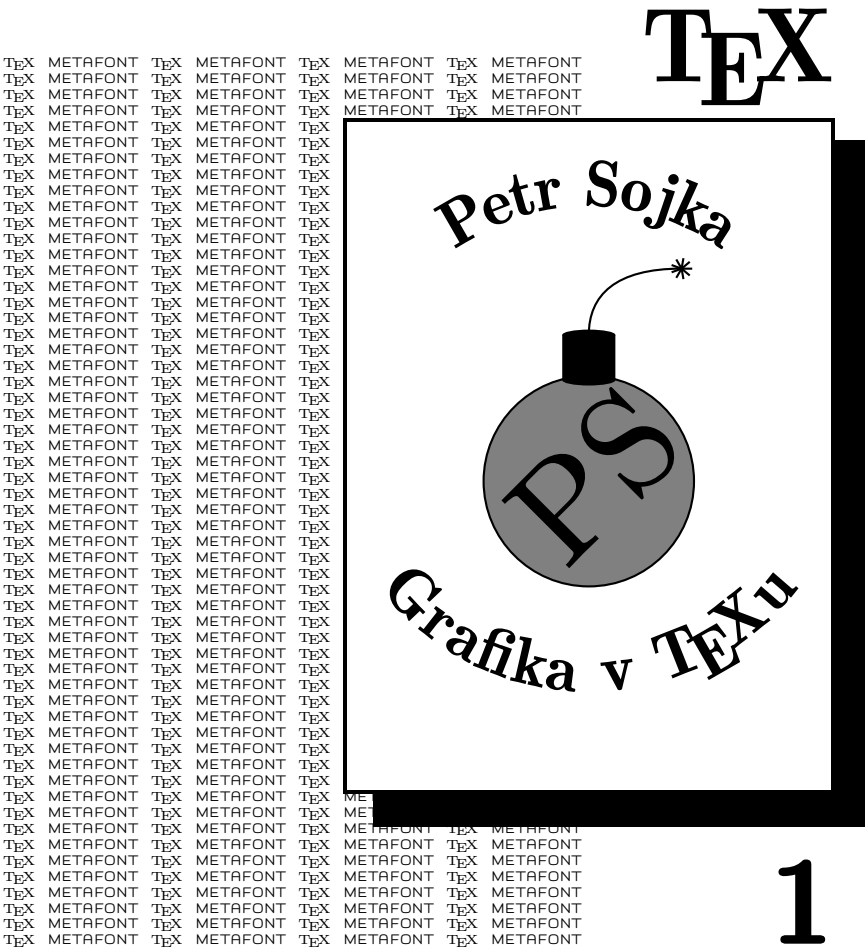


ZPRAVODAJ 94 1

ČESKOSLOVENSKÉHO SDRUŽENÍ UŽIVATELŮ TEXU

OBSAH

Frank Mittelbach: E-TEX: Sprievodca budúcimi rozšíreniami TEXu 1
Petr Sojka: Grafika v TEXu (2) 20
Petr Olšák: Kouzla s programem MNU 27

E-TEX: Sprievodca budúcimi rozšíreniami TEXu

Frank Mittelbach

Abstrakt

Ohlásením verzie TEX 3.0 opätoval Don Knuth (stále rastúcu) potrebu TEXovskej komunity ešte lepšieho systému. Ale súčasne dal jasne najavo, že nechce byť zaťahovaný do ďalšieho takého rozšírenia, ktoré by menilo *The TEXbook*.

TEX vznikol spočiatku ako systém určený na sádzanie vlastných publikácií autora. Medzičasom slúži stotisícom užívateľov. Nastal čas pohliadnuť sa po desaťročných skúsenostiach naspäť, či je, alebo nie je TEX 3.0 adekvátnou odpoveďou na sadzačské požiadavky deväťdesiatych rokov.

Výstup vytvorený TEXom má vyšší štandard než výstup automaticky generovaný väčšinou iných sádzacích systémov. Z toho dôvodu sa v tomto článku sústredíme na kvalitatívne štandardy predložené typografmi pre ručne sádzané dokumenty a spýtame sa, do akej miery sú tieto štandardy dosiahnuté TEXom. Budeme analyzovať možnosti TEXovských algoritmov a naznačíme nedostatky ako aj nové koncepcie.

1. Úvod

Minulý rok sme v Stanforde oslávili desiate narodeniny projektu TEX. TEX slúžil dodnes tisícom užívateľov a očakávame, že tak bude činiť aj v budúcnosti. Dlhovekosť TEXu spočíva

- v kvalite výstupu
- v univerzálnej dostupnosti
- v jeho stabilite.

V posledných niekoľkých rokoch stále viac a viac užívateľov prenieslo TEX z univerzít do priemyslu, kde bol postavený pred nové aplikácie [33]. Ale čas sa nezastavil a to, čo bolo vrcholom včera, môže byť zajtra už zastaralé. TEX je stále vzorom pre tie ciele, pre ktoré bol vytvorený, ale s rastúcimi skúsenosťami z niekoľkoročného používania môžeme lepšie porozumieť, kde neobstojí vo vysokej kvalite sadzby.

Ako výsledok tlaku užívateľov [27] oznámil Don Knuth v Stanforde novú verziu TEXu s priznaním skutočnosti, že nepredvídal potrebu pre 8-bitový vstup [19]. Súčasne dal najavo, že sa rozhodol odísť z tohto projektu a vrátiť sa k hodne opozdenému projektu „Umenia počítačového programovania“.

Takto je TEX v súčasnosti zakonzervovaný a každý jeho ďalší vývoj bude ústiť do iného systému, ktorý už nebude pod Knuthovým dohľadom. Preto hlavným cieľom tejto práce je podať prehľad požiadaviek pre vysokú kvalitu sádzania (bez ohľadu na to, či ich TEX spĺňa alebo nie), čím dúfame, že ovplyvníme budúci vývoj tak, aby neskončil niekoľkými nekompatibilnými systémami založenými na TEXu, ale snáď v jednom systéme s rovnakými charakteristikami (t.j. kvalita, prenosnosť a dostupnosť) ako súčasný program.

TEX bol navrhnutý ako formátovač nižšej úrovne, ako stabilné jadro sádzacieho systému, ktorého rozšírenia na oboch koncoch budú schopné akceptovať vývoj v technológii tlačenia (výstupný koniec) a spojenie s užívateľom (vstupný koniec) [14]. Sťažnosti na nepriateľskosť TEXu voči užívateľom sú neodôvodnené, lebo všetky požiadavky z tejto strany môžu byť ošetrené na vstupnom konci buď v reči TEXu (ako LATEX), a tým úplné prenositeľné, alebo pomocou externého jazyka, ako ArborText's Publisher alebo VAX Document, atď. Tieto systémy používajú TEX alebo sú založené na TEXu ako na konečnom formátovači a poskytujú užívateľsky priateľský interface [32].

Keď rozoberáme chýbajúce prvky, tak musíme obozretne rozlišovať medzi tým, čo môže a má byť ošetrené systémom na vstupnom konci, a tým, čo je naozaj úloha pre formátovač a nemôže byť ošetrené v TEXu 3.0. V nasledujúcich odsekoch budeme analyzovať požiadavky pre vysoko kvalitnú sadzbu a prediskutujeme, či môžu byť ošetrené TEXom alebo vhodným vstupným koncom, prípadne obidvoma. Ak takéto ošetrenie nie je možné, pokúsime sa nájsť cesty na dosiahnutie požadovaného výsledku. Konečne, v odseku 12 si všimneme samotný koncept jazyka TEX tým, že načrtneme základy toho, ako môže jazyk pre nový systém jasnejšie popisovať podkladové pojmy.

