Системийн шинжилгээний хэсэг

1. Онолын судалгаа

Windows үйлдлийн системийн ерөнхий зохион байгуулалтын талаар товчхон

Windows-н өмнөх хувилбарууд MS-DOS үйлдлийн системтэй сайн зохицон ажилладаг боловч үйлдлийн систем гэсэн утгандаа бүрэн хүрч чадахгүй байсан бол өнөөгийн Windows95 өөрөө 1 цогц үйлдлийн системийг илэрхийлж чадах болжээ.

Windows95 үйлдлийн систем нь ерөнхийдөө 3 дэд системд хуваагддаг.

1. Win KERNEL
2. Win GDI
3. Win USER

Windows95 нь 16 болон 32 битийн программуудыг ажиллуулахдаа системийн виртуаль машин болон дээрхи 3 дэд системийн тусламжтайгаар зохион байгуулдаг. Системийн виртуаль машин гэдэг нь Windows-н бүх ажлыг дэмжиж байдаг Windows95-н бүрэлдхүүнд ордог үйлдлийн орчин юм. Мөн дээрхи 3 дэд системийн ажлыг дэмжиж байдаг.

32 битийн win гэдэг нь санах ойн 32 битийн моделийг ашиглаж байдаг windows-н шинэ хэрэглээ юм. Харин 16 битийн win гэдэг нь бидний урьд өмнө хэрэглэж байсан windows-н хуучин хувилбарууд юм. Тэдгээр нь санах ойн сегментийн хаяглалтийг буюу чухамдаа 80286 процессорын санах ойн моделийг ашигладаг. Тэдгээр нь Windows 3.x –нх шиг ажиллахдаа Windows95 нх шиг нэгдмэл нэг орон зайг хуваан эзэмшдэг бөгөөд олон бодлогын зарчимтай зохицодоггүй. Windows95-н хэрэглээний программийн төвшинд Windows 3.x –н API бүрэн тохирч ажилладаг.

Win32 API нь Microsoft –н WinNT, Win32S, Win3.1-н өргөтгөл win32 битийн интерфейс юм. Windows95 үйлдлийн систем нь олон бодлогын аргад үндэслэн дээрх application-уудыг жолооддог. Бүрхүүл гэдэг нь хэрэглэгчийг системтэй харилцах үйлдлийг хангаж байдаг 32 битийн Windows application юм. Windows95-н бүрхүүл нь windows3.x-н программ менежер, файл менежер зэргийг нэгтгэсэн application юм.

Зураг А-г үзнэ үү !

KERNEL

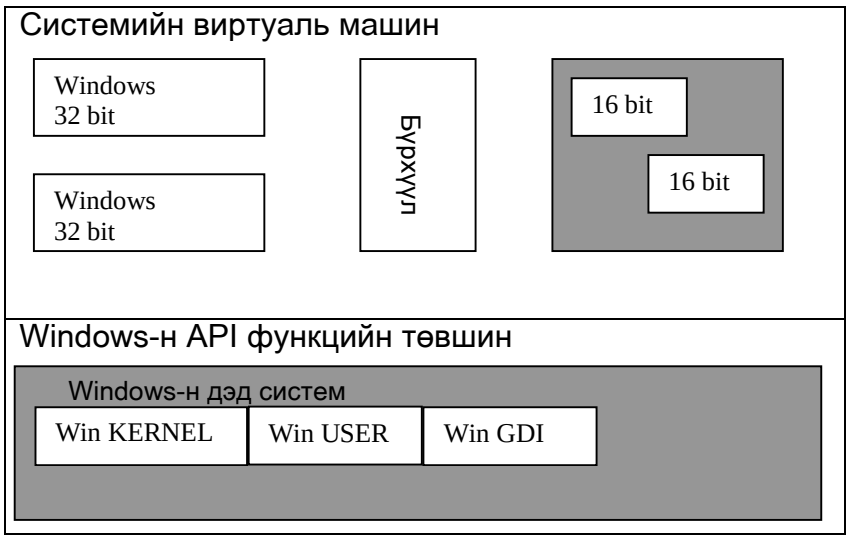
KERNEL нь windows-н доод түвшиний динамик санах ойн функцүүдээр хангаж өгч байдаг үйлдлийн системийн цөм юм. Энэ нь 16 ба 32 битийн windows-н үйлдлүүд тохирох
үйлчилгээг хангадаг.

GDI

GDI нь graphic device interface гэсэн товчилсон үг. Энэ нь Windows95-н график боломж болон дэлгэцэнд харуулах өнгө, хэлбэр дүрс, resolution-г хангаж байдаг. Win32 –т байдаг бүх шинэ боломжуудыг агуулна.

USER

USER нь дэлгэцийн диспетчер юм. Windows-н интерфейсийг үүсгэх, удирдах функцүүдийн цргц юм. Жишээ нь Хэрэглэгч хэд хэдэн MS-DOS –н Prompt гарган ашиглаж болно.
Хэрэглэгч арай илүү виртуалчилах боломжийг хангадаг хэд хэдэн давуу тал гарч ирлээ.



Зураг А

Дээр харлуулсан орон зайг хамтарч эзэмшинэ. Энэдээс Windows95 нь 16 битийн программуудад тусад нь орон зай гарган өгч орчинийг нь бүрдүүлж өгдөг нь харагдаж байна.

Файл удирдах дэд систем
Сүлжээ удирдах дэд систем
Үйлдлийн системийн үйлчилгээ
Виртуал машин удирдах дэд систем
Хөтлөгч хэрэгслүүд

Бүрхүүл

Энэ нь хэд хэдэн системийн удирдлагыг хэрэглэх нөөцүүдийн боломжуудаар хангах программ юм. Windows95-үйлдлийн системийн бүрхүүл нь дэлгэц системийн нөөцүүдтэй харилцан ажиллахыг удирдан зохион байгуулдаг.

API

Хэд хэдэн зуун API функцүүд windows-н бүх системийн хийх үйлдчилгээг хангаж өгдөг. Windows-н орчин бол интерфейс баазын дуудалтаар ханддаг. Үүнийг API гэдэг.

Системийн виртуаль машин

Windows-н бүх application ситемийн виртуаль машины тухайн хил хязгаарт үйлдлээ гүйцэтгэдэг. 16 битын application –ууд өөрийн нэгдмэл хаяглалын орон зайд ажилладаг.

Энэ нь application бүрийн өөрийн орон зайн хаяглалтыг 32 битын хаяглалтаар хангаж дэмжиж өгдөг.

MS-DOS-н виртуаль машин

Windows95 олон тооны MS-DOS-н программыг хамгаалалтын горим эсвэл 8086 виртуаль үйл ажиллагаа ажиллуулж дэмжиж байдаг. MS-DOS-н виртуаль машин хэд хэдийг үүсгэж болно. Зураг Б-г нягтлан үзнэ үү!

Виртуаль машины диспетчер

Виртуаль машины диспетчер нь үнэндээ үйлдлийн системийн зүрх юм. Энэ нь доод түвшиний санах ойн удирдлага болон драйверуудын виртуаль зохион байгуулалтын үйлчилгээг хянаж ажилллаж байдаг.

Windows95 –н диспетчерүүд

Windows95 –н виртуаль машины диспетчер дотор хоёр диспетчер ажиллаж байдаг.

Үүнд: Үндсэн диспетчер. (primary scheduler)

Энэ нь thread-дын приоретитүүдийг тооцоолж хариу өгнө.

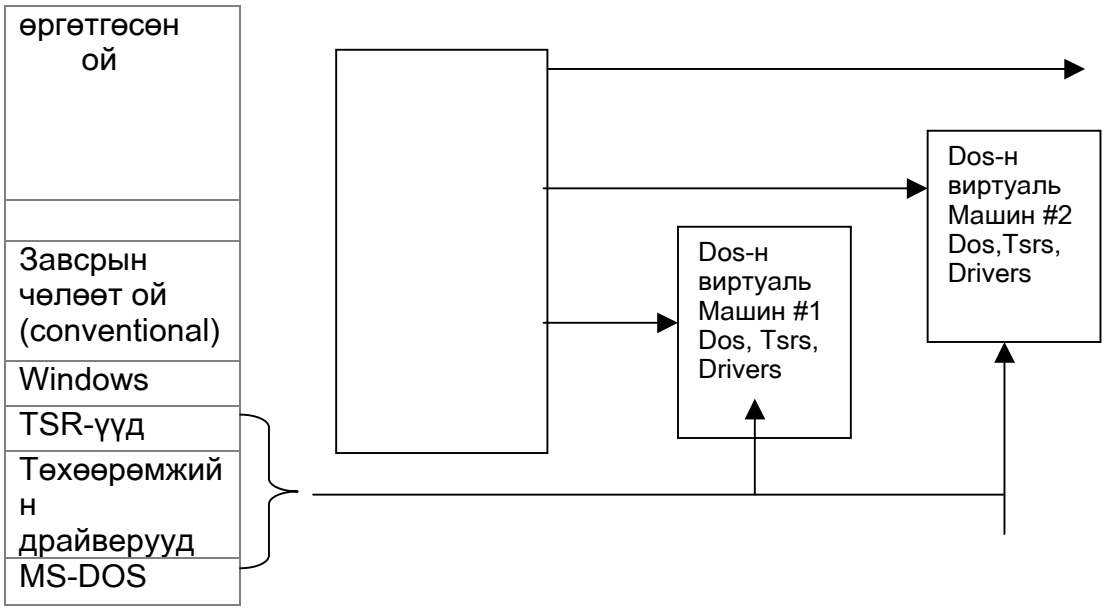
Квантын диспетчер (timeslice scheduler)

Энэ нь хугацааны квантуудад хувааж хэсэгчлэн ажилуулана.

