկ հաշվարկային քայլով:

1.2. Հասցեագրում

Անհրաժեշտ է հասկանալ այնպիսի հասկացություններ, ինչպիսիք են բացարձակ և հարաբերական հասցեավորումը, հիմնականում ծրագրերի (տե՛ս նկ. 2): Ամենից հաճախ այս հասկացություններն օգտագործվում են ծրագրերը և տվյալները RAM-ում տեղադրելիս, չնայած հասցեավորման տրամաբանության համաձայն, այս մոտեցումը կիրառելի է նաև այլ դեպքերում:



Նկար 2. Հասցեավորման համակարգ և փոխազդեցություն պրոցեսորի ռեգիստրների հետ:

Բացարձակ հասցեն բայթերի հաջորդականության մեջ 0-ից մինչև M-1 միջակայքում գտնվող բայթերի թիվն է, որտեղ M-ը հիշողության ծավալն է:

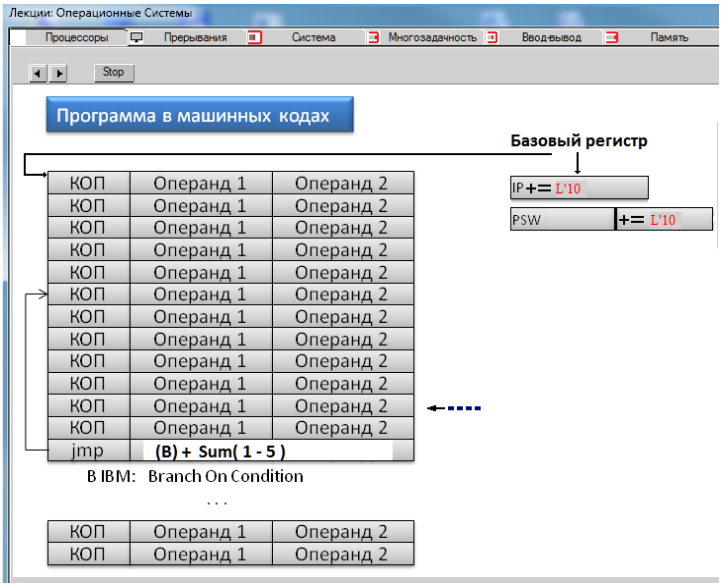
Հարաբերական հասցեն ձևավորվում է երկու ռեգիստրների և հաստատունի պարունակությունից՝ (B) + (I) + D, որտեղ B - ն այսպես կոչված բազային ռեգիստր է, որը պարունակում է բացարձակ հասցե, I՝ ինդեքս ռեգիստր, D՝ դրական թիվ է, որը կոչվում է օֆսեթ:

Որպես կանոն, բազային ռեգիստրի և օֆսեթի հասկացությունները առկա են բոլոր համակարգիչներում, երբ ձևավորվում է որևէ ծրագրի (կիրառական ծրագրի) համար հատկացված հիշողության մեջ գտնվող հասցե: Այս դեպքում, բազային ռեգիստրը, որպես կանոն, պարունակում է ծրագրի սկզբի բացարձակ հասցեն, ինդեքսային ռեգիստրները օգտագործվում են նույն IBM հաշվիչներում՝ ծրագրերի ներսում հասցեների դինամիկ փոփոխությունների ճկունության համար, այսինքն՝ այն դինամիկ ձևով նշում է բազայի նկատմամբ օֆսեթը, մինչդեռ բացարձակ օֆսեթը ցանկացած հասցեից հաստատուն օֆսեթի թիվն է: Որոշ պրոցեսորներ կարող են օգտագործել հասցեագրման կրճատ ձև՝ առանց ինդեքսային ռեգիստրի՝ A = (B) + D:

Հատուկ ուշադրություն պետք է դարձնել հասցեի համապատասխանեցման հայեցակարգին նմուշի բառի երկարությանը, քանի որ պրոցեսորը կարող է տվյալներ ընտրել RAM-ից միայն այն հասցեների դեպքում, որոնք բառի երկարության բազմապատիկներ են:

Ծրագրի աղբյուրի կոդերի կազմման պարամետրերում կա նաև հավասարեցման ցուցում (մասնավորապես՝ Visual Studio-ում), ինչը հաշվի է առնվում OOP-ում տվյալների կառուցվածքներ և օբյեկտներ ստեղծելիս՝ իրենց անդամներին տեղակայելիս և հասցեագրելիս:

Նկ. 3-ը ցույց է տալիս IP-ում հասցեների ձևավորումը: Հիմնական ռեգիստրը ցույց է տալիս ծրագրի սկիզբը (ընդհանուր առմամբ, հրամանների ցանկացած ընտրված հաջորդականություն): IP-ում և PSW-ում հասցեն ձևավորվում է հիմնական ռեգիստրի պարունակությունից, որը գումարվում է 10 հրամանների երկարությանը:



Նկար 3. Հասցեի հաշվարկով ցատկի հրահանգի (jmp) իրականացում:

Պրոցեսորի էմուլյատորի աջ կողմում գտնվող սլաքը ցույց է տալիս ընթացիկ գործարկվող հրամանը: Օպերանդներում "jmp" հրամանն ունի հարաբերական հասցե, որը ձևավորվում է բազային ռեգիստրի պարունակության և առաջին հինգ հրամանների երկարությունների գումարի հանրագումարով և հանդիսանում է անցման կետի ցուցում: Անցումային հրամանները կատարում են IP ռեգիստրի պարամետրերում հասցեների տեղադրում: Համապատասխանաբար, պրոցեսորի կողմից կատարվող հաջորդ հրամանը կլինի IP-ում նշվածը:

Ըստ էության, պրոցեսորը պոտենցիալ անվերջ աշխատող (գոնե այնքան ժամանակ, քանի դեռ կա սնուցման աղբյուր և այն գործունակ է) սարք է RAM-ից հրամանների քայլ առ քայլ մեկնաբանման համար (բացի արդյունավետությունից, ոչինչ չի խանգարում պրոցեսոր պատրաստել՝ հրամաններ կարդալու և դրանք որոշ ռեգիստրներում կատարելու համար մագնիսական սկավառակից, ժապավենից և այլն, այսինքն՝ այն վայրից, որտեղ հրամանները կգրվեն և հասանելի կլինեն կարդալու համար՝ նույնիսկ Ինկերի հանգույցավորված «գրառումից» (կիպու), եթե միայն սարք լիներ): Նույնիսկ պրոցեսորի կանգնեցումը հրամանով, որը բնորոշ է IBM պրոցեսորներին, միայն անցում է վիճակի, բայց ոչ մի կերպ պրոցեսորի դադարեցում: Ժամանակակից պրոցեսորների մեծ մասը գործարկում է այսպես կոչված Idle պրոցեսը՝ ՕՀ-ի միջուկի բաղադրիչ, որը կատարվում է բոլոր չզբաղեցված ենթապրոցեսորների (միջուկների) վրա՝ այն օգտագործվում է էներգիա խնայելու համար:

1.3. Ընդհատումներ

Պրոցեսորների կառուցման և գործունեության սկզբունքները համակարգչային կառավարման համակարգերի կառուցման էական հիմքերից մեկն են, բայց ոչ միակը:

Անվերապահ և որոշիչ նշանակություն ունեն այնպիսի բաղադրիչներ, ինչպիսիք են RAM սարքը, արտաքին I/O սարքերի հետ փոխազդեցության սարքը, լրացուցիչ սարքերի առկայությունը, ինչպիսիք են ապարատային ժամանակաչափերը և մի քանի ուրիշներ:

Նման բաղադրիչները ներառում են ապարատային ընդհատումների համակարգ, որն ապահովում է պրոցեսորի անհապաղ արձագանքը դրա ներսում և դրսում տեղի ունեցող իրադարձություններին: Ըստ էության, դա պրոցեսորի արձագանքման համակարգն է մի քանի

տրիգերների հավաքածուներից ստացված ազդանշաններին, որոնք ազդանշաններ են տալիս ընդհատվող իրադարձությունների մասին:

Եվ կրկին, այստեղ «միտումը որոշող» ուժը IBM հաշվիչներն էին, որոնց մշակողները որպես ընդհատումների աղբյուրներ նույնականացրին հետևյալ խմբերը.

- ինքնին պրոցեսորի ապարատային սխալները,
- RAM-ի ապարատային սխալներ (սովորաբար հիմնված են բայթերի հավասարության ստուգման կամ հիշողության կառավարման սխալների վրա),
- ներքին պրոցեսորի իրադարձության ստեղծում՝ օգտագործելով այսպես կոչված վերահսկիչի հրամանները (Supervisor Call, SVC *)),
- ծրագրային ապահովման ընդհատումներ ծրագրային սխալների պատճառով (օրինակ՝ զրոյի բաժանում, գերբեռնվածություն, դիմում «ուրիշի հիշողությանը» և այլն),
- I/O իրադարձություններ (թերևս իրադարձությունների ամենատարածված համակարգը՝ սարքերի, ինչպես նաև մուտքային/ելքային մեթոդների մեծ թվով տարբերությունների պատճառով),
- իրադարձություններ՝ սկսած ապարատային ժամանակաչափերից, բոլոր տեսակի «հեռակառավարման կոճակներից» և այլն, որոնք կոչվում են արտաքին ընդհատումներ:

Իհարկե, հնարավոր են ընդհատումները խմբերի բաժանելու այլ տարբերակներ, բայց, այս կամ այն կերպ, դրանք կարտացոլեն ընդհատումների ընդհանուր ըմբռնումը:

Ընդհատման մեխանիզմները չափազանց կարևոր են իրադարձություններին արագ և արդյունավետ արձագանքելու համար: Եվ ահա թե ինչու:

Սարքավորումների և ծրագրային ապահովման բազմազանությունը պահանջում է կառավարում: Ցանկացած կառավարում սկսվում է որոշակի սարքի վիճակի (առաջին հերթին՝ գործունակության) որոշմամբ, որոշակի գործողություններ կատարելու հնարավորություններով և այլն:

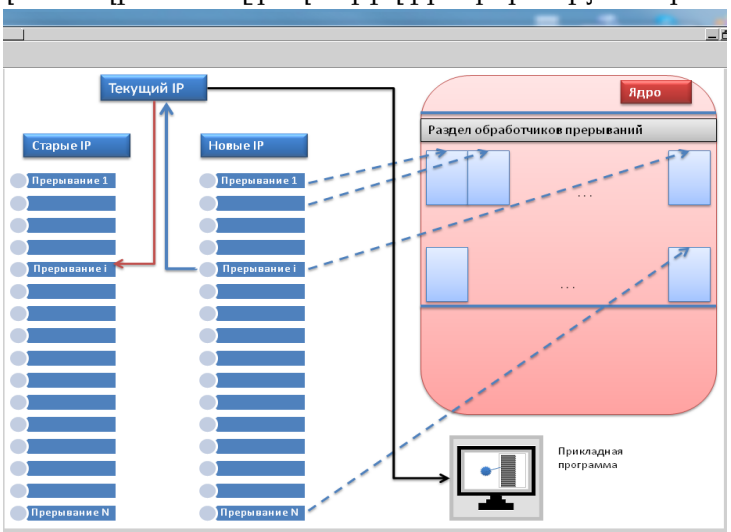
Այս առումով, փոխազդեցության գործողությունների նկատմամբ կան երկու հիմնարար մոտեցում՝ սինխրոն (այսինքն՝ սարքի կամ ծրագրի «շահադետի» կողմից) կամ ասինխրոն, որը սովորաբար գալիս է ֆունկցիայի կատարողի կողմից:

Այսպիսով, ցանկացած սարք կարող է հարցվել պրոցեսորի վրա աշխատող ծրագրի կամ միկրոծրագրի կողմից: Այս դեպքում խոսքը սինխրոն մեթոդի մասին է: Այս դեպքում պրոցեսորը (միկրոծրագիրը) կամ ծրագիրը սպասում է պատասխանի՝ վիճակի վերաբերյալ հնարավոր տվյալներով:

Սակայն սարքի վրա տեղի ունեցող որևէ իրադարձության դեպքում (օրինակ՝ գործողության մեկնարկ, գործողության ընթացքում տեղի ունեցող իրադարձություն, գործողության ավարտ և այլն), իրադարձության մասին ազդանշանը կարող է ընդհատման միջոցով պրոցեսորին ուղարկվել ասինխրոն, այսինքն՝ առանց իրադարձությանը սպասելու որևէ միտումնավոր գործողությունների:

Նկար 4-ում ներկայացված է պրոցեսորի ընդհատումների արձագանքման ընդհանուր սխեման: Այսպիսով, RAM-ում, նախապես նշված հասցեներում, տեղադրվում են հասցեներ՝ մուտքի կետեր այն ծրագրերի համար, որոնք կմշակեն ընդհատումները (առանձին կատեգորիա են միկրոծրագրերի գործարկմամբ մշակվող ընդհատումները, բայց այս դեպքում դրանք էական չեն): Սովորաբար, այս դիրքերը լրացվում են ՕՀ-երի սկզբնական բեռնման ծրագրերով և կարող են հետագայում փոփոխվել բեռնված ՕՀ-ի միջուկի կողմից: Ամեն դեպքում, այս հասցեները մատնանշում են ՕՀ-ի միջուկի այն հատվածները, որոնք նախատեսված են, առնվազն, ընդհատումների սկզբնական մշակման համար: Եթե պրոցեսորի կողմից համապատասխան ընդհատման տրիգերից ազդանշան է առաջանում, ըստ ընդհատման տեսակի, ընթացիկ IP արժեքը պահվում է RAM-ում (դառնում է "հին" IP արժեք), իսկ «նոր» IP-ների ցանկում ընդհատմանը համապատասխանող հասցեից՝ միջուկի ծրագրի բաժնի հաջորդ կատարվող հրամանի հասցեն բեռնվում է IP-ի մեջ (տե՛ս նկ. 4): Հաջորդ տակտի վրա պրոցեսորը, ինչպես արդեն նշվեց,

«կուրորեն» շարունակում է իր աշխատանքը նշված հրամանից: Ընդհատումների մշակիչը, մեկնարկելուց հետո, հասանելիություն ունի հին հասցեին, ռեգիստրի արժեքներին, որոնք կարող են ծառայել ընդհատումը մշակելուց հետո նախկինում աշխատող ծրագրին ճիշտ վերադարձի համար: Սա կարող է տեղի ունենալ որոշ ժամանակ անց, ինչպես սահմանված է առաջադրանքների կառավարման ենթահամակարգում (ընդհատումը մշակելուց հետո առաջադրանքների կառավարիչը կարող է վերահսկողությունը փոխանցել մեկ այլ ծրագրի, այլ ոչ թե այն ծրագրին, որի գործողության ընթացքում կանչվել է ընդհատումը), ռեգիստրների, այդ թվում՝ IP-ի վերականգնման միջոցով ընդհատման պահին պահպանված «հին» ռեգիստրներին: Մինչև ընդհատված ծրագրի վերականգնումը կարող են տեղի ունենալ բազմաթիվ իրադարձություններ:



Նկար 4. IP հասցեների փոփոխման և ընդհատումների մշակման գործընթացը:

Ընդհատումները միշտ մշակվում են կառավարման համակարգի կողմից (սա միշտ չէ, որ օպերացիոն համակարգն է), սակայն մշակիչներից ընդհատումների որոշակի տեսակներ կարող են անմիջապես փոխանցվել այդ ընդհատումներին սպասող կիրառական ծրագրերին:

Արտաքին ընդհատումների շարքում հատուկ տեղ են զբաղեցնում ապարատային ժամանակաչափերը, որոնք գրանցամատյաններ են, որոնք ունեն թվային արժեք գրելու հնարավորություն: Աշխատանքի սկիզբը սկսելուց հետո, յուրաքանչյուր որոշակի ժամանակային քվանտում գրանցամատյանի երկուական թվից կհանվի մեկը, մինչև հայտնվի զրոն որպես արժեք: Ժամանակաչափի զրոյացումը կառաջացնի ապարատային ընդհատում, որից հետո ՕՀ-ի միջուկի բաժինը մշակում է իրադարձության մասին տեղեկատվությունը այդ իրադարձությանը սպասող ծրագրին՝ կամ օպերացիոն համակարգի ծառայություններից մեկին, կամ օգտատիրոջ ծրագրին: Ըստ էության, ժամանակաչափերը ժամանակից կախված իրադարձություններ կազմակերպելու միջոցներ են: Մասնավորապես՝ թայմաուտներ:

Այս դեպքում խոսքը վերաբերում էր նվազող արժեքների վրա հիմնված նվազող ժամանակաչափերին: Օրինակ՝ համակարգիչներում կան աճող ժամանակաչափեր, որոնք որոշակի հաճախականությամբ մեկ են ավելացնում ընթացիկ արժեքին:

Պրոցեսորի ճարտարապետությունում սարքավորումների աշխատանքի ընդհատումներ առաջացնող ժամանակաչափերի (առնվազն մեկի) առկայությունը նշանակալի դեր է խաղում ինչպես ՕՀ-երի (ժամանակի կտրող բազմախնդրային համակարգեր, I/O կառավարում և այլն) այնպես էլ այն մասերում գտնվող ծրագրերի համար, որոնք ապահովում են իրենց սեփական կառավարումը, սովորաբար տեղեկատվության մուտքային/ելքային կառավարումը՝ իրադարձությունների սպասման ընթացքում:

IBM-ից բացի, ապարատային ժամանակաչափեր առկա են SPARK պրոցեսորների ճարտարապետությունում, DSP պրոցեսորների դասում (առնվազն չորս) և այլն: Միևնույն ժամանակ, պարզեցման և ծախսերի կրճատման ճանապարհով գնալով՝ Intel-ը ժամանակաչափեր չի ներառել

իր x86 պրոցեսորների ճարտարապետության մեջ, բացառությամբ միակի, որը կոչվում է համակարգային ժամանակաչափ, որը պարունակում է աստղագիտական ժամանակ՝ սկսած որոշակի ամսաթվից, ինչը զգալիորեն բարդացնում է ժամանակի վրա հիմնված իրադարձությունների ստեղծումը և հետևումը:

Դրանց բացակայությունը ինչ-որ կերպ փոխհատուցելու համար Microsoft-ը իր օպերացիոն համակարգերում ստեղծել է վիրտուալ ժամանակաչափերի մեխանիզմ, որոնք չափազանց անճիշտ են (սովորաբար սա համակարգի ժամանակաչափի ընթերցումն է սկզբնական պահին և ժամանակաչափի ընթերցումը ցիկլով, մինչև ընթացիկ արժեքի և սկզբնական արժեքի միջև տարբերության անհրաժեշտ քանակի միլիվայրկյանները առաջանան՝ շատ վատ եղանակ է սպասելու, որը զբաղեցնում է պրոցեսորը):

Նույն պատճառով (ոչ միակը), այս պրոցեսորներն ունեն չափազանց սահմանափակ բազմախնդրության հնարավորություններ: Սակայն այդ մասին ավելի ուշ՝ բազմախնդրության համակարգերի մշակման հիմունքներին նվիրված համապատասխան բաժնում:

Բնականաբար, կարող են առաջանալ նույն տեսակի կամ այլ տեսակի ընդհատումներ: Այս դեպքում ընդհատումները, որպես կանոն, հերթեր չեն ստեղծում, չնայած դա բացառված չէ: Այս առումով, ընդհատումների մշակման ծրագրերը կարող են արգելափակել և ապաբլոկավորել ինչպես բոլոր ընդհատումները, այնպես էլ դրանց առանձին խմբերը:

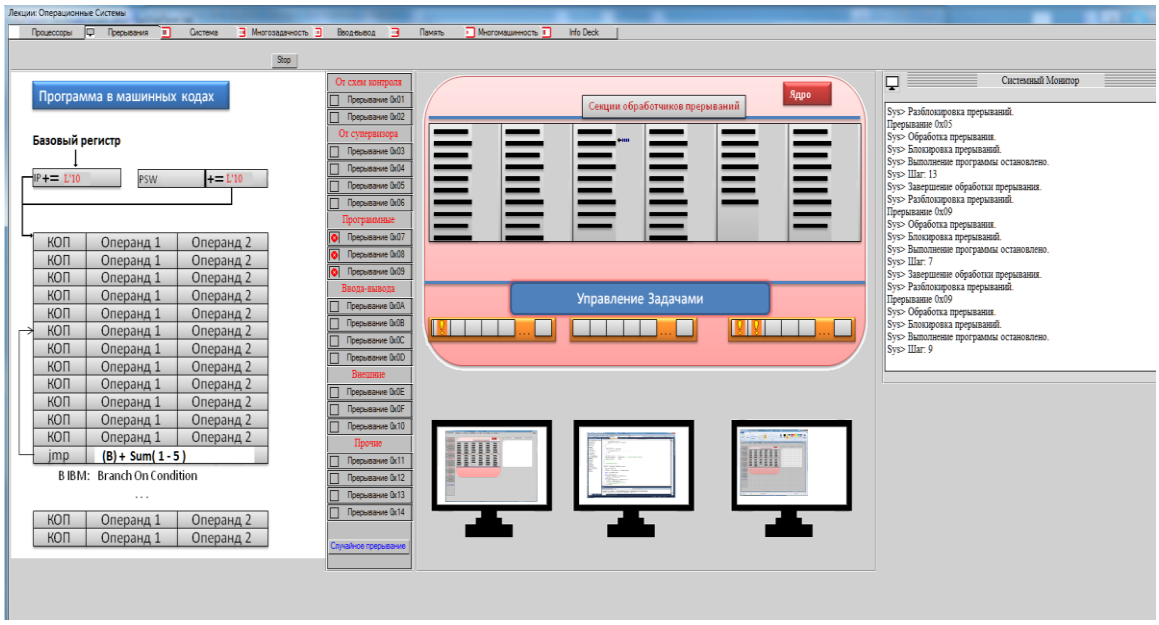
Հաճախ ընդհատումների «գլորումը» պարզապես վերակայվում է արդեն ապարատային մասով:

Որպես կանոն, ընդհատումների առաջացման ցածր մակարդակի կարգավորման և դրանց մշակման հաջորդականության նպատակներով, տարբեր տեսակի ընդհատումները կարող են ունենալ տարբեր առաջնահերթություններ կարևորության առումով: Այսպիսով, IBM-ում ամենաբարձր առաջնահերթությունը (ընդհանուր առմամբ, սա մնում է արդիական այլ ճարտարապետությունների և համակարգերի համար) պրոցեսորի կառավարման սխեմաներից առաջացող ընդհատումներն էին, որոնց առաջացումը գործնականում արգելափակում էր մնացած բոլորին: Պրոցեսորն ինքնին ուներ սարքավորումներում խափանումների պատճառները ստուգելու և բացահայտելու միջոցներ:

Սովորաբար, բարձր առաջնահերթության ընդհատումների տեսակները արգելափակվում էին ցածր առաջնահերթության ընդհատումների տեսակներով, այդ թվում՝ իրենց սեփական դասով: Այնուամենայնիվ, այստեղ կան հնարավոր լուծումներ: Մասնավորապես, բազմամիջուկային/բազմապրոցեսորային բազմախնդրության մեխանիզմի կիրառումը՝ մշակող ծրագրերի պարտադիր վերամուտքով (այս հասկացությունները կբացահայտվեն ավելի ուշ):

Նկ. 5-ը ցույց է տալիս ծրագրային ընդհատումների արգելափակումը մշակման պահին (մշակիչի բաժինների 3-րդ սյունակ) ընդհատումների սիմուլյատորի վրա:

Windows, UNIX և այլ UNIX-անման ՕՀ-ին ունեն հերթերում իրադարձություններ (ո ոչ միայն ընդհատումներ, այլև դրանք, ներառյալ) ստեղծելու միջոցներ: Windows-ում իրադարձությունների հերթերը (սովորաբար ոչ պակաս, քան 64 ԿԲ հիշողություն) ստեղծվում են առաջադրանքների կառավարման ենթահամակարգի կողմից (տե՛ս նկ. 5): UNIX-անման համակարգերի համար հերթերը ստեղծվում և կառավարվում են հատուկ գործառույթների միջոցով:



Նկար 5. Ընդհատման և մեքենայական կոդի կառավարման համակարգի ինտերֆեյս:

Բոլոր ժամանակակից օպերացիոն համակարգերում համակարգչային սարքավորումների հետ աշխատելու ծրագրերը (առաջին հերթին՝ ընդհատումների մշակիչները), համակարգչային ռեսուրսների և ծրագրերի կառավարման հիմնական սխեմաները իրականացնող ծրագրերը խմբավորվում են այսպես կոչված համակարգային միջուկում: Սակայն, որոշակի դեպքերում, որոշ ընդհատումների արձագանքը կարող է տեղի ունենալ նաև BIOS միկրոծրագրերի միջոցով, այսինքն՝ ընդհատումը կարող է կանչվել BIOS ֆունկցիաների (բաժինների) միջոցով:

Միջուկն ունի իր սեփական տվյալները, մասնավորապես՝ հիշողության բլոկները, որոնք նկարագրում և ներկայացնում են, առաջին հերթին, սարքային ռեսուրսները (հաճախ անվանում են UCB – Միավորի կառավարման բլոկ, որոշ համակարգերում՝ Սարքի կառավարման բլոկ): UCB-ն նշում է սարքերի տեսակները, կոնկրետ մոդելները, աշխատանքային պարամետրերը, արտադրողներին և թողարկման ամսաթվերը և այլն:

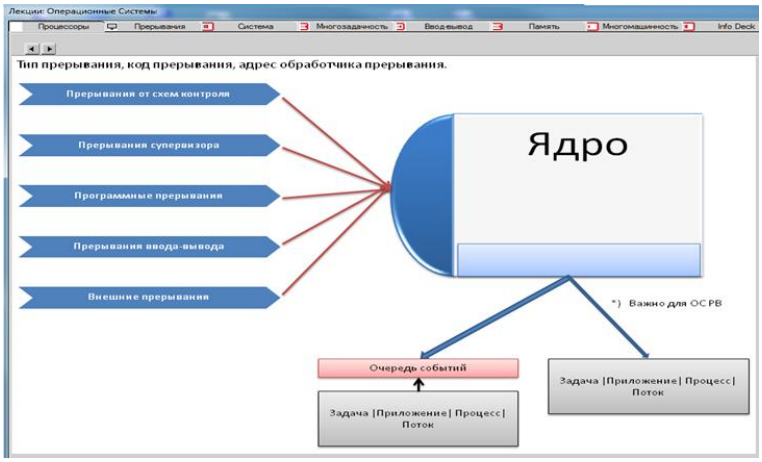
Միևնույն ժամանակ, համակարգի միջուկների ճշգրիտ սահմանում չկա, քանի որ դրանց գործառույթները կարող են տարբեր լինել համակարգից համակարգ՝ պահպանելով ընդհանուր հայեցակարգը:

Գրեթե բոլոր միջուկային մոդուլները և ընդհանուր առմամբ շատ օպերացիոն համակարգեր աշխատում են արտոնություններով. նրանք ունեն մուտքի իրավունք և անում են այն, ինչ արգելված է կիրառական ծրագրերի համար: Մասնավորապես՝ օգտագործում են մեքենայական հրամաններ, որոնք անմիջականորեն ազդում են պրոցեսորի և համակարգչի ռեսուրսների վրա որպես ամբողջություն: Որպես կանոն, դրանք ռեգիստրներ են, քեշ հիշողությունը, ընդհատումների մշակիչների սահմանման տարածքները, ֆիզիկական հիշողությունը, մուտքային-ելքային սարքերի (ենթամշակիչներ, կարգավորիչներ և այլն) հետ փոխազդեցությունը:

Մեքենայական հրամանները, որոնք ապահովում են թվարկվածը, կոչվում են և հանդիսանում են արտոնյալ: Դրանց կատարման հնարավորությունը ապահովվում է ապարատային եղանակով: Նման հրաման կատարելու փորձը ասեմբլերի լեզվով կամ բարձր մակարդակի լեզվով ստեղծված, բայց Ասեմբլերում այսպես կոչված ներգծային ներդիրներ ունեցող կիրառական ծրագրից հանգեցնում է արտակարգ ընդհատման:

Նկ. 6-ը ցույց է տալիս, թե ինչպես է միջուկի համապատասխան բաժինը ստանում ընդհատում և աննշան մշակումից հետո (ընդհատման փաստի գրանցում, վերլուծություն և նշանակում, եթե ընդհատումը վերաբերում է որոշակի ծրագրի) իրադարձության մասին տեղեկատվությունը տեղադրում կամ ծրագրի իրադարձությունների հերթում, կամ ուղղակիորեն կանչում է այդ ընդհատումը կարգավորելու համար նշանակված ծրագրի ֆունկցիան: Վերջինս

հիմնականում օգտագործվում է իրական ժամանակի համակարգերում (RTOS) և կապված է համակարգչին կից սարքերի վրա հնարավորինս արագ արձագանքի հետ (օգտագործվում է արդյունաբերության տեխնոլոգիական շղթաներում, զենքի համակարգերի մարտական տեղեկատվության և կառավարման համակարգերում և այլն):



Նկար 6. Ընդհատումների տեսակները և միջուկի կառուցվածքը:

***) SVC ընդհատումներ** – SuperVisor Call - պրոցեսորի հրամաններ թվային օպերանդով, որոնք ցույց են տալիս վերահսկիչի (միջուկի) ծրագրերից մեկը: Ընդհատում կանչելուց հետո, վերահսկիչի ծրագրի համապատասխան բաժինը սկսում է իր կատարումը:

Արտաքին ընդհատումներ - օրինակ՝ համակարգչի վրա՝ Reset և Shut down կոճակները (վերաբեռնում, միացում/անջատում), IBM համակարգիչների վահանակներում՝ պրոցեսորի Stop/Start կոճակները, RAM - ում գրելու և դրանից կարդալու կոճակները և այլն:

1.4. Պրոցեսորների քեշ հիշողություն

Ժամանակակից նորարարությունները ներառում են պրոցեսորներում տարբեր մակարդակների հիշողության քեշերի ի հայտ գալը (մշակողներն իրենք են որոշում յուրաքանչյուր մակարդակի ֆունկցիոնալ բեռնվածությունը): Եվ դա պայմանավորված է երկու հիմնական պատճառով՝ RAM-ի հետ փոխազդեցության արագությամբ և ծրագրերի կատարվող կոդը իրական ժամանակում վերլուծելու հարմարավետությամբ և արագությամբ:

Քեշ հիշողությունը արագ ներքին պրոցեսորի հիշողություն է, որը նախատեսված է հիմնականում RAM-ին մուտքը նվազագույնի հասցնելու համար: Պրոցեսորի ռեգիստրներում բեռնումը և բեռնաթափումը կատարվում է նմուշային բառի չափերով, հետևաբար, RAM-ին մուտք գործելու քանակը, օրինակ՝ ընթերցման համար, հավասար է ներբեռնվածի ծավալին, բաժանված բառի երկարությանը, մինչդեռ քեշի միանվագ ներբեռնումը (իհարկե, որոշակի ծավալից սկսած) շատ ավելի արագ է, քան բառի չափի բազմակի ներբեռնումը (հնարավոր է՝ ավելորդ տեղեկատվությամբ, քեշի չափերով, չափված մեգաբայթերով): Այսպիսով, քեշերը նպաստում են քեշից յուրաքանչյուր հրամանի արագ ընտրությանը և վերլուծությանը, տվյալներին արագ մուտք գործելուն:

Քեշ հիշողության նմուշների արագագործությունից բացի, հնարավոր դարձավ վերլուծել քեշում գտնվող ընթացիկ կիրառական ծրագրի կոդի մի հատված հաշվարկի գործընթացի ընթացքում: Մասնավորապես, վերլուծությունը կարող է իրականացվել ծրագրի հատվածի զուգահեռացման հնարավորության վերաբերյալ զուգահեռ հաշվարկների տեսության մեջ (մասնավորապես, գերսկալյար պրոցեսորներում) դինամիկ զուգահեռացման հայեցակարգի տարբեր մոտեցումների համատեքստում:

DSP պրոցեսորներում հրամանները քեշի միջոցով բաշխվում են համապատասխան ենթապրոցեսորներին՝ ըստ իրենց տեսակի *):

Կատարելի ծրագրային կոդի վերլուծությունը կատարվում է BIOS-ի միկրոծրագրերի կողմից:

Եթե քեշի հիշողության մեջ ծրագրի որևէ բաժին գերազանցում է ֆրագմենտի չափը (օրինակ՝ ցատկի հրամանով կամ դրսում գտնվող տվյալներին մուտք գործելով), ապա քեշի պարունակությունը փոխվում է՝ RAM-ից այն փոխանակելով, որպեսզի նշված հասցեները լինեն քեշում:

*) DSP պրոցեսորներում հրամանները բաժանվում են գումարային, բազմապատկիչ, համեմատական հրամանների և այլնի:
Դրանք համապատասխանում են մասնագիտացված ենթամշակիչներին:

Գլուխ 2. ՕՀ-եր, դրանց ստեղծման պատճառներ և տեսակներ

Ծանոթանալով համակարգիչների ընդհանուր սկզբունքներին և հիմնական կազմաձևին, կարող ենք խոսել նաև օպերացիոն համակարգերի ի հայտ գալու պատճառների մասին։

Հաշվարկների կազմակերպման զգալի մասը նվիրված է իրադարձություններին արձագանքների իրականացմանը, I/O գործողություններին, դինամիկ հիշողության տարրերի ընտրության հարցումներին, ծրագրերի բեռնմանը և կատարմանը և այլ բաների (օրինակ՝ միջպրոցեսորային և միջմեքենայային փոխազդեցության կազմակերպմանը)։

Այս առումով, մշակողներին և ծրագրավորողներին ստիպելը ամեն անգամ հորինել և մշակել վերը նշված բոլոր լուծումները, կամ «քարշ տալ» մեծ թվով ծառայողական և, ընդհանուր առմամբ, ռուտինային ծրագրային մոդուլներ ծրագրի հետ միասին բավականին թանկ և, ամենակարևորը, անօգուտ հաճույք է։ Հատկապես, եթե հաշվի առնենք, որ ծրագրավորողները սխալներ են թույլ տալիս տարբեր պատճառներով։ Ահա թե ինչու, նույն IBM ընկերությունում, ճարտարապետության մշակմանը զուգընթաց, մտածվել է նաև IBM-360 մոդելների շարքի ՕՀ-ի հայեցակարգը։ Անհրաժեշտ է ընդգծել այդ ժամանակների զարգացումների այս կարևոր առանձնահատկությունը. ճարտարապետությունը մշակվել է՝ հաշվի առնելով համակարգիչների կառավարման սկզբունքները, դրանց վրա խնդիրների լուծման սխեմաները։ Մենք այդքան հեռվից ենք սկսում այն փաստը, որ այս մոտեցումը մինչ օրս մնում է միակ ճիշտը։ Չհաշված այն փաստը, որ IBM համակարգիչների ճարտարապետության մեջ ներդրված լուծումները նույնպես գործնականում մնում են օրինակելի։

Հաշվողական ճարտարապետության կողմից բազմախնդրության աջակցության ամենակարևոր առանձնահատկությունը ծրագրերի կողմից կենտրոնական պրոցեսորի անորոշ ժամանակով գրավումը կանխելու հնարավորությունն է։ Ամենապարզ օրինակը ծրագրի բաժնի ցիկլային կրկնությունն է՝ առանց ՕՀ ծառայություններին մուտք գործելու գործառույթների օգտագործման, որոնք կարող են պարունակել ծրագրի մոնիթորինգի և ընդհատման միջոցներ։ Սա էապես խոչընդոտում է ՕՀ-ին կատարել ծրագրեր միջև ռեսուրսների վերաբաշխման առաջադրանքներ։

Այս առումով, ընդհատումների հայեցակարգից բացի, մշակվել են մուտքային/ելքային ենթապրոցեսորներ, որոնք հնարավորություն ունեն փոխազդել RAM-ի հետ՝ ուղղակիորեն գրելու/կարդալու համար, առանց կենտրոնական պրոցեսորի օգտագործման այդ նպատակների համար։ Սա ենթադրում էր բազմախնդրություն ստեղծելու ևս մեկ հիմք։ Այստեղ մենք կարող ենք համեմատություն անցկացնել Intel պրոցեսորների հետ, որոնցում ընթերցումը/գրելը էապես իրականացվում է CPU-ի կողմից։

IBM ենթապրոցեսորները կոչվում են I/O ալիքներ, ունեն իրենց սեփական մեքենայական լեզուն, որը բաղկացած է ֆիքսված երկարությամբ հրամանների փոքր, բայց բավարար հավաքածուից, և կառավարման համակարգ, որի ներկայացուցիչը CSW ռեգիստրն է (PSW-ի, Channel Status Word-ի նման), որը արտացոլում էր I/O ենթահամակարգի ընթացիկ վիճակը, կարգավորում էր RAM-ին մուտքը և այլն։

Ամենակարևորն այն է, որ ալիքների աշխատանքի ընթացքում պրոցեսորը մնում է ազատ և «Կանգնեցնել» հրամանով անցնում է սպասման ռեժիմի։ Հենց այս հնարավորությունն է դարձել օպերատիվ հիշողության մեջ միաժամանակ բեռնված բազմաթիվ ծրագրերի կառավարման ստեղծման սկզբնական պատճառը՝ պրոցեսորը ծրագրից ծրագիր անցնելով՝ սպասելով I/O ավարտին (սինխրոն ռեժիմում, ասինխրոն ռեժիմի համար տե՛ս այս տեքստի համապատասխան բաժինը)։

Այս կապակցությամբ, միտումնավոր բազմախնդրության ռեժիմի մշակման շնորհիվ, ստեղծվեց հիշողության այսպես կոչված «բանալիավորում», այսինքն՝ բոլոր օպերատիվ

հիշողությունները բաղկացած էին նույնական բլոկներից՝ ֆիքսված երկարությամբ, որոնք ունեն հիշողության բանալին պահելու էլեկտրոնային բիթեր:

ՕՀ-ի բեռնումից անմիջապես հետո սկզբնական պահին ամբողջ հիշողությունն ունի 0 բանալին:

Ըստ ծրագրերի բեռնումից՝ յուրաքանչյուրը ստանում էր RAM-ի հատվածներ, որոնք բաղկացած էին որոշակի բլոկներից, որոնց համար հատկացված էր մեկ ոչ զրոյական բանալի:

Ծրագրի ավարտից հետո, դրան հատկացված հիշողությունը ազատվում էր զրոյական բանալիով:

Հիշողության բանալիները դարձան հիշողության սահմանազատման և մուտքի կարգավորման հիմքը. ուրիշի բանալիով ծրագրից հիշողությանը մուտք գործելու փորձերը առաջացնում են սարքային ընդհատում (ապարատային մակարդակում համեմատվել են PSW-ում տեղակայված կիրառական բանալին և հասցեագրված հիշողության բանալին), այսինքն՝ չի պահանջվել հասցեային տարածքի ոչ տրիվիալ վիրտուալիզացիա և դրան մուտք գործելը:

Այսպիսով, IBM-ի գաղափարախոսները սկզբում նպատակ ունեին ստեղծել բազմախնդրություն կատարող համակարգիչ և դրա մեջ ներդրել էին անհրաժեշտ սարքավորումը: Ինչպես նրանք սկզբում նշել էին, նրանք ստեղծել են «իդեալական» ճարտարապետություն այդ սերնդի և դասի համակարգիչների համար, որը թույլ էր տալիս միաժամանակյա աշխատանք հարյուրավոր հեռակա օգտատերերի հետ (այն ժամանակ՝ 7 ալիք՝ յուրաքանչյուրի վրա 256 սարքով): Այս համակարգիչները հայտնի են որպես «մեյնֆրեյմներ» (mainframe):

Շատ բաղադրիչներ և լուծումներ հետագայում փոխառվեցին այլ մշակողների և ընկերությունների կողմից:

Նշենք, որ այս ճարտարապետությունը մինչ օրս պահպանվում է, գրեթե անփոփոխ:

IBM-390-ի վրա հիմնված ժամանակակից IBM համակարգիչները միայն ավելացրել են քանակական բնութագրերը (մասամբ, ռեգիստրների չափերը կրկնապատկվել են՝ պահպանելով էական բաղադրիչը):

Ճիշտ է, վերը նշվածը չի երաշխավորում մեկ ծրագրի կողմից պրոցեսորի գրավման խնդրից խուսափելը: Այս դեպքում կարևոր է համակարգի ադմինիստրատորի դերը՝ ենթադրելով նման հնարավորություն: Լուծումը, որպես կանոն, մեկն է՝ սարքային ժամանակաչափը սահմանել ծրագրի աշխատանքի համար հատկացված ժամանակի վրա, ինչը կհանգեցնի ընդհատման և ՕՀ-ի միջուկի վերականգնման՝ հետագա այլ առաջադրանքների անցնելու հնարավորությամբ:

Ընդգծենք կանոններից մեկը. որքան քիչ է համակարգչի և դրա ՕՀ-ի ճարտարապետությունը համապատասխանում դրա վրա լուծվելիք խնդիրներին, այնքան ավելի շատ ռեսուրսներ (ժամանակ, հիշողություն, օգտագործվող գործառույթների ծավալներ և այլն) կծախսենք դրանց իրականացման վրա:

Ծրագրերի կատարման համակարգումը, այդ թվում՝ դրանց բեռնման և RAM-ում տեղադրման հարցերում, համակարգիչների ներքին և արտաքին ռեսուրսների համատեղ օգտագործման մեջ, դրանց օգտագործման միատեսակ մեթոդների ապահովումը, ծրագրերի կատարման մոնիթորինգը, համակարգիչների և ծայրամասային սարքերի գործունակության և կայունության ապահովումը օպերացիոն համակարգերի հիմնական խնդիրներն են, որոնց լուծման արդյունավետությունը ընդգծում է ՕՀ-ի առավելություններն ու թերությունները:

Օպերացիոն համակարգերը, իրենց տեսական նախատրամադրվածությամբ, ռեսուրսների կառավարման մեջ այսպես կոչված մոնիտորներ են երեք տեսակների շարքում՝ սեմաֆորներ, մոնիտորներ, պահակներ՝ [1] *)-ում տրված դասակարգման մեջ: Այստեղ վարպետի և գործիքների պահեստի կառավարչի հետ համեմատությունը մեկ անձի մեջ բավականին տեղին է. առաջինը կառավարում է աշխատանքը, երկրորդը՝ տարբեր տրված գործիքների՝ ռեսուրսների տրամադրումը և վերահսկումը: Միևնույն ժամանակ, օպերացիոն համակարգերի առանձին բաղադրիչները կարող են օգտագործել կառավարման բոլոր երեք մեթոդները:

*) Սենտինել-հաշվիչ, որը սպասարկում է օգտվողներին և ծրագրերին ' նրանցից գործառույթների և ընթացակարգերի պահանջների կատարման միջոցով (հայտնի են որպես տարբեր ուղղվածության սերվերներ): Նման մոտեցումը ծրագրերից չի պահանջում լրացուցիչ ալգորիթմացում և անհրաժեշտ ալգորիթմների իրականացման համար ռեսուրսների տրամադրում: Այսպիսով, սենտինելները ներառում է տվյալների բազայի սերվերներ, որոնք աշխատում են "հարցում-պատասխան (հարցման արդյունքը տվյալների տեսքով)" սկզբունքով և չեն պահանջում որոնող ծրագրավորումից որոնում, դրա իրականացման և արդյունքների դասավորության ռեսուրսներ:

2.1. Բազմախնդրության տրամաբանություն

IBM-ի մեյնֆրեյմների համար ստեղծված և՛ ճարտարապետությունը, և՛ դրա համար ստեղծված ՕՀ-երը սկզբնապես կողմնորոշված էին հաշվողական միավորի բազմախնդրային աշխատանքի վրա (ի տարբերություն նախորդ ճարտարապետությունների): Սա արտահայտվում էր և՛ համակարգչի, և՛ ՕՀ-ի բոլոր բաղադրիչներում՝ պրոցեսորի մեքենայական հրամաններում, օպերատիվ հիշողության ապարատային աջակցությամբ՝ դրա բաժանմամբ հիշողության բանալիների (և հավելվածների), ծայրամասային սարքերի և ընդհանուր առմամբ I/O համապարփակ մոտեցման մեջ:

Մենք կբաժանենք օպերացիոն համակարգեր ստեղծելու մոտեցումները՝ ըստ ճարտարապետական առանձնահատկությունների.

1. Կենտրոնական պրոցեսորից (CPU) անկախ I/O ազդանշաններով ճարտարապետություններ՝ ենթամշակիչների համալիրի միջոցով անմիջապես RAM կամ RAM-ից (IBM):
2. Ժամանակի քվանտացումը կազմակերպելու ունակությամբ ապարատային ժամանակաչափերով ճարտարապետություններ (IBM պրոցեսորներ, SPARC, DSP պրոցեսորներ 4 կամ ավելի ժամանակաչափերով, մի քանի ուրիշներ):
3. Ճարտարապետություն՝ անկախ կենտրոնական պրոցեսորից (ենթամշակիչների համալիրի միջոցով) մուտքային-ելքային և առանձնացված դինամիկ և ստատիկ հիշողությամբ (օրինակ ՝ DSP պրոցեսորներ): Ընդհանուր առմամբ ՝ առանձին հիշողությամբ հավելվածների և I/O-ի համար:
4. Բազմապրոցեսորային համակարգեր (բազմակի կոմուտացվող պրոցեսորներ):
5. Բազմամեքենայական համակարգեր (հաշվողական մեքենաների համալիր, որոնք միացված են կապի միջոցներով, ներառյալ հաշվողական ցանցերը):

Այս ցանկում հատուկ տեղ է զբաղեցնում այսպես կոչված «կորպորատիվ բազմախնդրությունը», որը հիմնված է բավականին «վատ» ճարտարապետական ռեսուրսների վրա, որոնք չունեն ո՛չ I/O պրոցեսորից բաժանելու միջոցներ, ո՛չ էլ ընդհատումներով ապարատային ժամանակաչափեր: Հետևաբար, ցանկում մենք «-1» կնշենք Intel x86 ճարտարապետություն ունեցող տարրը:

IBM-ի համար բազմախնդրային օպերացիոն համակարգի ստեղծման գործում նշանակալի դեր խաղացին I/O գործընթացները.

- հատուկ ենթամշակիչների բաշխում սեփական հրամանային տողով և RAM-ում ուղղակիորեն կարդալու և գրելու ունակությամբ՝ առանց պրոցեսորի մասնակցության,
- հիշողության բաժանումը էլեկտրոնային բանալիներով, ինչը հնարավորություն տվեց բեռնել բազմաթիվ առաջադրանքներ, քանի որ բանալիները հնարավորություն տվեցին վերահսկել RAM-ին (հասցեագրում) մուտքը ապարատային մակարդակում,
- համարժեք ընդհատման համակարգ, ներառյալ ժամանակաչափեր,
- կենտրոնական պրոցեսորը կանգնեցնելու ունակություն, եթե ներկայումս չկան կատարման ծրագրեր, կամ այն մեկ այլ առաջադրանքի անցնելու հնարավորություն,
- բազմամշակող և ավելի ուշ՝ բազմամիջուկ պրոցեսորներ:

Վերոնշյալը դարձան «ճիշտ» բազմախնդրության նախապայմաններ, ի տարբերություն ավելի ուշ շրջանի ավելի պարզ ճարտարապետությունների վրա ստեղծված համակարգերի (Windows օպերացիոն համակարգի առաջին տարբերակների համար Intel ճարտարապետության համար OS-2-ը բավականին կեղծ-բազմախնդրություն է. իրավիճակը որոշ չափով բարելավվեց բազմամիջուկ պրոցեսորների ի հայտ գալով, երբ մուտքային/ելքային առաջադրանքների կատարումը կարող է վերապահվել նվիրված միջուկին(a):

IBM-360 ճարտարապետության մեջ սկզբնապես ներառված ամենակարևոր բաղադրիչը չորս կիլոբայթանոց հիշողության բլոկներում էլեկտրոնային բանալիներ (4 բիթ) տեղադրելու հնարավորությունն էր:

Այսինքն, ակնհայտորեն նախատեսվում էր հիշողությունը 16 բլոկների խմբերի բաժանելու հնարավորությունը, ինչը, իր հերթին, թույլ է տվել հիշողության բանալիների, դրանց ծրագրային կոդերի և տվյալների համար դինամիկորեն պահանջվող հիշողության դաշտերի հավաքածուների միջոցով տեղադրել միմյանցից անկախ և մեկուսացված ծրագրեր: Թույլատրվում էր մինչև 16 տարբեր ծրագրեր ներառականորեն՝ 0 բանալիներով օպերացիոն համակարգի համար, ոչ զրոյականներով՝ ծրագրերի համար: Այս տարբերակը կոչվում էր ֆիքսված թվով առաջադրանքներով համակարգ: Մի փոքր ավելի ուշ հայտնվեց փոփոխական թվով կիրառական առաջադրանքներով տարբերակ, իսկ ավելի ուշ՝ վիրտուալ էջերի համակարգեր և վիրտուալ մեքենայական համակարգեր:

Միևնույն ժամանակ, նրանք բոլորը հիմնվում էին էլեկտրոնային հիշողության պաշտպանության ուշագրավ հատկության վրա, որը թույլ չէր տալիս մեկ ծրագրին մուտք գործել մյուսի հիշողությանը: IBM-ի այս որակը զգալիորեն տարբերվում էր վիրտուալ հասցեավորմամբ համակարգերից (վերջիններս համեմատաբար ավելի դանդաղ էին, քան սարքային աջակցություն ունեցող համակարգը):

Բազմախնդրության իրականացման ամենակարևոր ենթահամակարգերից մեկը I/O տեղեկատվության և համապատասխան ՕՀ ծառայության կազմակերպումն է:

Խնդիրները կայանում են նրանում, որ պրոցեսորի աշխատանքի արագությունը և հաշվիչների ծայրամասերի սարքերի հետ փոխազդեցության արագությունը զգալիորեն տարբերվում և տարբերվում են մինչ այժմ: Երբեմն մի քանի կարգի մեծությամբ՝ պրոցեսորի հրամանների համար միկրովայրկյաններից և նանովայրկյաններից մինչև միլիվայրկյաններ, արտաքին սարքերի համար՝ վայրկյաններ և րոպեներ: Պրոցեսորի վրա կատարված ծրագիրը, խնդրելով կարդալ սարքից կամ գրել սարքին, կլինի.

- կամ կկասեցվի մինչև I/O գործողության ավարտը,
- կամ, I/O անհրաժեշտությունը նշելուց հետո, շարունակում է գործել մինչև որևէ իրադարձության (ներառյալ ընդհատման) տեղի ունենալը, որը համապատասխանում է ծրագրի տրամաբանությանը և կապված է ծրագրի կողմից կանչված I/O գործողության ավարտին սպասելու հետ: Սպասումը կարող է տեղի չունենալ, եթե I/O գործողությունն ավարտվել է մինչ այս պահը:

Առաջին դեպքում գործողությունը կոչվում է սինխրոն, երկրորդում՝ ասինխրոն: I/O արդյունքի հետ սինխրոնացման խնդիրը կարող է առաջանալ, եթե I/O գործողությունը չի ավարտվել պահանջվող պահին (սովորաբար ծրագիրը տեղափոխելով սպասման ռեժիմի՝ հարցման ցիկլ (վատ եղանակ է, բայց հաճախ պարտադրված), օրինակ՝ փոփոխականում համապատասխան դրոշը տեղադրելով կամ կառավարումը փոխանցելով ՕՀ սպասման ֆունկցիային, մինչև այն վերադարձնի կառավարումը ծրագրին, կամ, եթե կան համապատասխան միջոցներ, ծրագիրը տեղափոխելով I/O իրադարձության հատուկ սպասման ռեժիմի և այլն):

Նկար 7. Սինխրոն և ասինխրոն կանչեր:

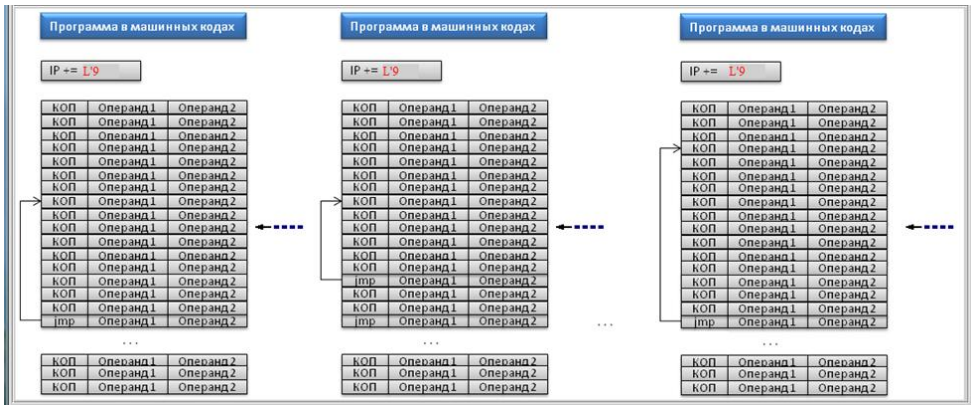
Պրոցեսորից անկախ I/O խնդրի որոշակի լուծում առաջարկվել է DSP թվային ազդանշանային պրոցեսորների ճարտարապետության մեջ, որոնք ունեն երկու տեսակի հիշողություն՝ ստատիկ և դինամիկ: Մուտք/ելքի կառավարման պրոցեսորները կարդում են ստատիկ հիշողություն մտնող կամ դուրս եկող տեղեկատվության հոսքերը՝ անկախ պրոցեսորից, մինչդեռ պրոցեսորն ինքը կարող է ընտրել կամ գրել տեղեկատվություն ստատիկ հիշողությունից դինամիկ հիշողության մեջ՝ հետագա մշակման համար, կամ բեռնաթափել այն ստատիկ հիշողության մեջ՝ ելքի համար:

Ժամաչափեր: Սարքավորումների ժամանակաչափերի առկայությունը թույլ տվեց ստեղծել ժամանակի քվանտացմամբ համակարգերի դաս, այսինքն՝ կախված հերթականությունից և առաջնահերթությունից, բեռնված ծրագրերին նշանակվում են ժամանակային քվանտներ, որոնց ընթացքում պրոցեսորը կարող է կատարել որոշակի առաջադրանքի մի հատված: Երբ ժամանակաչափը վերագործարկվում է, տեղի է ունենում սարքային ընդհատում, որը ընդհատվում է միջուկի կողմից, և այնուհետև առաջադրանքի վերահսկիչը անցնում է մեկ այլ առաջադրանքի, եթե այդպիսին կա: Քանի որ ժամանակային քվանտները, որպես կանոն, մնում էին փոքր, առաջանում էր ծրագրերի միաժամանակյա աշխատանքի պատրանք:

Վերջապես, բազմամշակումը, որը հայտնվել էր արդեն մեքենաների առաջին մոդելներում, ապահովվում էր «պրոցեսորից պրոցեսոր» շինայով, որը թույլ էր տալիս անմիջականորեն միացնել առնվազն երկու CPU-ներ: Նման կապերը ապահովվում էին ասեմբլերի հրամաններով՝ պրոցեսորների միջև փոխազդեցության համար: Բազմամշակումը, ընդհանուր առմամբ, այդ ժամանակ եզակի չէր, քանի որ արդեն գոյություն ունեին բազմաթիվ պրոցեսորներով և տարբեր ճարտարապետական լուծումներով համակարգիչներ: Սակայն սա առանձին դասընթացի թեմա է: Առայժմ նշենք, որ բազմամշակումը համատեղ հիշողությամբ (յուրաքանչյուր պրոցեսոր ունի իր սեփական ֆիզիկական հիշողությունը. նման ճարտարապետությունները ներառում են Motorola 68000 և ավելի բարձր պրոցեսորներ, որոնք ունեն տեղական շինա պրոցեսորի բլոկում, RAM և I/O միջոցներ. բլոկները կարող են տեղադրվել արտաքին շինայի վրա՝ բլոկների միջև փոխազդեցության համար. նման ճարտարապետությունը աջակցվում էր OS-9-ի կողմից) վերացնում է ծրագրերի բաժանման և կատարման հետ կապված բազմաթիվ խնդիրներ:

Բազմամշակումը համատեղ հիշողությամբ (ինչպես ներդրված է ժամանակակից բազմամիջուկային համակարգերում) լուծում է միայն բազմաթիվ առաջադրանքների միաժամանակյա կատարման խնդիրը, մինչդեռ մնում են կիրառական հիշողության բաժանման և պաշտպանության խնդիրները: Այդ թվում՝ նաև ապարատային մակարդակում, քանի որ տարբեր միջուկների կողմից հիշողությանը միաժամանակյա մուտքերի համաժամեցման խնդիրը պետք է լուծվի (և լուծվում է):

Այսպես թե այնպես, բազմամշակումը/բազմամիջուկը ապահովում է բազմաթիվ առաջադրանքների միաժամանակյա (իսկապես զուգահեռ) կատարում (տե՛ս նկ.):



Նկար 8. Մեքենայական կոդերում հրամանների և ընդհատումների մշակում:

Նույնը վերաբերում է բազմաֆունկցիոնալ հաշվողական կայանքներին: Կարևոր տարբերություն է այն փաստը, որ միջմեքենաների փոխազդեցությունը զգալիորեն դանդաղ է, քան միջհամակարգիչը, և ոչ միայն մեքենաների միջև հաղորդակցման միջոցների պատճառով, այլ նաև այն պատճառով, որ փոխգործակցության միջնորդները ՕՀ-երն են:

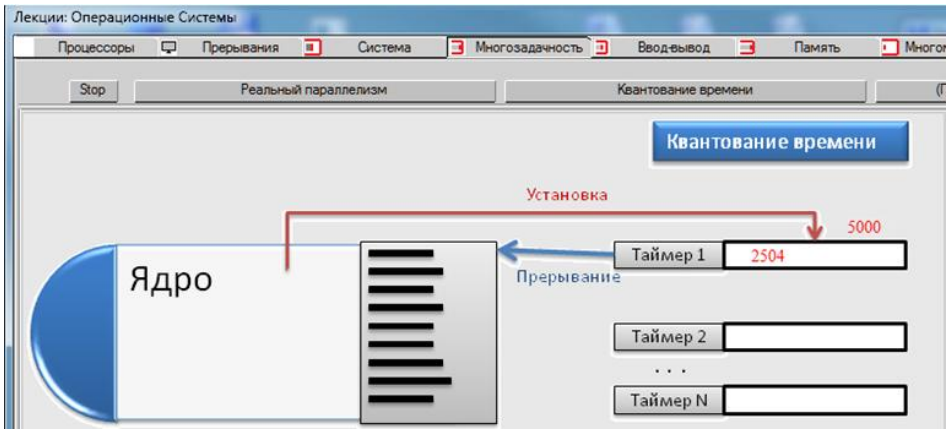
Այնուամենայնիվ, միաժամանակ կատարված գործընթացների կազմակերպման տրամաբանության տեսանկյունից, բազմամշակման և բազմամեքենաների միջև առանձնահատուկ տարբերություններ չկան:

Այսպիսով, մենք թվարկել ենք համակարգիչների հիմնական ճարտարապետական առանձնահատկությունները, որոնք հիմք են հանդիսացել ՕՀ-երի լիարժեք բազմախնդրության համար:

Դժբախտաբար, x86 պրոցեսորների մեծ պարզեցումների պատճառով, օպերացիոն համակարգերի մշակողները ստիպված էին մեծ ջանքեր գործադրել բազմախնդրության լուծումների սկզբունքները ձևակերպելու համար: Մասնավորապես, առաջացավ «համագործակցային» բազմախնդրության գաղափարը, որի հատկությունները մնացին և մնում են բավականին կասկածելի: Բայց դրա մասին առանձին բաժնում:

Պետք է նշել, որ պրոցեսորներն ու միջուկները կարող են չկատարել նույն առաջադրանքների հոսքերը նույն արագությամբ, քանի որ միջուկներից/պրոցեսորներից յուրաքանչյուրը կարող է բեռնվել տարբեր աստիճաններով և այլ առաջադրանքներով:

Իր սեփական առաջադրանքները (հատկապես ընդհատումների մշակումը) առանց որևէ խոչընդոտի կատարելու համար ՕՀ-ը կարող է բեռնման ժամանակ իր սեփական կարիքների համար պահեստավորել առնվազն մեկ միջուկ կամ պրոցեսոր: Նույնը վերաբերում է նաև հիշողությանը՝ բաժանելով ՕՀ-ի հիշողությունը և չզբաղեցված հիշողությունը, որը կարող է տրամադրվել հավելվածներին:



Նկար 9. Համակարգում ժամանակաչափերի և ընդհատումների կառավարում:

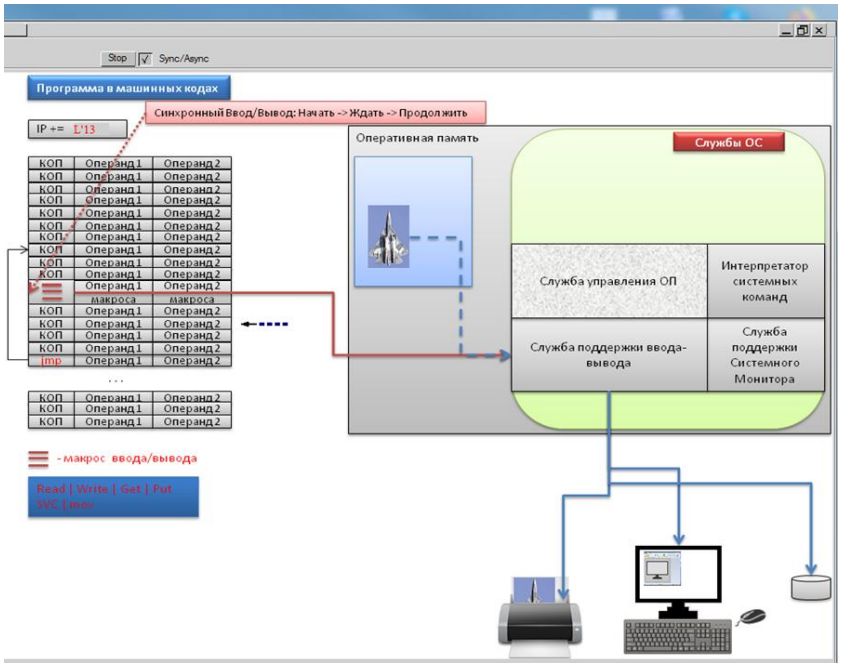
**) Ընդհանուր առմամբ, սինխրոն կանչերը նախաձեռնող ծրագրերից (մասնավորապես՝ հարցման սարքերից կամ այլ ծրագրերից) ընթացակարգերին ուղղված կանչեր են, մինչդեռ ասինխրոն կանչերը ենթադրում են

հարցումների ստեղծում՝ առանց ծրագրերում կամ սարքերում հետագա գործողությունները դադարեցնելու: Եթե սինխրոն կանչերը բավականին պարզ են և ներառում են կանգնեցում՝ կանչի արդյունքին սպասելով, ապա ասինխրոն կանչերը կպահանջեն լրացուցիչ գործողություններ՝ որոշակի կանչի ավարտի մասին տեղեկացնող իրադարձության պատասխանի ծրագրավորում կամ ավարտի իրադարձությանը սպասելու ռեժիմին անցում, եթե իրադարձությունը տեղի չի ունեցել (այսինքն՝ հարցումը չի կատարվել) մինչև արդյունքները ստանալու համար անհրաժեշտ ժամանակը: Նկատենք, որ ծրագիրը կարող է տեղեկատվություն ստանալ (հնարավոր է՝ ՕՀ-ից ընդհատման միջոցով կամ տեղեկատվական դրոշ սահմանելով) ասինխրոն իրադարձության մասին:

***) Շատ DBMS-ներ, երբ տեղադրվում են օպերացիոն համակարգերի սերվերների վրա, պահանջում են նշել տվյալների բազայի ընթացակարգերի կողմից օգտագործվող միջուկների/պրոցեսորների քանակը: DBMS-ի կարիքների համար օպերատիվ հիշողության քանակը նույնպես ֆիքսված է, քանի որ նման համակարգերն ունեն իրենց սեփական ռեսուրսների կառավարման գործիքները: Ըստ էության, սա համակարգ է համակարգի մեջ:

2.2. ՕՀ-երի ծառայություններ

Սովորաբար, օպերացիոն համակարգերի ամբողջ ֆունկցիոնալությունը զգալիորեն բաժանվում է ստատիկ և պոտենցիալ դինամիկ: Այստեղ առանցքային դեր են խաղում հնարավոր փոփոխությունները, ՕՀ-ի որոշակի բաղադրիչների փոփոխականությունը: Այսպիսով, ընդհատման կարգավորիչները, հիշողության կառավարման հիմնական սխեման, առաջադրանքների կառավարումը դժվար թե փոփոխվեն որոշակի գործառնական համակարգի ճարտարապետության շրջանակներում: Բայց նույն բաղադրիչների վերնագրային մասում (ենթադրենք՝ ծրագրերի ավարտից հետո RAM-ում «աղբի հավաքագրման» իրականացումը, վերադրումների և փոխանակման ֆունկցիաների կազմակերպումը, ծածկոցների և փոխանակման գործառույթների կազմակերպում), ռեսուրսների որոշ փորձարկումներ և այլն, կարող են իրականացվել միջուկից դուրս: Այսպիսով, դա արվում է տարբեր սարքերի համար I/O կառավարման խնդիրները լուծելու համար, հատկապես հաշվի առնելով դրանց բազմազանությունը, ինչը պահանջում է ծրագրային գործիքների տարբեր դասավորություններ և վերադասավորումներ: Նկ. 7 ներկայացված են ՕՀ-ի հիմնական ծառայությունները:



Նկար 10. ՕՀ-երի ծառայություններ:

Որպես կանոն, ՕՀ-ի միջուկից իրադարձությունների (ռեսուրսների հարցումներ և թողարկումներ) առաջացումը փոխանցվում է համապատասխան ծառայությանը: Մասնավորապես, հիշողության կառավարման ծառայությունը կարող է կանչվել, երբ, ինչպես նշված է, անհրաժեշտություն կա «աղբի հավաքագրման», երբ հիշողության առանձին հատվածները, իհարկե, սխալմամբ, մնում են չջնջված (օրինակ՝ ծրագրավորողի մոռացկոտության պատճառով): Այս երևույթը հայտնի է որպես «հիշողության արտահոսք»: Հիշողության կառավարման մեկ այլ ձև է որոշակի ծրագրերի կողմից դրա օգտագործման հավաքագրումը և հևրիստիկ գնահատումները: Օրինակ, Windows ՕՀ-ի որոշ տարբերակներ կարող են անմիջապես ֆիզիկապես չազատել RAM-ը, այլ անուղղակիորեն պահպանել այն որոշակի ծրագրի կրկնակի մեկնարկի համար: Հիմնված է, իհարկե, հևրիստիկայի վրա: Այս երկրորդական գործողությունները կառավարվում են համապատասխան ՕՀ-ի ծառայության կողմից: Այս դեպքում պահուստավորված հիշողությունը ներառվում է ազատ ֆիզիկական հիշողության չափի մեջ:

Շատ ավելի բարդ և բազմազան են I/O տվյալների կառավարման գործողությունները: Ինչպես նշվեց, սարքերի բազմազանության, դրանցում հասցեավորման, կարդալու/գրելու մեթոդների և այլնի պատճառով դրանք ավելի մանրամասն կցուցադրվեն այս դասընթացի համապատասխան բաժնում: Առայժմ կենտրոնանանք ՕՀ-ում համապատասխան ծառայության գոյության փաստի վրա:

ՕՀ-ի մեկ այլ ծառայություն է ՕՀ համակարգի հրամանների աջակցությունը: Ինչպես ցանկացած տեղադրման դեպքում, այս տեղադրումը կառավարելու համար անհրաժեշտ է խնդիրներ լուծող մեխանիզմ, գոնե տարրական համակարգ:

Սա լուծվում է օգտատիրոջ փոխազդեցության աջակցության ծառայության կողմից (ոչ միայն մարդկանց, այլև ծրագրերի հետ): Հիմնականում, ՕՀ հրամանային համակարգը կողմնորոշված է ծայրամասային մասերի վերակազմակերպմանը, տվյալների հավաքածուների ստեղծմանը և ջնջմանը (ֆայլային համակարգերի կառավարմանը), դրանց մասին տեղեկատվության ցուցադրմանը և այլն:

Համապատասխան թիմերի հետ նման գործողությունները ներառում են (հիմնական).

- Սարքերի միացում և ապամոնտաժում, այսինքն՝ նոր սարքերի միացում և գրանցում օպերացիոն համակարգում և դրանց անջատում: Օրինակ՝ արտաքին ֆլեշ հիշողության սարքերի միացումը USB միացքի միջոցով, երբ օպերացիոն համակարգը սկսում է ընդհատում (հրամանի տողի հնարավոր գեներացմամբ կամ ծրագրին ուղղակիորեն կանչով) I/O ծառայության վրա՝ նոր սարքը միացնելու համար: Որոշ համակարգեր կարող են արգելափակել սարքի ավտոմատ միացումը, և այդ դեպքում դա պետք է արվի ձեռքով՝ մուտքագրելով համապատասխան հրամանը: Անջատումը տեղի է ունենում նմանատիպ եղանակով՝ սարքը անմիջապես հանելով միակցիչից (վտանգավոր մեթոդ, քանի որ այն չի երաշխավորում դրան գրված վերջին տեղեկատվության պահպանումը. ավելի մանրամասն՝ մուտքագրման/ելքի և գրառման բուֆերացման խնդիրներին նվիրված բաժնում), որը, իհարկե, նույնպես կարող է ընդհատում առաջացնել և սկսել ապամոնտաժման գործընթացը, կամ կարող է և ոչ, և այդ դեպքում, երբ փորձում եք աշխատել հեռացված սարքի հետ, այն կկարգավորվի համակարգում առկա լինելու, բայց չաշխատելու ռեժիմով:
- անհրաժեշտ ծրագրերի մեկնարկը և դադարեցումը,
- համակարգում օգտագործողների ներկայության գրանցում և վերլուծություն,
- տվյալների հավաքածուների ստեղծում և ջնջում (դիրեկտորիաներ/թղթապանակներ, ֆայլեր),

Լայնորեն ներկայացված են տեղեկատվության դիտման միջոցները.

- դիրեկտորիաների և առանձին ֆայլերի մասին (ստեղծման ժամանակը/ամսաթիվը, վերջին թարմացման ժամանակը/ամսաթիվը, տվյալները օգտագործող օգտատերերի մասին, նրանց արտոնությունները և այլն).

- ներկայումս աշխատող ծրագրերի մասին,
- RAM-ի զբաղեցրած և ազատ տարածության ծավալների մասին,
- պրոցեսորի բեռնման մասին,
- այլ՝ կախված ՕՀ-ի ներդրումից և դրա ծառայություններից:

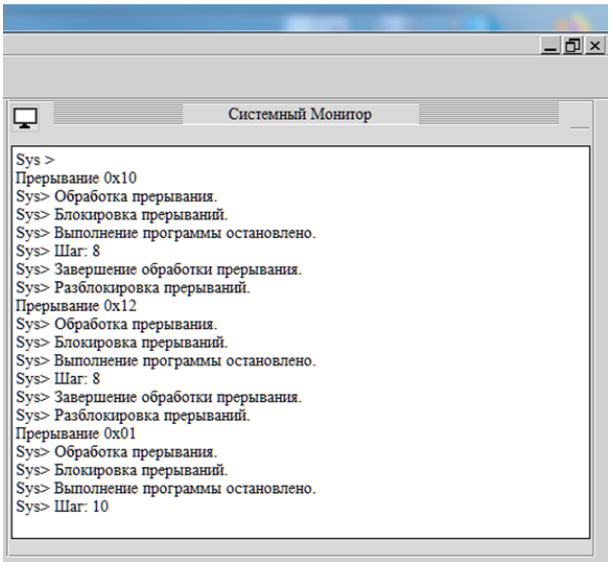
Բազմամշակիչ և բազմամեքենա սարքերի համար կառավարման և տեղեկատվական գործառույթների շրջանակը, անշուշտ, զգալիորեն ընդլայնվում է. անհրաժեշտ են գիտելիքներ համալիրի յուրաքանչյուր առանձին մեքենայի կամ պրոցեսորի, մեքենաների խմբերի, դրանց ընդհանուր և առանձին ռեսուրսների, փոխազդեցության միջոցների և այլնի մասին:

Այս կամ այն տեղեկատվության, ինչպես նաև հաշվողական տեղադրման ռեսուրսների հասանելիությունը պետք է կարգավորվի և գրանցվի:

Այսպես թե այնպես, ցանկացած համակարգչային տեղադրում պետք է ունենա ադմինիստրատորի ռեժիմ այն անձի համար, ով մուտք ունի բոլոր համակարգային ռեսուրսներին և կատարում է այլ օգտատերերի հարցումները:

Անհրաժեշտ հաշվողական միջավայր ստեղծելու համար օպերացիոն համակարգերը տեղադրում են հրամանների մեկնաբանիչներ (սովորաբար բոլոր հրամանները և դրանց պարամետրերը ներկայացվում են մեկ տեքստային տողի տեսքով) և համակարգի մոնիթորինգի աջակցության ծառայություն (համակարգի վահանակ, տե՛ս նկ. 6)՝ հրամաններ մուտքագրելու և պահանջվող տեղեկատվությունը արտածելու կամ ընթացիկ իրադարձությունների մասին տեղեկատվություն ավտոմատ կերպով արտածելու համար (տե՛ս նկ. 7):

Ցանկացած ՕՀ ունի համակարգի մոնիտոր՝ ծրագիր, որը թույլ է տալիս իր օպերատորին՝ ադմինիստրատորի կարգավիճակով, փոխազդել հաշվողական համալիրի հետ: Յուրաքանչյուր օպերատոր, բացի նույնականացման միջոցներից և գաղտնաբառից, ունի հաշվողական տեղադրման ռեսուրսների ամբողջ կամ մի մասի տնօրինմանն ու կարգավորումներին մուտք գործելու իրավունք:



Նկար 11. . Ընդհատման և հրամանների մշակման մոնիտոր:

2.3. ՕՀ-երի պահանջները

Օպերացիոն համակարգերին ներկայացվում են որոշակի բովանդակային պահանջներ, որոնց պետք է հետևեն ՕՀ-ի մշակողները:

1. Օպերացիոն համակարգը իր աշխատանքում պետք է նվազագույն ժամանակ պահանջի (գոնե ՕՀ-ի մշակողները պետք է ձգտեն դրան):
2. Օպերացիոն համակարգը իր աշխատանքում պետք է զբաղեցնի և/կամ օգտագործի ռեսուրսների նվազագույն քանակ՝ RAM, սկավառակի տարածք և այլն:

3. Կայունություն: ՕՀ-ն կարող է մտնել արտակարգ իրավիճակի («վթարի») միայն հաշվիչի սարքային խափանումների պատճառով (պրոցեսոր, կամ օպերատիվ հիշողություն, կամ համակարգային շինաներ կամ նմանատիպ):
4. ՕՀ-ն վերահսկում է հաշվիչի բոլոր ռեսուրսները և ապահովում է կիրառական ծրագրերի ընթացակարգային աջակցությունը՝ յուրաքանչյուր ռեսուրսին մուտք գործելու համար:
5. ՕՀ-ն թույլ չի տալիս դիմումներից մեկին (առաջադրանքին) տրամադրված ռեսուրսի վիճակի և տեղեկատվության փոփոխություն մեկ այլ դիմումի (առաջադրանքի) կողմից:
6. ՕՀ-ի գործառույթները միշտ պարտավոր են վերադարձնել արժեքը, երբ դիմում են հավելվածներից: ՕՀ-ի մոդուլների վթարային ավարտը անընդունելի է:
7. ՕՀ-ն թույլ չի տալիս համակարգի »կոտրումը» և «վիրուսների» ներթափանցումը/ազդեցությունը իր միջավայրի վրա:
8. Ռեսուրսների և դրանց սպասարկման ընթացակարգերի վերաբերյալ կառավարման և սպասարկման բոլոր տեղեկությունները ուղղակիորեն հասանելի չեն դիմումներից (ՕՀ-ի համար կողմնակի ծրագրեր):

Կան սխալ լուծումների օրինակներ, որոնք չեն բավարարում այս պահանջները: Մասնավորապես, Windows ՕՀ-ը թույլ է տալիս «փչացնել» պատուհանները՝ այլ ծրագրերի պատուհանները համընկնելով: Հաշվի առնելով, որ պատուհանները օպերացիոն համակարգի կողմից կառավարվող հիմնական տրամաբանական տարրերն ու ռեսուրսներն են, որոնք հատկացված են ծրագրերին ինտերֆեյսները կազմակերպելու համար: Այսպիսով, ծրագրերը՝ պատուհանների սեփականատերերը, ստիպված են «վերանկարել» դրանք՝ ուղարկելով փչացման մասին հաղորդագրություններ:

Մոտավորապես այսպես, որ մեկ ծրագիրը վնասում է մեկ այլ ծրագրի կողմից զբաղեցված ֆայլը, իսկ ՕՀ-ը հաղորդագրություն է ուղարկում՝ խնդրելով սեփականատիրոջը վերականգնել ֆայլը:

Նման որոշումների պատճառները պարզ են. ՕՀ-ի ներսում համընկնող պատուհանների պատճենների ստեղծումը և պահպանումը բավականին հիշողություն պահանջող խնդիր է: Այնուամենայնիվ, փաստը մնում է փաստ. ՕՀ-ը չի պահպանում ծրագրին տրամադրված ռեսուրսի անվտանգությունը, այլ այս խնդիրը թողնում է հենց ծրագրին: Ավելին, ժամանակակից համակարգիչների ռեսուրսների ծավալները անհամեմատ աճել են:

Կանոնների մեկ այլ խախտում է UNIX-ից ժառանգված դինամիկ կերպով հատկացված հիշողության միավորումների գրանցման բլոկներ ստեղծելու մեթոդը՝ միակողմանի ցուցակներ (այս մեթոդը իրականացվում է UNIX-անման օպերացիոն համակարգերում, այդ թվում՝ Windows-ում): Հիշողության հատկացման գործիքները կանչելիս (C/C++-ում - malloc() և դրա տարբերակները), հատկացված հիշողության տարածքին նախորդում է հատկացված տարածքի գրանցման բլոկը և ցուցակի տարրը պարունակող հատվածը: Այսպիսով, հիշողության տարրի կառավարման բլոկը գտնվում է օգտատիրոջ տարածքում, ինչը սխալի դեպքում կարող է հանգեցնել բլոկի և հաշվապահական ցուցակի վնասման:

Հենց դրա հետ է կապված ևս մեկ խախտում. համակարգային ֆունկցիաների հետագա կանչերը՝ հիշողությունը հատկացնելու և ազատելու համար, ավարտվում են վթարով՝ առանց առանց վերադառնալու իրենց կանչող ծրագրին: Առանց ախտորոշման, ծրագրում սխալներ գտնելը (ցանկի տարրի բովանդակության ոչնչացումը) դառնում է չափազանց դժվար և պահանջում է մեծ հնարամտություն և ժամանակ:

Նշենք, որ IBM համակարգերում ռեսուրսների կառավարման բոլոր բլոկները տեղակայված են ՕՀ-ի հիշողության մեջ, ինչը թույլ չի տալիս նրանց մուտք գործել հավելվածների կողմից:

Հատկանշական է մեկ այլ բան. IBM ՕՀ-երը չեն կարող կոտրվել, վիրուսները չեն ներթափանցում և կենսունակ չեն: Հենց այդ պատճառով էլ IBM համակարգիչները երբեք չեն

վերլուծվել կոտրվելու և վիրուսների նկատմամբ զգայունության տեսանկյունից, և չեն նշվել համապատասխան վարկանիշներում:

2.4. ՕՀ-երի բեռնում

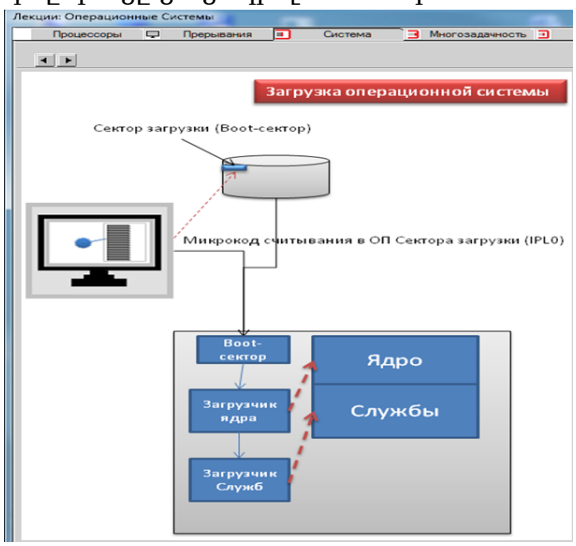
Ցանկացած հաշվողական համալիրի հիմնական աշխատանքը սկսվում է դրա կառավարման համակարգի բեռնումից: Մենք կկենտրոնանանք մեկ համակարգչի համար ՕՀ-երի բեռնման վրա, չնայած ավելի բարդ հաշվողական համալիրների համար համակարգերի բեռնման տրամաբանությունը բավականին համապատասխանում է ստորև ներկայացված սխեմային:

ՕՀ-ի բեռնումը սկսվում է BIOS-ում միկրոծրագրերի կատարմամբ, որոնք ստուգում են համակարգչի սարքավորումները: Դրանցից հետո գործարկվում է BIOS ծրագիրը՝ ըստ սարքերի հետազոտության, որոնք սահմանված են որպես գործառնական համակարգի հավանական կրիչներ: Կրիչների ցանկում առաջինը գտնելուց կամ սարքի նախապես սահմանված ցուցման համաձայն՝ բեռնվում է տվյալ ՕՀ-ի ծրագիրը: Եթե նշված սարքը չի համապատասխանում բեռնման սկզբունքներին (բեռնման համակարգի (մասնավորապես՝ բեռնման հատվածի) բացակայություն կամ սխալ գրանցում), դա հանգեցնում է սկզբնական բեռնման համակարգով մեկ այլ սարքի որոնմանը:

Հայտնաբերվածի դեպքում BIOS մոդուլը արտաքին կրիչի (Boot հատվածի) ֆիքսված տեղից RAM-ում բեռնում է նախնական բեռնման ծրագիրը (ավանդույթի համաձայն, Դրանք նշանակվում են IPL - նախնական ծրագրի բեռի հապավմամբ): Եվ պարտադիր չէ, որ դա լինի ՕՀ-ի ծրագիր: IPL-ը կբեռնի արտաքին կրիչի վրա գտնվող սկզբնական բեռնման հատվածում գրանցված ցանկացած ճիշտ կազմակերպված (BIOS-ի աշխատանքի ըմբռնմամբ) ծրագիր: Ծրագիրը բեռնվում է RAM - ի կանխորոշված վայրում (հնարավոր է՝ սկսած հիշողության զրոյական հասցեից, ավելի հաճախ՝ սկզբից ֆիքսված օֆսեթով - որոշ համակարգերի կողմից զրոյական հասցեներ օգտագործվում են ընդհատման վեկտորները տեղադրելու համար-ընդհատումների մշակման գործառույթների ցուցիչներ), ներբեռնված ծրագրի առաջին հրամանի ցուցիչը ձևավորվում է IP-ում:

Նման ծրագիրը կարող է լինել ՕՀ ընտրելու ծրագիր, եթե համակարգչում տեղադրված է մեկից ավելի ՕՀ (օրինակ՝ GRAB ծրագիրը UNIX-անման համակարգերի համար):

ՕՀ-ի բեռնումը սովորաբար իրականացվում է մի քանի փուլով. բեռնվում է ՕՀ բեռնիչը, որն իր հերթին բեռնում է ՕՀ-ի միջուկը, որը նախնականացնում է ընդհատման վեկտորները և այլ անհրաժեշտ տվյալները: Այնուհետև բեռնվում են ՕՀ-ի ծառայությունները: Համակարգի մոնիտորը կարող է միացվել բեռնման գործընթացը ցուցադրելու համար:



Նկար. 12. ՕՀ-ի սկզբնական բեռնումը:

Փուլերից և բեռնման ժամանակից մեկը դրայվերների բեռնումն է՝ ծրագրեր սարքերի հետ կառավարիչների միջոցով անմիջական փոխազդեցության համար:

Սարքերի ցանկը կարող է ֆիքսված լինել (տրամաբանական է այն պահել առանձին ներբեռնվող ֆայլում) կամ ձևավորվել միացված սարքերի վերաբերյալ հարցումներ անցկացնելով շինայի միջոցով: Սա դինամիկ դեպք է:

2.5. Առաջադրանքներ և ենթաառաջադրանքներ, գործընթացներ և հոսքեր

Հիմնականում Microsoft-ի շնորհիվ տերմինաբանության մեջ ձևավորվել է «տարաձայնություն», որը կապված է համակարգչային կայանքներում կատարված առաջադրանքների ըմբռնման հետ:

Տերմինաբանությունը ձևավորվել է IBM ընկերությունում 20 տարվա ընթացքում, ստուգվել և մտածվել է, բայց այն սկսել է «ընդհատել» Microsoft-ի անձնական համակարգիչների, օպերացիոն թաղանթների և օպերացիոն համակարգերի ընդլայնման հետ մեկտեղ՝ երբեմն շփոթելով արդեն իսկ ստեղծված կատեգորիաների ըմբռնումը:

Բացատրենք և կանոնակարգենք այն:

IBM-ում Յուրաքանչյուր հաշվարկ սկսվել և սկսվում է գործառնական համակարգի առաջադրանքով, որը նկարագրում է առաջադրանքի կատարման միջավայրը և այն ռեսուրսները, որոնք պետք է ներգրավվեն: Այս մոտեցումը թույլ է տալիս ՕՀ-ին պլանավորել (!) առաջադրանքի կատարում: Այս պատճառով է, որ ՕՀ-ի ամենակարևոր բաղադրիչը առաջադրանքների և առաջադրանքների ժամանակացույցն է, որը գնահատում է հաշվիչի ներկայիս հնարավորությունները և զբաղվում է կատարման միջավայրի պատրաստմամբ: Սա արձագանք է գտել կոնտեյներների, վերահսկիչների և հիպերվիզորների ժամանակակից մեխանիզմներում:

Առաջադրանքները թույլ են տալիս ոչ միայն պատրաստել անհրաժեշտ ռեսուրսները, այլև ձևավորել դրանց պաշտպանությունը չպլանավորված գործողություններից, ինչպես նաև մեկուսացնել որոշակի հաշվարկի միջավայրը ինչպես արտաքին ներխուժումներից, այնպես էլ հաշվարկի թաղանթի արտաքին ներթափանցումներից:

Միջավայրի նկարագրությունը ֆիքսելուց հետո, ՕՀ-ն սկսում է առաջադրանքը: Առաջադրանքը իր հերթին կարող է գործարկել ենթաառաջադրանք: MS համակարգերում առաջադրանքը համապատասխանում է գործընթացի հայեցակարգին, որը խմբավորում է հոսքերը: Երկու հասկացություններն էլ այնքան էլ հաջող չեն, քանի որ գործընթացի հայեցակարգը հատվում է զուգահեռ գործընթացի ավելի հիմնարար հայեցակարգի հետ զուգահեռ հաշվարկների տեսության և պրակտիկայի մեջ, որում յուրաքանչյուր գործընթաց դիտարկվում է մեկ խնդրի լուծմամբ զբաղվող այլ գործընթացների խմբում: "Հոսք" տերմինը նույնպես անհաջող է, քանի որ այս տերմինը համընկնում է զուգահեռ հաշվարկների հասկացություններից մեկի հայեցակարգի և տերմինների հետ. հոսքային հաշվարկներ (flow, flow-data, stream): Զուգահեռ հաշվարկման տեսության տերմինները հայտնվել են շատ ավելի վաղ, քան Ms-ի լուծումները: Ըստ էության, MS-ում գործընթացը IBM-ում առաջադրանք է, հոսքը՝ զուգահեռ գործընթաց կամ ենթաառաջադրանք:

Ենթաառաջադրանքները (հոսքերը) կարող են աշխատել իրենց սեփական հիշողությամբ, երբ զուգահեռացված հոսքը հիշողության հետ միասին ամբողջությամբ պատճենվում է ՕՀ-ի կողմից հատկացված նոր հիշողության մեջ: Այս կերպ գործարկված հոսքերը կոչվում են ծանրքաշային:

Այս տրամաբանությունը ժառանգվել է UNIX-անման համակարգերից: Միևնույն ժամանակ, իրադարձությունների հերթը մնում է տարածված (հիշեցնեմ, որ UNIX-ում իրադարձությունների հերթը կարող է ձևավորվել հատուկ ֆունկցիաների միջոցով): Այնուամենայնիվ, հնարավոր է պահանջել հոսքի սեփական իրադարձությունների հերթի ստեղծում:

Միևնույն ժամանակ, ՕՀ-ն հնարավորություն է տալիս հիշողությունը թողնել ընդհանուր հոսքերի համար: Նման հոսքերը կոչվում են թեթև, քանի որ դրանք ձևավորվում են ավելի հեշտ և արագ՝ առանց հրահրող հոսքի հոր հիշողության դաշտերը կրկնօրինակելու:

Յուրաքանչյուր հոսք կանչվում է ՕՀ Task Starter-ի կողմից, յուրաքանչյուր հոսքի (ենթաառաջադրանքների) համար ձևավորվում է իր սեփական կանչերի կույտը, որի սկզբում կա ֆունկցիայից մեկնարկային ֆունկցիային վերադարձի կետի՝ հոսքի մուտքի կետի նշում:

2.6. Ֆոնային և ռեզիդենտ ծրագրեր

Կա ծրագրերի որոշակի դաս, որոնք կարող են աշխատել այսպես կոչված ֆոնային ռեժիմում: Սրանք ծրագրեր են, որոնք չեն փոխազդում օգտատերերի հետ, դրանց ներկայությունը որևէ կերպ չի արտացոլվում կամ արտացոլվում է զուտ խորհրդանշականորեն էկրանին որևէ պատկերակի կամ ֆայլի առկայության միջոցով:

Սովորաբար, նման ծրագրերը ստեղծվում են իրադարձությունները հետևելու կամ որոշ առօրյա, ծառայողական գործառույթներ կատարելու համար, օրինակ՝ իրադարձությունների գրանցամատյանը վարելու և/կամ վերլուծելու համար:

Windows-ում ֆոնային ռեժիմով աշխատող օգտատիրոջ ծրագիրը կարելի է համարել այնպիսի ծրագիր, որը չի ստեղծում իր սեփական ինտերֆեյսի պատուհանը:

Միևնույն ժամանակ, գրեթե բոլոր ՕՀ-երը, բացառությամբ ամենապարզների (ինչպիսիք են MS-DOS-ը կամ ավելի պարզերը), թույլ են տալիս իրենց կազմում (ծառայություններում կամ նույնիսկ միջուկում) ներառել այսպես կոչված ռեզիդենտ ծրագրեր, որոնք, ավանդույթի համաձայն, ունեն արտոնյալ ռեժիմ (ինչպես ՕՀ-ի ծրագրերը), աշխատում են ՕՀ-ի հիշողության հետ (զրոյական ստեղներով հիշողություն) և ունակ են ընդհատումներ ընդհատել և մշակել:

Սակայն նման ծրագրերի գրելը մեծ ուշադրություն է պահանջում, հատկապես կրկնամուտքայինի (reenterability) առումով (կրկնամուտքայինի, դրա սահմանափակումների և ուշադրության համար տե՛ս «Բազմախնդրություն» բաժինը): Դրանք, անշուշտ, պետք է արդյունավետ լինեն թե՛ հիշողության, թե՛ կատարողականության առումով: Դրանք պետք է ռեենտերային լինեն, եթե սպասարկում են բազմաթիվ հավելվածներից ստացված հարցումներ:

Գլուխ 3. Առաջադրանքներ, հանձնարարություններ և դրանց կառավարում

Համակարգիչների վերջնական նպատակը ակնհայտ է՝ այնպիսի ծրագրերի կատարում, որոնք առաջանում են ինչպես արտաքին աղբյուրներից, այնպես էլ հանդիսանում են տեղադրման ներքին ռեսուրս: Այս գլխում մենք կքննարկենք արտաքին օգտատերերից ծագող ծրագրերի և հաշվողական առաջադրանքների կատարման տարբեր սխեմաներ: Միևնույն ժամանակ, ճարտարապետական և սխեմատիկ առանձնահատկություններից անկախ, բոլոր ունիվերսալ համակարգիչները և դրանց օպերացիոն համակարգերը ունեն նույն ֆունկցիոնալ հատկությունները՝ առաջադրանքների և հանձնարարությունների տեղադրում, դրանց ընտրություն կատարման համար, կատարվող կոդի և տվյալների բեռնում, առաջադրանքների մեկնարկ, դրանց կայուն և միանշանակ (անկախ համակարգչային միջավայրի վիճակից) աշխատանքի ապահովում, արդյունքների տեղադրում և առաջադրանքների կատարում:

3.1. Առաջադրանքների և հանձնարարությունների մեկնարկ

Օպերացիոն համակարգերը տարբեր կերպ են վերաբերվում առաջադրանքների գործարկմանը՝ ըստ հաշվիչների օգտագործման նպատակների և ճարտարապետական տրամաբանության:

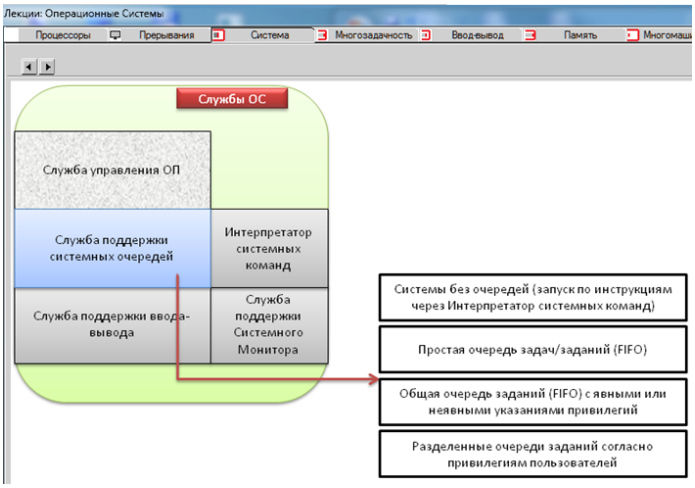
Առաջադրանքները կարող են ներկայացվել առաջադրանքով՝ գրառումների (փաթեթների) ամբողջություն, որը սահմանում է ՕՀ կարգավորումների և գործողությունների ամբողջություն, որոնք պետք է ընդունվեն որպես պարամետրեր, հրահանգներ, գործողություններ և որոնք պետք է կատարվեն ՕՀ ծառայությունների կողմից: Յուրաքանչյուր ՕՀ ունի իր սեփական հրահանգների հավաքածուները (IBM-ը այդ նպատակների համար ստեղծել է ամբողջական կառավարման լեզու՝ JCL - Job Control Language): Windows-ում և UNIX-անման համակարգերում հրահանգները և նկարագրությունները կարող են խմբավորվել այսպես կոչված խմբային ֆայլերում կամ թաղանթի սկրիպտներում (shell script):