2. Zalamovanie riadkov

Algoritmus TEXu na zalamovanie riadkov je očividne ústrednou časťou celého jeho systému. Namiesto toho, aby zalamoval odstavec riadok za riadkom, algoritmus považuje odstavce za jednotky a hľadá ‚optimálne riešenie‘ na základe hodnôt niekoľkých parametrov. V dôsledku toho

porovnanie výsledkov vytvorených TEXom a iným systémom dopadne obyčajne v prospech TEXu.

Taký prístup má však svoje úskalia, najmä v situáciách vyžadujúcich viac než blok štýlového textu pevnej šírky. Konečné zalomenia riadkov sú určené v čase, keď informácia o obsahu aktuálneho riadku je stratená (aspoň z pohľadu TEXu, t.j. jeho vlastného makrojazyka), takže TEX neposkytuje žiadne dodatočné spracovanie konečných riadkov z hľadiska ich obsahu.

Ďalej neexistuje spôsob ovplyvniť tvar odstavca vzhľadom na aktuálnu pozíciu na stránke, pretože táto informácia nie je *a priori* známa. V odseku 4 budeme diskutovať na túto tému.

Používanie len štyroch kategórií (tight, decent, loose, very loose) na rozlíšenie nastavenia glue pre susedné riadky sa zdá trochu neadekvátne. Počet týchto kategórií by mal byť zvýšený. Navyše, globálnejší prístup (dokonca za hranice odstavcov za istých okolností), ktorý by vzal do úvahy celkovú zmenu nastavenia glue, by mohol viesť k lepším výsledkom.

2.1 Parametre zalamovania riadkov

Aj keď algoritmus poskytuje množstvo parametrov na ovplyvnenie layoutu, niektoré dôležité parametre pre kvalitnú sadzbu chýbajú. Neexistuje spôsob, ako niečo urobiť s vertikálnymi pásami (riekami) vytvorenými medzislovnými medzerami zapadajúcimi do tej istej vertikálnej pozície. Podobný problém zahŕňa rovnaké slová nad sebou, najmä na začiatku alebo na konci riadku. Obidva problémy sú rozptylujúce pre oči čitateľa a zničia každú námahu na vytvorenie krásne zalomeného odseku. Dobrý príklad je udaný v druhom odstavci časti 5, kde slovo „strane“ sa opakuje na dvoch riadkoch napravo pod sebou.

Ďalší aspekt ušľachtilej tlače je istota, že posledný riadok odstavca nebude príliš krátky. To je mimoriadne dôležité pri layoutoch, ktoré používajú odsadenie na začiatku odstavca, kde by vznikla nežiadúca medzera v prípade, že posledný riadok odstavca bol kratší než odsadenie nasledujúceho paragrafu. Pre väčšinu používateľov TEXu je neznáme, že tomuto môžeme zabrániť špeciálnym nastavením parametrov TEXu pre zalamovanie riadku tak, ako je to ukázané v príklade 1 časti 14. Kým zvyšné časti tohto článku používajú toto nastavenie, v tomto odstavci je takýto nežiadúci efekt viditeľný.

Delenie slov v po sebe idúcich riadkoch je ošetrené do rozsahu dvoch

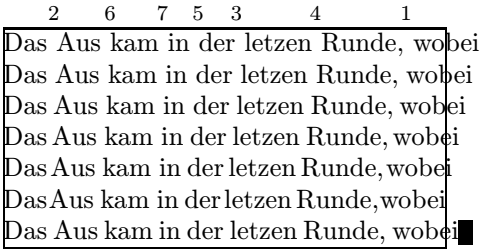
riadkov (`\doublehyphendemerits`), ale neexistuje možnosť za istých okolností zabrániť odstavcom takým ako tento a nasledujúci. Ako ľahko zbadáte, počet rozdelení v týchto odstavcoch je umelo zvýšený nastavením istých parametrov pre zalamovanie riadkov na nezvyklé hodnoty. Ale v neanglických jazykoch (s väčšou priemernou dĺžkou slova) predstavujú takéto situácie reálne životné problémy.

Iný problém predstavuje nezhoda medzi prvým a neskoršími riadkami odstavca, ktorá je dôsledkom implementácie odsadenia začiatku odstavca. Toto je podstatné v layoutoch s nulovým odsadením, pretože medzera na začiatku prvého riadku (napríklad z `\mathsurround`) nezmizne do okraja z dôvodu implicitného `\hbox`, ktorý vyjadruje odsadenie na začiatku odstavca (aj keď nie je viditeľné), zatiaľ čo rovnaká medzera bude zlikvidovaná na začiatku neskorších riadkov. To vedie k zvláštnym počiatočným medzerám.

3. Medzery

Keď je vytváraný blokový text, je nutné meniť medzislovnú alebo medziznakovú medzeru, alebo obidve. Kým medziznaková premenná medzera je prenechaná na expertov (až na malé výnimky), je úlohou zalamovacieho algoritmu naťahovať alebo zmenšovať medzislovnú medzeru od optimálnej hodnoty, ktorá je daná dizajnerom fontu, až pokiaľ nie je určená konečná poloha slova. TEX opäť má dobre navrhnutý algoritmus, ktorý berie do úvahy takúto rozťahovateľnosť. Navyše, každý znak má priradený tzv. `\spacefactor`, ktorý má vplyv na medzeru nasledujúcu za ním, čím je daná možnosť zväčšiť alebo zmenšiť medzeru medzi slovami za istými znakmi. Ako príklad porovnajte medzery za interpunkčnými znamienkami v tomto odstavci s ostatnými.

Ovšem neexistuje žiadne opatrenie na možnosť ovplyvnenia medzislovnej medzery vo vzťahu k aktuálnym znakom na obidvoch koncoch slova. Ak by bolo potrebné zúžiť daný riadok, nemali by sa zúžiť všetky medzery o rovnakú hodnotu. Namiesto toho by bolo lepšie zúžiť viac za čiarkou než napríklad medzi slovami ‚kam in‘ (viď obrázok 1) z dôvodu rôzneho tvaru znakov. Neexistuje spôsob ako dosiahnuť také jemné vyladenie v TEXu, s výnimkou manuálneho dodania `\hskip` v riadkoch, ktoré sú netolerovateľné. Jeden príklad takého prístupu je vidieť na obrázku 1. Takýto mechanizmus je samozrejme závislý od fontu a jeho implementácia by preto zmenila ako TEX, tak aj METAFONT, lebo najlepšie miesto, kde uložiť takúto informáciu, je `TFM` súbor. Ale aj tabuľky podobné k `\sfcode` (fixované formátom alebo makrom) by boli veľkým



Obrázok 1: Medzislovné medzery

Medzislovné medzery sú tak očíslované, že väčšie čísla označujú medzery, ktoré by sa menej zmenšili použitím pravidla podľa Siemoneita [28]. Posledný riadok ukazuje výsledný overfull box vytvorený v tejto situácii štandardným TEXom.

zlepšením, lebo väčšina používaných fontov má tendenciu mať podobné tvary znakov.

V TEXovskej koncepcii pre nastavovanie glue je urobený dôležitý rozdiel medzi glue pre naťahovanie a sťahovanie: kým to posledné je povolené len po isté pevné minimum (t.j. *natural width* mínus *shrink component*), každá hodnota naťahovacieho glue je automaticky povolená po ľubovoľnú hodnotu.[1]) Dôvodom pre toto správanie je, že toto umožňuje zalamovaciemu algoritmu dosiahnuť ‚núdzové výsledky', ak nie je nájdené iné vhodné zalomenie. Toto je však nežiadúce vo väčšine situácií, takže buď by malo byť ohraničené vo všetkých prípadoch naťahovanie podobne ako sťahovanie (výsledkom čoho by boli zmeny v algoritme pre zalamovanie riadkov a strán), alebo by mala byť dodaná ďalšia trieda glue, pre ktorú by veľkosť naťahovania mohla byť určená individuálne.

