

Simulirajmo tipkanje

(Blaž Kristan, Programer št. 4)

Marsikateri programer se je že znašel pred problemom, kako predstaviti svoj program na predstavitvah ali pa na sejnih privabiti mimoidoče z delujočim programom, ne da bi mu bilo treba sedeti pred računalnikom. Odgovor bi bil lahko program, ki bi deloval po vnaprej določenem demo poteku ali pa napisati poseben program, ki bi simuliral pritiske po tipkovnici. Pa si oglejmo prednosti in slabosti obeh. Za program, katerega demonstracijo želimo, moramo po prvi metodi preprogramirati večino kode (če ne kar vse), medtem pa po drugi metodi enostavno napišemo datoteko, ki vsebuje ukaze za simulacijo pritiskanja po tipkovnici, sam program pa ostane nespremenjen. Nadalje vzemimo, da smo v naš program dodali nekaj novih opcij. V tem primeru je potrebno demo program zopet preprogramirati (dodati nove opcije), kar pomeni dvojno delo. Pri simulaciji tipkanja pa enostavno dodamo ukaze, ki aktivirajo nove opcije. Sedaj pa pogledajmo še z druge strani. Demo program ima vse potrebno že vgrajeno in zaseda v pomnilniku ravno toliko kot običajen program (navadno malenkost več), simulacijski program pa mora imeti v pomnilniku prostor za izvršno kodo ter podatke in tako ostane drugim programom na voljo manj pomnilnika. Poleg tega pa lahko v demo program dodamo dodatne opcije kot na primer, da se pred izbrano akcijo na zaslon nariše okno, v katerega vpišemo sporočilo o izbrani akciji, itd. To seveda ne pomeni, da kaj takega ne moremo narediti v simulatorju, a to bi nam vzelo dosti več časa in prostora.

Sedaj, ko poznamo nekaj dobrih in slabih lastnosti obeh metod, se pojavi vprašanje katero izbrati. Iz lastnih izkušenj bom svetoval takole. Če je programov, katerih demonstracijo želimo, malo oz. niso kompleksni, potem je uporabna prva metoda, ko pa število programov naraste ali

postanejo kompleksni, svetujem drugo metodo. Sicer pa, naj vsak presodi sam. Za vse, ki se jim zdi druga metoda uporabna ali pa jih le zanima kako deluje tipkovnica na IBM PC kompatibilnih računalnikih, ponujam naslednji programček, ki počne ravno to, simulira tipkanje po tipkovnici.

Delovanje tipkovnice...

že v drugi št. Programerja je bilo opisano delovanje tipkovnice v članku *Ko črke uhajajo*, pa naj za tiste, ki ga nimajo na kratko ponovim in malce razširim opis.

Delovanje tipkovnice je razdeljeno na dva dela: *prekinitveni strežnik* in *vmesni pomnilnik*. Prekinitveni strežnik je del operacijskega sistema (dejansko BIOSa), ki skrbi da se pritisnjena tipka ne izgubi in jo shrani v vmesni pomnilnik, za kasnejšo obdelavo. Drugi del je vmesni pomnilnik, ki pa je razdeljen na več delov: *statusa tipkovnice*, *kazalcev na prvi in zadnji znak*, *medpomnilnika (buffer)*, *kazalcev na začetek in konec medpomnilnika ter statusa LED diod tipkovnice*. Vmesni pomnilnik se nahaja na segmentu 40H, vendar pa zaseda vsega 42 bytov (na splošno je segment 40H do 50H, 256 bytov, namenjen za BIOS spremenljivke). Status tipkovnice se nahaja na naslovu 40:17H in je bitno orientiran. Posamezni biti imajo naslednji pomen: 0 - desna tipka shift, 1 - leva tipka shift, 2 - tipka ctrl, 3 - tipka alt, 4 - tipka scroll lock, 5 - tipka num lock, 6 - tipka caps lock, 7 - ni uporabljen in je 0. Če je bit prižgan pomeni, da je tipka pritisnjena, drugače pa spuščena. Kazalec na prvi znak v medpomnilniku se nahaja na naslovu 40:1AH, kazalec na zadnji znak pa na 40:1CH (dejansko kaže na prazen prostor za zadnjim znakom). Medpomnilnik se nahaja na 40:1EH oz. kamor kažeta kazalca na začetek in konec medpomnilnika. Običajno je dolg 32 bytov (16 besed), kar zadostuje za 15 pritisnjenih tipk. Dva byta za eno tipko sta potrebna zato, ker v zgornji byte vpišemo *scan kodo*, v spodnji byte pa *ASCII kodo* pritisnjene tipke. Če tipka ne predstavlja ASCII kode (funkcijska tipka,...) jo enostavno pustimo na 0. Kazalec na začetek medpomnilnika je