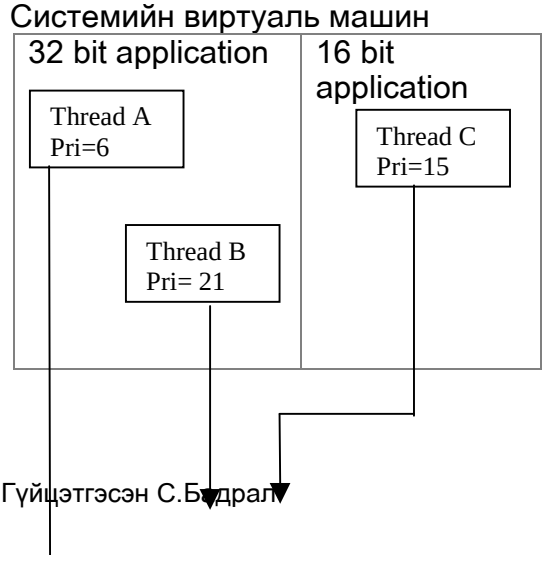
Үнэн хэрэгтээ квантын диспитчер аль thread-н ямар хувийг нь ямар хугацаанд процессороор үйлчлүүлэхийг тооцоолно.

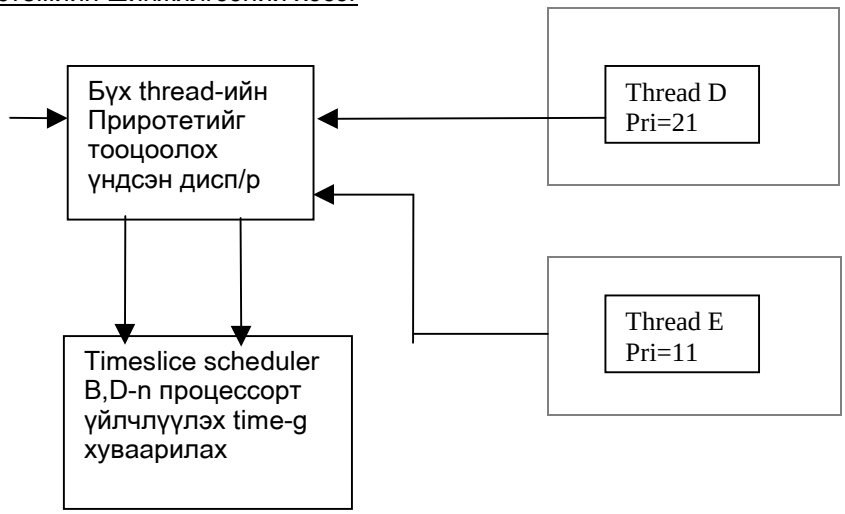
Процесс 0-31 хүртэл приоритеттой байдаг ба үүнийг үндсэн диспетчер бодоод квантын диспетчер лүү гаргана. Хэрвээ процессийн хугацааны квант 100 хувь болсон бол VMStat_Background эсвэл VMStat_High_Pri_Background гэдэг төлвийг үлдээдэг.

Зураг Б-г үзнэ үү.

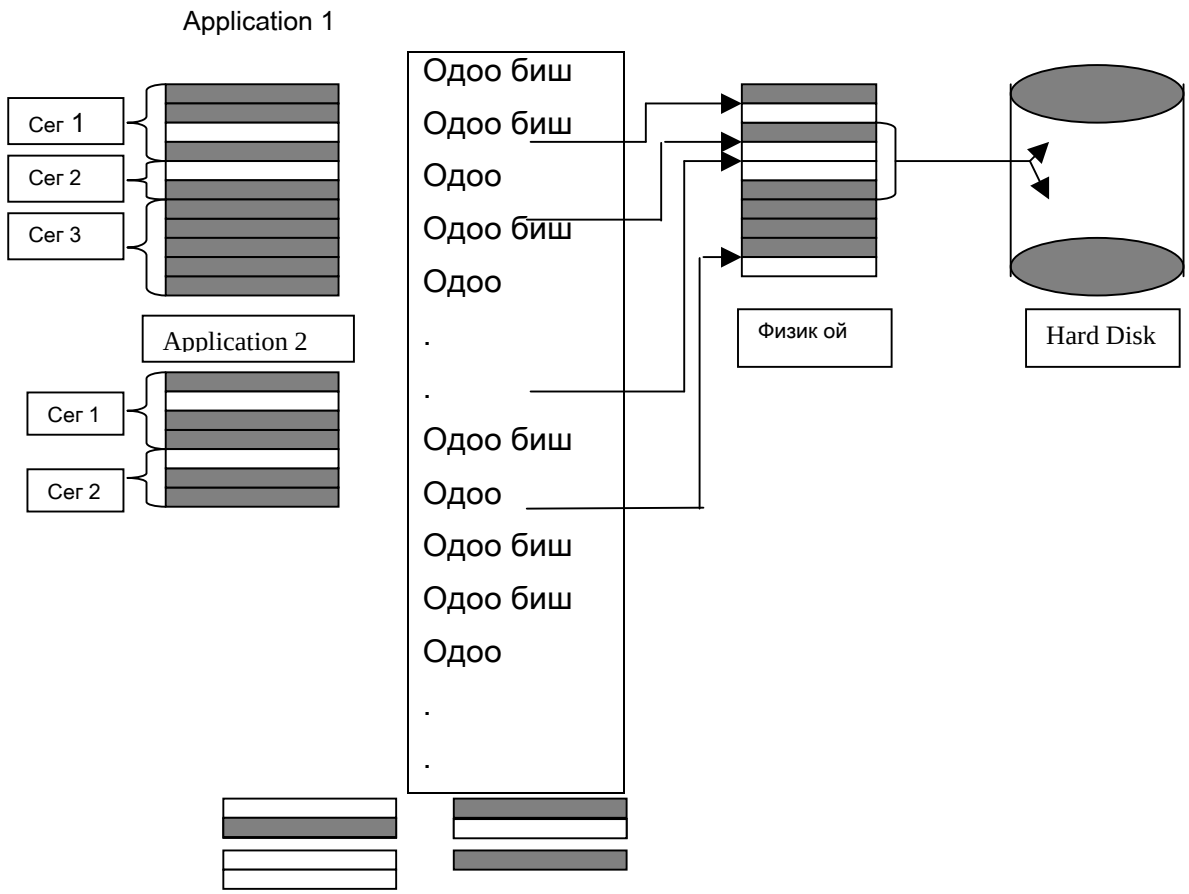


Зураг Б





Зураг В



Зураг Г

Windows95 үйлдлийн систем санах ойгоо сегмент хуудсын зохион байгуулалтаар зохион байгуулдаг. 80386 процессорын хувьд Windows3.1 физик ойгоо 4кв хэмжээтэй хуудсуудад хуваадаг. Тэдгээр хуудсууд нь Windows-н application бүрийн виртуаль хаяглалын орон зайн картуудад дүрслэгддэг.

Windows үйлдлийн систем олон бодлогын зарчимаа message дамжуулах аргаар боловсруулдаг.