Don Knuth [18, str. 394–395] podáva príklad ako dosiahnuť visiacu interpunkciu (spolu so špeciálnymi fontami, ako poznamenáva). Pretože toto je tiež znakom dobrej kvality sadzby, je otázne, či takáto schéma (ktorá urobí mechanizmus ligatúry čiastočne nepoužiteľný, spolu s inými efektmi) je vhodná, alebo či toto by nemalo byť priamym prvkom budúceho programu.[2])

[1]) Za normálnych okolností je však tomuto zabránené badness funkciou.

[2]) Počínajúc nasledujúcim odstavcom, používa tento článok visiacu interpunkciu. Zmena kvality je zreteľná, aj keď zlepšenie je možné jemnou adjustáciou všetkých znakov (napr. posunutím ‚r' máličko von, a pod.) dosiahnuť dokonalé zarovnanie.

4. Zalamovanie strán

Veľký problém v TEXu predstavuje jeho algoritmus na zalamovanie strán. Zalamovanie strán je obhospodarované asynchrónne presúvaním istých položiek v isté momenty zo zoznamu aktuálnych príspevkov na „aktuálnu stranu“, pokiaľ nie je tento zoznam naplnený viacerými jednotlivosťami, než sa to hodí na danú stranu v konečnej forme. Konečné zalomenie strany je zvolené po zvážení miery špatnosti (ako plná je strana, ak ju zalomíme na tomto mieste) a hodnoty pokút (ako drahé je to zalomiť práve tu). Takéto penalizácie sú uložené po niektorých riadkoch buď algoritmom na zalomenie riadku, alebo počas rozvoja makra.

Ovšem dobrý stranový layout vyžaduje vzatie do úvahy dvojíc oproti sebe stojacich strán tak, ako sú tieto videné čitateľom. Toto samo o sebe nie je reálnym obmedzením, lebo dvojstrana môže byť chápaná ako obrovský prípad dvojstĺpcového formátu, samozrejme za predpokladu, že obidve strany sa súčasne zmestia do pamäti. Nanešťastie žiadny z vnútorných mechanizmov TEXu nevie vhodne obhospodáriť viacstĺpcový layout, takže takýto prístup musí obísť všetky vnútorné prvky pre zalamovanie strán ako `\insert` a pod. Dobrými príkladmi, ktoré tiež ukazujú obmedzenia TEXu v tomto smere, sú výstup LATEXu [22] a implementácia viacstĺpcového layoutu [4, 24], hraničiace s nemožným.

Ale čo je viac dôležité, algoritmus na zalamovanie riadkov v spojitosti s algoritmom na zalamovanie strán predkladá neriešiteľné problémy. Keď je TEXom zvolené zalomenie strany, všetky odstavce, ktoré boli raz kandidátom pre aktuálnu stranu, sú už oddelené algoritmom na zalomenie riadku a toto oddelenie nemôže byť zrušené pre text prenesený na nasledujúcu stranu, lebo isté nutné informácie (napríklad medzera v mieste zalomenia riadku) sú stratené. Toto robí nemožným zmenu layoutu strany na istých miestach, napr. na vrchu novej strany, pre vynechanie miesta pre malý obrázok obkolesený textom. Len za veľmi obmedzených okolností môže byť nájdené riešenie vo vnútri TEXu [9], ale dokumenty priemernej zložitosti nemôžu byť spracované takýmto spôsobom. Všeobecné riešenie tohto problému môže byť zahrnuté do súčasného TEXu kompatibilným spôsobom smerom hore. Istý prototyp bol navrhnutý na univerzite v Mohuči krátko po konferencii v Stanforde [25].

Je otvorenou otázkou, či vôbec ostaneme pri TEXovskom zalamovacom algoritme pre strany, lebo bol nedotiahnutý z dôvodu pamäťových a časových obmedzení počítačov dostupných v čase jeho vývoja. Vo svojej PhD práci [26] vyšetroval M. Plass niekoľko globálnych optimalizačných stra-

tégií používajúcich dvojprechodový systém. Jeho výsledky otvorili široké pole pre ďalší výskum. Niektoré z jeho myšlienok boli použité v Type & Set systéme [3].

Hlavným príspevkom TEXu 82 k počítačovej sadzbe bol krok od riadok za riadkom odstavec zalamujúcim algoritmom ku globálne optimalizujúcemu algoritmu.[3]) Ďalším cieľom pre budúce systémy by malo byť vyriešenie podobného, ale komplexnejšieho problému globálneho zalamovania strán.

5. Layout strán

Pre úpravu strán poskytuje TEX koncepciu výstupných rutín spolu s vkladaniami (inzerciami) a značkami (márkami). Koncepty vkladania a značkovania sú šité pre potreby relatívne jednoduchého modelu layoutu strán, ktorý obsahuje len jeden stĺpec, poznámky pod čiarou a najviac tu a tam jednoduchý obrázok.[4])

Značkovací mechanizmus poskytuje akúsi informáciu o istých objektoch a ich relatívnom poradí na aktuálnej strane, alebo špecifickejšie, informáciu o prvom a poslednom z týchto objektov na aktuálnej strane a o poslednom z týchto objektov na ľubovoľnej predchádzajúcej strane. Taká informácia je nevyhnutná na konštrukciu určitých typov záhlaví, napr. takých s menom aktuálnej kapitoly alebo s informáciou o prvom a poslednom slove vysvetlenom na predchádzajúcej strane a pod.

Toto je globálny mechanizmus, avšak taký, že len jedna trieda objektov môže využívať výhody celého mechanizmu. Ak je implementovaná viac ako jedna trieda, niektoré prvky tohto mechanizmu sú stratené pre celú triedu.[5]) Ako dôsledok tohto nedostatku by bolo vhodné rozšíriť značkovací mechanizmus do systému nezávislých značiek, ktoré môžu byť alokované oddelene makro balíkom.

Zdá sa, že vkladací mechanizmus bol odvodený z ‚aplikácie poznámok

---

[3]) Musíme poznamenať, že podobný algoritmus bol nezávisle vyvinutý J. Achugbueom [2]. Porovnanie by mohlo viesť k ďalšiemu zvýkoneniu.

[4]) Termín ‚jednostĺpcový výstup' znamená, že celý text je skladaný z riadkov rovnakej šírky. Problémy s premenlivou šírkou riadku diskutujeme v časti 4. Toto samozrejme pokrýva široký rozsah viacstĺpcových layoutov, napr. spracovanie poznámok pod čiarou v originále tohto článku (viacstĺpcový layout). Ale podobný rozsah zaujímavých layoutov nie je definovateľný v TEXovskom modeli box – glue – penalty.

[5]) Implementácia LATEXu poskytuje rozšírený značkovací mechanizmus s dvoma druhmi nezávislých značiek s výsledkom, že jeden sa správa ako `\firstmark` a ten druhý ako `\botmark`. Informácia obsiahnutá v primitívnom pojme `\topmark` je stratená.

pod čiarou‘ a neskoršie rozšírený tak, aby umožnil istý jednoduchý typ pohyblivých vkladaní.[6]) Ovšem umiestnenie pohyblivých objektov vyžaduje viac než ich púhe uloženie v obrovskom boxe, z ktorého sa vyjmú v istom momente, keď je vyvolaná výstupná rutina. Pohyblivé objekty sú doprevádzané titulkami a podobnými záležitosťami, ktoré vyžadujú rozličný prístup v závislosti od ich konečného umiestnenia na strane. Pohyblivé objekty sa odlišujú, aj keď prislušia do tej istej triedy. Na druhej strane, objekty uložené v jednej triede môžu ovplyvniť, alebo dokonca zamedziť umiestnenie pohyblivých objektov z iných tried.