Միևնույն ժամանակ, առաջադրանքը կարող է գործարկվել համակարգի հրամանների մեկնաբանի համար հրամանի տող մուտքագրելուց հետո՝ համակարգի մոնիտորի (կամ ՕՀ-ի հրամանների մեկնաբանի) միջոցով: Windows-ը մշակել է «աշխատասեղանի» (desktop) վրա տեղադրված պատկերակների մեխանիզմ, որոնք ներկայացնում են առաջադրանքներ և հանձնարարություններ, որոնց, իրենց հերթին, նշանակվում են մեկնաբանի հրամանների տեքստերի տողեր (ցուցադրվում և խմբագրվում են պատկերակի հատկությունների ցուցադրման պատուհանը կանչելով):

Եթե հաշվողական համակարգը նախատեսված է մի քանի օգտատերերի սպասարկելու համար, ապա դրա վրա օգտագործվող կառավարման համակարգերը ուղեկցվում են օգտատերերի հաշվառման, ռեսուրսներին մուտքի վերահսկման, առաջադրանքների և հանձնարարությունների հերթերի ձևավորման և օգտատերերին առաջադրանքների կատարման մասին տեղեկացնելու ծառայություններով:

Կանգ առնենք առաջադրանքների և հանձնարարությունների հերթերի տարբերակներին (տե՛ս նկ. 10):

Ինչպես արդեն նշվեց, առաջադրանքների կամ հանձնարարությունների անհապաղ կատարման համակարգերի համար հերթեր չեն ստեղծվում: Այսպիսով, Windows օպերացիոն համակարգը անձնական տարբերակում անմիջապես սկսում է կատարել օգտատիրոջ կողմից ընտրված առաջադրանքը:



Նկար 13. Առաջադրանքների/հանձնարարությունների հերթերը կազմակերպելու տարբերակներ:

Իսկ ահա բազմաօգտատեր համակարգերը (սերվերներ, մեյնֆրեյմեր և այլն) սովորաբար ստեղծում են նման հերթեր մեկ կամ մի քանի ֆայլերում:

Ամենապարզ դեպքը մեկ ֆայլ է, որում հաջորդաբար գրված են առաջադրանքների/հանձնարարությունների փաթեթների հրահանգները, կամ հենց հրամանները կամ դրանց հետ կապված փաթեթները:

Հերթի ֆայլի բոլոր գրառումները կարող են ստեղծվել ըստ արտոնությունների՝ հնարավոր դասավորությամբ ըստ առաջնահերթությունների և արտոնությունների թվային արժեքների՝ անկախ համակարգի ադմինիստրատորների կողմից օգտվողներին նշանակված ստացման ժամանակից և նրանց առաջադրանքներից: Ավելի արագ տարբերակ է օգտատիրոջ առաջադրանքների համար բազմաթիվ հերթեր ստեղծելը: Միևնույն ժամանակ, առաջադրանքներն ու հանձնարարությունները կարող են ունենալ իրենց սեփական առաջնահերթությունը, որը նշանակվել է դրանց նախաձեռնողների կողմից և հնարավոր է՝ ճշգրտվել համակարգի կողմից:

Ելնելով օգտագործողների առաջնահերթություններից, նրանց առաջադրանքներից, հնարավոր է, հաշվարկային տեղադրման ծանրաբեռնվածությունից, ձևավորվում է այսպես կոչված: դիսպետչերական առաջնահերթություն, որի վրա հիմնված է կառավարիչը/վերահսկիչը/առաջադրանքների կառավարիչը ռեսուրսներ տրամադրելիս, առաջադրանքների բեռնման ժամանակը, պրոցեսորի օգտագործման ժամանակի քվանտի արժեքը և այլն:

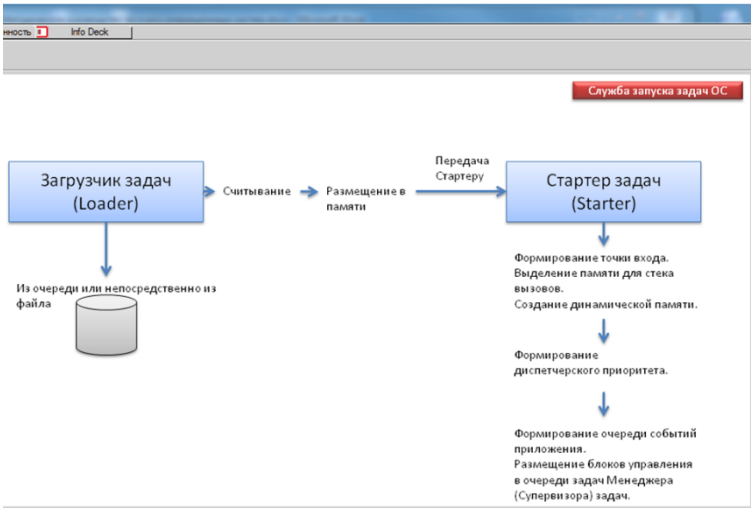
3.2. Բեռնիչ և առաջադրանքների մեկնարկիչ

Մի կողմ թողնելով այն դեպքերը, երբ հնարավոր է կանչել արդեն բեռնված առաջադրանքներ, օպերացիոն համակարգերում քայլերի հաջորդականությունը մոտավորապես նույնն է և կարող է կատարվել երկու բաղադրիչների կողմից՝ ծրագրի բեռնիչ և մեկնարկային ծրագրի միջոցով: Միևնույն ժամանակ, նշենք, որ ֆունկցիաների բաժանումը դրանց միջև բավականին կամայական է դրանց հաջորդականությունը կատարելիս:

Անվանումից արդեն պարզ է, որ Loader-ը ընտրում է (հնարավոր է՝ դրան նշված է) արտաքին պահեստավորման սարքի վրա հաջորդ առաջադրանքը՝ համաձայն ՕՀ-ում ընդունված տրամաբանության (հերթերով կամ առանց դրանց):

Առաջին հերթին որոշվում է առաջադրանքը բեռնելու համար անհրաժեշտ RAM-ի քանակը: Հիշողության չափը ձևավորվում է կատարվող կոդի իրական չափից (ֆայլի չափ), ինչպես նաև կանչերի կույտի չափից, դինամիկ հիշողությունից, ծառայության տեղեկատվությունից (որոշ մանրամասներ կնշվեն ավելի ուշ), մասնավորապես՝ ծրագրի կողմից պահանջվող դինամիկ գրադարանների ցանկից: Սովորաբար, ստեկի չափի մասին տեղեկատվությունը (սովորաբար լռելյայն արժեքները, բայց օգտագործողը, երբ ստեղծում է կոդը, կարող է նշել իր սեփական

արժեքները) և դինամիկ հիշողությունը պահվում են գործադիր կոդով ֆայլի վերնագրի կառուցվածքում, որը ձևավորվում է կազմողի ելքային տվյալների հիման վրա և հղումների խմբագրի կողմից կոդերի հետագա դասավորության հիման վրա: Այս տվյալները կարդալուց հետո, Loader-ը հիշողության կառավարիչից խնդրում է անհրաժեշտ հիշողությունը, ինչպես նաև ՕՀ-ի միջավայրում բեռնում է ծրագրի կողմից պահանջվող և նախապես հայտարարված դինամիկ գրադարանները: Նշենք, որ հավելվածները կարող են ինքնուրույն բեռնել և բեռնաթափել դինամիկ գրադարանները, բայց կրկին՝ ՕՀ-ի միջոցով, որը բեռնում և գրանցում է դինամիկ գրադարանները և մասամբ վերահսկում դրանց օգտագործումը: Որպես կանոն, դինամիկ գրադարանը առկա է մեկ օրինակով (այս կանոնը կարող է շրջանցվել), ինչը բխում է գրադարանների օգտագործման տրամաբանությունից՝ լինել ֆունկցիաների կրող՝ բոլոր հաշվիչների համար ընդհանուր օգտագործման համար:



Նկար 14. Առաջադրանքների բեռնման և կատարման մեկնարկի հաջորդականությունը:

Ծրագիրը հիշողության մեջ տեղադրելուց և պատրաստելուց հետո, առաջադրանքի մեկնարկիչը բեռնված ֆայլի վերնագրից ընտրում է մուտքի կետ՝ կամ հղումը խմբագրելու գործընթացում ուղղակիորեն սահմանված բացարձակ հասցե, կամ ֆայլի սկզբի նկատմամբ շեղում: Ձևավորվում է բազային ռեգիստրի արժեքը: Առաջադրանքների մեկնարկիչից իրականացվում է բեռնված ծրագրի ստանդարտ կանչ, որը արտացոլվում է մեկնարկիչի կանչող մոդուլի վրա ցուցիչ տեղադրելով և դրան վերադարձի կետ՝ ծրագրի աշխատանքի ավարտից հետո կանչերի կույտում (կան նրբերանգներ բազմահոսքային տարբերակների համար): Այսինքն, հենվելով C/C++ լեզվի ոճի և կառուցվածքների հիման վրա, **main(…)** ֆունկցիայի սկզբի հասցեն մուտքի կետն է, և դրանից **return** օպերատորը վերադարձնում է կառավարումը Starter ֆունկցիային՝ կանչի կույտի վերադարձի հասցեի վրա (հրամանի հասցեն, որը հաջորդում է կանչված հրամանին անցում կատարած հրամանին): Կոմպիլացված ծրագրերի մեքենայական կոդերում սա արտահայտվում է մուտքի կողմից նշանակված կոդի սեկցիայով և կոդի մի հատվածով, որն իրականացնում է վերադարձ դեպի կանչված հատված՝ ըստ առաջադրանքի մեկնարկի մոդուլի կանչի կույտի:

3.3. Առաջադրանքների կառավարման բլոկներ (հավելվածներ և հոսքեր)

Ինչպես արդեն նշվեց, օպերացիոն համակարգերում բոլոր ռեսուրսները ներկայացվում են կառավարման բլոկների միջոցով՝ տվյալների կառուցվածքներ, որոնք նկարագրում են ռեսուրսը՝ կառուցվածքի տարբերակը, ռեսուրսի տեսակը, հատկանիշները, ընթացիկ սեփականատերը (եթե ռեսուրսը հատկացված է), և այլն: Այս կառավարման բլոկները սովորաբար տեղակայվում են ցուցակներում (տվյալների տեսակների իմաստով), աղյուսակներում, զանգվածներում և այլն: Ցուցակներն առավել հարմար են անսահմանափակ ռեսուրսների պարագայում՝ ավելի դինամիկ

կառուցվածքներ ապահովելով, իսկ աղյուսակներն ու զանգվածները նախընտրելի են այն դեպքում, երբ տարրերի քանակի սահմանափակումները նախապես հայտնի են։

Մենք կդիտարկենք առաջադրանքների կառավարման բլոկների ամբողջությունը ցուցակային տեսքով (տե՛ս նկ. 12)։

Յուրաքանչյուր առաջադրանքի կառավարման բլոկում նշվում են.

- Ներբեռնման հասցեն,
- Առաջադրանքի մուտքի կետը,
- Ռեգիստրների պահպանման տարածքը (կամ հենց կառավարման բլոկում տեղադրված տարածքը),
- Քեշ հիշողության կամ դրա համապատասխան հատվածների պահպանման տարածքը,
- Հավելվածի կանչերի կույտի ցուցիչը,
- Հավելվածի դինամիկ հիշողության ցուցիչը,
- Դինամիկ հիշողության հարցված պուլների գրանցման/կառավարման բլոկների ցուցակը (վիրտուալ հիշողության դեպքում՝ էջերի աղյուսակը),
- Հավելվածի ենթառաջադրանքների ցուցակը,
- Բացված ֆայլերի ցուցակը,
- Ժամանակի քվանտի արժեքը (ժամանակաբաժան ռեժիմների դեպքում),
- Ծրագրի ակտիվության ժամանակը (պրոցեսորի զբաղվածության ժամանակը, գործող օպերատիվ հիշողության ծավալը և այլն),
- Առաջադրանքի առաջնահերթությունները,
- Եվ այլ պարամետրեր՝ կախված օպերացիոն համակարգի առաջադրանքների կառավարման մեխանիզմներից։

Նման բլոկների առանձին տարրերը կամ ամբողջ բովանդակությունը կարող են ներկայացվել նաև հավելվածների հոսքերի (ենթառաջադրանքների) ցուցակներում։ Այսպես, հոսքերը կարող են ունենալ.

- սեփական ժամանակի քվանտ,
- սեփական առաջնահերթություն,
- պարտադիր՝ սեփական կանչերի կույտ,
- իրենց տեղաբաշխման կետը (եթե հոսքը ստեղծվում է ծրագրային մոդուլի պատճենմամբ),
- սեփական մուտքի կետ,
- և այլ տեխնիկական տեղեկատվություն։

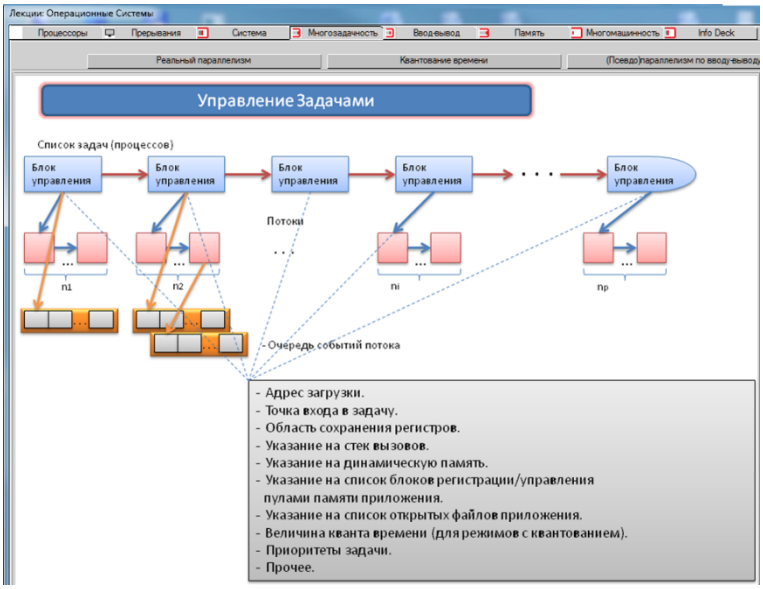
Windows օպերացիոն համակարգում հավելվածների գործարկման ժամանակ ստեղծվում են ֆիքսված երկարությամբ իրադարձությունների հերթեր (կառուցվածքների զանգվածներ), որոնցում տեղադրվում են ոչ միայն տվյալ հավելվածին վերաբերող, այլ նաև ամբողջ համակարգում տեղի ունեցող, սակայն տվյալ հավելվածի համար նշանակություն ունեցող իրադարձությունների մասին տեղեկություններ (օրինակ՝ մոնիտորի էկրանին փոփոխություններ, հավելվածի փակման պահանջներ, այլ հավելվածներից ստացված հաղորդագրություններ և այլն)։ Հետևաբար, հավելվածների և դրանց հոսքերի կառավարման բլոկներում ընդգրկվում են նաև այս իրադարձությունների հերթերի ցուցիչները։

Օպերացիոն համակարգերը, որպես կանոն, հնարավորություն են տալիս անմիջապես մշակել իրադարձությունները՝ հանձնարարված լինելով համապատասխան մոդուլներին (ֆունկցիաներ, ծրագրեր), որոնք օպերացիոն համակարգի միջուկի կողմից որսացված իրադարձությունների մշակմամբ են զբաղվում։ Նման դեպքերում հավելվածի կառավարման բլոկում արձանագրվում է իրադարձությունների մշակման մոդուլի հասցեն։ Որոշ օպերացիոն համակարգերում ձևավորվում են անցումների աղյուսակներ դեպի բեռնված ծրագրեր՝ իրադարձությունների մշակիչներ, որոնց վերահսկողությունը անմիջապես փոխանցվում է

համապատասխան ընդհատումների առաջացման դեպքում: Սովորաբար դա վերաբերում է իրական ժամանակի օպերացիոն համակարգերին:

Վիրտուալ մեքենան կարող է լինել ոչ միայն օպերացիոն համակարգ, այլև ծրագրավորման լեզվի աջակցող միջավայր: Օրինակ՝ նման մեքենաներ ունեն LISP և դրա նմանակ լեզուները, Java-ն, C#-ը, PROLOG-ը և այլն:

Վերադառնալով Windows-ին՝ իրադարձությունների մշակման ծրագիրը նշվում է հավելվածի բնութագրող կառուցվածքում, որը փոխանցվում է օպերացիոն համակարգի առաջադրանքների կառավարման ենթահամակարգին: Իրադարձությունների մշակիչները գործարկվում են համապատասխան մոդուլից՝ պատկերի մեջ տեղադրվելով վերադարձի կետի ցուցիչը:



Նկար 15. Ծրագրի կառավարման բլոկների, ենթաառաջադրանքների ցանկեր: Իրադարձությունների հերթեր:

Մեկ այլ կարևոր գործառույթ է իրադարձությունների հերթի ստեղծումը հավելվածի գործարկման պահին, ինչպես ընդունված է Windows օպերացիոն համակարգում: Սա, սակայն, բնորոշ չէ բոլոր օպերացիոն համակարգերին: Օրինակ՝ UNIX-ին նմանվող համակարգերը հերթերը որպես պարտադիր հատկանիշ չեն ապահովում:

Սա մեծապես կապված է այն հաշվարկային ճարտարապետությունների առանձնահատկությունների հետ, որոնց համար տվյալ օպերացիոն համակարգերը նախագծվել են: Հին ճարտարապետություններում առկա չէին գրաֆիկական էկրաններ և դրանց հետ փոխազդեցության միջոցներ, և այդ պատճառով ներկայում կիրառվում են ծրագրային վերաշերտեր X11 (X Window) պրոտոկոլի հիման վրա՝ Motif միջավայրը, Qt գրադարանը և այլն:

Անձնական համակարգիչների (PC) ճարտարապետությունները ի սկզբանե նախատեսված չեն եղել բազմագործառութայնության կազմակերպման համար (ինչպես նաև PDP համակարգերի առաջին սերունդները, թեև վերջիններս առնվազն ունեին ապարատային ժամանակաչափեր, որոնք հնարավորություն էին տալիս իրականացնել բազմագործառութայնություն ժամանակային քվանտավորմամբ): Մինչդեռ Windows օպերացիոն համակարգի համար նախատեսված հավելվածների գերակշիռ մեծամասնությունը հիմնված է բազմազան պատուհանների տրամաբանության վրա կառուցված միջերեսների ստեղծման սկզբունքի վրա (ի դեպ՝ այս գաղափարը Microsoft-ին չի պատկանում) և պատուհաններում տեղի ունեցող բազմաթիվ իրադարձությունների վրա: Այդ պատճառով նման հավելվածների ծրագրավորման առաջին քայլը սովորաբար լինում է սեփական պատուհանի և նրա տեսքի նկարագրությունը, պատուհանի իրադարձությունների մշակիչ ֆունկցիայի և ամբողջ հավելվածի իրադարձությունների մշակիչի սահմանումը: Իրադարձությունների մշակիչները կարող են մասնագիտացված լինել ըստ տիպի (օրինակ՝ վիրտուալ ժամանակաչափերից ստացվող իրադարձությունները): UNIX-ին նմանվող

համակարգերում սա կազմակերպվում է, օրինակ, Qt գրադարանների միջոցով՝ որպես օպերացիոն համակարգի վերաշերտ։

Այս համակարգերում ամենատարածված իրադարձություններն են՝

- մկնիկի սլաքի անցումը պատուհանի վրայով,
- մկնիկի կոճակների սեղմումը/արձակումը (հնարավոր է նաև սեղմման տևողության ֆիքսում), անիվիկի պտտում կամ սեղմում,
- ստեղնաշարի կոճակների սեղմում/արձակումը,
- այլ իրադարձություններ՝ նոր պատուհանների ստեղծում, դրանց չափերի կամ գրաֆիկական բաղադրիչների փոփոխություն, զուգահեռ գործընթացների գործարկում կամ ավարտ։

Կարևոր է նշել, որ իրադարձությունների հերթերը կարող են հարցում լինել ինչպես սինխրոն, այնպես էլ ասինխրոն եղանակով, սակայն դա իրականացվում է սինխրոն քայլերով՝ հավելվածից արված հարցումների միջոցով (այսինքն՝ հարցման նախաձեռնողը հենց հավելվածն է՝ անհրաժեշտ ֆունկցիայի կանչով, ինչը համարվում է սինխրոն քայլ)։

Այնուամենայնիվ, Windows API-ում GetMessage() ֆունկցիայի կանչը հանգեցնում է հավելվածի կանգնեցման՝ մինչև իրադարձության հայտնվելը (սինխրոնություն), մինչդեռ PeekMessage() ֆունկցիան անմիջապես վերադարձնում է վերահսկումը կանչող հոսքին։ Երկու դեպքում էլ իրադարձության առկայության և հերթում գտնվելու պարագայում դրա մշակումը սկսվում է նորից սինխրոն՝ DispatchMessage() ֆունկցիայի կանչով, որը համակարգի առաջադրանքների կառավարչին հանձնարարում է մեկնարկեցնել իրադարձության մշակումը։

Այս վերջին քայլը հանգեցնում է իրադարձության մշակման ֆունկցիայի կանչին, որը հավելվածը նախապես սահմանել է։ Նշենք, որ սինխրոն իրադարձության մշակումը հեշտությամբ կարելի է վերածել ասինխրոն մշակման՝ գործարկելով զուգահեռ գործընթաց (Windows-ում՝ հոսք), որը կստանձնի իրադարձության մշակումը, իսկ հիմնական մշակիչ ֆունկցիան կվերադառնա հավելվածին անմիջապես DispatchMessage() կանչից հետո։

Իրադարձությունների հերթում իրադարձությունների տեղադրումը կարող է իրականացվել նախապես սահմանված զտիչների հիման վրա։

Իրադարձությունների հերթի հարցումը հանգեցնում է իրադարձության դուրս բերմանը հերթից՝ անկախ այն հանգամանքից, թե արդյոք իրադարձությունը մշակվելու է, թե անտեսվելու։

Նկար 13-ում ներկայացված են Windows օպերացիոն համակարգում իրադարձությունների մշակման հիմնական բաղադրիչներն ու սխեման։ Ընդհանուր առմամբ, նշված ֆունկցիաներից յուրաքանչյուրը հանդիսանում է օպերացիոն համակարգին վերահսկողության փոխանցման միջոց և կարող է պատճառ դառնալ հոսքի փոփոխման։ Առավել քան մյուսները՝ GetMessage() ֆունկցիան, քանի որ իրադարձության սպասումը, գործնականում, երաշխավորված կերպով բերում է մեկ այլ հոսքի ակտիվացման։ Սա էլ հենց հանդիսացել է «համագործակցային բազմագործառութայնություն» տերմինի ձևավորման հիմքը՝ հավելվածը ինքնակամ փոխանցում է ռեսուրսը (ՕՀ ֆունկցիայի կանչով)՝ այլ հավելվածների օգտագործման համար։

Սակայն այստեղ կան նաև լուրջ ռիսկեր, որոնք հայտնի են օպերացիոն համակարգերի մշակողներին. հավելվածը կարող է գրավել պրոցեսորը և օգտագործել այն անսահման երկար՝ մինչև անգամ հաշվարկիչի ֆիզիկական մաշվածությունը, եթե արտաքին միջամտությամբ չկասեցվի։ Միակ արդյունավետ միջոցն այս դեպքում ապարատային ընդհատումն է։ x86 պրոցեսորների ճարտարապետությունը այս առումով բավականին պարզեցված է, այդ պատճառով հնարավոր էր շատ հեշտությամբ «կախել» ամբողջ ՕՀ-ը թե՛ հավելվածի, թե՛ նույնիսկ համակարգային ծառայության միջոցով («կապույտ էկրաններ»)։

Մասնավորապես, այն դեպքերում, երբ Caper լեզվի վիրտուալ մեքենային անհրաժեշտ էր ամբողջ պրոցեսորի ռեսուրսը՝ միաժամանակ գործարկվող տասնյակ կամ հարյուր հազարավոր գործընթացների կեղծ-զուգահեռ կատարման համար, օպերացիոն համակարգը «կախվում» էր

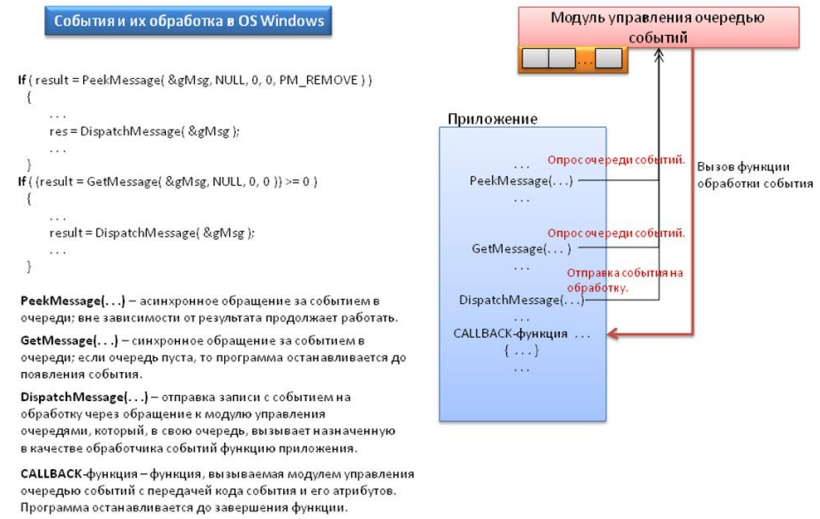
այնքան ժամանակ, քանի դեռ վիրտուալ մեքենան չէր ավարտում այդ բոլոր գործընթացների մշակումն ամբողջությամբ: Պատճառն պարզ է. բավական է չդիմել ՕՀ-ի մոդուլներին: Կարելի է նշել բազմաթիվ նմանատիպ խնդիրներ, երբ ՕՀ-ը պարզապես հեռանում է առաջին պլանից: Այդ թվում են կոմբինատոր խնդիրները, որոնք իրականացնում են որոնում օպերատիվ հիշողության մեջ՝ առանց որևէ ՕՀ ծառայության:

Այսպիսով, «համագործակցային բազմագործառութայնության» տրամաբանությունը կայանում է նրանում, որ հավելվածն ինքն է փոխանցում վերահսկողությունը ՕՀ-ի կառավարման մենեջերին, որը, օգտվելով պահից, կարող է փոխարկել առաջադրանքները կամ հոսքերը: Հակառակ դեպքում, յուրաքանչյուր հավելված ունի հնարավորություն՝ մոնոպոլիզացնելու պրոցեսորի ռեսուրսը, եթե ճարտարապետությունը չունի բարձր առաջնահերթության ընդհատումների հնարավորություններ, որոնք թույլ կտան խլել պրոցեսորը: Սա բացարձակ խախտում է ՕՀ կառուցման սկզբունքների, քանի որ համակարգիչի հիմնական ռեսուրսը կարող է անցնել միայն մեկ հավելվածի անսահման վերահսկողության տակ:

Այս իրավիճակի հիմնական պատճառը հաճախ ապարատային ժամանակաչափի բացակայությունն է, քանի որ հավելվածների միջև հերթափոխի ժամանակ կարելի է (և այդպես էլ արվում էր) նշանակել քվանտ հիշողության համար, որը լրանալուց հետո ընդհատում է հավելվածը:

Նշենք նաև, որ բազմամիջուկ պրոցեսորներն այս խնդիրները մասամբ վերացնում են, քանի որ լավ կառուցված օպերացիոն համակարգերը մեկնարկից սկսած յուրացնում են առնվազն մեկ միջուկ՝ այն բացառելով հավելվածների օգտագործումից: Սա, սակայն, համարվում է հարաբերականորեն «թանկ» լուծում:

Ընդհանուր առմամբ, նման խնդիրներ կարող են առաջանալ նաև այլ պրոցեսորների ճարտարապետությունների դեպքում, որոնք ունեն ընդհատումների ավելի բազմազան հավաքածու՝ կապված համակարգի ադմինիստրատորի անուշադրության կամ սխալի հետ:



Նկար 16. Windows ՕՀ-ում իրադարձությունների մշակման դիագրամ:

Այստեղ մենք չենք նկարագրում իրադարձությունների ստեղծման և տեղադրման ֆունկցիաները՝ ո՛չ հավելվածի սեփական իրադարձությունների հերթում, ո՛չ էլ այլ հավելվածների իրադարձությունների հերթերում: Սակայն արժե նշել, որ այդպիսի ֆունկցիաներ գոյություն ունեն՝ SendMessage() և PostMessage(), որոնք ապահովում են համապատասխանաբար սինխրոն և ասինխրոն իրադարձությունների տեղադրում՝ ինչպես սեփական, այնպես էլ այլ հավելվածների իրադարձությունների հերթերում:

UNIX-ին նմանվող համակարգերում իրադարձությունների հերթերը հնարավոր է կազմակերպել «ձեռնարկային» եղանակով՝ հատուկ ֆունկցիաների օգնությամբ և օգտագործել դրանք Windows-ում նշված ֆունկցիաների նմանակների միջոցով: Հատկապես պետք է ընդգծել, որ Windows համակարգում իրադարձությունների հերթերի ստեղծման և ջնջման ֆունկցիաներ առկա

չեն, քանի որ հերթերը ստեղծվում են ավտոմատ կերպով, իսկ դրանց մշակման ֆունկցիաները հասանելի են API-ում:

3.4. Առաջադրանքների և հոսքերի փոխարկում

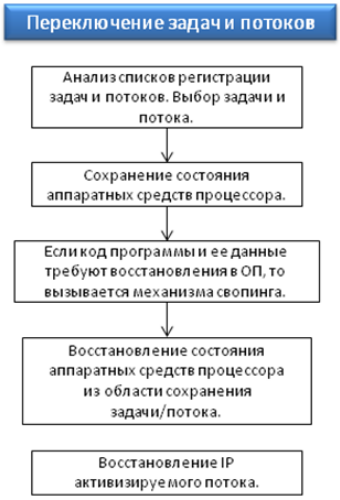
Առաջադրանքից առաջադրանքի, կամ հոսքից հոսքի փոխարկումը իրականացվում է օպերացիոն համակարգի առաջադրանքների սուպերվայզորի (կամ համապատասխան մենեջերի) կողմից՝ արտոնյալ ռեժիմում: Սովորաբար այս գործընթացը ուղեկցվում է օպերացիոն համակարգում իրադարձությունների մեծ մասի արգելափակմամբ կամ նվազեցմամբ, քանի որ փոխարկման ընթացքում տեղի է ունենում պրոցեսորի ապարատային բաղադրիչների հետ աշխատանք, ինչը պահանջում է սուպերվայզորի ուշադրության նվազագույն շեղում:

Միապրոցեսորային համակարգում փոխարկման մեխանիզմը բավականին պարզ է. ըստ օպերացիոն համակարգում սահմանված տրամաբանության (տե՛ս Նկ. 11) կատարվում է առաջադրանքների և հոսքերի գրանցման բլոկների ցուցակի վերլուծություն՝ դրանց առաջնահերթության հիման վրա, և ընտրվում է ակտիվացվող հոսքը: Այնուհետև պահվում է տվյալ պահին գործող առաջադրանքի ապարատային վիճակը (կառավարման ռեգիստրներ, ընդհանուր ռեգիստրներ, քեշ հիշողություն), որը տեղադրվում է առաջադրանքի կառավարման տարածքում:

Այնուհետև վերականգնվում են կատարման համար ընտրված առաջադրանքի (կամ ավելի վաղ դադարեցված հոսքի) վիճակը ներկայացնող տվյալները՝ ներառյալ պրոցեսորի քեշ հիշողության բովանդակությունը, որը պահպանվել էր հոսքի դադարեցման պահին:

Եթե տվյալ առաջադրանքի (կամ հոսքի) մեքենայական կոդը կամ տվյալները բացակայում են օպերատիվ հիշողությունում, ապա իրականացվում են սվոփինգի գործողություններ՝ անհրաժեշտ տվյալների ներբեռնում սվոփ-ֆայլերից օպերատիվ հիշողություն:

Վերջում վերականգնվում է հրամանների հաշվիչի (Instruction Pointer, IP) արժեքը, և պրոցեսորի կողմից հաջորդ կատարվող հրամանը կլինի ակտիվացված հոսքի հրամանը:



Նկար 17. Առաջադրանքների և հոսքերի փոխարկում:

Միջհավելվածային փոխազդեցությունը կարող է իրականացվել ինչպես օպերացիոն համակարգի միջոցով (Windows ՕՀ-ում դա իրականացվում է հաղորդագրությունների միջոցով, որոնք տեղադրվում են նույն կամ այլ հավելվածի իրադարձությունների հերթում, ինչպես նաև՝ հոսքի իրադարձությունների հերթում, եթե այդպիսիք ստեղծվել են հոսքի գործարկման պահին), այնպես էլ հատուկ ծրագրային գրադարանների օգնությամբ, որոնք ապահովում են միջհամակարգչային փոխազդեցություն (օրինակ՝ MPI ստանդարտի վրա հիմնված գրադարաններ):

3.5. Ծրագրային առաջադրանքներ և օպերացիոն համակարգի ռեսուրսների հարցումներ

Օգտագործողի ծրագրերի կատարողականի ընթացքում տարբեր պահերի կարող են պահանջվել օպերացիոն համակարգի տարբեր ռեսուրսներ: Հիշեցնենք, որ օպերացիոն համակարգը կառավարում է այն միջավայրի բոլոր ռեսուրսները, որտեղ աշխատում են հավելվածները: Հետևաբար, բոլոր կիրառական ծրագրերը անհրաժեշտության դեպքում դիմում են համապատասխան ծառայություններին:

Ռեսուրսների հարցման տրամաբանությունը կարելի է ներկայացնել պարզ սխեմայով՝ համապատասխան ֆունկցիաների միջոցով.