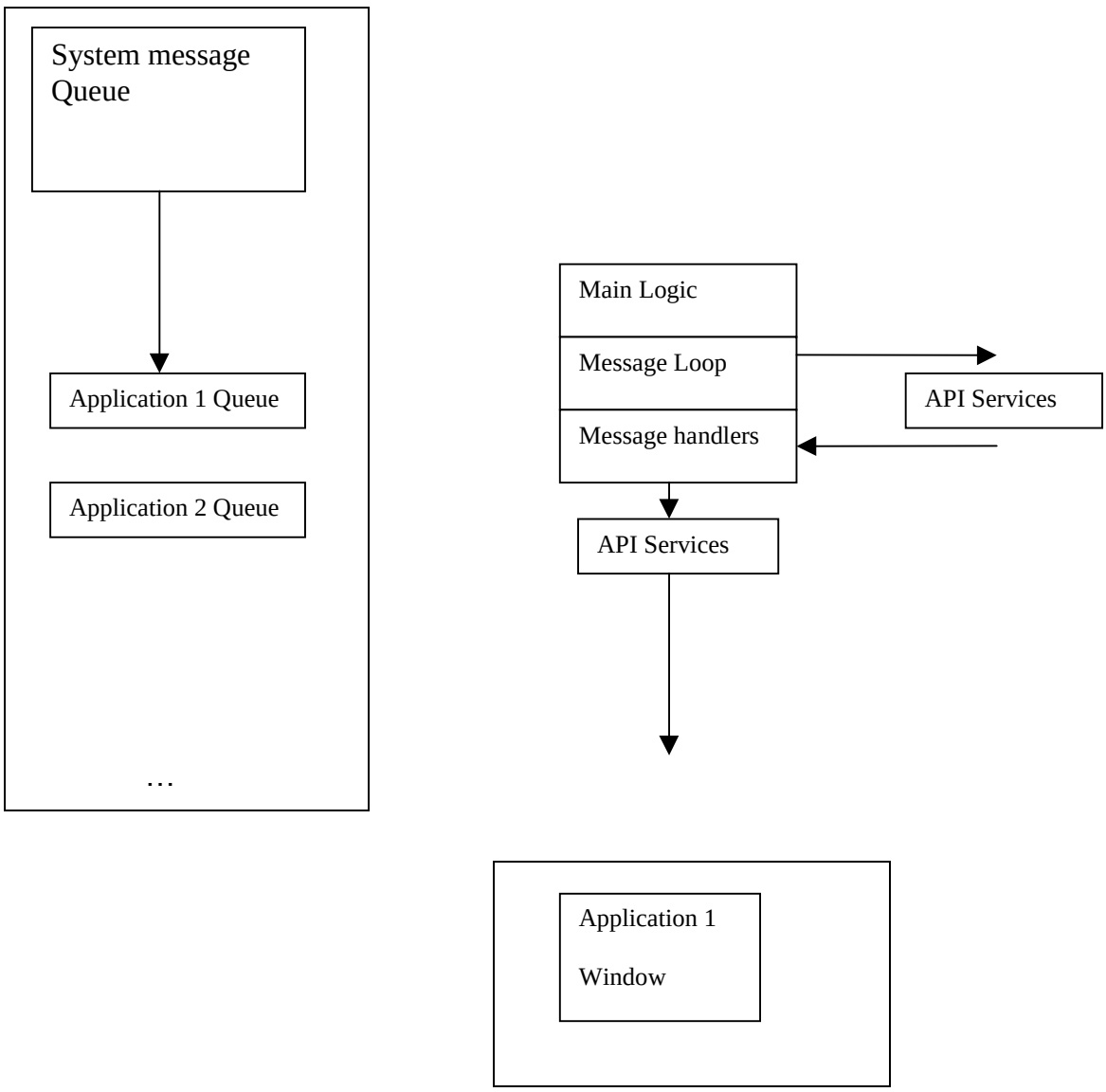
Хүнтэй аналогоор тодорхойлбол : Message бол windows-н системийн программын цус нь юм. Getmessage()/Dispatchmessage() циклийн функцүүд зүрх нь цонхны функцүүд болон windows программийн түүнтэй холбоотой функцүүд артер венийн судас ба хялгасан судсууд нь юм гэж хэлж болно. Судасны үүрэг мэдээж message дамжуулах. Хүн цусгүй бол үхэхийн адил windows программ message боловсруулахаа зогсвол мөн үхнэ.

Windows үйлдлийн системийн Message боловсруулалт

Windows программ бүр кодуудын блокуудыг агуулах ёстой бөгөөд Windows тэдгээрийг шинжилээд application-руу message-үүдийг илгээдэг. Программ цонх бүрдээ нэмэлт кодын блоктой ба тэдгээр нь программыг дэлгэцэнд үзүүлэх ба удирддаг.

Энэ кодууд application-уудын нэмэлт жижиг хэсгүүдийг хэвийн биелүүлэхэд хэрэгтэй. Эхний бүлэг кодуудыг message цикл гэдэг ба бусад группууд (цонх бүрд 1)-ыг messade handler гэдэг.

WINDOWS



Message-н ерөнхий бүтэц

```
Typedef struct tagMSG
{  HWND        hwnd;
   UINT        message;
   WPARAM wParam;
   LPARAM      lParam;
   DWORD       time;
   POINT       pt;
} MSG;
```

MSG бүтэц нь message-н мэдээллийг thread-н message дарааллаас авдаг.

Hwnd талбар

Ямар цонх харуулах хэрэгтэйг зааж өгнө. Энэ талбар зөвшөөрөгдөөгүй цонхны логик номерийг агуулж болно.

Жишээ нь : Message-г дараалалд PostAppMessage() функцээр NULL цонхтой оруулая. WndProc –д message-н шуудан үргэлж боловсруулагддаг ба тэд үнэндээ системийн цонхны удирдлагаас шалтгаалдаггүй.

Message талбар

Өгөгдсөн message-н өөрийгөө төлөөлөх message код байна.

Жишээ нь : 0x000F-нь WM_PAINT юм. WM_USER (0x0400) microsoft-н ерөнхий системд өргөн ашиглагддаг. Windows-д системийн message-н цар хэмжээ хүрэлцээтэй их биш учир заримдаа ерөнхий нийтлэг message-д хандахдаа массив ба заагч функц ашиглан хандаж болно.

wParam lParam талбар

Message-н тухай (Message талбарын утга) нэмэлт өвөрмөц мэдээллийг агуулна.

Жишээ нь: Хэрвээ message нь WM_MOUSEMOVE байсан гэвэл wParam нь ямар товчлуур дарсан мэдээллийг агуулах ба lParam нь заагчийн координатыг (x,y) агуулна.

Үүнчлэн Message талбарын утга ямар байхаас хамаарч wParam, lParam-н утга тодорхойлогдоно.

Time талбар

Систем асаагдсанаас хойшхи хугацааг миллисекундээр илэрхийлнэ.

Pt талбар

Курсорын дэлгэц дэх байрлалыг агуулна.

Ер нь Windows программд message -үүдийг 5 төрөлд хуваж программчилж болно.

- QS_KEY, QS_MOUSEMOVE, QS_MOUSEBUTTON message-үүдийг GetQueueStatus() функц тусад нь тооцдог боловч эдгээр нь бүгд оролтын message –н жишээ болно. Оролтын message –д аппарат төхөөрөмжийн гар, хулгана гэх мэт message-үүд орно. Тэд системийн Message дараалалд хуваагдан хадгалагддаг. Хойно системийн message дараалалыг тайлбарласан байгаа.
- QS_POSTMESSAGE message-г программын message дараалалд PostMessage() эсвэл PostAppMessage() функцийн тусламжтайгаар хийнэ. Программ (application) бүр зөвхөн нэг өөрийн message дараалалтай байна. Программын message дараалалыг хожим үзнэ.
- QS_PAINT message QS_TIMER message шиг хүлээхгүйгээр өөрийн дарааллаа боловсруудаг. Энэ message программд хэзээ л шаардагдана тэд зайлшгүй байж боловсруулагддаг. Системийн удирдах цонх зайлшгүй шинэтгэх онцгой цонхны тодорхойлолтыг асуудаг. Салбар цонх гажуу болоход хэрэглэгч системд мэдээ илгээнэ. Энэ нь цонх дахин зурах хэрэгтэй юу гэдгийг заадаг QS_PAINT төлвийг тогтоодог байна. Программ QS_PAINT төлвийг шалгаад хэрэв WM_PAINT message олдвол тэр даруй шинэ message асуудаг.

  Программын message дараалалд message байхгүй байхад дахин зурагдах message боловсруулах эсэхийг сайтар тэмдэглэх нь чухал.
- QS_TIMER message –г шаардлагатай үед дуудахдаа GetMessage() ба PeekMessage () функцийг ашигладаг. Энэ message программын болон системийн message дараалалд хадгалагддаггүй. Тиймээс таймерийн message системийн болон программын дарааллыг дүүргэх боловч хадгалагддгүй.
- QS_SENDMESSAGE

  SendMessage() API message илгээх цонхонд баталгаа өгөөд хүлээн авах цонхноос удалгүй хариу авна. Нэг программаас хоёр цонхны хооронд message илгээх нь тийм ч хэцүү биш. Энэ ажлыг энгийн дуудлага функцын тусламжтайгаар гүйцэтгэдэг. Тийм дуудлага функцүүдийн нэг SendMessage() функц нь тиймч хялбар бус. Хоёр өөр бодолгын (Task) хооронд message дамжуулах нь хэцүү.

Программын message дараалал

Хэрвээ Windows message –г системээс авая гэвэл дээрх зүйлсийг бүгдийг сайн эзэмшсэний дараа хэрэглээний системийн 2 төрлийн дарааллыг мэдэх хэрэгтэй.
InitApp() функцээр программын USER модульд хугацаанд нь инитциализацилж программийн message дараалалыг зохион байгуулна. CreateQueue() ба CreateQueue2() USER модулийн дотоод функцийн тусламжтайгаар message дараалалыг санах ойд инитциализалж байрлуулна. Message дараалалыг санах ойн сегментэд зохион байгуулахдаа глобалаар бөөнд нь байрлуулна. GetTaskQueue() функцийн тусламжтайгаар идэвхитэй message дараалаллын логик дугаарыг авч болно.
HANDLE FAR PASCAL GetTaskQueue(HANDLE htask) функцын htask параметрт 0 утга дамжуулбал одоогийн (идэвхитэй) бодлогын (Task) логик номерийг буцаана.
Цонх бүр системд холбоотой программын message дараалаллыг тусгаарлана. Message дараалалыг логикоор байлгах талбарыг олоход WND өгөгдлийн бүтцийг харах хэрэгтэй.
Хэзээ ямар дараалалд тэднийг оруулахыг message дараалалын логик номерыг WND бүтцэд өгнө.
Desktop цонх хүртэл message дараалалтай холбоотой. Message дараалал үндсэндээ бүрэн нарийн томьёолсон desktop цонхтой холбоотой. Үүнийг хийхийн тулд өгөдлийн бүтцийн цонх агуулах өгөгдлийн сегмент модулийг ангилах хэрэгтэй.
Өөрсдөө найруулагч (Mixer) эсвэл desktop цонхны логик номерийг олох хэрэгтэй. Desktop цонхны утган дээр 18h –г нэмж USER модулийн сегмент дэх холилдсон өгөгдлийг ангилах хэрэгтэй.
Хулганы товчлуур товших, программ ажиллуулах ба бусад даалгавар биелүүлэхэд desktop цонхны message дарааллын хаана хэзээ шилжин орж байгааг мэдэх хэрэгтэй. Программын дараалалд message байрлуулахад PostMessage() функцийг хэрэглэнэ.