Niektoré triedy vkladaní, ako marginálne poznámky, nemôžu byť vôbec obhospodárené pomocou primitívnych pojmov. Napríklad LaTeX pre poskytnutie takých možností definuje vlastnú správu pamäti pre pohyblivé objekty. Takýto mechanizmus je prirodzene pomalý a náročný na pamäť. Navyše je ďalej redukovaná kvalita zalomení strán, lebo je ťažké udržovať voľne všetku informáciu poskytovanú koncepciou vkladania.

Iným problémom je rozhodnutie, že zalamovací mechanizmus strán je, aspoň v jeho rozhodujúcej fáze, dostupný len vo vonkajšom vertikálnom režime, a tak napríklad priestor pre vsúvanie vkladaní sa neberie na zreteľ, keď sa delí `\vbox` pomocou `\vsplit`.

Pre navrhovateľa je TeXovský model medziriadkového určenia glue úplne neznámy, lebo nedovoľuje špecifikáciu medzery od základnej čiary k základnej čiare v stranovej špecifikácii bez použitia zdĺhavých a komplikovaných interných výpočtov (viď obrázok 2 na ďalšej strane). To znamená, že je skoro nemožné implementovať mrežovo orientované špecifikácie, t.j. kde (skoro) všetky účaria padnú na dopredu určené pozície. Tento článok používa mrežovo orientovanú špecifikáciu (ktorá je pripravená ručne), aby sme ukázali tento aspekt vysoko kvalitnej sadzby. Detaily sú udané v príklade 3.

Návrh vhodných primitívnych pojmov pre tento komplex musí ísť ruku v ruke s novým algoritmom pre zalamovanie strán, pravdepodobne predstavujúcim najdrastickejšiu zmenu v TeXovom systéme.

6. Penalizácia — rozhodovacia miera

Zalamovanie riadkov a strán v TeXu je hlavne determinované vážením „badness“ výsledného výstupu[7]) a penalizácie za zalomenie v aktuálnom

[6]) Toto je len dohad zo štúdia [17].

[7]) Toto je v istom zmysle miera rozdielu medzi optimálnym a aktuálnym množstvom bieleho priestoru v riadku a na strane akurát spracovávaných.

Obrázok 2: Priestory medzi účariami

Pre implementáciu dimenzie medzi účariami, napríklad medzi odstavcom a názvom (označené otáznikom), by hodnota `\parskip` mala byť určená v závislosti od `\baselineskip` druhého odstavca. Nanešťastie, používaná hodnota `\baselineskip` je aktuálna z konca druhého odstavca, zatiaľ čo `\parskip` musí byť vypočítaný na začiatku.

bode. Takáto penalizácia je buď zadaná ručne užívateľom (počas rozvoja makra), alebo dodaná neskoršie istými formátovacími rutinami TEXu.

Hlavný problém spôsobený implicitnými penalizáciami je, že tieto nemôžu byť odstránené. Ak napríklad algoritmus pre zalomenie riadku rozhodne uložiť penalizáciu po riadku (napr. z `\widowpenalty`), neexistuje spôsob, ako zamedziť na tomto mieste zalomeniu strany prostredníctvom rozvoja makra, prirodzene s výnimkou nastavenia predmetnej penalizácie na nekonečno. Toto je výsledkom TEXovského algoritmu, že následné penalizácie p_1 a p_2 sa správajú ako $p_3 := \min(p_1, p_2)$.

Lokálne zmeny prehrešujúcich sa penalizačných parametrov majú tendenciu k chybám a sú časovo náročné, lebo za každou zmenou môže nasledujúce zalomenie strany padnúť na inú pozíciu. Navyše, budúce editovanie dokumentu je zložitejšie, lebo každá korekcia tohto typu môže produkovať nežiadúce výsledky za každou jednotlivou zmenou.

Ak myslíme v reláciách súčasného TEXu (t.j. predpokladajúc, že všetky hlavné algoritmy zostanú nezmenené), tak by bolo lepšie prijať inú stratégiu v prípade za sebou nasledujúcich penalizácií, buď $p_3 := \max(p_1, p_2)$, alebo $p_3 := \frac{1}{2}(p_1+p_2)$. Pri obidvoch funkciách vyžadujú hraničné prípady $p_i = \pm\infty$ špeciálnu starostlivosť. Reťazec penalizácií môžeme preraziť ako obyčajne zoskupením alebo pomocou `\kern0pt`, alebo podobnými procedúrami, ako sa už používajú pri ligatúrach, atď.

Ako sme už povedali, toto je riešenie v rámci TEX82. Ak bude na-

vrhnutý úplne iný algoritmus pre zalomenie strán, koncepcia lokálnej penalizácie by mala byť tiež preskúmaná a pravdepodobne nahradená inou stratégiou.

7. Rozdeľovanie slov

Pri sadzbe textu, špeciálne s úzkymi stĺpcami, je rozdeľovanie slov nevyhnutné, aby sa zabránilo nečitateľným medzerám. Ovšem čitateľnosť má veľa podôb, jedno zo základných pravidiel hovorí [28]: „Vyhni sa viac než dvom rozdeleniam v riadkoch za sebou.“ Ako sme už uviedli v časti 2, toto nie je možné špecifikovať v TEXu (pokiaľ dokonca úplne neznemožníte dve rozdelenia za sebou).

Iným problémom je rozdeľovanie slov na miestach, ktoré sú síce prípustné, ale menia význam:

Stiefel-tern	Steif-eltern
Spargel-der	Spar-gelder[8])

Mali by sme pravdepodobne vždy zabrániť takýmto problematickým rozdeleniam zvolením vhodných vzorov v Liangovom algoritme [23]. Čitateľnosť však môže byť narušená aj rozdelením veľmi krátkych slabík, ktoré nedávajú skoro žiadnu informáciu o rozdelenom slove, a preto spomaľujú proces čítania. V TEXu 3.0 je teraz možné nastaviť minimálny počet písmen napravo a naľavo od rozdeľovacieho znamienka. Toto je dôležité pre veľa jazykov, ktoré majú často dlhé slová a napríklad veľa dvojpísmenových slabík podobne ako nemčina. Neexistuje ale spôsob ako priradiť váhu rôznym rozdeľovacím miestam, t.j. jedná sa o jednoduchú áno alebo nie situáciu. Jeden z možných spôsobov riešenia tejto situácie by mohol byť prostredníctvom novej triedy demerits

$$\frac{\text{užívateľova hodnota}}{\text{dĺžka zalamovanej časti}}$$

alebo podobnej funkcie. Pritom ovšem musíme pravdepodobne vo formule rozlišovať medzi textom pred zalomením a po ňom (a/alebo jeho dĺžkou).

Pretože kvalita niektorých miest zalomenia tiež závisí aj od slova (t.j. významu slovných častí), mali by sme premýšľať, či by taká informácia mala byť poskytovaná rozdeľovacím algoritmom.

[8]) Slová znamenajú „nevlastní rodičia“ a „úspory“, kým prvé časti slov v ľavom stĺpci znamenajú ‚čižmy‘ a ‚špargľa‘.