na naslovu 40:80H, na konec pa na naslovu 40:82H. Status LED pa na 40:97H in je zopet orientiran bitno, 0 - scroll lock, 1 - num lock, 2 - caps lock, ostali biti služijo za komunikacijo s tipkovnico in jih pustimo pri miru.

Zaradi lažjega branja sem poimenoval kazalce takole: *KbStat* pomeni status tipkovnice, *KbHead* kaže na prvi znak v medpomnilniku, *KbTail* na zadnjega, *KbBot* je kazalec na začetek medpomnilnika, *KbTop* pa na konec.

Delovanje medpomnilnika je naslednje. Ko pritisnemo tipko, se le ta vpiše kamor kaže *KbTail*, sam kazalec pa se poveča za 2, če preseže vrednost *KbTop*, se zmanjša na *KbBot*. Podobno velja za *KbHead*, le da se ta povečuje, ko preberemo tipko (z BIOS klicom). Kadar sta *KbHead* in *KbTail* enaka, pomeni da je medpomnilnik prazen (iz tega sledi, da imamo lahko zapomnjenih le 15 tipk in ne 16). Iz napisanega sledi, da je medpomnilnik organiziran v obliki kroga (prstana), ko pridemo do konca preskočimo na začetek (wrap around).

... in njena simulacija ...

Sedaj, ko vemo kako deluje tipkovnica, pa si oglejmo, kako jo bomo simulirali. Že iz samega delovanja medpomnilnika lahko vidimo, da bomo morali povečati *KbTail*, ko bomo vpisali tipko. Za operacije s kazalcem *KbHead*, pa poskrbi sam BIOS, tako da nam ne bo treba skrbeti zanj. Postopek pa je naslednji. Najprej vpišemo na mesto kamor kaže *KbTail* željeno tipko, nato povečamo *KbTail* (za 2). Če je pri tem postal *KbTail* večji od *KbTop*, ga moramo zmanjšati na vrednost *KbBot*, nadalje primerjamo *KbTail* in *KbHead*. V primeru, da sta kazalca enaka imamo medpomnilnik poln in moramo postaviti *KbTail* nazaj na prejšnjo vrednost, poleg tega pa si moramo zapomniti katero tipko smo vpisali v medpomnilnik. Če pa kazalca nista enaka, imamo v medpomnilniku že vpisano tipko in lahko celoten postopek ponovimo.

Do sedaj je (teoretično) vse v redu, v praksi pa se pokaže, da nekaterim programom le ni všeč, če vpisujemo tipke v medpomnilnik,

dokler le ta ni poln, zato se poslužimo trika, da v medpomnilnik vpisujemo le po en znak naenkrat in počakamo, da ga drugi programi prebavijo. To pomeni, da čakamo toliko časa, da postaneta *KbHead* in *KbTail* enaka in šele nato vpišemo tipko v medpomnilnik. Poleg tega pa imamo zaradi tega še eno prednost, v simulatorju se koda poenostavi, saj primerjamo le enakost obeh kazalcev in nam ni treba ugotavljati ali je medpomnilnik poln ali ne.