- ռեսուրսի հարցում,
- ռեսուրսի օգտագործում՝ ծառայության կանչի միջոցով,
- ռեսուրսի ազատում:

Որոշ դեպքերում ռեսուրսի օգտագործումը կարող է իրականացվել առանց օպերացիոն համակարգի ծառայությունների մասնակցության: Օրինակ՝ հնարավոր է հիշողության հարցում՝ հիշողության կառավարման ծառայությունից, բայց հիշողության օգտագործումը կարող է իրականացվել անմիջապես ծրագրից՝ ցուցիչների (pointers) միջոցով: Սակայն, ֆայլերի հետ աշխատանքի դեպքում՝ ֆայլը կամ նրա հատվածը օպերատիվ հիշողության բուֆերում տեղակայելուց հետո, կարդալու և գրելու բոլոր գործողությունները կատարվում են ՕՀ ծառայության միջոցով:

Երբ ռեսուրսը այլևս անհրաժեշտ չէ, այն պետք է ազատել, չնայած՝ առաջադրանքի ավարտից հետո օպերացիոն համակարգը ինքնուրույն ազատում է հավելվածին տրամադրված բոլոր ռեսուրսները (առնվազն՝ պարտավոր է): Սակայն «բարեխիղճ ծրագրավորման» և պրոֆեսիոնալիզմի նշան է, երբ հավելվածն ինքն է ազատում այլևս անհրաժեշտ չեղած ռեսուրսները: Դա վերաբերում է ինչպես հիշողության բաժիններին, այնպես էլ բացված ֆայլերին և այլ ռեսուրսներին:

Մեկ այլ ռեսուրս, որով կարող է օգտվել հավելվածը, դա դինամիկ գրադարաններն են: Դրանք նույնպես բեռնվում և ազատվում են օպերացիոն համակարգի ծառայության միջոցով՝ հավելվածի կողմից համապատասխան ֆունկցիայի կանչով: Սակայն նման գրադարանները դառնում են օպերացիոն համակարգի ռեսուրս (յուրաքանչյուրը՝ միակ օրինակով) և կարող են տրամադրվել այլ հավելվածներին: Այդ պարագայում գրադարանի ազատումը կամ հեռացումը կարող է դառնալ անհնար:

Պետք է ընդգծել, որ որոշ առանձին կիրառական ծրագրեր և նույնիսկ ամբողջական հավելվածներ կարող են դառնալ օպերացիոն համակարգի ռեսուրս: Այստեղ դիտարկվում է ռեսուրսի երկակի բնույթը՝ երբ կիրառական ծրագիրը կամ տվյալները, որոնք սկզբում դիտարկվում են որպես հավելվածի ռեսուրս, դառնում են օպերացիոն համակարգի կողմից կառավարվող ռեսուրս:

Ի վերջո, որոշ տեսանկյունից, դա վերաբերում է նաև կիրառական ծրագրերին, քանի որ բեռնավորման պահից դրանք որոշ իմաստով դառնում են օպերացիոն համակարգի վերահսկողության տակ գտնվող ռեսուրսներ:

Գլուխ 4. Հիշողություն և դրա կառավարում

Օպերատիվ հիշողության կառավարումը ցանկացած ունիվերսալ հաշվարկչային համակարգի ամենակարևոր և ամենաբարդ բաղադրիչներից է։ Այն նույնիսկ ավելի կարևոր է, քան մյուս սարքավորումները, այդ թվում՝ պրոցեսորը, քանի որ հենց օպերատիվ հիշողությունում են տեղակայվում ծրագրերը, և հենց այդտեղ են պահվում դրանց տվյալները, իսկ յուրաքանչյուր ծրագիր «խանգարում» է մյուսներին։

Խնդիրները սկսվում են հերթական ծրագրի բեռնման պահից. ինչպես և որտեղ տեղակայել այն հիշողությունում, ինչպես տեղավորել բոլոր անհրաժեշտ տվյալները՝ ինչպես նախնական, այնպես էլ հաշվարկման ընթացքում առաջացող։ Միևնույն ժամանակ պահպանվում է պահանջը՝ օպերատիվ հիշողության մուտքը պետք է լինի հնարավորինս արագ, քանի որ դրանից է կախված հաշվարկների կատարման արագությունը, և այդպիսով՝ հաշվիչի ընդհանուր արդյունավետությունը։

4.1. Հիշողության ձևավորումը հաշվարկիչներում և հավելվածներում

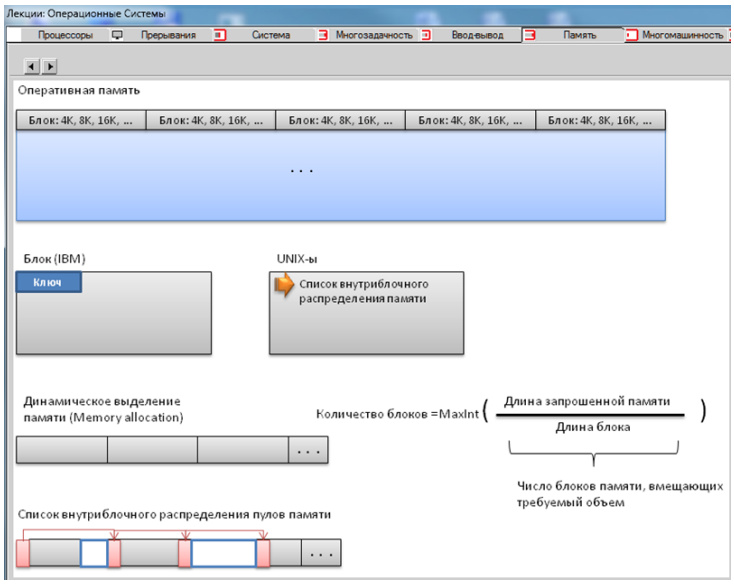
Հիշողության կազմակերպման վրա մեծ ազդեցություն է թողել IBM-360 համակարգի կողմից կիրառված համակցված հիշողության մոդելը, որն իրականացվում էր էլեկտրոնային բանալիներով բլոկների տեսքով և ամբողջ հիշողության բաժանումով՝ ըստ բանալիների՝ ՕՀ-ի և հավելվածների միջև։ Այստեղ բանալու երկարությունը (սկզբնական տարբերակում՝ 4 բիթ 4 ԿԲ-անոց բլոկի համար) կարևոր չէ։ Կարևոր է հիշողության բլոկային տրամաբանությունը՝ հիշողության հատկացման ընթացքում, այլ ոչ թե առանձին բայթերի խմբերով։

Հիշողության էլեկտրոնային բանալին ունի պաշտպանիչ գործառույթ, երբ հիշողությունն օգտագործվում է տարբեր հավելվածների կողմից։ Այս տրամաբանությունը ժառանգաբար անցել է UNIX ՕՀ-ին, իսկ հետո՝ նաև դրա ածանցյալներին։

Արտաքին միջամտությունից պաշտպանությունը յուրաքանչյուր բայթի մակարդակով՝ առանց բլոկային կազմակերպման, մեղմ ասած՝ բարդ է (պահանջվում է գրանցել հսկայական քանակությամբ հատվածներ և «փոքր հատվածներ», ինչը կարող է պահանջել ավելի շատ հիշողություն, քան այդ հատվածների ընդհանուր ծավալը)։ Այդ իսկ պատճառով պաշտպանվում են բլոկները կամ դրանց համակցությունները։

Բլոկների չափերը կարող են տարբեր լինել՝ կախված համակարգչային ճարտարապետությունից և դրա հնարավորություններից։ Մեծ բլոկներ ունեցող հիշողությունների պարագայում դինամիկ հարցվող հիշողությունը կարող է տրամադրվել այդ բլոկներից, որոնք արդեն նախապես պահված են հավելվածի համար։ Այստեղ կա որոշակի ավելցուկայնության վտանգ. բլոկը կարող է լինել ավելին, քան հավելվածին անհրաժեշտ է։ Բայց իդեալական լուծումներ գոյություն չունեն. ալգորիթմավորումն ու ծրագրավորումը միշտ էլ բալանսների և փոխզիջումների որոնում է։

Ինչպես արդեն նշվեց, որոշ ՕՀ-ներում հիշողության բլոկների ցուցակները տեղակայվում են նույն բլոկներում, ինչը նրանց կառավարման բլոկները դարձնում է խոցելի՝ հենց հավելվածի կողմից (տե՛ս ստորև)։



Նկար 18. ՕՀ-ում հիշողության բաշխման մոդելը:

Հիշողության բլոկների էլեկտրոնային բանալիներն ու մուտքի վերահսկման ապարատային մեխանիզմները թույլ են տալիս ավելի «ազատորեն» բաշխել հիշողությունը (ինչը լայնորեն կիրառվում է IBM համակարգերում) և կիրառել տարբեր մոդելներ հիշողության կառավարման համար: Սակայն իդեալական սխեմաներ գոյություն չունեն: Ամեն անգամ օպերացիոն համակարգի մշակման ընթացքում ուսումնասիրվում և նախագծվում են (երբեմն՝ բավականին բարդ) ալգորիթմներ՝ հիշողության բաժինների հատկացման և ազատման համար:

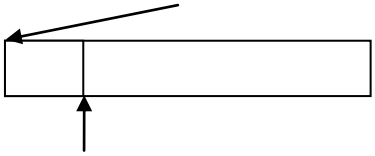
Պատահական (ոչ հարակից) հիշողության բաժինների տեղաբաշխումը ստացել է անվանումը՝ ոչ հարակից հիշողության կառավարում: Գոյություն ունի նաև հարակից հիշողության կառավարում, որի դեպքում ստուգվում է ազատվող հատվածի հարևանությունը, և եթե գոնե մեկը հարակից բաժիններից ազատ է, ապա այդ բաժինները միավորվում են:

Հիշողության հատկացման և ազատման գործընթացներում առաջանում են տարբեր չափերի «խորշեր»՝ հիշողության անկանոն ֆրագմենտացիա: Սա հանգեցնում է հիշողության կառավարման վրա ծախսվող ժամանակի աստիճանական աճի: Այդ պատճառով որոշ օպերացիոն համակարգերում կիրառվում են դեֆրագմենտացման ծառայություններ, որոնց նպատակը հիշողության հատվածների քանակի նվազեցումն է: Դրանք գործում են պարբերաբար՝ հիմնվելով նախապես սահմանված հեուրիստիկ կանոնների վրա: Մոտավորապես այսպես են աշխատում նաև արտաքին հիշողության դեֆրագմենտացման ծառայությունները՝ հիմնականում սկավառակների համար:

Հիշողության հատկացման ժամանակ կառավարման ծառայությունը (օրինակ՝ C/C++ լեզուներում սա իրականացվում է malloc(), calloc() ֆունկցիաներով, իսկ realloc() ֆունկցիան օգտագործվում է հիշողության հատվածի չափը փոփոխելու և նրա բովանդակությունը պահպանելու համար) որոնում է համապատասխան չափի հատված՝ տարբեր բաժիններում: Եթե հիշողության կառուցվածքը բլոկային է, ապա, ինչպես նշվեց, յուրաքանչյուր բլոկ ստուգվում է՝ արդյոք հնարավոր է հատկացնել հիշողություն դրա ազատ մասերից:

calloc() ֆունկցիան առանձնանում է նրանով, որ հատկացված բոլոր բայթերը ինիցիալիզացնում է (լռությամբ՝ զրոյացնում): UNIX-ին նման օպերացիոն համակարգերում, ինչպես նաև Windows-ում (որը ժառանգել է այս լուծումը), հատկացված հիշողության հատվածից առաջ տեղադրվում է կառավարման բլոկ՝ հիշողության պաշարի ցուցակի տարրի համար:

Կառավարման բլոկ հիշողության պաշարի հաշվառման ցանկում



Pointer - Նկարում նշված ուղղահայաց սլաքը ցույց է տալիս հատկացված հիշողության ցուցիչը:
pointer = malloc(...); // ցուցիչ հատկացված հատվածին

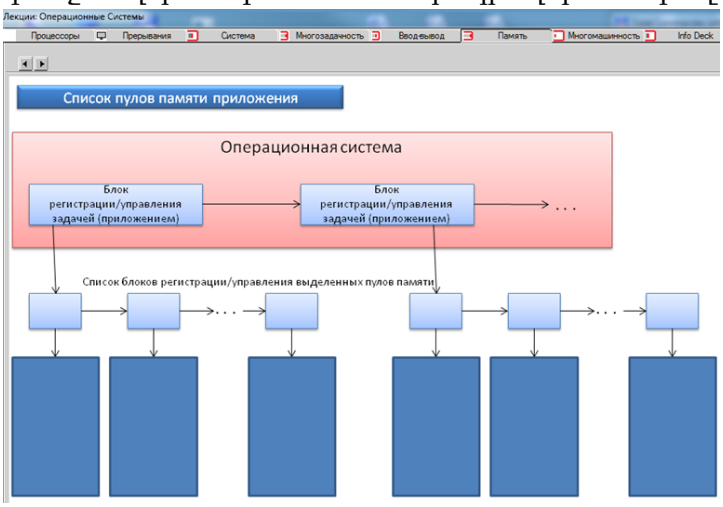
Ցուցիչից «ձախ» կողմ կատարված սխալով տեղաշարժը (օրինակ՝ բացասական ինդեքսով) կարող է վնասել կառավարման բլոկի տվյալները, ինչը կբերի հիշողության հատկացման (malloc) և ազատման (free) ֆունկցիաների վթարային ավարտին: Պատճառն ակնհայտ է. ցուցակները միաուղղ են, և երբ ցուցակի տարրերն աղավաղվում են, ապա հաջորդ ֆունկցիաների կատարման ժամանակ ուղին խախտվում է:

Ինչպես արդեն նշվել է, պատճառը պարզ է. խախտվում է օպերացիոն համակարգերի կառուցման հիմնարար սկզբունքներից մեկը՝ համակարգային ռեսուրսների կառավարման տվյալները տեղակայված են կիրառական ծրագրի տեսանելի տարածքում, այսինքն՝ հասանելի են: Իսկ երբ համակարգային ֆունկցիան փորձում է օգտագործել այդ տվյալները, դրանք արդեն կարող են վնասված լինել, ինչը անընդունելի է օպերացիոն համակարգերի համար (տե՛ս ՕՀ-ի պահանջները):

Առկա է նաև օգտակար առաջարկություն՝ աշխատել միայն զանգվածների տարրերի ուղղակի հասցեավորմամբ, այսինքն՝ ինդեքսներով (array[i])՝ առանց ցուցիչների օգտագործման, հատկապես՝ երբ հիշողության պուլերի հետ է կապված: Սա վերաբերում է նաև կառուցվածքներին (structures), և ցանկալի է նաև ապահովել ինդեքսների արժեքների վերահսկում, եթե ծրագրավորման միջավայրը դա թույլ է տալիս: Այսինքն՝ ապահովել, որ ինդեքսը լինի սահմանված միջակայքում (օրինակ՝ i >= 0 և i < N, որտեղ N-ը զանգվածի չափն է):

Եթե վերահսկում չկա, ապա C լեզվով, օրինակ, արտահայտությունը array[i] = 10; կարող է հանգեցնել սխալի, եթե i-ի արժեքը դուրս է սահմաններից: Որոշ կոմպիլյատորներ հնարավորություն են տալիս ավտոմատ կերպով ներառել նման ստուգումներ՝ համապատասխան պարամետրերի սահմանման միջոցով:

Ընդհանուր առմամբ, ցուցիչները վտանգավոր գործիք են, որը պահանջում է զգուշություն և մեծ ուշադրություն: Նույնիսկ դինամիկ հիշողությունում փոփոխականների հետ կարող են խնդիրներ առաջանալ, եթե, օրինակ, վերցվի փոփոխականի հասցեն և ապա տվյալներ գրանցվեն այդ հասցեում՝ առանց պատշաճ վերահսկման: Սա ևս ծրագրավորման սխալների հետևանք է:



Նկար 19. Գրանցման և հիշողության բաշխման մոդել:

4.2. Օվերլեյներ և սվոփինգ

Հաշվարկման գործընթացում օպերատիվ հիշողության պակասը համակարգիչների ստեղծման օրերից սկսած համարվում է լուրջ խնդիր: Ժամանակի ընթացքում այս խնդիրը սկսել են լուծել կիրառական ծրագրավորողները՝ կիրառելով այսպես կոչված օվերլեյներ: Սակայն դա հավելյալ բարդություններ է ստեղծել ծրագրերի մշակման գործընթացում:

Օվերլեյների հիմնական գաղափարն այն է, որ ծրագիրը կառուցվի այնպես, որ հնարավոր լինի դրա որևէ մոդուլ բեռնաթափել օպերատիվ հիշողությունից և դրա փոխարեն ներբեռնել մեկ այլ մոդուլ: Մեծապես հենց այս գաղափարից է ծագել Նիկլաուս Վիրտի ծրագրերի մոդուլայնության վերաբերյալ կոնցեպցիան:

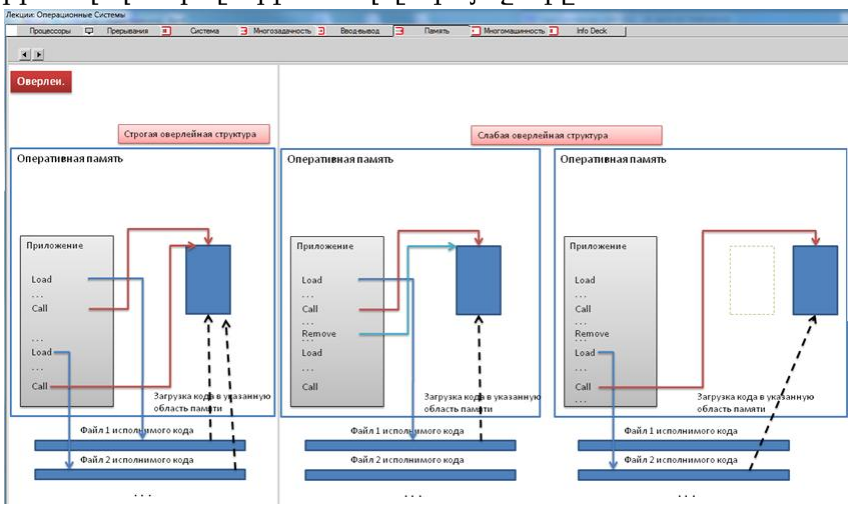
Նկարում ներկայացված են օվերլեյների երկու տարբերակ՝ «խիստ» և «թույլ» :

- Խիստ օվերլեյի դեպքում օպերատիվ հիշողության մեջ նախապես առանձնացվում է մի հատված, որտեղ կարող են հերթով տեղադրվել բոլոր օվերլեյ մոդուլները: Մոդուլները հերթով բեռնվում են այդ հատվածում՝ արտաքին հիշողությունից, օգտագործվում են, ապա նույն տարածքում տեղադրվում է հաջորդ մոդուլը: Եթե նախորդ մոդուլի աշխատանքը դեռ ավարտված չէ և անհրաժեշտ է պահպանել նրա վիճակը հետագայում վերականգնելու համար, ապա այն նախապես բեռնաթափվում է:

Ծրագրավորման պրակտիկայում այս մոտեցման զարգացման արդյունքում նման տեխնոլոգիան, որոշ փոփոխություններով, ընդգրկվել է IBM-ի օպերացիոն համակարգերում: Սկզբնական շրջանում օպերացիոն համակարգն աշխատում էր ֆիքսված երկարության էջերով (memory pages), որոնք պահվում էին սկավառակներում: Օպերատիվ հիշողությունում նախապես նախատեսվում էր ֆիքսված հատված այդ էջերի համար՝ ինչպես ներբեռնելու, այնպես էլ հետագա բեռնաթափման նպատակով:

- Թույլ օվերլեյը նույնպես հիմնված է ներբեռնման և բեռնաթափման սկզբունքի վրա, սակայն այն թույլ է տալիս մոդուլները բեռնել օպերատիվ հիշողության տարբեր հատվածներում՝ առանց պարտադիր նույն տարածքը կրկնակի օգտագործելու: Սա կիրառվում է այն դեպքերում, երբ նախապես հատկացված տարածքը բավարար չէ նոր մոդուլի բեռնելու համար: Եթե ակնկալվում է, որ մոդուլը կրկին վերադարձվելու է հիշողություն, ապա տարածքը կարող է պահվել (ռեզերվավորվել): Սակայն կրկնենք, որ նման մոտեցումը լիարժեք «օվերլեյ» չի համարվում:

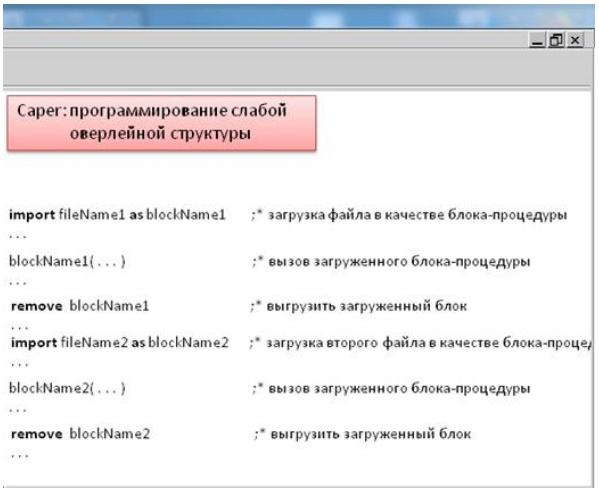
Խիստ օվերլեյի պահպանման նպատակով կարելի է օպերացիոն համակարգին հատուկ առաջադրանքով նախապես պահանջել օվերլեյների համար անհրաժեշտ հիշողության տարածք՝ հաշվի առնելով ներբեռնվող մոդուլների առավելագույն չափը:



Նկար 20. Կոդի բեռնման գործընթացներ և հիշողության կառավարում:

Օրինակով ներկայացված է Caper լեզվով գրած կոդի հատված, որը ցուցադրված է Նկարում: Այդ կոդն ընդգրկում է ծրագրային մոդուլի ներբեռման (import <ֆայլի անունը> as <պրոցեդուրային բլոկի անունը>) և բեռնաթափման (remove <պրոցեդուրային բլոկի անունը>) հնարավորություններ: Ներբեռնվող մոդուլը դիտարկվում է որպես անունով ներկայացված պրոցեդուրաների ամբողջություն, որը նույնպես հանդիսանում է պրոցեդուրա:

Այս մեխանիզմը, ըստ էության, բնութագրվում է թույլ օվերլեյայնությամբ, երբ մոդուլների ներբեռնումը և հեռացումը իրականացվում է ծրագրի գործարկման ընթացքում՝ առանց հիշողության խիստ նախապես ամրագրված տարածքների:



Նկար 21. Caper-ում թույլ վերադրման ծրագրավորում:

4.3. Վիրտուալ հիշողության կազմակերպում

Հաշվարկների օվերլեյային կազմակերպման որոշակի ժառանգորդ է օպերացիոն համակարգերում վիրտուալ հիշողության կառուցման եղանակը:

Հիշողության բաշխման մեջ առավել տարածված մեթոդներից մեկն է այսպես կոչված էջային կազմակերպման մեթոդը: Այս մոտեցումը մշակվել է IBM ընկերության կողմից՝ SVS (Single Virtual Storage) վիրտուալ հիշողության համակարգի տարբերակի ստեղծման ժամանակ, որը շուտով վերափոխվեց VM (վիրտուալ մեքենաների համակարգ՝ խորհրդային տերմինաբանությամբ):

Փոքր հաշվիչ համակարգերը՝ x86 տիպի պրոցեսորներով, փոխհատուցում էին «բարդ» (հետևաբար՝ թանկարժեք) հիշողության բացակայությունը՝ օգտագործելով վիրտուալ հասցեավորում, որը հատուկ փոխակերպիչների օգնությամբ վերածվում էր իրական հասցեների: Այդ փոխակերպիչները հիմնվում էին հիշողության սեգմենտների աղյուսակների վրա, որոնք սահմանվում էին երկու բայթանոց ռեգիստրներում (տե՛ս Նկ.): Այս մոտեցումն ընդլայնվեց նաև DEC ընկերության կողմից ստեղծված VAX համակարգիչների համար նախատեսված օպերացիոն համակարգերում:

Էջային հիշողության գաղափարը կառուցվում է հետևյալ տրամաբանությամբ.

- ամբողջ ֆիզիկական (իրական) հիշողությունը բաժանվում է ֆիքսված երկարությամբ, չհատվող հատվածների (էջերի),
- ծրագրերը և դրանց տվյալները բեռնվում են իրենց հատկացված էջային տարածքում,
- հիշողության հետ բոլոր գործողությունները սահմանափակվում են միայն այդ հատկացված տարածքով և ներսում եղած հասցեներով,
- այն դեպքերում, երբ տվյալ տարածքի չափը բավարար չէ ծրագրի ամբողջ կոդի կամ տվյալների տեղաբաշխման համար, իրականացվում են դրանց մասայական բեռնումներ:

Այս նպատակով սկավառակային ֆայլեր են ձևավորվում՝ օպերատիվ հիշողությունից դուրս բերվող ծրագրային և տվյալային հատվածները ժամանակավոր պահելու համար, ինչը հնարավորություն է տալիս օպերատիվ հիշողությունը ազատել նոր ֆրագմենտների ներբեռնումների համար: Այս գործընթացը կոչվում է սվոփինգ (SWAP՝ անգլերեն «փոխանակել», ոչ հապավում):

Ավելի վաղ գործարկվող մեխանիզմներից է հիշողության դամփինգը (dump)՝ օպերատիվ հիշողության կամ նրա հատվածների բովանդակության ուղղակի արտահանում արտաքին հիշողության վրա: Սովորաբար այն կիրառվում էր հիշողության բովանդակությունը դիտելու և սխալները գտնելու նպատակով: Սակայն այս տեքստի հեղինակը օգտագործել է դամփինգը՝ պարբերաբար արտաքին հիշողության վրա հավելվածների ընթացիկ վիճակների պահման համար, որպեսզի վթարային իրավիճակներից (օրինակ՝ հոսանքի անջատումից) հետո հնարավոր լինի վերականգնել վիճակը: Այս մեթոդը կարևոր է հատկապես երկարատև հաշվարկների դեպքում:

Սվոփինգի և դամփինգի գործընթացները կարող են իրականացվել տարբեր եղանակներով և տարբեր կրիչների վրա: Կարևոր է, որ այդ կրիչները լինեն բարձր արագությամբ՝ օրինակ HDD կամ SSD: Վերջիններս բարձր արագություն ապահովելուց զատ ունեն նաև թերություն՝ գրանցումների սահմանափակ քանակ: Արտաքին կրիչի արագագործությունը առանցքային դեր ունի սվոփինգի ընթացքում առաջացող դադարների տևողության վրա:

Էջայնացման մեխանիզմի հիմնական առավելություններից մեկն է էջերի ֆիքսված չափսը օպերատիվ հիշողությունում, ինչը հնարավորություն է տալիս այդ էջերը պահպանել կամ կարդալ «մեկ շարժումով»՝ միևնույն գրանցման կամ ընթերցման գործողությամբ:

Օպերացիոն համակարգերում հաճախ կիրառվում է արդեն վերլուծված հիշողության հաջորդական բլոկների միացման տրամաբանությունը, որոնք նույնպես կոչվում են էջեր:

Այդուհանդերձ, էջերում կարող են ավելացվել նոր տվյալներ կամ հեռացվել արդեն առկա տարրեր: Տվյալների այս դինամիկ փոփոխությունները ժամանակի ընթացքում հանգեցնում են հիշողության ֆրագմենտացիայի՝ ինչպես օպերատիվ հիշողությունում, այնպես էլ սվոփ ֆայլերում: Սա իր հերթին դանդաղեցնում է տվյալների որոնման և մշակման գործընթացը:

Այդպիսի դեպքերում, հեուրիստիկ կանոնների հիման վրա իրականացվում է դեֆրագմենտացիա՝ էջերի և բլոկների վերակազմակերպում, որը ժամանակատար գործողություն է:

Հաշվի առնելով այս հանգամանքը՝ բացի փոփոխականների հետ աշխատանքն ու դրանց վերահսկումը հեշտացնելու նպատակից (հիշեցնենք, որ C/C++ լեզուներում դինամիկ ստեղծվող բարդ փոփոխականները պահանջում են ազատում դրանց օգտագործումից հետո, ինչը պահանջում է հավելյալ ուշադրություն), որոշ տարածված ծրագրավորման լեզուներում ներդրվում է ընդլայնվող ներքին հիշողություն, որտեղ հաշվառվում են փոփոխականները և դրանց հետ կապված բոլոր տվյալները:

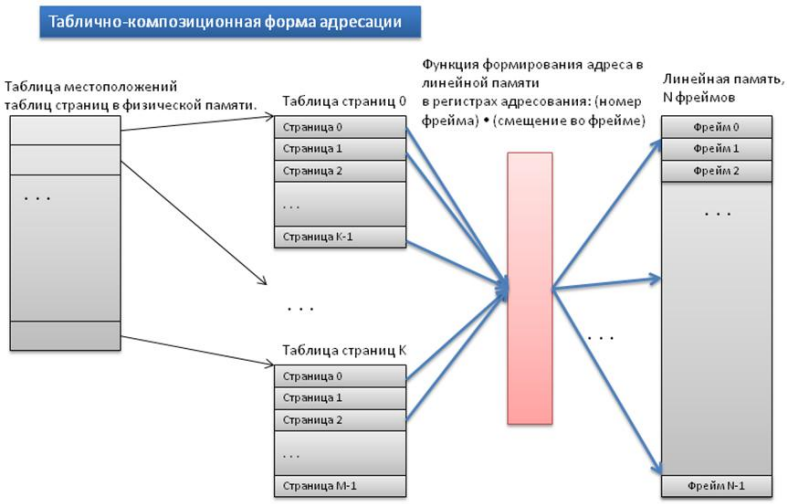
Այսպիսի լեզուները հիմնված են սեփական վիրտուալ մեքենաների առկայության վրա, որոնք նաև կառավարում են հավելվածների հիշողությունը: Այդ միջավայրերում կիրառվում է «աղբահանության» (garbage collection) գործընթացը, որի տրամաբանությունն ի սկզբանե մշակվել է LISP լեզվում:

Դեֆրագմենտացիայի գործընթացը մեծապես նման է հիշյալ մեխանիզմներին, սակայն բարդանում է այն դեպքում, երբ տարբեր զուգահեռ հոսքեր կիսում են ընդհանուր տվյալների դաշտեր: Այս պարագայում անհրաժեշտ է վարել օգտվողների հաշվառում, կիրառել վերահսկման հաշվիչներ և այլ մեխանիզմներ:

Վիրտուալ հիշողությունը և վիրտուալ հասցեավորման համակարգերը ձևավորվում են բազմաշերտ կառուցվածքով՝ տարբերակների միջոցով: Սովորաբար ամեն ինչ սկսվում է էջերի (ֆրեյմերի) աղյուսակից, որը գտնվում է ֆիզիկական հիշողությունում: Յուրաքանչյուր վիրտուալ հասցե բաղկացած է էջի (ֆրեյմի) համարից և էջի ներսում տեղաշարժից: Այս երկուսի միջոցով ձևավորվում է հասցե՝ իրական օպերատիվ հիշողությունում:

Այսպիսով, ձևավորվում են հավելվածների հասցեական տարածքներ, որոնք փակվում են էջերի համակցված կառուցվածքով:

Ընդհանուր առմամբ, այստեղ գործ ունենք հարաբերական հասցեավորման տարբերակի հետ:



Նկար 22. Վիրտուալ հասցեն ֆիզիկական հասցեի փոխակերպումը:

Ընդհանուր առմամբ, ժամանակակից 32-բիթային ճարտարապետություններում մենք ունենք 16 բիթ էջերի համարակալման և 16 բիթ՝ offset-ի համար. մենք ունենք 4 ԳԲ հասցեական տարածք (երբեմն սահմանափակվում է 2 ԳԲ-ով):

4.4. Վիրտուալ մեքենաներ և վիրտուալ օպերացիոն համակարգեր

Վիրտուալ մեքենա (VM, ՎՄ) կոչվում է ծրագրային համալիր, որը հիմքում ունի սեփական մոտեցումը հաշվարկների ներկայացման և իրագործման առումով, և որը թույլ է տալիս իրականացնել այլ ծրագրերի կողմից սահմանված գործողությունները՝ ծրագրավորման լեզուների օգնությամբ: Վիրտուալ մեքենաները սովորաբար ունեն իրենց հատուկ լեզու, որը կառուցվածքով մոտ է ասեմբլերներին. այս լեզուն տարբեր համատեքստերում կոչվում է՝ «բայթ-կոդ» (օրինակ՝ Java-ում), «միջանկյալ լեզու» (Intermediate Language, օրինակ՝ C#-ում կամ F#-ում): Այս տեքստի հեղինակն առաջարկում է սեփական անվանումը՝ «կեղծ-ասեմբլեր», օրինակ՝ Caper լեզվի համար:

Վիրտուալ մեքենաների հիմնական նպատակը և դրանց տարբերությունը պրոցեսորների ասեմբլերներից այն է, որ դրանք հնարավորություն են տալիս մեկնաբանելու ցածր մակարդակի հրահանգները՝ սեփական տրամաբանությամբ:

Վիրտուալ օպերացիոն համակարգերը (ՕՀ) իրենց կառուցվածքով կրում են բոլոր հիմնական հատկանիշները, որոնք բնորոշ են սովորական օպերացիոն համակարգերին: Սակայն, վերջիններս հիմնվում են հիմքային օպերացիոն համակարգի վրա՝ օգտագործելով դրա ռեզիդենտ բաղադրիչները՝ մասնավորապես՝ միջուկի տարրերը, ֆիզիկական սարքավորումների սպասարկման մեխանիզմները, ընդհատումների նախնական մշակումն ու այլ ռուտինային ծառայությունները: Նշենք, որ ընդհատումների մեծ մասը փոխանցվում է վիրտուալ օպերացիոն համակարգին:

Միևնույն ժամանակ, անհրաժեշտ է ընդգծել, որ վիրտուալ օպերացիոն համակարգն ունի իր առանձնահատուկ տրամաբանությունը՝ հաշվողական ռեսուրսների և գործընթացների կառավարման տեսանկյունից, որը կարող է էապես տարբերվել հիմքային ՕՀ տրամաբանությունից: Հակառակ դեպքում, վիրտուալ ՕՀ ներդրումը իմաստ չի ունենա:

Գլուխ 5. Օպերացիոն համակարգերի ծրագրերի մշակման առանձնահատկությունները

Օպերացիոն համակարգերի բազմաառանցքային և բազմաթելային բնույթի պայմաններում առանձնահատուկ նշանակություն է ստանում այն ծրագրավորման տեխնիկան, որն ուղղված է ՕՀ բաղադրիչների նախագծմանը, որոնք պետք է արձագանքեն տարբեր իրադարձությունների կամ ՕՀ-ին ուղղված դիմումների՝ գալիս դրանք լինեն հավելվածներից։

Բնական հարց է առաջանում՝ ինչպիսի՞ն պետք է լինի օպերացիոն համակարգերի ծրագրերի կազմակերպումը, որպեսզի դրանք կարողանան ապահովել միաժամանակ իրականացվող բազմաթիվ դիմումների (թելերի/առաջադրանքների) մշակում։

Այս խնդրի լուծման համար կիրառվում են մի շարք տեխնիկա, մասնավորապես՝

- ծրագրերի ռեենտերականություն (reenterability),
- ծրագրային մոդուլների կրկնօրինակման մեթոդներ,
- մյութեքսների (mutexes) օգտագործում։

Խնդիրն այն է, որ բազմաառաջադրանքային համակարգերում, երբ միևնույն ծրագրին կամ նրա որոշ հատվածներին հնարավոր է դիմում կատարեն տարբեր ծրագրեր նույն պահին կամ կարճ ժամանակային ընդմիջմամբ, կարող են առաջանալ կոնֆլիկտներ, քանի որ բոլոր այդ դիմումները կարող են փոխել ընդհանուր հիշողության նույն հատվածները։

Նշենք, որ ստորև ներկայացված մեթոդները կիրառելի են ոչ միայն օպերացիոն համակարգերում, այլև լայնորեն օգտագործվում են կիրառական ծրագրերի մշակման մեջ։

5.1. Ռեենտերական ծրագրեր

Երբ կանչվող ծրագիրը օգտագործում է տվյալներ (փոփոխականներ, հիշողության հատվածներ և այլն), որոնք ունեն ստատիկ բնույթ (այսինքն՝ դրանք ամրագրվում են ծրագրին դրա գործարկման սկզբում և մնում են ակտիվ մինչև ավարտ), կամ երբ այն փոխազդում է սարքավորման հետ, ապա նոր կանչը կարող է խաթարել արդեն ընթացող գործողությունները, փոփոխել տվյալները և խախտել ծրագրի ընթացքը։

Այս պատճառով, այն ծրագրերը, որոնք ենթակա են բազմակի կանչերի, պետք է լինեն ռեենտերական, այսինքն՝ ունակ լինեն ապահովել կանչերի անվտանգ և անկախ մշակում։

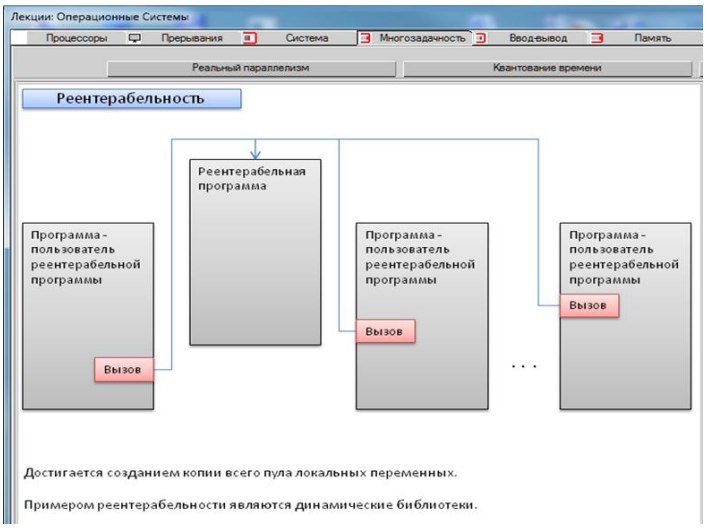
Լուծումները հետևյալն են.

- Չկիրառել ստատիկ տվյալներ կամ այդ տվյալներն օգտագործել այնպես, որ չազդեն հիմնական ալգորիթմի ընթացքի վրա։
- Օրինակ՝ կարելի է կիրառել հաշվիչ փոփոխական, որը հաշվարկում է ծրագրի մուտքերի քանակը՝ աճելով մուտքի և նվազելով ելքի ժամանակ։

Ռեենտերական ծրագրերում յուրաքանչյուր կանչի համար ստեղծվում են նոր օրինակներ փոփոխականներից և համապատասխան տվյալների դաշտերից։ Պարզ փոփոխականները տեղադրվում են կանչի կույտի (stack) մեջ, իսկ զանգվածները և կառուցվածքները պահանջում են դինամիկ հիշողության հատկացում։

Ռեենտերական ծրագրերում բացառվում է նաև ծրագրի կոդի փոփոխումը՝ ինչպես ինքն իրենով, այնպես էլ արտաքին մոդուլների միջոցով։ Դա հնարավոր չէ ժամանակակից այն ճարտարապետություններում, որտեղ ծրագրային կոդն ու տվյալները պահվում են տարբեր հիշողական հատվածներում և ծրագրային հատվածները ունեն գրառման արգելք։

Այսպիսով, բազմակի կանչերը հաճախ հնարավոր է սպասարկել մեկ ընդհանուր ծրագրային կոդի միջոցով, եթե այն նախագծված է որպես ռեենտերական։



Նկար. 23. Ռեենտերական ծրագրերի կանչեր տարբեր պահերին

5.2. Ծրագրային մոդուլների պատճեններ

Ռեենտերականության այլընտրանքային լուծումներից մեկն ավելի պարզ և ակնհայտ մեթոդ է՝ յուրաքանչյուր կանչի համար ստեղծվում է համապատասխան ծրագրային կոդի սեփական օրինակ: Այս մոտեցման դեպքում ծրագրի սկզբնական օրինակը կարող է արդեն գտնվել օպերատիվ հիշողության մեջ կամ դեռևս ներբեռնվել արտաքին կրիչից:

Նկար 14-ում ներկայացված են ծրագրային մոդուլների ներբեռնման մեխանիզմները, որոնք օգտագործվում էին IBM-ի ՕՀ-ում՝

- Load (ներբեռնում՝ անհրաժեշտ մոդուլի բեռնում օպերատիվ հիշողություն),
- Copy (արդեն բեռնված մոդուլի պատճենում հիշողության այլ հատվածում),
- Call (հիշողության մեջ գտնվող մոդուլի կանչ),

ինչպես նաև fork և exec հրամանները, որոնք կիրառվում են UNIX-համանման օպերացիոն համակարգերում:

Այս մեխանիզմների հիման վրա էլ կառուցվել է մուլտիֆունկցիոնալությունը (մուլտիթասքինգը) UNIX համակարգերում, երբ ստեղծվում էր ոչ միայն օգտվողի ծրագրի պատճեն, այլև ՕՀ միջուկի ընթացիկ վիճակի պատճենը, ինչպես նաև որոշ ծառայությունների:

5.3. Մյութեքսներ

Միաժամանակ միևնույն ծրագրային կոդին ուղղված բազմաթիվ դիմումների խնդիրների լուծման ևս մեկ՝ ավելի ուշ ի հայտ եկած արդյունավետ մեխանիզմ է մյութեքսների (mutexes) կիրառումը:

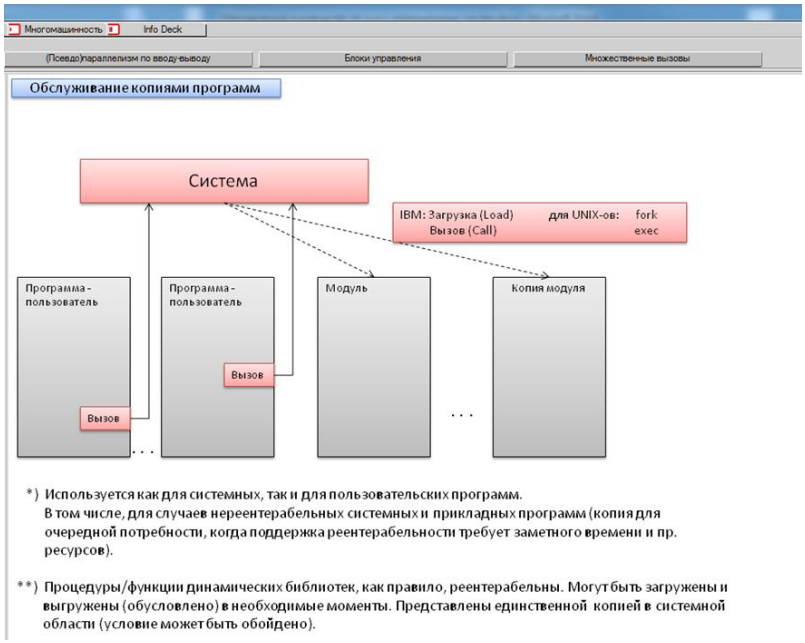
Մյութեքսները սովորաբար ներկայացվում են որպես ծրագրի որոշակի ֆունկցիաներ, պրոցեդուրաներ կամ կոդի հատվածներ, որոնք կարող են սահմանափակել միաժամանակյա մուտքը ընդհանուր ռեսուրսներին:

Մյութեքսը գրանցվում է օպերացիոն համակարգում՝ կիրառական ծրագրի կողմից, որը ՕՀ-ին դիմում է համապատասխան ֆունկցիայով՝ նշելով այն ռեսուրսը կամ կոդի հատվածը, որը պետք է դիտարկվի որպես մյութեքս: Նշված ծրագիրն էլ համարվում է այդ մյութեքսի «սեփականատերը» (release գործողությունը նույնպես իրականացնում է նույն ծրագիրը կամ ՕՀ-ը՝ արտակարգ դեպքերում, օրինակ՝ սխալի դեպքում կամ մյութեքսի մոռացված բացման ժամանակ):

Մյութեքսները հիմնականում օգտագործվում են մի քանի հաշվարկային հոսքերի կողմից ընդհանուր փոփոխականների կառավարման նպատակով, սակայն դրանց կիրառման շրջանակը կարող է լինել ավելի լայն:

Եթե որևէ հոսք փորձում է մուտք գործել արդեն զբաղված մյութեքսին, ապա վերջինս բլոկավորվում է, այսինքն՝ դադարում է իրագործվել մինչև մյութեքսը ազատվի:

Բացի հոսքերի կանգից, մյութեքսային մեխանիզմների կիրառումը կարող է բերել նաև հետաձգումների, որոնք պայմանավորված են մյութեքսների վերահսկողության համար պահանջվող ընթացակարգերով:



Նկար 24. Բազմաթիվ կանչերի մշակման կազմակերպում՝ ծրագրային կոդի օրինակների ստեղծման միջոցով

Գլուխ 6. Ներմուծում-ելք: Ֆայլային համակարգեր: Մուտքի մեթոդներ

Հաշվողական ցանկացած համակարգի հիմնական բաղադրիչներից մեկը մշտական հիշողության և տվյալների արտապատկերման միջոցներն են: Այս միջոցներն ունեն մեծ բազմազանություն և կիրառման տարբեր մեթոդներ:

6.1. Ներմուծման-ելքային սարքերն ու մուտքի մեթոդները

Հաշվողական սարքերի հետ փոխազդեցությունը սկսվել է սարքի վահանակի՝ տարբեր անջատիչների, կոճակների և նման տարրերի միջոցով, իսկ արդյունքները ցուցադրվում էին նույն վահանակի լամպերով (ինդիկացիա):

Ավելի ուշ հայտնվեց տպող մեքենա, որն իրականացնում էր թե՛ մուտքագրում, թե՛ տպում: Դրանից հետո՝ հատուկ տպող սարքեր, մուտքի սարքեր պերֆոկարտերով և պերֆոժապավեններով, ապա՝ տառաթվային էկրաններ ստեղնաշարով, պլոտտերներ, գրաֆիկական էկրաններ, սկաներներ և այլ մասնագիտացված սարքեր: Այս ամենը ներառվեց «պերիֆերիա» հասկացության մեջ՝ մուտքի և ելքի սարքերի ամբողջություն, որոնք միացված են համակարգին առանձին կամ համակցված (տվյալներ + կառավարում) ավտոբուսներով:

Սարքերի բազմազանությունն ու դրանց իրականացման եղանակները հանգեցրեցին սարքերի և համակարգչի միջև ինտերֆեյսների ստանդարտացմանը:

Ծրագրային մակարդակում, որը ապահովում էր սարքերի և համակարգչի փոխազդեցությունը, առաջատար դարձան IBM ընկերության մասնագետները՝ մշակելով ներմուծում-ելքի կազմակերպման մեթոդների ամբողջական փաթեթ:

Դիսկավորիչները, մագնիսական ժապավեններով կուտակիչները, պերֆոկարտերը, պերֆոժապավենները և նույնիսկ սովորական թուղթը դարձան ելքային տեղեկատվության կուտակման միջոցներ: Միևնույն ժամանակ, դրանք ծառայեցին նաև որպես մուտքային աղբյուր՝ հիշողություն ներբեռնելու համար (թուղթը՝ սկաներով):

Ներմուծում-ելքի կապը սարքերի հետ ապահովվում է վերահսկիչների (փաստորեն՝ միկրոպրոցեսորների) կամ հատուկ պրոցեսորների միջոցով, որոնք ունեն իրենց սեփական ծրագրավորման լեզուները (հաճախ՝ պարզեցված ասեմբլերներ):

Սարքերը դասակարգվում են՝

- ըստ արագագործության՝ արագ և դանդաղ,
- ըստ հասցեագրման եղանակի՝ հաջորդական և ուղիղ,
- ըստ գործառույթի՝ միայն ելք, միայն մուտք, մուտք-ելք,
- ըստ ներքին բուֆերների առկայության կամ բացակայության:

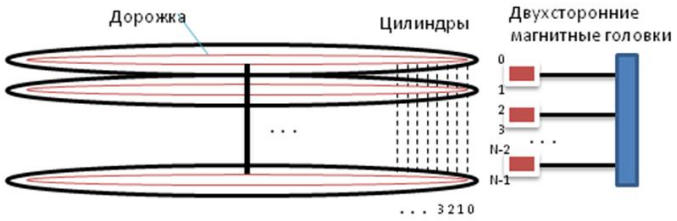
Օրինակ՝ ստեղնաշարը, մուկը՝ դանդաղ են (համեմատելի մարդու գործողությունների արագության հետ): Իսկ կոշտ սկավառակները, գրաֆիկական քարտերը՝ արագ: Մնացած սարքերը՝ միջանկյալ կամ դանդաղ, ինչպես օրինակ՝ ինքթթանիչ տպիչները, պլոտտերները:

Տվյալներին հասանելիության եղանակով սարքերը լինում են երկու տեսակի՝

- ուղիղ մուտք ունեցող սարքեր (Random Access Method՝ թեև «Random» բառը սխալ է օգտագործվում, իրականում որևէ պատահականություն չկա),
- հաջորդական մուտք ունեցող սարքեր:

Ուղիղ մուտքը բնորոշ է օպերատիվ հիշողությանը (բացառությամբ այն դեպքերի, երբ այն կիրառվում է որպես վիրտուալ սկավառակ):

Մշակվել են սկավառակային կուտակիչներ, որոնց հասցեագրման համակարգը ներառում է երեք բաղադրիչ՝ գլանափոսի համարը, դրա մեջ գտնվող ուղու համարը և տվյալ ուղու վրա գրառման համարը: Այսպիսի սկավառակները կառուցված են միմյանց վրա դասավորված մագնիսական շերտ ունեցող սկավառակներից:



Նկար 25. Մագնիսական սկավառակի կառուցվածքը

Յուրաքանչյուր սկավառակ իր մագնիսական մակերեսին (հիշեցնենք, որ երկու ծայրադիր սկավառակների արտաքին կողմերում մագնիսական շերտ չկա) ունի հարյուրավոր համակենտրոն շրջանաձև ուղիներ՝ մագնիսական ուղիներ: Յուրաքանչյուր ուղու վրա գտնվում են մագնիսական գրառումներ, որոնք, որպես կանոն, բաժանվում են ֆիքսված երկարությամբ հատվածների՝ կլաստերներ, սեկտորներ կամ բլոկներ:

Այդ ուղիները, որոնք տեղակայված են սկավառակների վրա և համընկնում են միմյանց ուղղահայացով, ձևավորում են մեկ գլանափոս: Ուղիների համարակալումը սկսվում է 0-ից՝ գլանափոսի առաջինից մինչև վերջինը (ինչը ներկայացվում է ուղղահայաց՝ հևաշիրքային գծերով, ինչպես նշված է համապատասխան նկարում): Գլխիկների զույգերը նույնպես ունեն համապատասխան համարներ: Այդ պատճառով սկավառակի հասցեն կազմվում է եռյակով՝ (Cyl, Head, Record)՝ գլանափոսի համարը, գլխիկի համարը, և գրառման համարը ուղու վրա: Ուղու սկիզբը (կամ տվյալ ուղու գրառման սկիզբը) նշվում է հատուկ մարկերներով:

Գլխիկների ամբողջ համակարգը կարող է տեղաշարժվել սկավառակների մակերեսի խորքը՝ սկսած 0-րդ գլանափոսից մինչև վերջինը:

Սկավառակը պտտվում է շատ բարձր արագությամբ՝ արդյունքում ստեղծելով օդային բարձիկ գլխիկների և սկավառակի մակերեսի միջև՝ կանխելով ֆիզիկական շփումը: Երբ գլխիկը շարժվում է դեպի նշված գլանափոսը, այն կատարում է տվյալ ուղուց ընթերցում կամ գրառում: Այս մեթոդը, երբ անմիջական հասանելիություն է ստացվում անհրաժեշտ գլանափոսին, կոչվում է ուղիղ մուտքի մեթոդ:

Սակայն պետք է նկատի ունենալ, որ սկավառակային ուղիները կամ գրառումները կարող են վնասված լինել դեռ արտադրության պահից, սահմանված թույլատրելի քանակով: Այդ իսկ պատճառով սկավառակների ֆորմատավորման ժամանակ թողնվում են պահուստային գլանափոսեր, որոնք կարող են կիրառվել՝ վնասված ուղիների կամ գրառման բլոկների վերաուղղորդման համար:

Նման վերաուղղորդումը կատարվում է ՕՀ-ի ախտորոշիչ գործիքների կամ անկախ օգտակար ծրագրերի միջոցով: Վերաուղղորդման վերաբերյալ տեղեկատվությունը պահվում է սկավառակի սարքում կամ դրա վերահսկչի մեջ: Այսինքն, եթե օգտագործվում է հասցե (c1, h1, r1), ապա իրականում ընտրվում է (c2, h2, r2):

Գլխիկների շարժման ժամանակը առաջինից մինչև վերջին գլանափոս ժամանակակից սկավառակներում կազմում է մոտավորապես 5-ից 15 միլիվայրկյան: Պահուստային գլանափոսերի քանակը՝ սովորաբար 20-ից ավելին, գտնվում է սկավառակի վերջում, ինչպես նաև դրանց հասանելիության ժամանակը՝ տեխնիկական պատճառներով:

Ըստ ընդունված կարգի՝ 0-րդ գլանափոսի 0-րդ ուղու 0-րդ գրառումը պարունակում է ՕՀ-ի բեռման սեկտորը (տես՝ ՕՀ բեռնումին նվիրված բաժինը):

Նշենք, որ նույնիսկ ուղիղ մուտք ապահովող սարքերի դեպքում դիսկերից ընթերցումը կամ գրառումը, ինչպես նաև ֆայլերի հետ աշխատանքը, իրականացվում է հաջորդական եղանակով:

Ուղղահայաց հասանելիությունից տարբեր մոտեցում է հաջորդական մուտքի մեթոդը, որը կիրառվում է այն տեղեկատվական կրիչների վրա, որտեղ տվյալները տեղակայված են հաջորդաբար, և որոնցում տվյալների հասանելիությունը ապահովվում է դրանց դասավորվածության հերթականությամբ: Առանց նախորդ տվյալներին հասնելու՝ հնարավոր չէ հասնել պահանջվողին:

Դասական օրինակներ են՝

- պերֆոժապավեններ,
- մագնիսական ժապավեններ (որոնք դեռևս կիրառվում են որոշ ոլորտներում)։

Նմանատիպ տարբերակներ կան նաև տվյալների կառուցման ժամանակ՝

- զանգվածներում՝ ուղիղ մուտք ըստ ինդեքսի՝ array[i],
- ցուցակներում՝ հաջորդական մուտք՝ քայլելով ցուցիչներով ցանկալի տարրի ուղղությամբ։

Ծրագրավորման մեջ այս մոտեցումները նույնպես ունեն համապատասխանություններ։ Օրինակ՝

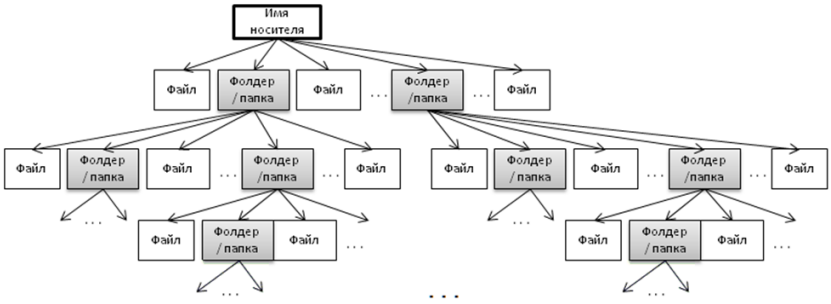
- զանգվածներ, որտեղ հնարավոր է անմիջապես հասնել ցանկացած տարրի,
- ցուցակներ, որտեղ անհրաժեշտ է հաջորդաբար անցնել տարրից տարր, աջ կամ ձախ ուղղությամբ (երկկողմ ցուցակների դեպքում) մինչև հասնել նպատակակետին։

Արագացման նպատակով երբեմն օգտագործվում են ցուցակային քարտեզներ՝ ցուցիչների զանգվածներ, որոնք հաջորդական մուտքը վերածում են ուղիղի։

Եվ իհարկե՝ օպերատիվ հիշողությունը, որն ունի ուղիղ հասցեագրման և անմիջական հասանելիության հնարավորություն հիշողության բջիջներին կամ ընտրվող բառերին, համարվում է ամենահայտնի օրինակներից։

6.2. Ֆայլային կառուցվածքներ

Օպերացիոն համակարգերում յուրաքանչյուր սարք ներկայացվում է սիմվոլային կերպով, իսկ եթե այն տվյալներ պարունակող կրիչ է, ապա նրա պարունակությունը ներկայացվում է ծառաձև տրամաբանական կառուցվածքով, որը արտացոլում է տվյալ ֆայլերի և դրանց խմբավորումների դասավորությունը կրիչի վրա։



Նկար 26. Ծառաձև ֆայլային կառուցվածք

Բազմաթիվ կրիչների համար կարող է ձևավորվել.

- ծառային գրաֆիկ, որի գագաթները՝ արմատի հաջորդ մակարդակում, ներկայացնում են սարքեր, իսկ դրանց մեջ՝ ֆայլեր (նախնական տեսքով՝ տվյալների հավաքածուներ), կամ
- անտառային գրաֆիկ, այսինքն՝ ծառերի ամբողջություն։

Գրաֆիկ կառուցվածքների ցուցակը կարող է պահպանվել՝

- ցուցակների տեսքով, կամ
- աղյուսակներում, որտեղ յուրաքանչյուր գրառման մեջ նշվում է այն պանակը, որին պատկանում է տվյալ ֆայլը։ Նույնը վերաբերում է նաև պանակներին։ Որոշ օպերացիոն համակարգերում պանակներն ու ֆայլերը միասին կոչվում են «ֆայլեր»՝ պարզ կամ համակցված։

Յուրաքանչյուր գագաթին կարող է համապատասխանել տվյալների կառուցվածք կամ աղյուսակ, որում, սովորաբար, նշվում են հետևյալ տվյալները՝

- ֆայլի տեսակը՝ պանակ կամ պարզ ֆայլ,
- ֆիզիկական հասցեն կրիչի վրա,

- ստեղծման ժամանակը,
- վերջին թարմացման ժամանակը,
- թարմացնող օգտատերը,
- վերջին օգտագործման ժամանակը (հարկ է տարբերել թարմացումից, քանի որ կարող է լինել միայն ընթերցում),
- թույլատրված օգտվողները և նրանց մուտքի իրավունքները,
- այլ դաշտեր՝ կախված ՕՀ-ի տրամաբանությունից և մշակողի որոշումներից։

Այդպիսով, ծառաձև նշումներով գրաֆիկները թույլ են տալիս նկարագրել ներդրված տվյալների ամբողջություններ, վերահսկել դրանց օգտագործումն ու մատչելիությունը։

Հատկապես կարևոր է նշել, որ ֆայլը կարող է.

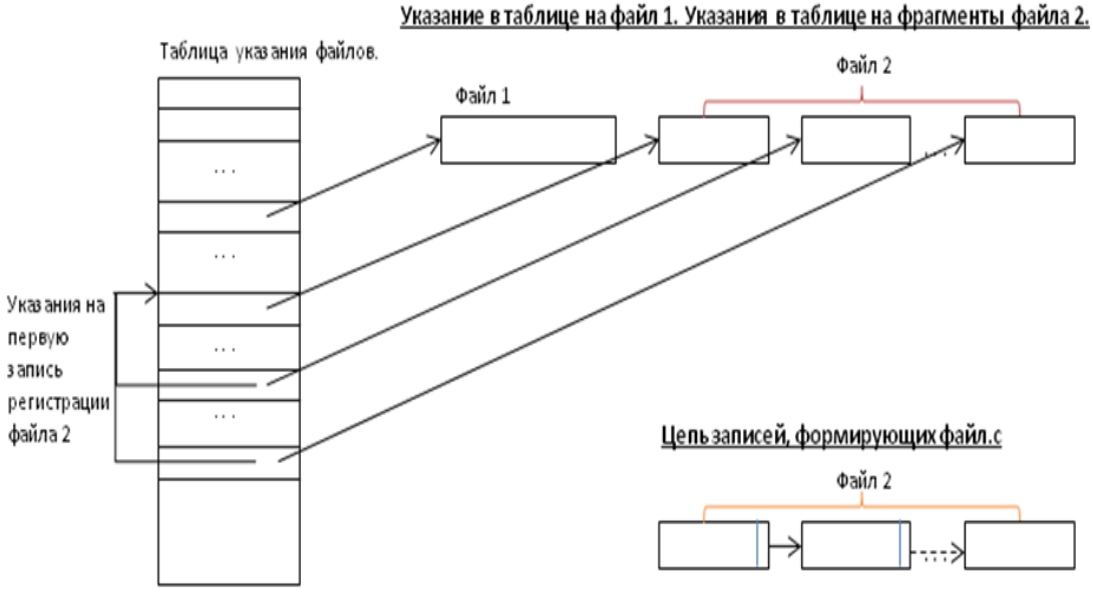
- պահպանվել որպես անընդմեջ բայթերի հաջորդականություն,
- կամ լինել բազում հատվածներից կազմված՝ պայմանավորված ֆրագմենտացիայով (բաժանված տարածքներով, որոնք առաջանում են ֆայլերի ջնջումից հետո)։

Այս պարագայում անհրաժեշտ է պարբերաբար իրականացնել դեֆրագմենտացիա՝ ֆայլերը վերագրանցելու և մեծ ազատ տարածքներ ստեղծելու նպատակով։

Ֆրագմենտավորված ֆայլի յուրաքանչյուր հատված կարող է.