8. Rotácia boxov

Koncepcia TEXovskej reprezentácie dokumentov je orientovaná prísne horizontálne a zľava doprava. Popri probléme spracovania dokumentov, ktoré obsahujú jazyky orientované sprava doľava alebo zhora dole (ktoré môžu byť do istej miery spracované špeciálnymi verziami TEXu [21]), spôsobuje toto v štandardných aplikáciách tiež zbytočné obmedzenia. Odhliadnuc od použitia `\special` (pre PostScriptové zariadenia) je nemožné otočiť isté časti dokumentu. Kým ľubovoľné otočenie je naozaj temer nemožné, pre väčšinu výstupných zariadení môže byť otočenie o 90° urobené jednoduchým spôsobom otočením znakových buniek. Malo by byť jednoduché zahrnúť istý druh primitívneho pojmu `\rotate` do jazyka TEXu, ktorý by dovolil otáčanie vodorovných a zvislých boxov o násobky 90 stupňov.[9]) To by umožnilo zahrnutie otočených tabuliek a pod. do dokumentov bez toho, aby sme potrebovali skutočné lepidlo a nožnice na pridanie čísla strany alebo záhlavia.

9. Informácia o fontoch

ISO Draft Standard [1] obsahuje stovky charakteristík popisujúcich fonty. Zatiaľ čo niektoré z nich sú dostupné v TEXu prostredníctvom `\fontdimen`, väčšina z nich nie je. Zdá sa, že by bolo záhodné pridať viac z nich do množiny parametrov TEXu (napríklad doporučené `\baselineskip`), aby sa dali robiť jednoduchšie zmeny v triedach fontov.

9.1 Virtuálne fonty

Použitie tried fontov v TEXu, ktoré sa odlišujú v pozícii znaku, atď. od štandardu z triedy Computer Modern, je ťažké, ale môže byť dosiahnuté, ako to bolo ukázané v niekoľkých projektoch [7, 35]. Navrhované použitie virtuálnych fontov [20] môže zjednodušiť mnohé záležitosti v tomto smere. Keby bolo možné dohodnúť sa na štandardoch pre pozíciu a na spôsobe prístupu k istým znakom s diakritikou v najspoločnejšej abecede latinkou píšúcich jazykov, tak by bolo možné sádzať viacjazyčné dokumenty (použitím nových prvkov TEXu 3.0) bez zavádzania zbytočných variant štandardných fontov, odlišujúcich sa len v dostupnosti istých znakov s diakritikou ako ‚reál-

[9]) Toto by vyžadovalo zmeny v `dvi` jazyku, a preto zmeny v programoch všetkých ovládačov.

nych‘ znakov.[10]) Dobrý prehľad znakov s diakritikou v jazykoch používajúcich latinku s návrhom k ich prístupu cez ligatúry dal Haralambous [8].

9.2 Ligatúry a kerny

Ligatúry a kerny sa na nešťastie odlišujú od jazyka k jazyku. Uveďme ako príklad ligatúru ffl, ktorá sa nepoužíva v tradičných nemeckých dokumentoch. Na druhej strane tieto dokumenty používajú ligatúry ch, ck a ft na dosiahnutie lepšieho písma. Nasledujúce príklady ukazujú rozdiel:

Druckschrift	Druckschrift	(štandardne)
Druckschrift	Druckschrift	(nemecká ligatúra)

Pretože tieto špeciálne ligatúry nezahŕňajú nové tvary písmen (aspoň nie vo väčšine tried fontov), je možné dosiahnuť žiadúce výsledky jednoducho použijúc kerning. V triede fontov Computer Modern [15] sú obidva prípady ch a ck zahrnuté do kerningových programov, ale len pre písmo patkové. Iné triedy fontov vykazujú podobné nedostatky. Preto na sádzanie nemeckých dokumentov buď potrebujeme špeciálne fyzické fonty (alebo aspoň virtuálne fonty), alebo spôsob na manipuláciu s ligatúrami a kerningovými programami v rámci programu TEX. Z dôvodu prenosnosti sa zdá, že ovládateľný prístup k ligatúrnym a kerningovým programom počas zavádzania fontov je lepší.

10. Tabuľky

Pekne proporcionálne tabuľky je ťažko sádzať aj pre skúseného ručného sadzača. Primitívne pojmy TEXu `\halign` a `\valign` robia báječnú službu v tomto smere, a to i v komplikovaných situáciách. Existuje však dôležitá podtrieda tabuliek, s ktorou sa vôbec ťažko pracuje, s výnimkou možnosti ručného prispôsobovania. Nie je možné špecifikovať kombináciu horizontálne a vertikálne rozpätých stĺpcov, napr. otvorenú krútenú zátvorku cez niekoľko riadkov v jednom stĺpci, kým na obidvoch jej stranách pokračuje riadková štruktúra.

Iným často požadovaným prvkom sú tabuľky zaberajúce niekoľko strán. Toto je ťažko dosiahnuť, lebo primitívne príkazy TEXu pre tabuľky za normálnych okolností prečítajú celú tabuľku, kým určia šírku stĺpca atď.

[10]) Použitie diakritických znakov ako primitívnych pojmov sa nedoporučuje pre štandardnú diakritiku jazyka [18, str. 54] z dôvodu znemožnenia rozdeľovania slov.

Nespôsobuje to ale neriešiteľné problémy s pokročilými prvkami TEXu 3.0.

11. Matika

Matematická sadzba je jednou z hlavných TEXovských domén, kde ju žiadny iný automatický sádzací systém nebol schopný dohoniť. Ale aj v tejto oblasti by sa dalo niekoľko vecí zlepšiť.

Zdrojový program $\mathcal{AMS}$-TEXu [30] poskytuje veľa zaujímavých príkladov, kde Spivak preľstil obmedzenia TEXovských formátovacích pravidiel vytvorením komplexného programu na definovanie funkcií, ktoré majú za úlohu realizovať štandardné úlohy matematickej sadzby. Detailná analýza týchto problémov (dvojitá diakritika, spodná diakritika, umiestnenie číslovania rovníc, atď.) by ľahko zaplnila niekoľko strán; isté poznámky môžete nájsť v Spivakovej dokumentácii [29].

Kým TEXovské pravidlá pre medzerovanie v matike sú celkom dobré, zdá sa, že je prinajmenšom otázne, ak množstvo z nich je natvrdo zadrôtovaných do programu namiesto toho, aby boli dosiahnuteľné prostredníctvom parametrov. Tabuľka pre medzerovanie medzi rôznymi matickými atómami je pravdepodobne najdôležitejší príklad tohto druhu.

Iným problematickým prvkom TEXovských sádzacích rutín pre matiku je, že subformuly sú vždy zaboxované v prirodzenej šírke, a to aj vtedy, keď najvyššia úroveň math listu je predmetom roztahovania alebo sťahovania. To môže za určitých okolností viesť k ohyzdným výsledkom. Koncepcia zaboxovania subformúl má dodatočnú nevýhodu, že tieto časti formúl nemôžu byť zalomené cez riadok. Preto programátorské štruktúry ako `\left...\right`, ktoré automaticky určujú výšku delimiterov s variabilnou výškou, nemôžu byť použité pri zložitých výstupoch.

12. Jazyk TEXu

Jazyk TEXu je rozdelený na dve časti, ktoré sa popisujú ako ústa a žalúdok TEXu [18]. Tento rozdiel je rozhodujúci vo veľa aplikáciách, lebo výstup produkovaný gastronomickými rutinami nemôže byť zavedený opäť do jeho úst, t.j. do jeho scanneru. V skutočnosti sú skonštruované boxy post-procesovateľné v ohraničenom rozsahu (cez `\lastbox`, `\unpenalty`, atď.), ale ľubovoľné konštrukcie nemôžu byť spracované týmto spôsobom, lebo chýbajú primitívne pojmy pre manipuláciu s prvkami ako písmená, linky, atď. Tento rozdiel vo všeobecnosti je dôvodom pre veľa prekážok

v TEXovskom programovaní, čo niekedy úplne znemožňuje akékoľvek riešenie. Nový systém by mal odstrániť túto dvojtriednu spoločnosť vnútorných príkazov.