Upam, da je sedaj vsem jasno kako stvar deluje, če pa še komu ni, ni nič hudega saj obstaja še ena metoda vpisovanja tipk v medpomnilnik. V tej drugi metodi se poslužimo klica BIOSa, natančneje dela, ki skrbi za tipkovnico (INT 16H), funkcijo št. 5 (AH=05H). Pri tem klicu navedemo v registru CH scan kodo v registru CL pa ASCII kodo željene tipke. Na ta način nam tudi ni treba skrbeti za vse kazalce, vendar je pomanjkljivost te metode v tem, da obstaja le na AT in PS/2 kompatibilnih računalnikih (ker pa je XTjev danes že zelo malo, mislim, da to niti ni taka pomanjkljivost). Kot rezultat tega klica, imamo v zastavici prenosa (carry flag) vrednost 0, če je vpis uspel, oz. 1, če je medpomnilnik že poln.

... ter izvedba v jeziku C

Iz napisanega je razvidno, da bomo simulator najlažje realizirali kot pritajen (TSR, terminate & stay resident) program.

Tipke je potrebno vpisovati v medpomnilnik v časovnih razmikih, zato uporabimo PCjevo *timer C* prekinitev (interrupt), katere vektor ima oznako 1CH (INT 1CH). Rutina (oz. del programa), na katero kaže ta vektor, se izvede 18.2 krat na sekundo (zato bi morali simulator dodatno upočasniti pri vpisovanju, če bi vpisovali več tipk naenkrat). Poleg tega vektorja (in prekinitve) pa uporabimo še vektor 60H, ki je *user interrupt* (uporabniška prekinitev) in ga uporabimo za ugotavljanje prisotnosti programa v pomnilniku računalnika ter za njegovo brisanje iz pomnilnika. Da bomo simulator med delom lahko prekinili, ali pa ga ponovno zagnali uporabimo vektor 15H

(INT 15H). To prekinitvev (funkcijo 4FH) kliče hardware-ska prekinitvev 09H, ki se sproži vsakič, ko pritisnemo katerokoli tipko oz. jo spustimo (se izvede dvakrat!). V primeru vektorja za *timer C* in vektorja za tipkovnico, moramo *obvezno* klicati obe rutini, na kateri kažeta originalna vektorja. Delovanje programa pa je naslednje. Najprej preverimo ali ni mogoče v pomnilniku že naložena kopija simulatorja. Če ta kopija obstaja in smo v ukazni vrstici podali parameter *-OFF*, zberemo to kopijo iz pomnilnika, drugače pa javimo napako in končamo. Če kopija simulatorja ne obstaja in smo podali kot parameter ime datoteke, prečitamo dano datoteko v pomnilnik, izračunamo količino zasedenega pomnilnika, nastavimo prekinitvene vektorje in končamo tako, da ne sprostimo zasedenega pomnilnika. Izračunavanje količine zasedenega pomnilnika ni enostavno, saj ne vemo vnaprej koliko bo sam program dolg in kolikšna bo dolžina datoteke, zato se poslužimo naslednjega trika. Ko rezerviramo prostor za datoteko s klicom funkcije *malloc()*, se ta del rezevira *nad* simulatorjem (na višjih lokacijah), ker pa za simulator natančno vemo, kje se nahaja (spremenljivka *_psp* oz. lahko preko DOS funkcije GET PSP, INT 21H, AX=62H, rezultat v BX) in vemo tudi naslov rezerviranega bloka z datoteko, lahko izračunamo celotno količino zasedenega pomnilnika po naslednji formuli:

$$\text{velikost} = (\text{seg_pom_z_dat} - \text{_psp}) * 16 + \text{ofs_pom_z_dat} + \text{dolž} + 16$$

Kjer pomeni *seg_pom_z_dat* segment rezerviranega pomnilnika kjer se nahaja prebrana datoteka, *ofs_pom_z_dat* pa pomeni offset istega dela pomnilnika. *dolž* pomeni dolžino prebrane datoteke. Na koncu prištejemo še 16 zaradi poravnavanja na paragraf (16 bytov).