PostMessage() функцийг дуудахад ард нь Windows-н дотоод функцүүд жишээлбэл DefWindowProc() ажилаж байдаг.

Хэрвээ программын message дараалалд бодлого байрлуулахыг хүсвэл (түүний HWND мэдэгдэхгүй эсвэл бодлого цонхгүй байх үед) PostAppMessage() функцийг ашиглаж болно.

Дээд төвшиний том программын хувьд message дарааллын хэмжээг нэмэгдүүлэн хэрэглэж болно. Энэ тохиолдолд SetMessageQueue() функцыг хэрэглэдэг. Энэ функцийг дуудахын өмнө дурын цонх үүсгэж Message дарааллын эхнээс USER модулийг устгаж шинээр үүсгэх нь чухал. Хэрвээ эхнийхийг хэрэглэсэн бол энэ нь системд Message дамжуулж болно. SetMessageQueue нь Win.ini файлд тодорхойлогдсон байдаг ”DefaultQueueSize” түлхүүрийг өөрчилдөг.

Программ бүрийн бодлогын өгөгдлийн баазтай (TDB) программын Message дараалал нягт холбоотой. Программын Message дараалал TDB-н нэг хэсгийг бүрэлдүүлнэ.

DLL Message дараалалтай байж болохгүй. Учир нь DLL нь сангийн код. Программын гүйцэтгэлийн үед л бүрэлдүүлдэг. Тиймээс тэд обьект эзэмшиж болохгүй. Жишээ нь: Message дараалал, файлуудын логик дугаарын таблиц Message программын message дараалалаас өөрийн уншилтийг хүлээж байдаг. Харагдах байдал нь:

DWORD	Extrainfo
HWND	hwnd
WORD	message
WPARAM	wParam
LPARAM	lParam
DWORD	time
POINT	pt

Бодлогын дунд Message боловсруулахад SendMessage()-г хэрэглэх ба энэ өгөгдлийн бүтцийн логик номерийг GetTeskQueue() функц буцаана.

…

…

…

Message –н циклийн буфер

Энэ муж зарчмийн хувьд ROM-BIOS-н гарны буфертэй адил. Дараалалаас унших ба бичих хоёр заагчийг агуулна. Хэрвээ эхлэлийн заагч нэмэгдсээр Message –н сүүлчийн талбарт очиход заагчийг хамгийн эхэнд буцаан шилжүүлдэг. Хоёр заагч хэзээ тэнцэнэ тэр үед дараалал хоосон байна. Хэрвээ дараалалд message болон өгөгдөл байх үед бодлогын үйл ажиллагааны дундуур message илгээх болбол SendMessage() функцийг хэрэглэнэ.

Программийн гүйцэтгэл дуусахад программын төлвийн үг байх уу? Мэдээж байна. Хэрвээ ажиллаж дууссан бол түүний дууссан код бидэнд ирнэ.

Программын Message дараалал гэж Message-г дараалалд оруулж хадгалдаг энгийн муж юм.Энэ нь цонхны логик номероор зарим бодлоготой холбогддог. Энэд хадгалагдах битүүдийн амьдралын байдал нь GetMessage ба PeekMessage функцүүдэд чухал хэрэгтэй байдаг.

Системийн Message дараалал

Системийн message дараалал программын message дарааллын ах нь юм. Оролтийн төхөөрөмжүүдийн message эсвэл аппаратурын message-үүд ерөнхийдөө бүх message -үүд системийн message дараалалд баригддаг. Ө.Х системийн message дараалалд бүх message оролтийн тогтолцооний мэдээ юм. (Дигитайзороор message дамжуулсан болдог.)

Программийн message дараалалын адил USER модулиар найруулж инициализациладаг. Системийн message дараалалын формат нь түүнд хадгалагдаж буй message-н форматаас бусад нь программын message дараалалтай адил. Ерөнхийдөө системийн дараалал нь мэдээллийн буферийн циклт зохион байгуулалтанд хамаарагдах хэсгийг л хамардаг байна.

Системийн message дараалал программын дарааллаас их хэмжээний message илүү хадгална. Энэ нь аппаратурын илэрлийн хэлбэлзэл нилээд өндөр байдагтай холбоотой.

Windows 3.1 системийн message дараалал 120 аппаратурын message хадгалж чаддаг.

Системийн message дараалалд орж ирж байгаа мэдээлэл нь ямар нэг шинэ цонхонд хамаарагдахгүй. Энэ нь windows3.x-г Windows NT-с ялгаж өгдөг ганц ялгаа юм.

Жишээ нь: WM_LBUTTONDOWN message фокусыг өөрчилдөг. Дараалалд буй дараагийн message нь хуучин цонх руу биш шинэ цонх руу явна.

Системийн message дараалалд дараах форматууд орно.

DWORD нэмэлт мэдээлэл
WORD meesage-н номероос шалтгаалж утга нь өөрчилөгдөнө.
WORD Мэдээллийн номер. Үнэндээ WM нь мэдээллийн тоо юм.
WORD meesage-н номероос шалтгаалж утга нь өөрчилөгдөнө.
DWORD Цаг. Систем асаагдсанаас хойшхи миллисекундийн тоо

хэмжээ.

GetMessage /PeekMessage функцүүдийн кодод 5 төрлийн мэдээлэл хэрхэн боловсруулагдахыг авч үзье.

QS_SendMessage

Getmessage2 нь хэд хэдэн удаа CheckForNewInput() функцийг дуудна. Түүний үндсэн үүрэг нь илгээсэн мэдээллийг шалгахад оршино. Хэрвээ GetMessage() нь бодлогыг SleepHq-р унтуулсанаар дуусаж байгаа бол илгээсэн мэдээлэл нь SleepHq-р шалгаж байна гэсэн үг.

QS_POSTMESSAGE

ReadMessage функц программийн message дараалаас message салган авна. GetMessage эсвэл PeekMessage функцийн дуудалтаар өгөгдсөн хаягаар мэдээллийн талбар хэвлэгдэнэ.

QS_MOUSE & QS_KEY

ScanSysQueue функц системийн message дараалалаас message-г гарган авдаг. GetMessage эсвэл PeekMessage функцийн дуудалгаар өгөгдсөн хаягаар мэдээллийн талбар хэвлэгдэнэ.

QS_PAINT

DoPaint функц WND бүтцээс дахин зурах параметрийг гарган авдаг. GetMessage эсвэл PeekMessage функцийн дуудалгаар өгөгдсөн хаягаар мэдээллийн талбар хэвлэгдэнэ.

QS_TIMER

DoTimer() функц эхлээд синтезэлж дараа нь программийн message дараалалд TIMER-аас бичнэ. Дараа нь GetMessage2 функц эхнээс эхэлнэ. QS_PostMessage-н message байсан юм шигээр таймераас message-г хайж эхэлнэ.

Дээрхээс дүгнэвэл:

1. GetMessage эсвэл PeekMessage функцээс дуудагдаж буй бодлого хэрвээ түүнийг хүлээж буй message байгаа бол өөр ямарч дараалалд холбогдохгүй.
2. Message-үүдийн приоритетийг тогтооё. SendMessage функцээр илгээгдсэн message дандаа хамгийн өндөр приоритеттэй байна. Дараа нь PostMessage-н тусламжтайгаар дараалалд орсон message. Дараа нь оролтын системд орж буй мэдээлэл. Түүнийг дагаад WM_PAINT message орно. WM_PAINT-д орсон message-н шалгалт бусад төрлийн message шалгагдсаны дараа явагддаг. Учир нь бусад message-г шалгах явцад нэмэлт илрэх магадлалтай. Ашиггүй шалгалтыг багасгахын тулд бусдыг нь эмхэтгэсний дараа шалгаж

боловсруулдаг. Түүний дараа WM_TIMER шалгагддаг. Сүүлийн 2 message-г шалгах журам нь маш чухал. Хэрвээ WM_WTIMER –нь өмнө WM_PAINT-аас өмнө шалгагдсан бол CLOCK болон түүнтэй төстэй программууд ажиллаж чадахгүй байх байсан.(!?)

Message ба Message дараалалын гол функцууд

1. BroadcastSystemMessage

BroadcastSystemMessage функц тодорхойлогдсон хүлээн авагчууд руу message-г илгээдэг. Хүлээн авагчууд нь application-ууд , install драйверууд, windows-н сүлжээний драйверууд, системийн түвшиний төхөөрөмжүүдийн драйверууд, эсвэл эдгээрийн бүрдэл хэсгүүдийн эвлүүлэг (Комбинаци) байна. Ер нь л бүх цонх руу гэж ойлгож болно.