Iným vážnym problémom jazyka je jeho neúplnosť ohľadne štandardných programovacích konštrukcií (takých, ako isté podmieňovacie príkazy, prijateľný aritmetický parser atď.), ako aj špeciálnych konštrukcií vhodných pre sadzbu. Napríklad na určenie dĺžky posledného riadku odstavca (čo je robené TEXom automaticky pri sadzbe samostatných formúl), musíme použiť komplikovaný a zdĺhavý výpočet tak, ako je ukázané v príklade 2. Tento príklad tiež ukazuje jednu nedôslednosť jazyka TEXu: `\prevgraf` musí byť menený použitím pomocného registra, lebo priame použitie príkazu `\advance` je zakázané. Veľa problémov tohto druhu nájdete v prílohe D *The TEXbooku* [18], str. 373–401, ktorá má názov „Špinavé triky“. V skutočnosti deväť z týchto desiatich príkladov je použitých v implementácii LATEXu [22], čo ukazuje, že tieto príklady sú zďaleka nie tak exotické, ako predslov tejto prílohy naznačuje. Ako príklad chýbajúcich programovacích konštrukcií uveďme podmieňovací príkaz typu `\ifmathopen`.

Takéto problémy vysvetľujú skutočnosť, že všeobecný aplikačný software napísaný v TEXu (ako LATEX) ľahko zaberie viac ako tretinu prístupnej pamäti bez toho, aby sme napísali jediné písmeno. Pre lepšie a stabilnejšie vstupné a výstupné konce potrebujeme jazyk, v ktorom takéto úlohy môžu byť špecifikované oveľa elegantnejším spôsobom.

Niektoré obmedzenia jazyka TEXu sú spôsobené reprezentáciou dimenzií ako ‚reálnych čísiel‘ vo väčšine inštalácii TEXu, ktoré sú strojovo závislé.[11]) Na zabezpečenie strojovej nezávislosti TEXu sa Knuth pokúsil predísť tomu, aby sa strojovo závislé výsledky generované v žalúdku TEXu rozplynuli do častí prístupných scanneru, alebo aby vnútorne ovplyvňovali akékoľvek rozhodnutia o zalamovaní riadkov alebo strán. Výsledkom tejto stratégie je, že použitie programu z príkladu 2 spolu s konečným `\parfillskip` (ako v príklade 1) skoro vždy produkuje hodnotu `\maxdimen` namiesto decentnej.[12]) Z toho istého dôvodu povedal Knuth v rozhovore s autorom na univerzite v Stanforde, že nemôže dovoliť odstránenie ľubovoľných položiek v zostrojovaných zoznamoch, lebo toto by viedlo k prístupu k aritmetike s pohyblivou rádovou čiarkou.

[11]) Toto sa de facto vzťahuje len na interné dimenzie reprezentujúce naťahovanie a sťahovanie glue, počítaných kernov pre diakritiku a niektorých ďalších prípadov.

[12]) Dôvod je daný v module 1148 programu TEX [16, str.470].

Ovšem existuje iný spôsob ako dosiahnuť strojovú nezávislosť, ktorý by tiež odstránil uvedené obmedzenia, a to prechod od aritmetiky v pohyblivej rádovej čiarke k aritmetike s pevnou rádovou čiarkou,[13]) ktorý môže byť urobený jednoduchým spôsobom, ako sám Knuth s poďakovaním kvitoval [16, str. 46].

TEX je makrojazyk so všetkými výhodami i nevýhodami. Ktokoľvek raz napísal relatívne dlhší program v TEXu, vie, že jeho odladovanie je mimoriadne ťažké. Prehľadné programovanie tak, ako bolo navrhnuté autorom TEXu [10], je temer nemožné: nie je problém napísať tri riadky programu v TEXu, ktorý je nezrozumiteľný dokonca pre TEXperta bez toho, že by si ho prečítal dvakrát trikrát. Je však ešte ťažšie napísať program v TEXu, ktorý vykoná požadovanú akciu, aby tento bol zrozumiteľný pre priemerného užívateľa. Príklady uvedené v časti 14 sú dobrými testovacími príkladmi; sú to jednoduché programy v TEXu, ale ich presné pochopenie je ťažké bez vysvetľujúceho textu.

Veľa problémov povstáva z dizajnérskych rozhodnutí založených na úplne rozdielnych sémantických konštrukciách, ktoré majú podobnú alebo identickú syntaktickú štruktúru. Najdôležitejšími príkladmi sú krútené zátvorky a znak dolára.[14]) Krútené zátvorky sa používajú jednak na delimitáciu argumentov počas rozvoja makier, a jednak ako začiatok a koniec blokovej štruktúry, ktorá definuje rozsah istej deklarácie. V matematickom móde majú dodatočný význam delimitovania rozsahu subformúl. Dva za sebou stojace doláre obyčajne začínajú a končia výpis formúl na samostatnom riadku, ale v obmedzenom horizontálnom móde označujú jednoducho prázdnu matematickú formulu. Takéto koncepcie by mali byť z dôvodu jasnosti rozmotané.

Jazyk TEXu je vhodný pre jednoduché programátorské práce. Je to ako krok od strojového jazyka (formátovača) k asembleru. Pre komplexné programátorské úlohy všeobecného aplikačného softwaru, najmä z hľadiska logicky označených dokumentov [5], bol by výhodnejší mocný jazyk s dobre definovanou koncepciou väzby premenných, procedúr atď. Kým tento aspekt môže byť dosiahnutý na vstupnom konci programovacieho jazyka (ktorý sa skompiluje do jazyka TEXu), je lepšie, z dôvodu prenosnosti, zahrnúť ho do jadra TEXu. Podľa autorovho názoru, ideálny jazyk by mal kombinovať výhody procedurálneho jazyka s prospešnými

[13]) Snáď použitie vždy toho istého programu pre aritmetiku v pohyblivej rádovej čiarke (buď dostupného v knižnici kompilátora, alebo simulovaného programom) by bolo ešte lepšie.

[14]) Aby sme boli presnejší, tri znaky s `\catcode` jeden, dva a tri.

črtami interpreteru.[15]) Vedľajším efektom by bolo, že takýto jazyk by bol čiastočne kompilovateľný.

13. Záver

Súčasný TEX nie je dostatočne mocný na realizovanie všetkých potrieb vysokokvalitnej (ručnej) sadzby. Autor zdieľa Knuthov sen o stabilnom formátovači na úrovni nižšieho jazyka, ktorý je schopný vytvárať dokumenty najvyššej kvality. Na rozdiel od Knutha však chápe súčasný TEX len ako veľmi dobrý prototyp na ceste k tomuto cieľu.

Ako je to naznačené v tomto článku, veľa dôležitých koncepcií sadzby vysokej kvality nie je podporovaných TEXom 3.0. Pre ďalší výskum je nutné navrhnúť sádzací jazyk, ktorý vie vhodne riešiť tieto problémy.

Pospolitosť užívateľov TEXu potrebuje prístupnosť a nezaujatosť pre nový rozvoj, ktorý udrží ‚systém TEX' vrcholovým v oblasti počítačovej sadzby. Keďže Knuth už nie je zapojený do výskumu v typografii, je dôležité pre TUG, aby sa zjednotila v *podpore* a *pestovaní* ‚najlepšieho sádzacieho programu' a nielen v propagovaní programu, ktorý dal Knuth svetu. Ak sa nezmobilizujeme pre ešte lepšiu kvalitu, môže naša pospolitosť upadnúť do bezvýznamnosti.

Jeden dôležitý krok pre TUG by bolo inicializovať a (ak je to vhodné) podporovať ďalšie výskumné projekty, ktoré zdvihnú rukavicu hodenú Stanfordským projektom.