Sedaj se moramo dogovoriti le še o datoteki simulacijskih tipk. Do sedaj njene strukture nisem omenil niti z besedico, zato je prav, da ji posvetimo malo pozornosti. Uporabili bomo standardni ASCII format, ker je

najlažje dostopen in nam bo delal najmanj težav. Tako datoteko lako urejamo s kakršnimkoli urejevalnikom (Turbo C, Pascal, DOS, Wordstar...). Ker pa ima ASCII koda le 128 različnih znakov (nekateri še sedaj mislijo, da jih ima 256), bomo uporabili naslednji pristop. Uvedli bomo posebne znake, ki bodo pomenili *ukaze* in bomo z njihovo pomočjo dosegali vse mogoče kombinacije tipk. Struktura datoteke bo naslednja:

- vsi ASCII znaki med ' ' (ASCII 32) in '~' (ASCII 126) se prenašajo taki kot so, z izjemo znakov '#', '\$' in '@', ki bodo kontrolni znaki (ukazi)
- vse ostale znake v datoteki ignoriramo
- znak '#' služi za vnos ASCII kode poljubnega znaka, vnesemo tri mestno število, ki predstavlja *extended* ASCII (0-256) kodo. Tako vnesemo '#013' za pritisk na tipko <ENTER>, oz. '###' za vnos znaka '#' v medpomnilnik.
- znak '\$' nam bo služil za vnos scan kode tipke, in bomo z njim dosegali funkcijske, kurzorske, itd tipke. Ravno tako sledi tri mestno število scan kode oz. še en znak '\$', Če želimo vnos le tega.
- znak '@' (afna) pa nam bo služil za vnos ukazov, ki nimajo neposredne zveze z vnosom v medpomnilnik. Kombinacija '@A' bo povzročila trenutno prenehanje izvajanja simulatorja (abort). Kombinacija '@Wnnn', pa služi za vnos pavze (delay) med naslednjim vpisom v medpomnilnik. *nnn* je trimestno število, ki podaja število osemnajstink sekund za pavzo (tako bi vnesli '@W018' za sekundo pavze).

Ukazi '#', '\$' in '@' naj se vsi zaključijo s presledkom ali kakim drugim neizpisnim znakom (CR, LF, TAB, ...) zaradi lažjega pretvarjanja in možnih razširitev. Ko smo že pri razširitvah, si lahko sami dodate posebne ukaze in izboljšate program ter sporočite vaše dopolnitve.

ZAKLJUČEK

Tako, to bi bilo vse o delovanju tipkovnice in njeni simulaciji. Program je bil testiran predvsem s Clipperjem in DOSom in ni delal kakšnih posebnih težav, za ostale programe pa boste morali malo sami eksperimentirati, čeprav mislim, da ne bi smelo biti težav. Sam program z izvorno kodo (zelo bogato komentirano) dobite na vseh podpornih BBS, poleg tega pa sem tudi vsem, ki jih program zanima, ali pa bodo imeli kakršnekoli težave na voljo v konferencah C, Assembler in EMail na vseh BBS, ki so vključeni v AdriaNet. Za tiste, ki pa nimate modema, pokličite na uredništvo Programerja.

Literatura

- [1] Schaepers, Arne: Turbo Pascal 5.5, Tips & Tricks
- [2] Duncan, Ray: MS DOS FUNCTIONS, programmers reference
- [3] Duncan, Ray: IBM ROM BIOS, programmers reference
- [4] Hogan, Thom: PC Sourcebook
- [5] Quarterdeck: Manifest (program)