Бүтэц нь

```
long BroadcastSystemMessage(
   DWORD dwFlags,
   LPDWORD lpdwRecipients,
                                  UINT uiMessage,
                                  WPARAM wParam,
   LPARAM lParam
  );
```

хэлбэртэй.

2. DefWindowProc

DefWindowProc функц дурын цонхны процедурыг дуудаж тэр программд боловсруулаагүй ямар нэгэн цонхны message-үүдэд боловсруулагдах нөхлийг бүрдүүлж өгнө. Энэ функц баталгаа өгсөн бүх message боловсруулагдана. DefWindowProc бол цонхны процедураас хүлээн авсан зарим параметрүүдтэй дуудагддаг.

Бүтэц нь:

```
LRESULT DefWindowProc(
   HWND hWnd,
   UINT Msg,
   WPARAM wParam,
   LPARAM lParam
  );
```

3. DispatchMessage

DispatchMessage функц цонхны процедур луу message-г илгээдэг. Энэ функц ерөнхийдөө GetMessage функцээр буцаасан message илгээхэд хэрэглэгддэг.

Бүтэц:

```
LONG DispatchMessage(
    CONST MSG *lpmsg
    );
```

4. GetInputState

GetInputState функц mouse-даралт эсвэл гарны message-н дуудалтад thread-н message дараалал үүссэн үү үгүй юу гэдгийг шалгана.

Бүтэц:

```
BOOL GetInputState(VOID)
```

5. GetMessage

GetMessage функц нь дуудагдсан thread-н message дараалалд тодорхой бүтцэд байрласан message-г хүлээн авдаг. Энэ функц PostThreadMessage-тэй хоёулаа хамтран тодорхой цонх ба thread message-үүдийг хүлээн авч чадна. Функцийн хүлээн авсан message-үүдийн дотор утгууд нь агуулагдана. GetMessage бусад цонхнууд, thread-үүд эсвэл application-уудын message-үүдийг буцаахгүй.

Бүтэц:

```
BOOL GetMessage(
   LPMSG lpMsg,
   HWND hWnd,
   UINT wMsgFilterMin,
   UINT wMsgFilterMax
  );
```

6. GetMessageExtraInfo

GetMessageExtraInfo функц GetMessage эсвэл PeekMessage функцүүдээр буцаасан message-тэй холбоотой нэмэлт мэдээллийг буцаана. Энэ мэдээлэл гар эсвэл заах төхөөрөмжөөр message-рүү нэмэгдэж магад.

Бүтэц:

```
LONG GetMessageExtraInfo(VOID)
```

7. GetMessagePos

GetMessagePos функц курсорын дэлгэцэн дэх байрлалыг өгдөг long утга буцаана. Энэ байрлал GetMessage-р буцаагдсан сүүлийн message хэзээ курсор эзэмшсэнийг заана.

Бүтэц:

```
DWORD GetMessagePos(VOID)
```

8. GetMessageTime

GetMessageTime функц систем асаагдсанаас хойшхи хугацааг миллисекундээр хэмжиж Long integer төрлөөр буцаана.

Бүтэц:

```
LONG GetMessageTime(VOID)
```

9. GetQueueStatus

GetQueueStatus функц дуудагдсан thread-н message дарааллын message-үүдийн шинж ба төлвийг буцаана.

Бүтэц:

```
DWORD GetQueueStatus(
    UINT flags
  );
```

10. PeekMessage

PeekMessage функц thread message дарааллыг шалгаж тодорхой бүтцэд message хадгалагдаж уу үгүй гэдгийг тодорхойлно.

Бүтэц:

```
BOOL PeekMessage(
    LPMSG lpMsg,
    HWND hWnd,
    UINT wMsgFilterMin,
    UINT wMsgFilterMax,
    UINT wRemoveMsg
  );
```

11. PostMessage

PostMessage функц message дараалалд thread-тэй тодорхой цонх үүсгэж холбогдсон ба хүлээлтгүй бол thread-д боловсруулах message-г буцаана.

Message-үүдийг message дараалалаас GetMessage эсвэл PeekMessage функцийг дуудаж авна.

```
BOOL PostMessage(
   HWND hWnd,
   UINT Msg,
   WPARAM wParam,
   LPARAM lParam
  );
```

12. PostQuitMessage

PostQuitMessage функц thread төгсгөх хүсэлт гаргавал windows-руу зааж өгнө. Энэ нь ерөнхийдөө WM_DESTROY message-д хариу хандахад хэрэглэгдэнэ.

Бүтэц:

```
VOID PostQuitMessage(
   int nExitCode          );
```

13. PostThreadMessage

PostThreadMessage функц тухайн thread-н message дараалалд message-г байрлуулаад thread message-г боловсруулах хүртэл хүлээж байгаад утга буцаана.

Бүтэц:

```
BOOL PostThreadMessage(
   DWORD idThread,
   UINT Msg,
   WPARAM wParam,
   LPARAM lParam
  );
```

14. ReplyMessage

ReplyMessage функцийг SendMessage функцийн илгээсэн message рүү тухайн илгээсэн SendMessage функцд удирдлагыг буцаахгүйгээр хариулахад (буцаахад) хэрэглэдэг.

Бүтэц:

```
BOOL ReplyMessage(
   LRESULT lResult
);
```

15. SendMessage

SendMessage функц нь өгсөн message-үүдийг цонх болон цонхнууд руу илгээдэг. Функц тухайн цонхны цонхны процедурыг дуудах ба цонхны процедур message боловсруулж дуусах хүртэл утга буцаахгүй.

```
LRESULT SendMessage(
   HWND hWnd,
   UINT Msg,
   WPARAM wParam,
   LPARAM lParam
  );
```

16. SendMessageCallback

SendMessageCall функц нь өгсөн message-үүдийг цонх болон цонхнууд руу илгээдэг. Функц тухайн цонхны цонхны процедурыг дуудаад утга буцаана. Дараа нь цонхны процедур message боловсруулаад, систем тухайн callback функцыг (Message хүлээн авах) дуудаад, message боловсруулалтын үр дүнг программд тодорхойлсон хувьсагчаар callback функц рүү дамжуулна.

Бүтэц:

```
BOOL SendMessageCallback(
   HWND hWnd,                          // handle of window
   UINT Msg,                           // message to send
   WPARAM wParam,                      // first message parameter
   LPARAM lParam,                      // second message parameter

   SENDASYNCPROC lpResultCallBack,    // function to receive message value
   DWORD dwData                        // value to pass to callback function
  );
```

17. SetMessageExtraInfo

SetMessageExtraInfo функц идэвхитэй thread-дэхь message-н нэмэлт мэдээллийг өөрчилнө.

Бүтэц:

```
LPARAM SetMessageExtraInfo(
```

```
    LPARAM lParam
);
```

18. TranslateMessage

TranslateMessage функц message-үүдийн virtual-key-нүүдийг дамжуулдаг. (Тэр тусмаа CHAR message-үүдийн) CHAR message-үүд дуудагдсан thread-н message дараалал руу илгээгдэх ба дараагийн хугацаанд нь thread GetMessage эсвэл PeekMessage функцээр уншдаг.

Бүтэц:

```
BOOL TranslateMessage(
  CONST MSG *lpMsg
                    );
```

Процесс ба Thread

Процессийг ихэвчлэн биелэгдэж буй программын экземпляр (хувь) гэж тодорхойлдог. Win 32-т процесст 4 гб хүртэлх хаягийн орон зайг ногдуулдаг. Win 32-ын процесс яг өөрөө ”юу ч хийдэггүй” гэж хэлж болно, үнэндээ EXE файлын код болон өгөгдлийг агуулсан 4гб хаягийн орон зайг эзэмшдэг. Энэ хаягийн мужид шаардлагатай DLL-ийн өгөгдөл ба код мөн ачаалагдана. Хаягийн мужаас гадна процессийн мэдэлд файл динамик санах ойн хэсгүүд, Thread зэрэг нөөцүүд байна. Процессийн ”амьдрах хугацаанд” үүссэн нөөцүүд нь түүнийг ажиллаад дуусахад заавал устгагдах ёстой.

Процесс ямар нэгэн үйл хийхийн тулд түүн дотор ямар нэгэн Thread үүсгэх хэрэгтэй. Үнэн хэрэгтээ Thread-үүд нь процессийн хаягийн муж дахь кодын биелэлтийг хариуцдаг. Үндсэндээ процесс хэд хэдэн Thread-тэй байж болох ба энэ тохиолдолд тэдгээр нь процессийн хаягийн муж дахь кодыг зэрэг биелүүлэх болно. Үүний тулд Thread болгон өөрийн гэсэн стек процессоруудын региструудын бүрдэлтэй байдаг.

Бүх Thread-ууд ажиллаж байхын тулд үйлдлийн систем нь тэдгээрт тус болгонд нь тодорхой процессорын хугацаа олгодог. Thread-уудыг хугацааны нэгж хэсэг болох ”квант”-аар биелүүлдэг. Нэг Thread-ын кодыг нэг ”квант” хугацаанд биелүүлээд дараагийн Thread-рүү шилжинэ гэсэн үг.
Процесс үүсэхэд түүний анхдагч (үндсэн) Thread автоматаар үүсдэг. Цаашид энэ Thread өөр Thread үүсгэх боломжтой г.м.