14. Príklady

Príklad 1. Nasledujúci program môže byť použitý na zabránenie skoro prázdnych riadkov na konci odstavca:

```
\parfillskip \hsize
\advance\parfillskip by -1.5\parindent
\advance\parfillskip by 0pt minus \parfillskip
\advance\parfillskip by 0pt minus -1em
```

Toto nastavenie bolo použité v tomto článku a napríklad viedlo k istým zmenám v druhom odstavci abstraktu. So štandardným nastavením (napr. `0pt plus 1fil`) by tento odstavec ukončil časťou slova ‚kov' (od ‚ro-kov').) Nanešťastie toto riešenie nie je tiež dokonalé, lebo vyprodu-

[15]) Akýsi druh jazyka lispovského typu, ale s primitívnymi pojmami vhodnými pre sadzbu.

kuje úsmevný výsledok s riadkami pozostávajúcimi z dvoch krátkych slov.

Príklad 2. Nasledujúci program určuje dĺžku posledného riadku predchádzajúceho odstavca, využívajúc istý prvok TEXu, ktorý je zabudovaný do vypisovania matematických formúl na samostatný riadok. Tento program môže byť určený napríklad na určenie množstva bieleho miesta pred zoznamom jednotlivostí alebo niečoho podobného. Ukážkový program nie je naozaj vhodný na priame použitie takéhoto druhu, lebo jednoducho vypíše nájdenú hodnotu na terminál. Dá sa však ľahko rozšíriť.

```
\def\getlastlinewidth{\ifhmode $$%
  \predisplaypenalty\@M \postdisplaypenalty\@M
  \abovedisplayskip-\baselineskip \belowdisplayskip\z@
  \abovedisplayshortskip\abovedisplayskip
  \belowdisplayshortskip\belowdisplayskip
  \showthe\predisplaysize
  $$\count@\prevgraf \advance\count@-\thr@@
  \prevgraf\count@ \else\typeout{*Not hmode}\fi}
```

Tento príklad ilustruje niekoľko dôležitých vecí. Po prvé, neexistuje žiadna elementárna metóda na výpočet takýchto dôležitých informácií. Po druhé, je to jeden (nie neobvyklý) z prípadov, keď informácia o sádzacom procese môže byť získaná jedine zavedením nežiadúcej (zrejme miesto zaberajúcej) penalizácie, glues a nulových boxov vo výstupe.

Príklad 3. Pri zavedení mrežovo orientovanej špecifikácie musia byť všetky flexibilné glue odstránené (s výnimkou `\skip\footins`) a `\vsize` musí byť adjustovaná. Nadpisy sú nastavené s rozpaľom `8pt + 4pt=` `\baselineskip` a musíme zabezpečiť, že vrchný priestor je zachovaný po každom zalomení strany. Zoznamy sú nastavené s `6pt + 6pt`, takže vnútorné linky sú o polovicu riadku posunuté. Zalomenia strán vo vnútri zoznamov vyžadujú špeciálny prístup, napr. zvýšenie `\topskip` na udržanie podmriežky. Obrázky a príklady s rozdielnymi typmi veľkostí sú vyvážené a nevyhnutné kerny pridané na udržanie okolitého materiálu v súlade. Tento prístup opäť funguje, jedine ak nezasiahne žiadne zalomenie strany, čo je aj prípad tohto článku. Pri použití TEXovského počítania špatnosti na určenie zalomenia strán môže byť použité naťahovacie `\topskip`. Počas výstupnej rutiny musí byť toto extra natiahnutie opäť zrušené.

Literatúra

1. *Information Processing — Font InformationInterchange, ISO/IEC JTC 1/SC 18/WG8 N1036*, February 1990
2. Achugbue, James O. "On the line breaking problem in text formatting." *Proc. of the ACM SIGPLAN/SIGOA,* 2 (1, 2), 1981.
3. Asher, Graham. "Type & Set: TEX as the engine of a Friendly Publishing System." In *TEX applications, uses, methods*, 91–100, Malcolm Clark [6].
4. Benson, Gary, Debi Erpenbeck, and Jannet Holmes. "Inserts in a multiple-column format." In *1989 Conference Proceedings*, 727–742, Christina Thiele [31].
5. Bryan, Martin. *SGML: an author's guide to the standard generalized markup language*, Addison-Wesley, Woking, England; 1988, Reading Massachusetts, second edition.
6. Clark, Malcolm, editor. *TEX applications, uses, methods*, Chichester, West Sussex, England; 1990, Ellis Horwood Limited. Exeter conference July 1988.
7. Conrad, Arvin C. "Fine typesetting with TEX using native autologic fonts." In *1989 Conference Proceedings*, 521–528, Christina Thiele [31].
8. Haralambous, Yannis. "TEX and latin alphabet languages." *TUGboat,* 10 (3), November 1989, 342–345.
9. Hoenig, Alan. "Line-Oriented Layout with TEX." In *TEX applications, uses, methods*, 159–184, Malcolm Clark [6].
10. Knuth, Donald E. "Literate programming." *The Computer Journal,* 27 1984, 97–111, An expository introduction to WEB and its underlying philosophy.
11. Knuth, Donald E. *Computers & Typesetting*, Addison-Wesley, Reading, Massachusetts; 1986, Consists of [18, 16, 13, 12, 15].
12. Knuth, Donald E. "METAFONT: The Program." In *Volume D of Computers & Typesetting* [11], 1986.
13. Knuth, Donald E. "The METAFONTbook." In *Volume C of Computers & Typesetting* [11], 1986.
14. Knuth, Donald E. *Talk given at Gutenberg Museum Mainz*, September 1987.
15. Knuth, Donald E. "Computer Modern Typefaces." In *Volume E of Computers & Typesetting* [11], July 1987, Reprint with corrections.

16. Knuth, Donald E. "TEX: The Program." In *Volume B of Computers & Typesetting* [11], May 1988, Reprint with corrections.
17. Knuth, Donald E. "The errors of TEX." In *Technical Report STAN-CS-88-1223*, Stanford University, Department of Computer Science, Stanford, California 94305; September 1988.
18. Knuth, Donald E. "The TEXbook." In *Volume A of Computers & Typesetting* [11], May 1989, Eighth printing.
19. Knuth, Donald E. "The now versions of TEX and METAFONT." *TUGboat,* 10 (3), November 1989, 325–328.
20. Knuth, Donald E. "Virtual Fonts: More fun for Grand Wizards." *TUGboat,* 11 (1), April 1990, 13–23.
21. Knuth, Donald E. and Pierre MacKay. "Mixing right-to-left text with left-to-right text." *TUGboat,* 8 (1), April 1987, 14–25.
22. Lamport, Leslie. `latex.tex`, February 1990, LATEX source version 2.09.
23. Liang, Franklin Mark. "Word Hy-phen-a-tion by Com-put-er." In *PhD thesis*, Stanford University, Depertment of Computer Science, Stanford, CA 94305; August 1983, Report No. STAN-CS-83-977.
24. Mittelbach, Frank. "An enviroment for multicolumn output." *TUGboat,* 10 (3), November 1989, 407–415.
25. Mittelbach, Frank. "Letter to Don Knuth." In *Suggestions for the TEX 3.0 release*, September 1989, Published by R. Wonneberger in [34].
26. Plass, Michael Frederick. "Optimal Pagination Techniques for Automatic Typesetting Systems." In *PhD thesis*, Stanford University, Depertment of Computer Science, Stanford, CA 94305; June 1981, Report No. STAN-CS-81-970.
27. Rynning, Jan Michael. "Proposal to the TUG meeting at Stanford." *TEXline,* 10 May 1990, 10–13, Reprint of the paper that triggered TEX 3.0.
28. Siemoneit, Manfred. *Typographisches Gestalten*, Polygraph Verlag, Frankfurt am Main; 1989, second edition.
29. Spivak, Michael. `amstex.doc`, 1990, Comments to [30].
30. Spivak, Michael. `amstex.tex`, 1990, $\mathcal{AMS}$-TEX source version 2.0 (without comments).
31. Thiele, Christina, editor. *1989 Conference Proceedings*, volume 10#4 of TUGboat, December 1989, TEX Users Group.
32. Wittbecker, Alan E. "TEX enslaved." In *1989 Conference Proceedings*, 603–606, Christina Thiele [31].