- պահպանվել աղյուսակում՝ կապված ֆայլի նախորդ մասի հետ,
- կամ հենց այդ հատվածում պահել հղում դեպի հաջորդ հատված։

Նման մեթոդներ կիրառվում են նաև օպերատիվ հիշողության մեջ՝ զանգվածների կառուցման ժամանակ։



Նկար 27. Ֆրագմենտավորված ֆայլերի հասցեագրման տարբերակներ։

6.3. Բուֆերացված ներմուծում-ելք

Համակարգիչների ժամանակակից սարքերի մի մասը՝

- փոխանցում կամ ընդունում է մեկական բայթ,
- մյուս մասը՝ տվյալների մի ամբողջ բառ (word)։

Այս տարբերակները կիրառվում են կամ մուտքագրման/ելքի պարզեցման համար, կամ այն դեպքերում, երբ տվյալների փոխանակումը պահանջում է անմիջական արձագանք՝ ինչպես ծրագրերի, այնպես էլ սարքի միկրոշրջանակների կողմից։

Ներքին բուֆերով սարքերը թույլ են տալիս կուտակել տեղեկատվություն իրենց մեջ՝ բայթերով։ Դրանք հիմնականում դանդաղ պերիֆերիկ սարքեր են՝ տպիչներ, սկաներներ և այլն։

Օրինակ՝ տպիչին կարելի է ուղարկել տեքստ կամ պատկեր, որը կուտակվում է բուֆերում և ապա տպվում:

Բուֆերացում կիրառվում է նաև արագ սարքերում, օրինակ՝ կոշտ սկավառակների դեպքում. բուֆերում նախապես կուտակվում է ընթերցված տվյալները և միայն հետո ուղարկվում է օպերատիվ հիշողություն՝ նվազեցնելով գլխիկի շարժման ժամանակը:

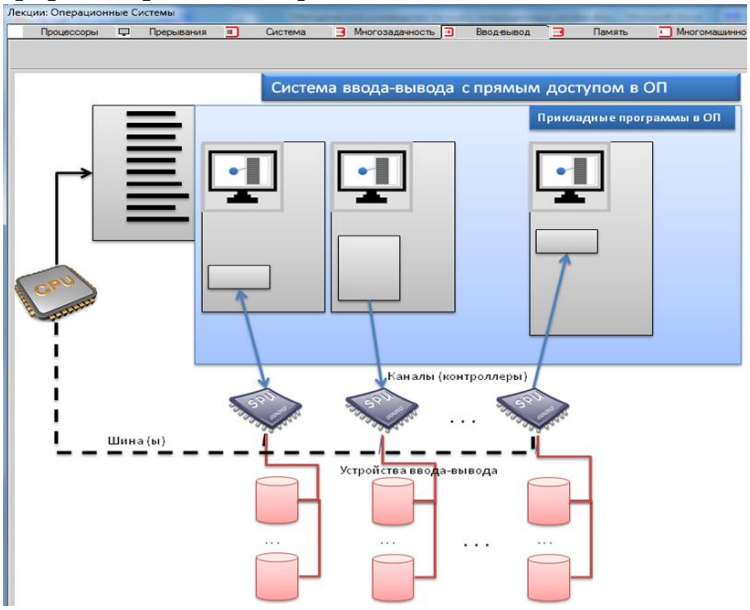
Բուֆերների նպատակն է՝

- նվազեցնել ընթերցման/գրառման գործողությունների քանակը,
- ազատել կենտրոնական պրոցեսորը նման ծանրաբեռնվածությունից:

Բացի սարքերում առկա բուֆերներից, բուֆերներ կիրառվում են նաև ծրագրային մակարդակում՝ որպես հիշողության հատվածներ՝ տարբեր ծրագրերի համար:

Այստեղ բուֆերը կարող է լինել լոգիկական միավոր՝ առանց ֆիզիկական առանձնացման, որը օգտագործվում է՝

- տվյալների միջանկյալ պահպանման,
- սարքերի հետ փոխազդեցության կամ
- միջծրագրային փոխանակման համար:



Նկար 28. I/O համակարգի ճարտարապետություն

Նկ. 16-ում ներկայացված է մուտքագրման/ելքագրման (input/output) իրականացման տարբերակ, երբ տվյալները անմիջապես կարդացվում են մուտքային սարքերից կամ գրվում են ելքային սարքերին՝ առանց կենտրոնական պրոցեսորի մասնակցության: Այդ գործընթացը իրականացվում է հատուկ ենթապրոցեսորների միջոցով (նկարում նշված են որպես SPU – Special Processing Units):

Ակնհայտ է, որ նման պարագայում անհրաժեշտ է ապահովել հիշողության հատվածների պաշտպանություն, մասնավորապես՝ երբ տվյալները պետք է գրանցվեն հիշողության մեջ (իսկ որոշ դեպքերում՝ նույնիսկ կարդալը «օտար» տարածքներից կարող է լինել անթույլատրելի և պահանջել հատուկ ընթերցման/գրանցման թույլտվության նշիչներ՝ ֆլագեր):

Բուֆերային մուտք/ելքը ի սկզբանե կիրառվել է որպես միջոց՝ ուղղակի սարքերի հետ կատարվող գործողությունների քանակը նվազեցնելու համար: Օրինակ՝ ելքային տվյալները հավաքվում են բուֆերում և դուրս են բերվում միանգամից՝ բուֆերի լրացման պահին կամ հատուկ հրամանով: Այս մեթոդը թույլ է տալիս նաև իրականացնել ասինխրոն մուտք/ելք՝ առանց սպասման՝ մինչև գործողության ավարտը:

Բուֆերների լրացումը և մաքրումը սովորաբար ուղեկցվում են ընդհատումներով (ընդհանուր դեպքում՝ ՕՀ իրադարձություններով), եթե բուֆերները կառավարվում են օպերացիոն համակարգի

կողմից կամ ունեն ապարատային աջակցություն։ Դա ապահովում է աշխատելու ասինխրոնության պահանջները։

Բուֆերները նաև կիրառվում են տվյալների միջանկյալ կուտակման և պահպանման համար։ Օրինակ՝ պատկերների հետ աշխատելիս, որոնք տեղակայված են օպերատիվ հիշողությունում։

Ուշադրության է արժանի մի քանի հատկանիշ՝ կապված վիդեոտեղեկատվության ցուցադրման հետ մոնիտորների վրա։ Նախ՝ օգտագործվում է հատուկ պրոցեսոր՝ գրաֆիկական պրոցեսոր (GPU, Graphics Processing Unit) [9], որն ունի սեփական հիշողություն։ GPU-ները դասվում են Ֆլինի դասակարգմամբ SIMD (Single Instruction, Multiple Data) ճարտարապետության՝ մատրիցային պրոցեսորների շարքին։

Վիդեոպրոցեսորը որոշակի հաճախականությամբ սկանավորում է իր վիդեոհիշողությունը կամ դրա հատվածը (դրական վիդեոհիշողության ծավալը թույլ է տալիս միաժամանակ պահել բազմաթիվ պատկերներ), և այդ բովանդակությունը արտացոլվում է մոնիտորի վրա։ Պատկերները սովորաբար ներկայացվում են RGB ձևաչափով, որից հետո գեներացվում են վիդեոսիգնալներ։

GPU-ները կարողանում են մեծ արագությամբ իրականացնել պատկերի փոխակերպումներ հենց իրենց հիշողության ներսում՝ շեյդինգ (shading), ռաստերիզացիա (rasterization) և այլ գործողություններ, որոնք պատկանում են վիզուալիզացիայի (rendering) գործընթացներին։

Չնայած փոխակերպումների մեծ արագությանը, պատկերի մշակման բոլոր խնդիրները GPU-ներով չեն լուծվում, և հաճախ այդ գործողությունները կատարվում են կենտրոնական պրոցեսորի (CPU) վրա։

Այս առումով միջանկյալ բուֆերներում կարելի է պահել պատկերի տարբեր հատվածներ, մի քանի պատկերներ հետագա կոմպոզիցիայի կամ փոփոխության նպատակով։ Սակայն պատկերների հաճախակի պատճենումը օպերատիվ հիշողությունից դեպի GPU-ի հիշողություն առաջացնում է «փայլատակման» (blinking) էֆեկտ, երբ պատկերը կարճ ժամանակով անհետանում է կամ դողում է։

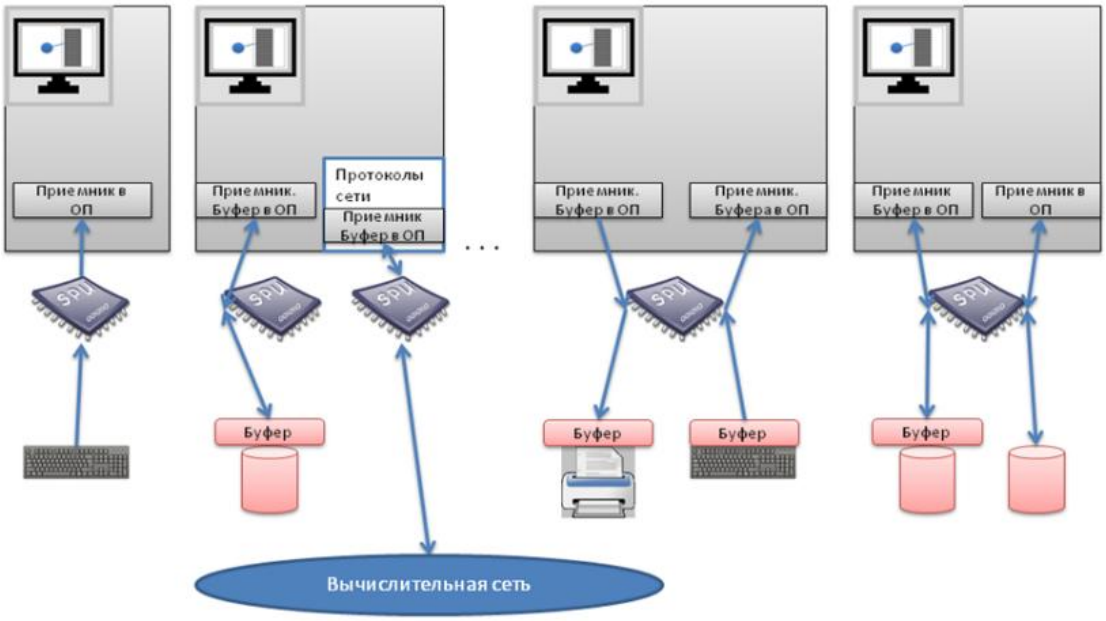
Պատկերի ձևավորումը օպերատիվ հիշողությունում (բուֆերում) և միայն վերջնական տարբերակի փոխանցումը GPU-ին՝ վիդեոհիշողություն, լուծում է այդ խնդիրը։

Շատ օպերացիոն համակարգերի ֆայլային համակարգերը հիմնված են այն սկզբունքի վրա, որ բացված ֆայլերը ամբողջությամբ կամ մասնակիորեն պատճենվում են բուֆերներ, որոնք տեղակայված են օպերատիվ հիշողությունում։ Բոլոր կարդալ/գրել գործողությունները կատարվում են այդ բուֆերների բովանդակության հետ, այլ ոչ՝ անմիջապես սարքերի հետ (նպատակն է՝ նվազեցնել սարքերին ուղղակի հասանելիության գործողությունների քանակը)։

Այդ պայմաններում գրելու գործողությունները սարքին ունեն հետաձգված կատարում։ Այդ պատճառով միշտ խորհուրդ է տրվում սարքերը անջատել ՕՀ-ի միջոցով, այլ ոչ պարզապես ֆիզիկապես՝ օրինակ, USB սարքի հանմամբ։ Քանի որ տվյալները կարող են լինել գրանցված միայն օպերատիվ հիշողության բուֆերում և դեռևս չփոխանցված ֆիզիկական սարքին, այսինքն՝ հնարավոր է տվյալների կորստի ռիսկեր։

Եթե սարքը անջատվում է ՕՀ-ի ծառայությամբ, ապա բուֆերը հարկադիր կերպով փոխանցվում է սարքին՝ առանց սպասելու սահմանված ուշացման ավարտին։

Հնարավոր է սահմանել զրոյական ուշացում, ինչի դեպքում յուրաքանչյուր գրանցման գործողություն անմիջապես կատարվում է սարքի վրա։ Բայց սա հանգեցնում է մուտք/ելք իրականացնող հավելվածի աշխատանքի զգալի դանդաղեցման։



Նկար 29. I/O՝ բուֆերով և առանց: Բուֆերներ ՕՀ-ում:

Գլուխ 7. Բազմամեքենայական և բազմապրոցեսորային համալիրներ

Երկու և ավելի հաշվիչ մեքենաներ միավորելու գաղափարը գրեթե համընկնում է համակարգիչների ստեղծման հետ: Սկզբնական շրջանում միացման միջոց էին հանդիսանում հատուկ մալուխներ՝ տվյալների փոխանցման կարողությամբ ավտոբուսներ (շիներ): Ավելի ուշ սկսվեցին փորձարկումներ և անալոգա-թվային և թվա-անալոգային փոխարկիչների մշակում: Վերջիններս հնարավորություն տվեցին օգտագործել տվյալ ժամանակաշրջանում լայնորեն կիրառվող անալոգային կապի միջոցները՝ հեռախոսային, հեռագրային կապ, ռադիոհաղորդում և այլն:

Փոխազդող հաշվիչները, տեղեկություն փոխանցելով միմյանց, հնարավորություն էին տալիս ապահովել դրա ավելի հուսալի պահպանությունը (օրինակ՝ երկու համակարգչում պահել տվյալը ավելի հուսալի է, քան մեկում), փոխանցել նաև ծրագրերի կոդերը, ապահովել հաշվարկների կրկնօրինակված կատարում, դրանց համադրում և սինխրոնացում: Համակարգիչների առաջին սերունդները առանձնապես հուսալի չէին, և հաճախ կիրառվում էին վթարների հավանականության հաշվարկներ՝ հիմնված վիճակագրական տվյալների հավաքագրման վրա:

Սակայն բազմապրոցեսորային և բազմամեքենայական միավորումների հիմնական նպատակը հաշվարկների բաշխումն է՝ արագացման հնարավորությամբ, ինչպես նաև զուգահեռ ծրագրերի աջակցությունը, որոնք իրականացնում են անկախ կամ մասնակիորեն կախյալ վարքագիծ ունեցող օբյեկտների մոդելներ: Հաշվարկների մի քանի կատարողների առկայությունը, անկասկած, ոչ միայն արագացնում է գործընթացը, այլ նաև պարզեցնում է ալգորիթմացումը և ծրագրավորումը:

7.1. Բազմամեքենայական և բազմապրոցեսորային համակարգերի ձևավորում

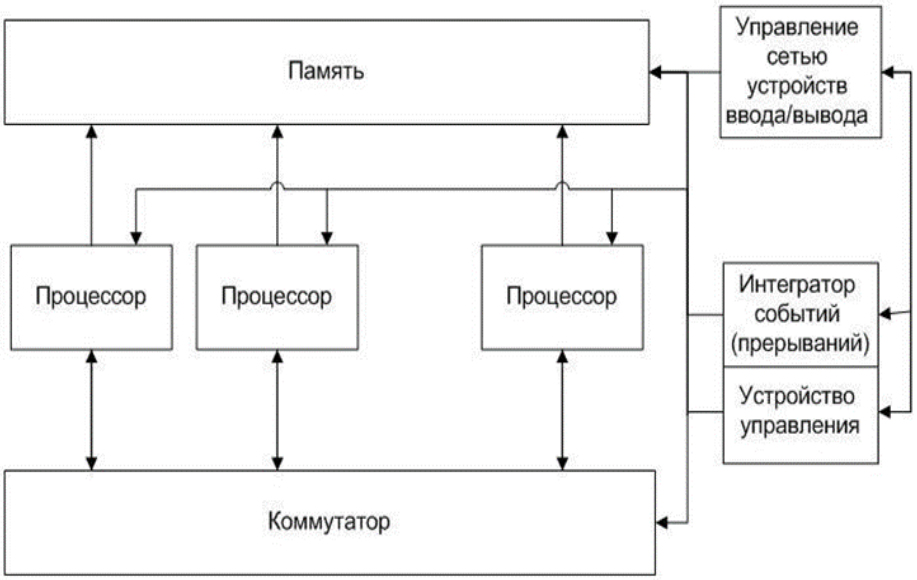
Այս գաղափարը առավել հիմնավորված ու իրագործված կերպով արտահայտվեց IBM/360 համակարգի հայեցակարգում:

Կիրառվեցին հետևյալ միացման տեսակները՝

1. Պրոցեսոր – պրոցեսոր,

2. Ավտոբուսային ալիք – ալիք,
3. Ընդհանուր սարքավորում և տվյալների կրիչ,
4. Ցանցային միացում,
5. Հեռահաղորդակցական կապեր։

Առաջին տարբերակի դեպքում՝ պրոցեսորները միացվում են մալուխ-շիներով։ Հրամանների համակարգում ներառված են հրամաններ՝ միջպրոցեսորային կապի նախաձեռնման և տվյալների փոխանակման համար՝ տվյալ միացման միջոցով։



Նկար 30. Բազմապրոցեսորային ճարտարապետություն՝ ընդհանուր հիշողությամբ և կենտրոնացված կառավարմամբ

Հետագայում մշակվեցին կոմմունիկատորներ, որոնք հնարավորություն էին տալիս միացնել ոչ միայն պրոցեսորներ, այլև արտաքին սարքեր՝ թե՛ ամբողջությամբ, թե՛ որոշակի ընտրությամբ։

«Ալիք–ալիք» միացումները թույլ են տալիս մեկ հաշվիչ համակարգիչը օգտագործել որպես մուտքագրման/ելքագրման սարք մեկ այլ հաշվիչի համար։ Հետագա աշխատանքը կատարվում է «Master/Slave» (ղեկավար/ենթարկվող) մոդելով՝ գլխավոր համակարգը կառավարում է ենթարկվող համակարգիչը։

Ընդհանուր տվյալների կրիչի միջոցով միացումը իրականացվում է այն ձևով, որ մուտքագրման/ելքագրման սարքը միացված է մի քանի հաշվիչների։ Սովորաբար որպես այդպիսի սարք օգտագործվում է կարծր սկավառակ, որպես ամենաարագ սարքավորում։ Նման կառուցվածքի նմանակը կարելի է համարել RAID զանգվածները։ Տվյալների փոխանակումը կատարվում է կրիչում գրանցումների և տարբեր համակարգերի կողմից դրանց ընթերցման միջոցով։

Հաշվիչ ցանցերի հայտնվելով (ինչը ենթադրում է փոխազդեցության կանոնակարգված պրոտոկոլներ) միացումները անհրաժեշտության դեպքում իրականացվում են դրանց միջոցով։

Տելեկոմունիկացիոն միացումները ապահովվում են հեռահաղորդակցական կայանների միջոցով, որոնք համարվում են հաշվիչի պերիֆերիկ սարքեր։ Այսինքն՝ տարբեր հաշվիչ համակարգեր փոխազդում են միմյանց հետ հեռավորության վրա տեղադրված տելեկոմունիկացիոն սարքերի միջոցով՝ լինի դա լարային, ռադիո կամ օպտիկամանրաթելային կապ։

Այս բոլոր միացումների կառավարումը իրականացվում է համապատասխան օպերացիոն համակարգի կոմպոնենտների կամ ծառայությունների միջոցով։

Ընդհանուր առմամբ, բոլոր միացումները կարող են կառուցվել ինչպես կառավարչական և ենթարկվող համակարգիչների տարանջատումով, այնպես էլ հավասար իրավունքներով համակարգերի փոխգործակցության ձևով։ Վերջին դեպքում համակարգերը «պայմանավորվում» են փոխազդեցության պայմանների շուրջ։

Այս փոխազդեցության տրամաբանական հիմքում պետք է լինի միասնական լեզու և նրա միանշանակ մեկնաբանություն բոլոր մասնակիցների համար։ Այս սկզբունքը հետագայում ձևակերպվեց որպես պրոտոկոլներ՝ ներառելով միավորված սիմվոլիզմ, ընդհանուր տվյալների կառուցվածքներ և արձագանքման (ինտերպրետացիայի) միևնույն կանոններ։

Սկզբնական փուլում բազմամեքենայական համալիրները բաժանվում էին՝

- համասեռ (հոմոգեն)՝ կազմված նույն ճարտարապետությամբ և օպերացիոն համակարգերով համակարգիչներից,
- տարասեռ (հետերոգեն)՝ ներառելով տարբեր ճարտարապետություն ունեցող (ոչ պարտադիր) համակարգիչներ՝ տարբեր օպերացիոն համակարգերով։

Այստեղ հիմնական դեր է խաղում օպերացիոն համակարգը, քանի որ հենց ՕՀ-ի վերահսկողությամբ է կատարվում փոխազդեցությունը, տեղեկատվության փոխանակումը և հաշվարկային գործընթացների շարունակականությունը։

Ճարտարապետությունների համանմանությունը ապահովում է միևնույն մեքենայական կոդի կատարումը բոլոր համակարգիչների վրա։ Հակառակ դեպքում անհրաժեշտ կլիներ իրականացնել տարբեր ասեմբլերների կոմպիլացիա և կապերի խմբագրում, ինչը զգալիորեն կբարդացներ գործընթացը։

Տարբեր օպերացիոն համակարգերի առկայությունը բերում է հավելյալ խնդիրների՝ համաձայնեցված աշխատանքի կազմակերպման և կիրառական ծրագրերի մակարդակում։ Յուրաքանչյուր ՕՀ ունի իր առանձնահատկությունները, և կիրառական ծրագրերը պետք է ադապտացվեն դրանց։

Վիրտուալ մեքենաների հայտնվելը, որոնք հնարավոր է տեղակայել տարբեր հարթակների վրա (օրինակ՝ POSIX-համատեղելի համակարգեր) և ունեն միևնույն վարքագիծ, դարձավ տարասեռության խնդրի որոշակի լուծում։

Արդեն 1980-ականներին IBM-ը մշակեց վիրտուալ մեքենայի օպերացիոն համակարգ, որն այսօր համարվում է հիմնականներից մեկը և ներառում է UNIX, Linux, OS/2 և այլ տարբերակներ։

Վիրտուալ մեքենայի աշխատանքի արագությունը ի սկզբանե դանդաղ է բազային ՕՀ-ի համեմատ, քանի որ բոլոր գործողությունները կատարվում են միջանկյալ շերտով՝ բազային ՕՀ-ի միջուկի միջոցով։ Սիստեմում կարող են գործել մի քանի վիրտուալ մեքենաներ, որոնք ունեն իրենց միջուկներ, ընդհատումների մշակման մոտեցումներ և այլն։

Բազմամեքենայական համակարգերի տարածված օրինակ են կլաստերները՝ համասեռ համակարգիչների խմբեր, որոնք միացված են հատուկ հաշվողական ցանցով։ Այս ցանցերը օպտիմիզացված են հաղորդագրությունների արագ փոխանակման համար և չունեն շերտավորված ստանդարտացված պրոտոկոլների բոլոր մակարդակները, ինչն էլ պարզեցնում է կառուցվածքը։

Կլաստերները դասվում են SPMD ճարտարապետության (Single Procedure – Multiple Data)՝ ըստ Ֆլինի ընդլայնված դասակարգման։ Այսինքն՝ նույն ծրագիրը տարածվում է բոլոր համակարգչային հանգույցների վրա, իսկ տարբեր հանգույցներ մշակում են տարբեր տվյալներ։

Կլաստերների գաղափարն ազդեցիկ զարգացում ապրեց՝ համակարգիչների գնի նվազման շնորհիվ։ Այդպիսով, հնարավոր դարձավ ձևավորել լայնածավալ համակարգիչների միացումներ՝ միավորված մեկ ցանցով։

Կլաստերները, սակայն, զգալիորեն ավելի դանդաղ են, քան բազմապրոցեսորային համակարգերը, քանի որ վերջիններում պրոցեսորները միացված են ներքին ավտոբուսներով կամ արագ հաղորդակցիչներով, իսկ կլաստերները՝ ցանցային պրոտոկոլներով։ Տվյալների փոխանակման արագությունները կարող են տարբերվել մի քանի կարգով։

Կլաստերների կառուցվածքը սովորաբար ունի կենտրոնացված կառավարում։

Կլաստերների վառ օրինակն է Titan-ը, որը բաղկացած է 18 688 հանգույցից և կառուցված է Cray XK7 հիբրիդային հարթակի վրա։ Հիբրիդային ճարտարապետությունն ընդգրկում է 16-միջուկանի AMD Opteron պրոցեսորներ՝ ինտեգրված NVIDIA Tesla K20x GPU-ների հետ։ Այս

ձևաչափը ապահովում է ինչպես ամբողջ թվերի 32 և 64 բիթանոց գործողությունների, այնպես էլ լողացող կետով օպերատորների բարձր արդյունավետություն։

Ծրագրային կոդի բաշխումն ըստ պրոցեսորների տեսակի որոշվում է կոմպիլյացիայի փուլում։

Նշենք, որ Titan-ը, որը 2012 թվականին զբաղեցնում էր TOP-500 վարկանիշի առաջին տեղը, այժմ զիջել է այն Չինաստանի Tianhe-2 կլաստերին, որն օգտագործում է Intel Xeon պրոցեսորների մոդիֆիկացիաներ։

Երկու կլաստերն էլ կառավարվում են Linux ՕՀ-ի տարբերակներով։ Ընտրությունը պայմանավորված է ոչ այնքան Linux-ի բացառիկ որակներով, որքան այն փաստով, որ Linux-ը անվճար է, պարզ և բաց կոդով։

TOP-500 վարկանիշի սուպերհամակարգիչների ամենատարածված հարթակն է IBM-ի արտադրած Blue Gene շարքը։ Այս շարքում կան բազմաթիվ սերունդներ։ Blue Gene/Q հարթակի հիմքում են PowerPC A2 պրոցեսորները՝ 18 միջուկով.

- մեկ միջուկը սպասարկում է կառավարումը,
- մեկ այլը՝ հուսալիությունը,
- իսկ 16 միջուկները նախատեսված են հաշվարկների համար։

Blue Gene-ի հանգույցներում տեղակայվում է CNK (Compute Node Kernel) օպերացիոն համակարգը, իսկ մուտք/ելքի գործողությունների ապահովման համար՝ Linux-ի հատուկ միջուկային տարբերակ՝ INK (I/O Node Kernel)։

7.2. Կառավարման կառուցվածքներ բազմամեքենայական համալիրներում և օպերացիոն համակարգի առաջադրանքներ

Բազմապրոցեսորային և բազմամեքենայական համակարգերում սկզբունքային նշանակություն ունի համալիրի կառավարման կազմակերպումը։

Ակնհայտ է, որ կառավարման համակարգերի նախագծման ժամանակ մշակումներն ուղղորդվում են տվյալ համակարգիչների ճարտարապետությամբ։ Սակայն ճարտարապետությունից զատ, որոշ դեպքերում կարող է անհրաժեշտ լինել մշակել աջակցող միջաշերտեր, որոնք ապահովում են տարբեր հաշվարկային մոդելների կազմակերպումը։

Կառավարման համակարգում բազային հարցերից մեկն է ընտրությունը կենտրոնացված կամ բաշխված կառավարման միջև։

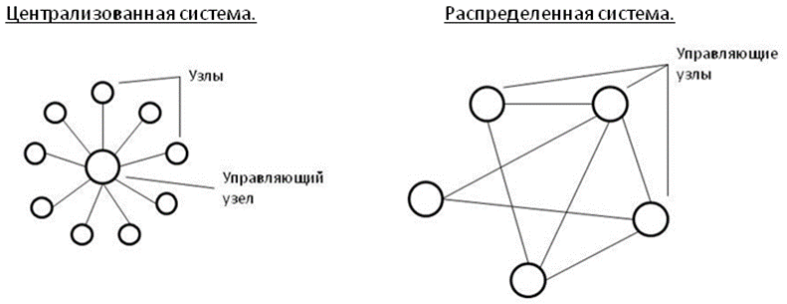
Նկարում ներկայացված է կենտրոնացված կառավարման տրամաբանական սխեմա՝ «աստղաձև» կառուցվածքով։ Այստեղ չի նշվում ֆիզիկական միացման եղանակը համակարգիչների միջև՝ դրանք կարող են լինել առանձին ավտոբուսներ, հատուկ ալիքներ կամ հաշվարկային ցանցեր։

Միացումները կարող են լինել նաև տիպային տարբերություններով։

Կենտրոնական համակարգիչը (կենտրոնական հանգույցը) իրականացնում է հետևյալ գործառույթները.