Win 32 хоёр төрлийн application-ыг дэмждэг : GUI (график интерфейс дээр үндэслэгдсэн) болон CUI (консоль) GUI төрлийн application-ны гадаад интерфейс нь цэвэр график байдаг : цонх, меню, диалог цонх зэргийг агуулсан байдаг. Консоль төрлийн application нь MS-DOS-ын текст горимд ажилладаг программтай төстэй. Хэдийгээр консоль application экран дээр цонхон дотор байрлах боловч зөвхөн текстийг л харуулна. Command. Com бол консоль application-ны нэг жишээ юм.
Систем нь процессийн хаягийн орон зайд ачаалагдаж буй дурын EXE эсвэл DLL модульд процессийн экземплярийн тодорхойлогчийг олгодог. EXE файлын экземплярийн тодорхойлогч нь WinMain функцийн эхний параметрээр дамжигддаг.

WinMain функцийн hinstExe параметр нь EXE файлын ”image”-ийн процессийн хаягийн муж дотор байрлах бааз хаягийг тодорхойлдог. Жишээ нь, хэрэв систем биелэгдэх файлыг нээж түүнийг 0x400000 гэсэн хаягаар ачаалласан бол, hinstExe 0x400000 гэсэн утгатай байна.

Application-ны ачаалагдах бааз хаяг нь Linker-ээр тооцоологддог. Win 32-д процесс болгонд өөрийн гэсэн хаягийн орон зай олгогддог учир тэдгээр нь ажиллах явцдаа дангаараа системийг эзэмшиж байна гэж ”боддог”.

Виртуаль санах ойн орон зай

Win 32-т процессийн виртуаль санах ойн орон зайн хэмжээ нь 4гб байдаг. Иймд 32 бит-ийн заагчийн утга нь 0x00000000 – 0xFFFFFFFF хооронд дурын тоо байна.

MS-DOS болон 16-ын Windows-д бүх процессууд нь нэг санах ойн орон зайд байрладаг. Энэ нь ямар нэгэн процесс өөр нэг процессийн эзэмшиж буй санах ойн хэсэгт бичилт эсвэл уншилт хийх боломжтой гэсэн үг юм. Ингэхлээр аливаа нэг процесс бусад процессуудаас ямар нэгэн хамааралтай байж болзошгүй байна гэсэн үг. Жишээ нь, А процесс тохиолдлоор Б процессийн өгөгдлийг дараад бичсэн тохиолдолд Б процесс тодорхойгүй байдалд орж магадгүй ба эсвэл бүр нурна.

Win32-т энэ асуудал нь процесс болгон өөрийн хаалттай санах ойн орон зайтай болсноор шийдэгдсэн билээ. Тэгэхлээр процесст ямар нэгэн Thread биелэгдэж байна гэж үзвэл, тэр нь зөвхөн түүнийг агуулж буй процессийн санах ойн орон зайн мужид хандах боломжтой байдаг. Хэдийгээр өөр процессийн хаягийн

мужид халдах боломжгүй ч Windows 9x ашиглаж байх үед түүний дотоод өгөгдлүүд (үйлдлийн системийн цөм) байрлах санах ойн муж руу хандах боломжтой. Windows NT-д энэ боломж хаагдсан байдаг. (?)

Windows 95 санах ойн орон зайг хэрхэн хуваадаг вэ?

0x00000000 – 0x003FFFFF хэсэг:
Процессийн санах ойн орон зайн доод хэсэгт байрлах 4мб хэмжээтэй энэ хэсэг нь MS-DOS болон 16-тын Windows-ыг дэмжихэд ашиглагддаг.

0x00400000 – 0x7FFFFFFF
2 143 289 344 байт хэмжээтэй энэ хэсэгт процессийн хасалттай санах ойн хаягийн муж нь байрладаг. Нэг ч Win32 процесс энэ хэсэгт байрлах өөр процессийн өгөгдөлд хандах боломжгүй. Win32 процессийн өгөгдлийн ихэнх хувь нь энэ хэсэгт хадгалагддаг.

0x80000000 – 0xBFFFFFFF
1 гб хэмжээтэй энэ хэсэгт систем нь бүх Win32 процесст хүртээмжтэй өгөгдлүүдийг хадгалдаг. Жишээлэхэд, яг энэ хэсэгт системийн DLL болох KERNEL32. DLL, USER32. DLL, GDI32. DLL ачаалагддаг. Процесс болгоны хувьд эдгээр DLL нь ижил хаягаар ачаалагдсан байна. Үүнээс гадна санах ойд проекцлогдож буй бүх файлууд энэ хэсэгт буулгагдана.

0xC0000000 – 0xFFFFFFFF
1гб хэмжээтэй энэ хэсэгт үйлдлийн системийн цөм VxD драйверууд санах ойг болон файлын системийг удирдах доод түвшний код зэрэг мэдээллүүд байрлах ба дээр дурдсанаар энд байгаа код нь бүх Win32 процесст хандалтыг зөвшөөрдөг.

Application-ыг ачаалахад системийн гүйцэтгэх үйлдлийн дараалал нь дараах байдалтай байна:

Биелэгдэх файлыг нээж application-ны код болон өгөгдлийн хэмжээг тодорхойлдог. Дараа нь санах ойн мужид нэг хэсгийг нөөцөнд авч (reserving) түүнд харгалзах физик санах ойг нь EXE файл өөрөө байна гэж тэмдэглэнэ. Ингэснээр application-ыг ачаалах нь маш хурдан болдог.

Санах ойн мужийн тодорхой хэсгийн (бүс) харгалзах физик санах ой нь болох hard дискен дээр байрлах биелэгдэх файлын (EXE эсвэл DLL) ”image”-ийг memory-mapped file гэж нэрлэдэг.

Message ба message-ийн дараалал

Ердийн Microsoft Windows-д зориулан бичигдсэн application нь (программ) үйл явцаар удирдагддаг (event-driven). Тэдгээр нь оролтыг (input) хүлээж авахдаа С-тэй адил шууд функц дууддаггүй, харин Windows тэдгээрт оролтыг илгээхийг хүлээдэг. Windows-д цонх болгон ”window procedure” (цонхны процедур) нэртэй функцтэй байдаг ба систем тодорхой цонх оролтыг эзэмшилдээ авах болгонд түүнийг цонхны процедурыг дууддаг. Цонхны процедур оролтыг (mouse-ийн хөдөлгөөн, товч дарах) боловсруулаад эргээд удирдлагыг Windows-д шилжүүлдэг.

Windows цонхонд оролтыг дамжуулахдаа message (мэдээ) хэлбэртэйгээр дамжуулдаг. Message нь Windows эсвэл application-ээр үүсгэгдэн гардаг. Оролтын үйл явц болгонд Windows message гаргадаг (mouse хөдөлгөхөд, товч дарахад, ямар нэгэн control дарахад). Мөн системд орсон ямар нэгэн өөрчлөлтийн хариуд Windows message үүсгэн гаргадаг. Application message-ийн тусламжтайгаар өөрийн үүсгэсэн цонхнуудтай харьцаж тэдгээрийг удирддаг.

Windows message-ийг цонхны процедурт 4 параметрийн багцаар илгээдэг : цонхны handle, message-ийн ID, хоёр 32 битийн утга бүхий параметрууд (message параметрууд). Цонхны handle нь аль цонхонд очих гэдгийг нь тодорхойлно. Message ID бол message нь өөрөө түүний утгаас хамаарч цонхны процедур тохирох боловсруулалт хийдэг.

Message параметрууд нь message боловсруулалтанд ашиглагдах өгөгдөл эсвэл өгөгдлийн байршлыг тодорхойлдог. Message-ийн төрлөөс хамаарч message параметруудын утга нь өөр өөр байна.

Windows message-ийг цонхны процедурт чиглүүлэх 2 аргыг хэрэглэдэг.

1. FIFO зарчмын ажиллагаатай message-ийн дараалалд (message queue) message-ийг илгээх
2. Шууд цонхны процедур явуулах

Windows нь тухайн агшинд хэд хэдэн цонх харуулах боломжтой байдаг. Яг тухайн цонхонд mouse эсвэл гарны оролтыг өгөхийн тулд Windows нь message дарааллыг хэрэглэдэг.

Windows нэг системийн message дараалалтай ба хэд хэдэн энгийн message дараалалтай байдаг (Thread message queue). GUI Thread болгон нэг message дараалал үүсгэдэг. Хэт их message дараалал үүсгэхгүйн тулд Thread-ууд анх үүсэхдээ message-ийн дараалалгү\й байдаг. Харин Thread Win32 API USER эсвэл GDI функцуудын аль нэгийг дуудахад Windows түүнд зориулж message дараалал үүсгэдэг. Хэрэглэгч mouse хөдөлгөх, mouse-ны товч дарах, гарны товч дарах зэрэг үйлдэл болгонд mouse болон гарны драйверууд оролтын сигналыг message болгон хувиргаад системийн message дараалалд байрлуулдаг. Агшин тутам системийн message дарааллаас нэг message чөлөөлөгддөг ба түүнийг харгалзах цонхыг нь тогтоогоод тэр цонхыг үүсгэсэн Thread-ийн message дараалал руу message-ийг илгээдэг. Thread өөрийн message дарааллаас message-уудыг авч харгалзах цонхны процедурт илгээхийг Windows-д даалгадаг.