- առաջադրանքների բաշխման կառավարում,
- տեղեկատվության փոխանցում,
- պերիֆերիկ հանգույցների վիճակի վերահսկում,
- համալիրին առնչվող իրադարձությունների ինտեգրում,
- առանձին հանգույցների միացում կամ դուրս բերում և այլն։

Այսպիսի կառավարման կառուցվածքի կիրառմամբ ամբողջ համալիրն աշխատում է որպես մեկ կազմակերպված միավոր՝ կախված կենտրոնական հանգույցի աշխատունակությունից և վերահսկման արդյունավետությունից:



Նկար 31. Կենտրոնացված և բաշխված կառավարման սխեմաներ:

Բաշխված կառավարման համակարգերը ներառում են գործառութային և կառավարման հնարավորություններով հավասարազոր հանգույցներ: Այսպիսով, յուրաքանչյուր հանգույց կարող է նախաձեռնել առաջադրանքներ այլ հանգույցների համար, իսկ վերջիններս, իրենց հերթին, կարող են մերժել այդ առաջադրանքները: Հանգույցների միջև փոխազդեցությունը կազմակերպվում է երկխոսության և համաձայնեցված աշխատանքի ռեժիմի հաստատման միջոցով:

Կենտրոնացված և բաշխված կառավարման միջև որակական տարբերությունն այն է, որ կենտրոնացված կառավարման դեպքում կենտրոնական հանգույցը վերապահված է հրամայական իրավասություններով (պերիֆերիկ հանգույցը պարտավոր է կատարել նրա հանձնարարականը), իսկ պերիֆերիկ հանգույցները սովորաբար զբաղված են բացառապես կենտրոնական հանգույցից ստացած առաջադրանքներով: Իսկ բաշխված կառավարման դեպքում փոխազդեցությունը կատարվում է դիմումների միջոցով՝ առանց պարտադիր կատարման պահանջի, ինչը հանգույցներին հնարավորություն է տալիս հիմնվել իրենց՝ սովորաբար հեուրիստիկ կանոնների վրա, և մերժել կատարման դիմումները:

Յուրաքանչյուր հանգույց կարող է ունենալ իրեն միացված և իրեն ենթակա պերիֆերիկ հաշվիչներ՝ հանդես գալով որպես հյուրընկալ համակարգիչ (host), նույնիսկ առանց սերվերի կարգավիճակի:

Հանգույցի դերը համակարգում որոշվում է նրա դիրքով կառավարման համակարգում, և հանգույցը կարող է ներկայացվել ցանկացած բարդության հաշվիչով, ներառյալ՝ ամբողջական կլաստերով:

Կառավարման կառուցվածքների համեմատական առավելություններ

- Կենտրոնացված կառավարման առավելությունը այն է, որ պերիֆերիկ համակարգիչները ազատվում են կառավարման ֆունկցիաներից, և առաջադրանքների բաշխումը կատարվում է արագ:
- Բաշխված կառավարմամբ համակարգերը ավելի ճկուն և կենսունակ են. հանգույցներից որևէ մեկի կամ նրա կապի միջոցների խափանումը չի հանգեցնում համակարգի ընդհանուր աշխատանքի խափանման: Այս դեպքում մնում ենք առանց մեկ հանգույցի, իսկ կապի խնդիրները կարող են լրացվել այլընտրանքային ուղիներով:

Իսկ ահա կենտրոնական հանգույցի խափանումը բերում է ամբողջ համալիրի աշխատանքի դադարեցման: Այդ պատճառով ընդունված է կիրառել հայելային (միրորային) համակարգիչներ, որոնք կրկնօրինակում են կենտրոնականի ֆունկցիոնալությունը և ավտոմատ կերպով ստանձնում են կառավարումը խափանման դեպքում:

Ընդհանուր առմամբ, բազմամեքենայական համալիրներում կիրառվում է պահեստային հանգույցների ռազմավարություն, որոնք միացվում են ընդհանուր համակարգին՝ խափանման դեպքում:

Կոդի բաշխում և դրա ռացիոնալ կազմակերպում

Պարալել կատարման համար մեքենայական կոդի բաշխումը հանգույցների միջև պահանջում է հատուկ մոտեցում։

- Մեքենայական կոդը ուղարկել հանգույցից հանգույց՝ հաղորդագրությունների միջոցով անարդյունավետ է, քանի որ այդպես ծանրաբեռնվում է ցանցը, իսկ բուն կոդային ֆայլերը կարող են ունենալ մեծ ծավալ։
- Դրա փոխարեն առավել ռացիոնալ մոտեցում է, երբ յուրաքանչյուր հանգույց ինքնուրույն ընթերցում է անհրաժեշտ կոդը ընդհանուր ֆայլային սերվերից, որտեղ ծրագիրը նախապես տեղադրվել է։

Նշենք, որ այս տրամաբանությամբ են կառուցված նաև ժամանակակից գրաֆիկական պրոցեսորները (GPU)՝ մասնավորապես NVIDIA և AMD-ի արտադրած։ GPU-ները, ըստ էության, մատրիցային պրոցեսորներ են, որոնք ունեն բազմաթիվ հաշվարկային մոդուլներ (կորներ) և կենտրոնական կառավարման միավոր։ Վերջինս կառավարում է ներքին հիշողությունը, փոխազդեցությունը CPU-ի հետ, հիշողության ներմուծման/արտածման գործընթացները և այլն։

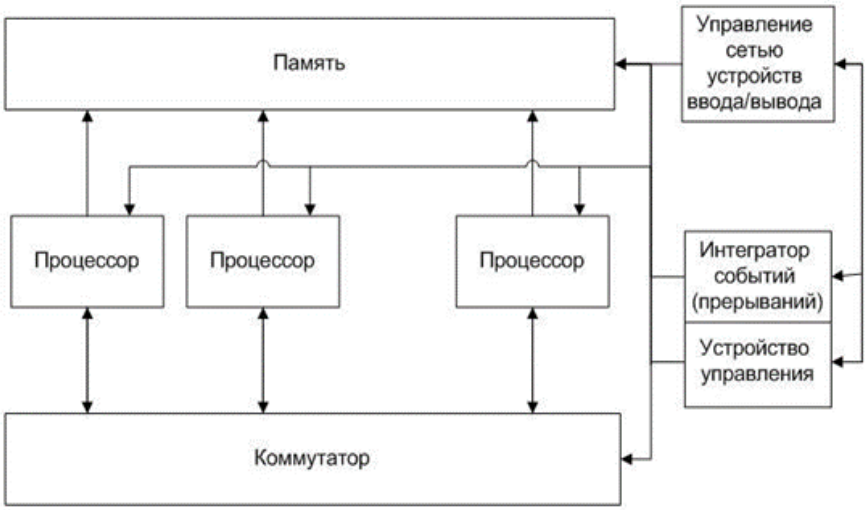
Caper պարալել լեզուն նույնպես հիմնված է մոդուլային կառուցվածքի վրա։ Նրա վիրտուալ մեքենան թույլ է տալիս դինամիկ կերպով բեռնել օբյեկտային մոդուլներ, կապել դրանք և իրականացնել։ Նաև աջակցվում է օպերացիոն համակարգի դինամիկ գրադարանների հետ աշխատանքը։

Այսպիսով, Caper լեզուն հնարավորություն է տալիս մոդուլները տեղադրել սերվերի վրա, և օգտագործել դրանք առանձին հանգույցներում՝ ըստ անհրաժեշտության։ Այս մոտեցումը բացառում է ամբողջական մոնոլիտային կոդի ձևավորումը։ Տվյալները կարելի է անհրաժեշտության դեպքում փոխանցել հանգույցից հանգույց, քանի որ այդ մոդուլներն ունեն փոքր ծավալ։

[5]) Այս տեքստում դիտարկվում է հիմնականում գործնական կիրառությունը. բացառվում է էկզոտիկ մոտեցումը, երբ հանգույցները սկզբում ստանում են սկզբնաղբյուր կոդը, այնուհետև այն կոմպիլացնում են տեղում և կատարում։ Նման մոտեցումը ռեսուրսատար է և խանգարում է կլաստերի հիմնական նպատակին՝ բարդ խնդիրների լուծում՝ բազմաօգտվողային ռեժիմով։ Կոմպիլյացիան կարող է իրականացվել հաճախորդի մեքենայի վրա։

[6]) Մոդուլային կառուցվածքով ծրագրավորման լեզուները ենթադրում են նաև մոդուլային կառուցվածքով կոմպիլացված կոդ, որը կարող է լինել առանձին ֆայլերում՝ ըստ անհրաժեշտության բեռնվող։ Նույնը վերաբերում է նաև դինամիկ գրադարաններին, որոնք լայնորեն աջակցվում են ժամանակակից ծրագրավորման լեզուներում։

7.3. Ընդհանուր հիշողության կազմակերպում բազմամեքենայական համալիրներում



Նկար 32. Ընդհանուր հիշողությամբ ճարտարապետություններ

Նշենք, որ մի քանի պրոցեսորների կամ միջուկների կողմից ընդհանուր օպերատիվ հիշողության ընթերցումն ու, հատկապես, տվյալների գրանցումը, կարգավորվում են ապարատային միջոցներով և պահանջում են հիշողության հետ աշխատանքի ընթացքում գործընթացների արդյունավետ սինխրոնացման մեթոդներ (Նկարում սինխրոնացման մեխանիզմները չեն ներկայացված)։

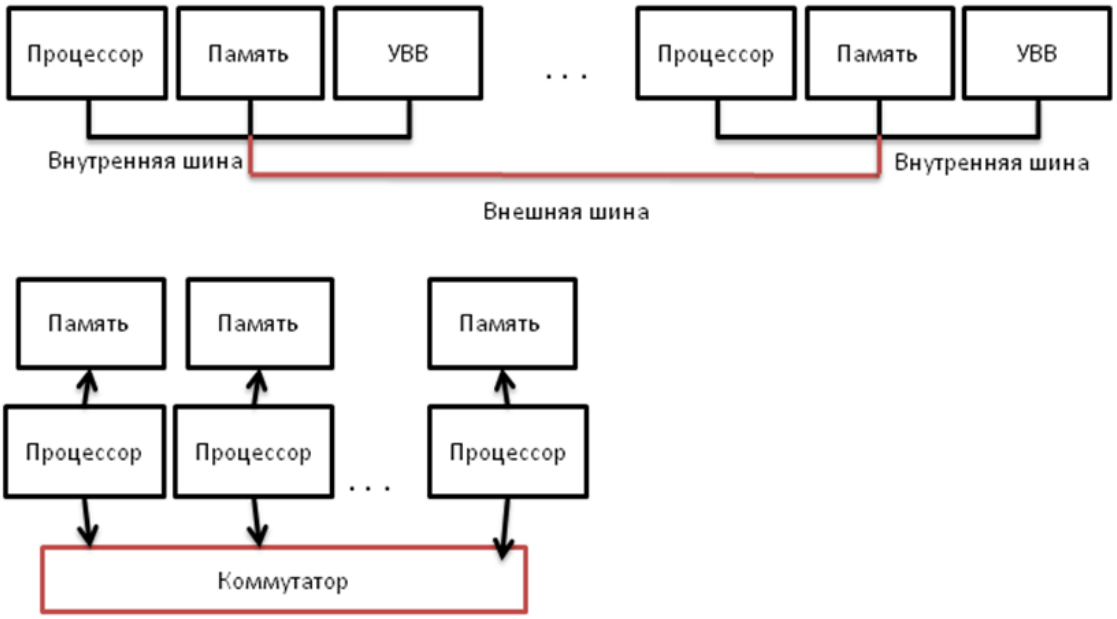
Կոմմուտատորը (switch) ապահովում է պրոցեսորների կառավարման և միջպրոցեսորային փոխազդեցության ֆունկցիաները։ Ինտեգրատոր իրադարձությունների, իր հերթին, անհրաժեշտ է, եթե յուրաքանչյուր պրոցեսոր ունի սեփական ընդհատումների համակարգ։

Ընդհանրապես, բազմապրոցեսորային հաշվիչ համակարգի արդյունավետ աշխատանքի համար (այս մոտեցումը հետագայում նաև կիրառվել է կլաստերներում) ելակետ է հանդիսանում ընդհանուր կառավարող հանգույցի գոյությունը։

Այսպիսի կառուցվածքը հատկապես անհրաժեշտ է, երբ կազմակերպվում են SIMD / SPMD դասի հաշվարկներ (Single Instruction, Multiple Data / Single Program, Multiple Data), որոնց օրինակներն են՝ GPU-ները և այլ մատրիցային պրոցեսորներ։ Այդ պրոցեսորները հանդես են գալիս որպես գործողությունների բազմապատկման և տարբեր տվյալների վրա տարածման գործիքներ։

Օրինակ՝ NVIDIA պրոցեսորների դեպքում, գործարկվող ֆունկցիայի (կամ «քերնելի») իտերացիոն մարմինը փոխանցվում է կառավարման միավորին (պրոցեսորին), որն այնուհետև տարածում է հրահանգները ողջ սարքի հաշվարկային միջուկների միջև։

NVIDIA-ն մշակել է CUDA լեզուն՝ այդպիսի հաշվարկների կառավարման ծրագրավորման նպատակով։ Նույն դերն է կատարում նաև OpenCL ծրագրային հարթակը, որը ստանդարտացվել է խաչաձև հարթակների համար (cross-platform) և աջակցվում է գրեթե բոլոր առաջատար ծրագրային արտադրողների կողմից։



Նկար 33. Բաժանված հիշողությամբ ճարտարապետություններ

Նշենք, որ օպերատիվ հիշողության տարրերն իրենց հերթին կարող են համալրվել կոմմուտատորով: Նման դեպքերում յուրաքանչյուր պրոցեսոր կամ՝ ընտրված խմբաքանակը, կարող է մուտք գործել ոչ միայն «սեփական» հիշողություն, այլ նաև այլ պրոցեսորների՝ այսպես կոչված «օտար» հատվածներ: Այս դեպքում հաճախ օգտագործվում է հիշողության բայթերի կամ բառերի ամբողջական՝ սկիզբից վերջ սկավառակի (сквозная) համարակալում:

Բազմամեքենայական և բազմապրոցեսորային համալիրներում բաժանված հիշողության (օր.՝ Motorola 68 0xx տեսակի) կիրառման կարևոր խնդիրներից է՝ ընդհանուր օպերատիվ հիշողության ստեղծումը, որը կաջակցի ընդհանուր փոփոխականների և տվյալների կառուցվածքների բաշխմանը տարբեր պրոցեսորների կամ մեքենաների միջև:

Այդ նպատակով կիրառվում է ընդհանուր հիշողության տարրերի համարակալման մեթոդ. եթե տվյալ համակարգչին (համարը՝ a) հատկացվում է հիշողության հատված L չափով, ապա որոշակի բջջի հասցեն կարող է գրվել հետևյալ ձևով՝

$$A = a \times L + b,$$

որտեղ b–ն օֆսեթն է տվյալ համակարգչի հատվածում:

Այս մեթոդը նման է վիրտուալ հիշողության հասցեավորմանը, սակայն լրացուցիչ պարունակում է համակարգչի համարը:

Մուտքը տվյալ տարրին իրականացվում է մոնիտոր-ֆունկցիայի միջոցով, որը վերահսկում է հիշողության ընթերցման և գրանցման գործողությունները: Հնարավոր է ավելի բարդ կազմակերպում՝ նման մեկ համակարգչի հիշողության կառավարմանը:

Այլընտրանքային տարբերակ է մի համակարգչի ընտրությունը որպես ընդհանուր տվյալների պահոց, որտեղ տվյալները պահպանվում են նրա օպերատիվ հիշողությունում: Դիմումները տվյալ փոփոխականներին կատարվում են պահոցի ծրագրային մոնիտորի միջոցով: Այս եղանակով փոփոխականները ներկայացվում են սիմվոլիկ անուններով, իսկ դրանց հղումները պահպանվում են՝

(անուն, հիշողության հասցե) զույգերով:

Արտաքին կրիչներում ընդհանուր հիշողության կազմակերպումը, որոշ տեխնիկական մանրամասներից բացի, շատ չի տարբերվում օպերատիվ հիշողության կազմակերպումից: Նման դեպքում տվյալները պահպանվում են ֆայլերում, իսկ գործողությունները կատարվում են հաղորդագրությունների միջոցով, որոնք պարունակում են հրահանգներ:

Վերջապես, հնարավոր է ևս մեկ տարբերակ՝ առանձին համակարգչի հատկացում ընդհանուր հիշողության ստեղծման, պահպանման, փոփոխման և փոխանցման համար: Այս

համակարգիչն աշխատում է ըստ պահանջի՝ ստեղծելով հիշողության դաշտեր («փոփոխականներ»՝ համապատասխան ստորագրություններով), որոնք հայտարարվում են համակարգի այլ հանգույցների կողմից։ Մուտքը նման փոփոխականին տրանսլացվում է որպես ֆունկցիա՝ դեպի այդ համակարգիչ, որով իրականացվում են գրանցման, ընթերցման, ստեղծման կամ հեռացման գործողությունները։

Եզրափակիչ դիտարկում

Ուշադրության է արժանի, որ ցանցային կապերով իրականացվող դիմումները, այդ թվում՝ ընդհանրապես պրոցեսոր-հիշողություն փոխազդեցությունը, պահանջում են ժամանակ, ինչն էլ հանգեցնում է գործընթացների դանդաղեցման։ Սակայն այս ժամանակային ծախսերը հաճախ բազմակիորեն փոխհատուցվում են՝ ամբողջական հաշվարկները մեկ համակարգչի շրջանակներում իրականացնելու հնարավորությամբ։

ԵԶՐԱԿԱՑՈՒԹՅՈՒՆ

Սույն տեքստը կազմվել է ՀՀ ԳԱԱ Ինֆորմատիկայի և ավտոմատացման հիմնարար գիտական կենտրոնում ընթերցվող «Օպերացիոն համակարգեր» դասընթացի հիման վրա։

Ներկայացված նյութում փորձ է արված համապարփակ կերպով լուսաբանել հաշվիչ համակարգերի ճարտարապետական առանձնահատկությունները, դրանց ռեսուրսային բաղադրիչները և դրանց կառավարման մեթոդները, երբ առկա են բազմաթիվ հաշվարկային գործընթացներ։

Հատուկ ուշադրություն է դարձվել. իրադարձային մեխանիզմներին՝ ընդհատումների և իրադարձությունների մշակման սխեմաներին, օպերացիոն համակարգերին՝ որպես ռեսուրսների բաշխման ծրագրային համալիրների, պերիֆերիկ սարքերի կառավարմանը, սինխրոն և ասինխրոն ներմուծման/արտածման մեթոդներին, տեղեկատվության պահպանման և հասանելիության միջոցներին։

Ներկայացվել է նաև բազմամեքենայական համալիրների կառուցվածքը, մասնավորապես՝ կլաստերների, ինչպես նաև նրանց կառավարման կազմակերպման մեթոդները։

Վերհանվել են ծրագրավորման որոշ խնդիրներ, որոնք առաջանում են տարբեր տեսակի հաշվիչների և օպերացիոն համակարգերի համար ծրագրեր մշակելիս։

Ուզում եմ շնորհակալություն հայտնել Վերոնիկա Մինասյանին և Նազելի Մկրտչյանին, ովքեր զգալի օգնություն ցուցաբերեցին այս տեքստի պատրաստման գործում։

ԳՐԱԿԱՆՈՒԹՅՈՒՆ.

1. Алгоритмы, математическое обеспечение и архитектура многопроцессорных вычислительных систем. М: “Наука”, 1982, 336 с.
2. Карп Р.М., Миллер Р.Е. Параллельные схемы программ. Москва, “Мир”: Кибернетический сборник. Вып. 13. 1976. сс. 5-61.
3. Flynn M. J. Very high speed computers. Proc IEEE, 1966, 54. — P. 1901—1901.
4. Джермейн К. Программирование на IBM/360. Пер. с англ. Изд. 2, 1973. 870 с.
5. Документация по zOS: https://www.ibm.com/docs/en/zos
6. М.Б. Кузьминский. Z-архитектура. https://www.osp.ru/os/2001/10/180514
7. Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. СПб.: Питер, 2015, 1120 с.
8. Стивенс У. Р., Раго С. UNIX. Профессиональное программирование. 3-е изд. СПб.: Питер, 2018, 944 с.
9. Устройство графических процессоров. https://club.dns-shop.ru/blog/t-99-videokartyi/122040-ustroistvo-graficheskih-protsessorov-gpu/?utm_referrer=https%3A%2F%2Fwww.google.com%2F
10. Digital Signal Processing Solutions. Texas Instruments Incorporated, 2000.
11. Рихтер Дж. Windows для профессионалов: создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows. СПб.: Питер, 2001, 752 с.
12. Вартанов С.Р. Язык программирования CAPER. - Киев, 1997 (Препр. 97-5, Национальная Академия Наук Украины, Институт Кибернетики им. Глушкова).
13. Vartanov S.R. On Parallel Programming Language Caper. Lect. Notes in Computer Sci., HCPN-2001, p. 501-503.

ՀԱՎԵԼՎԱԾ 1. ՏԵՐՄԻՆԱԲԱՆՈՒԹՅՈՒՆ

Ապարատային ընդհատում (Аппаратное прерывание) - ազդանշան է, որը պրոցեսորին է հասնում արտաքին ապարատային սարքերից (ծայրամասային սարքեր)

Վերահսկիչ (Контроллер) - սարք է, որը կառավարում է այլ սարքեր կամ համակարգեր՝ մշակելով մուտքային տվյալները և ստեղծելով կառավարման ազդանշաններ

Ասինխրոն I/O (Асинхронный ввод-вывод) - I/O մշակման մի ձև է, որը թույլ է տալիս գործընթացին շարունակել կատարումը՝ առանց տվյալների փոխանցմանը սպասելու

Ապահատվածավորում (Дефрагментация) - դա կոշտ սկավառակի վրա տվյալների կազմակերպման գործընթաց է, որտեղ ֆայլերի կտորները միավորվում են ավելի տրամաբանորեն դասավորված բլոկների մեջ:

Բազմամեքենայական հաշվիչ համալիր (Многомашинный вычислительный комплекс) - հաշվողական մեքենաների խմբի ապարատային և ծրագրային համադրություն է, որոնցից յուրաքանչյուրը գործում է իր սեփական օպերացիոն համակարգով:

Բուֆերացում (Буферизация) - համակարգիչներում և այլ հաշվողական սարքերում տվյալների փոխանակումը կազմակերպելու մեթոդ, որն օգտագործում է բուֆեր՝ տեղեկատվությունը ժամանակավորապես պահելու համար

Բազային ռեգիստր (Базовый регистр) - միջզերատեսչական էլեկտրոնային փոխգործակցության համակարգ է, որը պարունակում է հանրային ծառայությունների մատուցման համար անհրաժեշտ տեղեկատվություն:

Բազմախնդրություն (Многозадачность) - միաժամանակ մի քանի բան անելու կամ առաջադրանքների միջև արագ անցնելու ունակությունն է:

Բազմահոսքայնություն (Многопоточность) - ծրագրի կարողությունն է միաժամանակ կատարել մի քանի հոսքեր (խնդիրներ): Հոսքերը կատարման միավորներ են գործընթացի ներսում, որոնք կիսում են ընդհանուր հիշողություն և ռեսուրսներ:

Բաժանված հիշողություն (Разделённая память) - գործընթացների միջև տվյալների փոխանակման ամենաարագ միջոցն է:

Բազմաթիվ հրահանգներ, մեկ տվյալ (MISD, Множественные инструкции, отдельные данные) - Համակարգեր, որոնցում կա բազմակի հրահանգների հոսք և մեկ տվյալների հոսք:

Բազմաթիվ հրահանգներ, բազմակի տվյալներ (MIMD, Несколько инструкций, несколько данных) - զուգահեռ ճարտարապետություններից մեկը, որը թույլ է տալիս մի քանի պրոցեսորների միաժամանակ կատարել հրահանգների տարբեր հաջորդականություններ տարբեր տվյալների հոսքերի վրա

Բազմաթիվ ընթացակարգեր, բազմաթիվ տվյալներ (MPMD, Несколько процедур, Несколько данных) - թույլ է տալիս մի քանի պրոցեսորների միաժամանակ աշխատել մի քանի տվյալների հոսքերի վրա

Բազմաթիվ ընթացակարգեր, մեկ տվյալ (MPSD, Несколько процедур, Одни данные) - զուգահեռ հաշվարկման ճարտարապետության տեսակ, որի դեպքում մի քանի պրոցեսորներ կատարում են տարբեր հրահանգներ տվյալների մեկ հոսքի վրա

Ենթապրոցեսորներ, ենթահամակարգեր (Подпроцессор /Субпроцессор) - սարքեր, որոնք սովորաբար ունեն սահմանափակ և մասնագիտացված ֆունկցիոնալություն՝ հրամանների նեղ շրջանակով:

Սակայն ժամանակի ընթացքում որոշ ենթապրոցեսորներ (օրինակ՝ պրոցեսորի միջուկները) սկսել են ունենալ ամբողջական հրամանաշար, գրանցիչներ և այլ բաղադրիչներ՝ դարձնելով դրանք ունակ ինքնուրույն ծրագրային կոդ կատարել:

Զուգահեռ հաշվարկներ (Параллельные вычисления) - տվյալների մշակման մեթոդ է, որի դեպքում առաջադրանքները բաժանվում են ավելի փոքր մասերի և կատարվում են միաժամանակ մի քանի պրոցեսորների կամ միջուկների վրա:

Զանգված (Массив) - Համակարգչային ծրագրավորման և տեղեկատվական գիտության մեջ տվյալների կառուցվածք, որը պահպանում է նմանատիպ տարրերի կարգավորված հավաքածու:

Ընդհատում (Прерывание) - կառավարման փոխանցումը գործող ծրագրից համակարգ (և դրա միջոցով համապատասխան ընդհատումների մշակման ծրագրին), երբ տեղի է ունենում որոշակի իրադարձություն:

Իրադարձություն (Событие) - հայտնաբերելի իրադարձություն կամ համակարգի վիճակի փոփոխություն, ինչպիսիք են օգտագործողի մուտքագրումը, ապարատային ընդհատումները, համակարգի ծանուցումները կամ տվյալների կամ պայմանների փոփոխությունները, որոնց համար նախատեսված է համակարգը վերահսկելու համար

Իրադարձությունների ինտեգրատոր (Интегратор событий) - բաղադրիչ, որն իրականացնում է համակարգի իրադարձությունների և ծրագրերի ինտեգրում

Ինդեքս ՕՀ-ում (Индекс в ОС) - կարող է պարունակել տեղեկատվություն ֆայլային տվյալներ պարունակող սկավառակի բլոկների մասին

Ծրագրային մոդուլ (Программный модуль) - ծրագրի անկախ և ֆունկցիոնալ առումով ամբողջական մասն է, որը նախագծված է որպես կոդի անկախ հատված

Կիրառական ծրագիր, հավելված (Прикладная программа, приложение) - ծրագիր է, որը նախատեսված է համակարգիչների և այլ թվային սարքերի վրա օգտատիրոջ որոշակի առաջադրանքներ կատարելու համար:

Կանոնակարգ (Протокол) - կանոնների և համաձայնագրերի ամբողջություն, որը սահմանում է տեղեկատվության (տվյալների) փոխանակման գործընթացը մեկ համակարգչի ներսում կամ համակարգչային ցանցի տարբեր համակարգիչների վրա տարբեր ծրագրերի միջև

Կոմպիլյացիա (Компиляция) - բարձր մակարդակի լեզվով գրված ծրագրի սկզբնական կոդը համակարգչի համար հասկանալի մեքենայական կոդի փոխակերպման գործընթաց

Հաշվիչ (Вычислитель) - սարք, որի միջոցով կատարվում են հաշվարկներ

Հաշվիչ սարք (Вычислительное устройство) - սարքավորումներ և համակարգեր, որոնք նախատեսված են մաթեմատիկական, տրամաբանական և հաշվողական գործողություններ կատարելու համար

Հաշվիչ համալիր (Вычислительный комплекс) - հաշվողական գործիքների մի շարք, որոնք նախատեսված են տարբեր խնդիրներ լուծելու համար

Հոսք (Поток) - կոդի կատարման անկախ միավոր մեկ գործընթացի շրջանակներում

Համակարգային կանչ (Системный вызов) - կիրառական ծրագրի կանչը ՕՀ-ի միջուկին՝ որևէ գործողություն կատարելու համար

Միջհամակարգչային փոխազդեցություն (Межкомпьютерное взаимодействие) - ցանցային ռեսուրսները երկու կամ ավելի համակարգիչների միջև բաժանելու (համօգտագործելու) գործընթաց

Մատրիցային պրոցեսոր (Матричный процессор) - մասնագիտացված հաշվողական սարք, որը ավելի պարզ պրոցեսորների ցանց է, որոնք աշխատում են զուգահեռ և տեղեկատվություն են փոխանակում իրենց ամենամոտ հարևանների հետ

Միջուկ (Ядро) - ՕՀ-ի կենտրոնական մասը, որը հավելվածներին ապահովում է համակարգչի ռեսուրսներին համակարգված մուտքով

Մեկնաբանություն (Интерпретация) - ծրագրի սկզբնական կոդի կատարումը

Մեկ հրաման, բազմակի տվյալ (SIMD, Одна команда, несколько данных) - զուգահեռ հաշվարկային ճարտարապետություն, որտեղ նույն հրահանգը միաժամանակ կատարվում է տվյալների մի քանի հավաքածուների վրա

Մեկ ծրագիր, բազմակի տվյալ (SPMD, Одно приложение, множество данных) - զուգահեռ հաշվարկման մոդել, որի դեպքում մեկ ծրագիր միաժամանակ աշխատում է մի քանի տվյալների տարրերի վրա

Մյութեքս (Мьютекс) - Ծրագրավորման մեջ համաժամեցման պրիմիտիվ է: Այն մեկ հոսքի կամ գործընթացի համար տրամադրում է բացառիկ մուտք դեպի համօգտագործվող ռեսուրս:

Պրոցեսոր (Процессор) - համակարգչի հիմնական բաղադրիչը, որը կատարում է բոլոր հաշվարկները և կառավարում է այլ համակարգերը

Պլանավորիչ (Планировщик) – ՕՀ-ի մոդուլ, որը ընտրում է համակարգ մուտքագրվող հաջորդ առաջադրանքները և գործարկվող հաջորդ գործընթացը

Ռեենտերայնություն (Реентерабельность) - ծրագրի հրահանգների նույն պատճենը հիշողության մեջ կարող է համօգտագործվել մի քանի օգտատերերի կամ գործընթացների կողմից

Սվոփինգ (Swapping, Свопинг) - համակարգչի RAM-ի և երկրորդական պահեստի (սովորաբար կոշտ սկավառակ կամ SSD) միջև տվյալների կամ ծրագրերի տեղափոխման գործընթաց

Սարքի դրայվեր (Драйвер устройства) - ծրագրակազմ, որը հնարավորություն է տալիս փոխազդել Օհ-ի և համակարգչի սարքավորումների միջև

Սերվեր (Сервер) - Ցանցային համակարգիչ, որը մշակում է տեղական կամ գլոբալ ցանցի այլ համակարգիչներից ստացված հարցումները: Այս տերմինը վերաբերում է նաև ծրագրակազմին, որը մշակում է օգտատիրոջ հարցումները:

Վերահսկիչ (Контроллер) - էլեկտրոնային սարք, որը նախատեսված է տարբեր համակարգեր և գործընթացներ կառավարելու համար

Վիրտուալ հիշողություն (Виртуальная память) - Համակարգչային հիշողությունը կառավարելու մեթոդ, որը թույլ է տալիս գործարկել ծրագրեր, որոնք պահանջում են համակարգի ունեցածից ավելի շատ օպերատիվ հիշողություն՝ ծրագրի մասերը ավտոմատ կերպով տեղափոխելով հիմնական հիշողության և երկրորդական պահեստի (օրինակ՝ կոշտ սկավառակի) միջև

Ցուցիչ (Указатель) - փոփոխական, որը պահպանում է հիշողության բջջի հասցեն, այսինքն՝ մատնանշում է այն

Ցանցային միացում (Сетевое соединение) - հաշվողական սարքերը և համակարգերը միացնելու մեթոդ՝ տվյալներ և հաղորդակցություններ կիսելու համար

Հաշվողական ցանց (Вычислительная сеть) - համակարգ, որը հնարավորություն է տալիս տվյալների փոխանակում կատարել հաշվողական սարքերի՝ համակարգիչների, սերվերների, ռոութերների և այլ սարքավորումների կամ ծրագրերի միջև

Քեշ հիշողություն (Кэш-память) - ծրագրի կամ սարքի ժամանակավոր պահեստ, որտեղ պահվում են օգտագործված տվյալները արագ մուտք գործելու համար

Օպերացիոն համակարգ (Операционная система) - համակարգչի կամ այլ սարքի աշխատանքը և օգտատիրոջ փոխազդեցությունն ապահովող ծրագրերի ամբողջություն

Օբյեկտային կոդ (Объектный код) - փոխակերպված ծրագրի տեքստը պատրաստ է համակարգչի կողմից կատարման համար

Ֆայլային համակարգ (Файловая система) - համակարգիչների, ինչպես նաև այլ էլեկտրոնային սարքավորումների՝ թվային տեսախցիկների, բջջային հեռախոսների և այլնի վրա տվյալները կազմակերպելու, պահպանելու և անվանակոչելու մեթոդ է

Ֆոնային պրոցես (Фоновый процесс) - համակարգչային գործընթաց, որը գործում է ֆոնային ռեժիմով և առանց օգտագործողի միջամտության

OpenCL՝ բաց հաշվարկային լեզու (OpenCL, открытый язык вычислений) - ունիվերսալ ծրագրավորման հարթակ, որը թույլ է տալիս մշակողներին ստեղծել կոդ, որը կարող է աշխատել

տարբեր հաշվողական սարքերի վրա, ներառյալ CPU-ներ, GPU-ներ, FPGA-ներ և այլ արագացուցիչներ։

CUDA՝ NVIDIA հաշվարկային հարթակ (вычислительная платформа NVIDIA) - Զուգահեռ հաշվողական հարթակ և API NVIDIA-ից: Թույլ է տալիս օգտագործել GPU-ներ ոչ միայն գրաֆիկայի մատուցման, այլև ընդհանուր նշանակության հաշվարկների (GPGPU) համար։