Зарим message-ийг систем шууд цонхны процедур луу явуулдаг. Ингэхдээ системийн message дараалал ба Threade message дарааллаар аль алинаар нь дамжуулахгүйгээр. Windows голдуу сонордуулгын утгатай message—дыг ийм маягаар явуулдаг. Жишээ нь, хэрэглэгчийн цонхыг идэвхжүүлэхэд Windows хэд хэдэн төрлийн message илгээдэг: Үүнд: WM_ACTIVATE, WM_SETFOCUS, WM_SETCURSOR. Эдгээр message нь тухайн цонх идэвхжиж байна, гарны оролт түүн рүү шилжиж байна, mouse-ийн курсор энэ цонхны хил хязгаар руу нэвтэрлээ гэсэн сануулгын мэдээнүүд байдаг. Гэхдээ хүсвэл программаас SendMessage функцийн дуудалтаар цонхны процедур луу message-ийг шууд илгээж болно. Application түүний Thread—дын message дарааллуудад ирсэн message-уудыг авч боловсруулах үүрэгтэй байдаг. Үүний тулд message цикл (message loop)-ийг ашигладаг. Нэг Thread-тэй application нэг message циклтэй байх ба үүн дотроо message-ийг дарааллаас авах зөв цонхны процедур луу түүнийг илгээх зэрэг үйлдлийг гүйцэтгэнэ.

Энгийн message цикл дараах функцуудээс бүрдсэн цикл байдаг.

```
MSG  msg;
While (GetMessage (& msg, NULL, O,O))
{
  TranslateMessage (&msg);
   DispatchMessage (&msg);
```

}

GetMessage функц нь message-ийг дарааллаас авч MSG төрлийн структурт хуулдаг. Энэ функц WM_QUIT message-ийг тааралдтал true буцаадаг ба эсрэг тохиолдолд false буцааж циклийг дуусгадаг.

Хэрэв Thread тэмдэгтийн оролтыг гарнаас хүлээж авдаг бол message цикл нь TranslateMessage функцийг заавал агуулсан байх ёстой. Windows нь хэрэглэгч товч дарах болгонд виртуаль товчны message (virtual-key message) гаргадаг. TranslateMessage функц нь виртуаль товчны message-ийг тэмдэгт message (character message)- WM_CHAR- руу хөрвүүлээд message дарааламд нь буцааж байрлуулдаг.

DispatchMessage функц нь MSG структурын цонхны handle-д харгалзсан цонхны процедур луу message-ийг илгээдэг.

Appliction-ны үндсэн Thread application-ыг инициализацлагдаж ядаж нэг цонх үүссэний дараа message циклийг эхлүүлдэг.

Цонхны процедур

Энэ нь цонхонд явуулсан бүх message-уудыг хүлээн авч боловсруулах үүрэгтэй функц юм. Цонхны класс (window class) болгон цонхны процедуртай ба тухайн цонхны класстай үүссэн цонх болгон ижил цонхны процедур ашиглана.

Цонхны процедур нь message ID-ийг шалгаад message параметруудаар дамжигдсан мэдээллийг ашиглаад боловсруулалт хийдэг.

Цонхны процедур message-ийг няцааж огт боловсруулахгүй тохиолдол гаргахгүй байх ёстой ба үүний тулд DefWindowProc функцийг ашигладаг. Энэ функц тухайн ирсэн message-ийг систем рүү буцааж default боловсруулалтанд хамруулдаг. Ихэнх цонхны процедур цөөн тооны message-ийг гардан боловсруулдаг ба үлдсэнийг нь системд даатгадаг.

Тодорхой нэг цонхны класст хамрагдах бүх цонхнууд дундаа нэг цонхны процедуртай байдаг учир цонхны процедур нь хэд хэдэн цонхноос ирсэн message-уудыг боловсруулах боломжтой байдаг. Яг аль цонхных нь вэ гэдгийг цонхны handle-ээр тогтоогоод боловсруулалтаа хийдэг.

Цонхны класс

Цонх болгон нэг цонхны классын гишүүн байдаг. Цонхны класс гэдэг нь тодорхой аттрибутуудын багц ба үүнийг Windows нь шинэ цонх үүсгэх болгондоо загвар болгон ашигладаг.

Классын бүтэц:

1. Class name
2. Window procedure address
3. Instance handle
4. Class cursor
5. Small class icon
6. Class background brush
7. Class menu
8. Class styles
9. Extra class memory
10. Extra window memory

Цонхны класс 3 төрлийн байдаг. Үүнд:

1. System global classes
2. Application global classes
3. Application local classes

Window Procedure Subclassing

Application шинэ цонх үүсгэх үед үйлдлийн систем санах ойд нэг хэсэг нөөцөлж түүн дотроо цонхонд холбогдолтой мэдээллийг хадгалдаг. Энд мөн цонхны процедурын хаяг хадгалагддаг. Хэрэв Windows message-ийг цонхны процедурт илгээх шаардлагатай болоход яг энэ хаягийг ашигладаг юм.

Subclassing гэдэг нь тухайн цонх message-ийг боловсруулахаас өмнө нь түүнийг барьж авч боловсруулах буюу цонхны процедурын хаягийг өөр процедурын хаягаар солих механизм юм. Энэ механизмыг ашигласнаар application цонхны үйл ажиллагааг бүрэн өөрчлөх боломжтой. Ер нь application системийн дурын цонхыг subclass-лаж болно. Үүнд, system global class төрлийн цонхнууд ч хамрагдах болно.

Application цонхыг subclass-лахдаа түүний оригинал цонхны процедурын хаягийг өөр шинэ цонхны процедурын хаягаар сольдог. Үүний дараа тухайн цонхонд чиглэсэн бүх message шинэ цонхны процедураар боловсруулагдах болно. Сольсон шинэ цонхны процедурыг subclass процедур гэж нэрлэдэг.

Subclass процедур нь хүлээж авсан message дээр гурван төрлийн үйлдэл гүйцэтгэдэг.

1. Message-ийг оригинал цонхны процедур луу явуулах
2. Message-ийг боловсруулаад дараа нь оригинал цонхны процедур луу явуулах
3. Message-ийг боловсруулаад дараа нь оригинал цонхны процедур луу илгээхгүй байх

Windows нь 2 төрлийн subclassing дэмждэг: Instance ба Global

Instance subclassing гэдэг нь тухайн нэг цонхны цонхны процедурын хаягийг солихыг хэлнэ. Яг одоо оршиж буй (харагдаж байгаа) цонхыг subclass-лахдаа instance subclassing-ыг ашиглах хэрэгтэй.

Global subclassing гэдэг нь тухайн цонхны харьяалагдах цонхны классын WNDCLASS структур дахь цонхны процедурын хаягийг солихыг хэлнэ. Энэ тохиолдолд тухайн классаар үүссэн бүх цонхнууд шинэ цонхны процедурыг ашиглах болно.

Instance Subclassing

Application нь цонхыг subclass-лахдаа SetWindowLong функцийг хэрэглэдэг. Ингэхдээ SetWindowLong руу GWL_WNDPROC флаг, тухайн цонхны handle ба шинэ цонхны процедурын хаяг зэргийг дамжуулна. Subclass процедур нь application-ны модульд эсвэл DLL-д байрлана. SetWindowLong нь оригинал цонхны процедурын хаягийг буцаадаг. Application нь энэ цонхыг халгалаад дараа нь энэ хаягийг CallWindowProc функцээр дамжуулан message-уудыг оригинал цонхны процедур луу явуулахад ашигладаг. Мөн энэ хаяг нь subclass процедурыг цонхноос авч хаяхад хэрэглэгддэг (оригинал цонхны процедурыг сэргээхэд). Үүний тулд application нь дахин SetWindowLong функцийг дуудах ба оригинал цонхны процедурын хаягийг параметр болгон түүнд дамжуулна.

Нэг чухал нөхцөл биелэгдсэн тохиолдолд л subclassing-ийг хэрэгжүүлэх боломжтой. Энэ нь SetWindowLong функцийг дуудаж буй процесс нь тухайн subclass-лагдаж байгаа цонхыг үүсгэсэн процесстой ижил байх ёстой. Өөрөөр хэлбэл, процессийн хаягийн орон зайн дүрмийг зөрчихгүй байх ёстой.

Hooking

Hook гэдэг нь Windows-ын message боловсруулалтын механизм дахь онцгой хэрэгсэл ба үүнийг хэрэглэснээр message-ийн дамжуулалт, урсгалыг хянаж зарим тохиолдолд хүрэх ёстой цонхны процедурт message очихоос өмнө түүнийг барьж авч боловсруулах боломжтой болдог. Хамгийн гол нь Hook-ийн тусламжтайгаар нэг процесс өөр процессийн санах ойн орон зай руу хандах бололцоотой болдог.

Hook суурилагдсаны дараа Hook-ийн төрлөөс хамаарч message гармагц Windows түүнийг түрүүлээд Hook процедур луу явуулдаг. Дараа нь жинхэнэ цонх руу нь явуулдаг.

Windows нь Hook-ийн төрөл бүрт ”hook chain”-ыг бий болгодог. Hook chain гэдэг нь Hook процедуруудыг заасан жагсаалт юм. Hook төрөлд хамрагдах message гармагц систем түүнийг Hook chain дахь Hook процедур болгон руу ээлж ээлжээр явуулдаг.

Hook тавихын тулд SetWindowsHookEx функцийг хэрэглэх ба энэ нь Hook процедурыг суурилуулдаг. Hook процедур дараах syntax-тай байдаг.

```
LRESULT CALLBACK HookProc
( int Code,
  WPARAM wParam
  LPARAM lParam
  )
```

HookProc-ийн оронд application дурын нэр өгч болно.

Code нь Hook-ийн хийх үйлдлийг нь тодорхойлох Hook код юм. Үүний утга Hook төрлөөс хамаардаг. Сүүлийн хоёр параметрууд нь цонхны процедурын параметруудтай ижил үүрэгтэй.

SetWindowsHookEx нь Hook процедурыг Hook chain-ны эхэнд суурилуулдаг. Тухайн Hook процедур нь ирсэн message-ийг дараагийн процедурт шилжүүлэх эсэхийг өөрөө шийддэг. Дараагийн процедурт шилжүүлэхдээ CallNextHookEx функцийг дууддаг.

Гэхдээ тухайн Hook процедур CallNextHookEx функцийг дуудсан дуудаагүй үл хамааран Windows message-ийг Hook процедур болгон руу илгээдэг.

Hook процедур нь Global эсвэл Thread-ийн хүрээний байна.

Global Hook процедур нь систем дахь бүх Thread-ийн message-ийг хянана.

SetWindowsHookEx (WH_GETMESSAGE, GetMsgProc, hinstDLL, NULL) функцийн үйл ажиллагааг судлая.

WH_GETMESSAGE нь Hook-ийн төрлийг тодорхойлно. GetMsgProc - Hook процедурын хаяг (энэ хаяг нь бидний процессийн хаягийн орон зайд байрлана!). hinstDLL нь Hook процедурын байгаа DLL-ийг тодорхойлдог. Win32-т hinstDLL-ийн утга нь тухайн DLL-ийн процессийн хаягийн орон зайд проекцлогдсон виртуаль санах ой дахь 32 битийн хаяг юм. Сүүлчийн параметр тухайн Hook ямар Thread-д зориулагдсаныг нь тодорхойлно. NULL дамжуулснаар системийн Global Hook тавигдаж байна. Энэ бүхэн яаж явагддагийг харуулья.

1. B процессийн Thread ямар нэгэн цонх руу message илгээх гэж байна гэж үзье.
2. Систем тухайн Thread-ийн хувьд WH_GETMESSAGE Hook тавигдсан эсэхийг шалгана.
3. Дараа нь GetMsgProc функцийг агуулж буй DLL B процессийн хаягийн орон зайд проекцлогдсон эсэхийг тогтооно.
4. Хэрэв тухайн DLL проекцлогдоогүй бол түүнийг B процессийн хаягийн орон зайд буулгана.
5. Систем A болон B процесст холбоотой hinstDLL-ийн утгууд нь таарч байгааг шалгана. Хэрэв hinstDLL-ийн утга нь хоёр процесст хоёуланд нь ижил байвал

GetMsgProc хаяг мөн хоёуланд нь ижил байна гэсэн үг. Энэ тохиолдолд систем А процессийн хаягийн орон зайд GetMsgProc-ийг дуудах боломжтой.

6. Хэрэв hinstDLL-ийн утгууд нь ялгаатай бол систем В процессийн хаягийн орон зай дахь GetMsgProc функцийн хаягийг дараах томьёогоор олдог.
   GetMsgProc B = hinstDLL B + (GetMsgProc A – hinstDLL A)
7. В процессийн хаягийн орон зай дахь GetMsgProc дуудагдана.

Hook механизмын хэрэгжилтийн үед бүтэн DLL-ийн өөр процессийн хаягийн орон зайд проекцлогдоно. Ингэснээр В процессийн Thread-уудэд DLL-ийн бүх функцууд хүртээмжтэй гэсэн үг.

Иймд өөр процессийн Thread үүсгэсэн цонхыг subclass-даа эхлээд WH_GET_MESSAGE hook-ийг тэр thread-д зориулж тавина. Дараа нь GetMsgProc функц дуудагдахад SetWindowLong ашиглаж subclassing-ийг хийх хэрэгтэй. Мэдээж subclass процедур нь энэ DLL-д байх шаардлагатай.

Registry

Registry нь Microsoft Windows-ийн 32 битийн хувилбарууд болох Windows 9x, NT системүүдийн тохиргоо болон үйл ажиллагаанд хамаатай олон янзын мэдээллийг агуулсан өгөгдлийн сан юм. Үүнд: hardware, software, хэрэглэгчид болон PC-ийн ихэнх тохиргоо, мэдээллийг агуулдаг. Хэрэглэгчийн системд оруулсан өөрчлөлтүүд мөн энд бүртгэгдэж хадгалагддаг.

Registry нь иерархи буюу шаталсан бүтэцтэй. Харахад хэцүү зохион байгуулалттай боловч бүтэц hand дискний директорийн зохион байгуулалттай ижилхэн.

Registry үндсэн 6 салаанаас бүрддэг (Эдгээр нь default)

: HKEY_CLASSES_ROOT:

Энэ салаа төрөл (class), OLE зэрэг мэдээллийг агуулдаг

HKEY_CURRENT_USER:

Энэ нь HKEY_USERS салааны тухайн үед холбогдсон хэрэглэгчид хамаатай мэдээлэлтэй яг ижил.

HKEY_LOCAL_MACHINE:

Энэ салаа нь hardware, software төрөл болон өөр янз бүрийн тухайн машинд хамааралтай мэдээллийг агуулдаг.

- HKEY_USERS:
  Энэ салаа нь хэрэглэгч тус бүрээр ялган орчны хувьсагчид, программын бүлгүүд, өнгө, принтер, сүлжээний холболт, application-ны тохиргоо зэрэг төрлийн мэдээллүүдийг агуулдаг.

- HKEY_CURRENT_CONFIG:
  Энэ салааны агуулга нь HKEY_LOCAL_MACHINE тухайн hardware-ийн конфигурацид хамаарах хэсэгхэн мэдээлэлтэй ижилхэн.

- HKEY_DYN_DATA:
  HKEY_LOCAL_MACHINE - Windows-ийн Plug and Play механизмтай холбоотой хэсгийг заадаг.

Цонхны процедурыг зохион байгуулах жишээ

```
LRESULT CALLBACK MainWndProc(
   HWND hwnd,        // handle of window
   UINT uMsg,        // message identifier
   WPARAM wParam,    // first message parameter
   LPARAM lParam)    // second message parameter
{
   switch (uMsg)
   {
      case WM_CREATE:
         // Initialize the window.
         return 0;

      case WM_PAINT:
         // Paint the window's client area.
         return 0;

      case WM_SIZE:
         // Set the size and position of the window.
         return 0;

      case WM_DESTROY:
```

```
            // Clean up window-specific data objects.
            return 0;

        //
        // Process other messages.
        //

        default:
            return DefWindowProc(hwnd, uMsg, wParam, lParam);
    }
    return 0;
}
```

Цонхны процедурыг цонхны классд харгалзуулах жишээ

```
int APIENTRY WinMain(
    HINSTANCE hinstance,  // handle of current instance
    HINSTANCE hinstPrev,  // handle of previous instance
    LPSTR lpCmdLine,      // address of command-line string
    int nCmdShow)         // show-window type
{
    WNDCLASS wc;

    // Register the main window class.
    wc.style = CS_HREDRAW | CS_VREDRAW;
    wc.lpfnWndProc = (WNDPROC) MainWndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hinstance;

    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = GetStockObject(WHITE_BRUSH);
    wc.lpszMenuName =  "MainMenu";
    wc.lpszClassName = "MainWindowClass";

    if (!RegisterClass(&wc))
      return FALSE;
```

```
    //
    // Process other messages.
    //
}
```

Sublassing-ийн жишээ

```
WNDPROC wpOrigEditProc;

LRESULT APIENTRY EditBoxProc(
    HWND hwndDlg,
    UINT uMsg,
    WPARAM wParam,
    LPARAM lParam)
{
    HWND hwndEdit;

    switch(uMsg)
    {
        case WM_INITDIALOG:
            // Retrieve the handle of the edit control.
            hwndEdit = GetDlgItem(hwndDlg, ID_EDIT);
             // Subclass the edit control.
            wpOrigEditProc = (WNDPROC) SetWindowLong(hwndEdit,
                GWL_WNDPROC, (LONG) EditSubclassProc);

            //
            // Continue the initialization procedure.
            //
            return TRUE;

        case WM_DESTROY:
            // Remove the subclass from the edit control.
            SetWindowLong(hwndEdit, GWL_WNDPROC,
                (LONG) wpOrigEditProc);
            //
            // Continue the cleanup procedure.
```

```
            //
            break;
    }

    return FALSE;
        UNREFERENCED_PARAMETER(lParam);
}
// Subclass procedure
LRESULT APIENTRY EditSubclassProc(
    HWND hwnd,
    UINT uMsg,
    WPARAM wParam,
    LPARAM lParam)
{
    if (uMsg == WM_GETDLGCODE)
        return DLGC_WANTALLKEYS;

    return CallWindowProc(wpOrigEditProc, hwnd, uMsg,
        wParam, lParam);
}
```