33. Wonneberger, Reinhard. "TEX in an industrial enviroment." In *Brüggemann-Klein, Anne, editor, 1989 EuroTEX*, Conference Proceedings1990, To appear.
34. Wonneberger, Reinhard. "TEX yesterday, today and tomorrow." *TEXhax*, 90 (5), January 7, 1990.
35. Youngen, R. E., W. B. Woolf, and D. C. Latterner. "Migration from computer modern fonts to times fonts." In *1989 Conference Proceedings*, 513–519, Christina Thiele [31].

Frank Mittelbach
Electronic Data Systems (Deutschland) GmbH
Eisenstraße 56 (N 15), D-6090 Rüsselsheim,
Spolková republika Nemecko
Tel. +49 6142 803267
Bitnet: `pzf5hz@drueds2`

z anglického originálu (TUGBOAT 11 (1990), 337–345)
přeložil *Štefan Porubský*

Grafika v TEXu (2)

PETR SOJKA

V dnešním pokračování seriálu o grafice v sázecím systému TEX se zaměříme na grafické možnosti jazyka POSTSCRIPT[1]) a na způsoby integrace grafiky vytvořené v jazyce POSTSCRIPT do TEXových dokumentů.

POSTSCRIPT

Jazyk POSTSCRIPT je poměrně jednoduchý programovací jazyk s bohatými grafickými možnostmi. Byl vyvinut firmou ADOBE za účelem popisu vzhledu textu, grafiky a bitových map na tištěné stránce (POSTSCRIPT) nebo stránce zobrazované na obrazovce (DISPLAY POSTSCRIPT). Popis stránky je nezávislý na používaném zařízení (tiskárně,

[1]) POSTSCRIPT, DISPLAY POSTSCRIPT, ADOBE jsou ochranné známky firmy ADOBE Systems Incorporated.

obrazovce). Tato zařízení však musí být schopna interpretovat příkazy jazyka PostScript (většinou hardwarově vestavěným interpretem).

Některé možnosti a vlastnosti jazyka PostScript:

- Vytváření kubických křivek (tento mocný mechanismus je například použit pro definice obrysů znaků PostScriptových písem).
- Operátory, které umožňují křivky jakékoliv tloušťky, vyplňované jakoukoliv barvou (v některé z norem RGB, CMYK či CIE), jakýmikoliv vzory.
- Text plně integrovaný s grafikou umožňuje operace jako lineární transformace, zvětšování, zmenšování, rotace, sklánění, zrcadlení apod., nezávisle na rozlišení výstupního zařízení.
- Programy v jazyce PostScript jsou reprezentovány pomocí standardní sady znaků ASCII[2]) a mohou proto být jednoduše přenášeny informačními kanály (email apod.).

Tyto vlastnosti popisu a zobrazování stránek v jazyce PostScript, spolu s otevřenou licenční politikou firmy Adobe způsobily masové rozšíření jazyka a vytvoření de facto standardu pro popis stránky.

Jazyk PostScript je přirovnáván k zásobníkovému jazyku Forth a používá postfixovou notaci pro zápis programů. Pro čtenářovu představu o příkazech a syntaxi jazyka uvádíme malý příklad PostScriptového programu včetně výstupu na obrázku 1.

Většina dnešních interpretů již umí zpracovat normu jazyka zvanou *Level II*, která je oproti prvotní verzi jazyka z počátku osmdesátých let rozšířena o primitiva práce s barvou atd. Další nezávislá rozšíření mají interprety Display PostScriptu určeného pro interaktivní komunikaci s PostScriptovým zařízením.

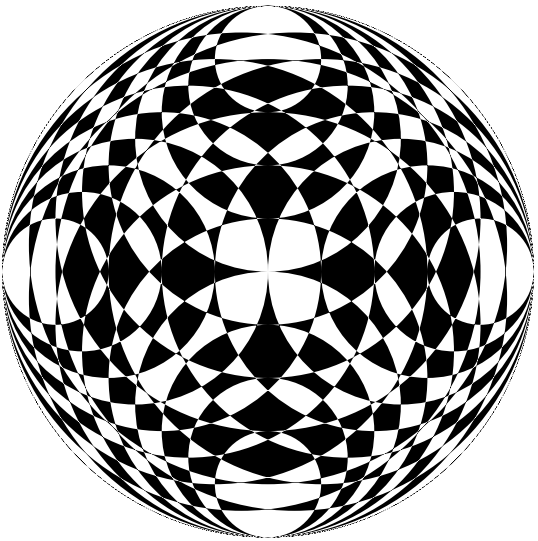
Integrace do TEXu

TEX jako takový žádné příkazy (primitiva) pro práci s PostScriptem nemá. Umožňuje však „vkládat“ do `.dvi` souborů příkazy pro ovladače (drivery) výstupních zařízení příkazem `\special`. Tyto informace (například příkazy v jazyce PostScript) pak jsou zpracovány ovladačem či poslány na příslušné (třeba barevné) výstupní zařízení. Pro PostScriptová zařízení existuje ovladač `dvips`.[3]) Tradiční model práce je

[2]) American Standard Code for Information Interchange

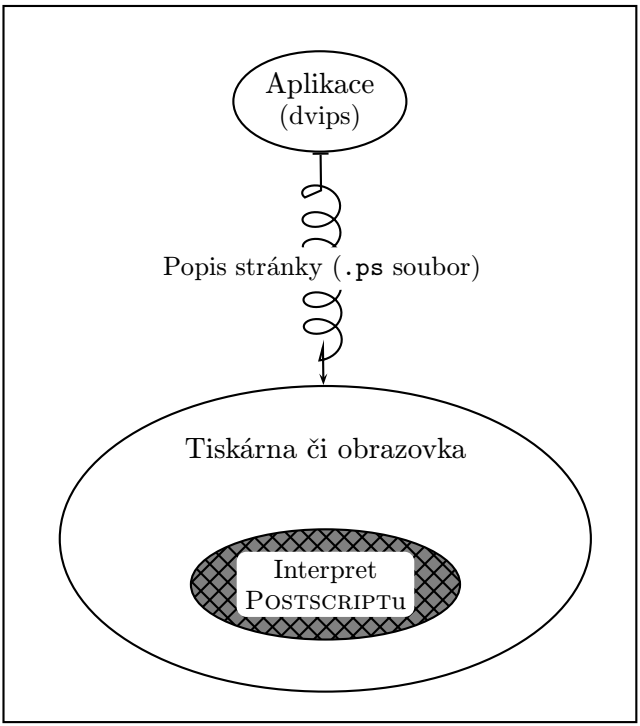
[3]) `ftp.muni.cz`, adresář `pub/tex/drivers/dvips`.

```
/const 10 def   /quadrant 4 def
-90 rotate
quadrant {
 -100 0 translate
 0 0 moveto
 2 1 const {
   100 const div mul
   dup 0 exch
   -180 180 arc
 } for
 100 0 translate
 360 quadrant div rotate
} repeat
eofill
```



Obr. 1. Příklad PostScript programu a jeho výstupu

znázorněn na obrázku 2. Verzi pro Display PostScript znázorňuje obrázek 3.



Obr. 2. Tradiční model práce s PostScriptem

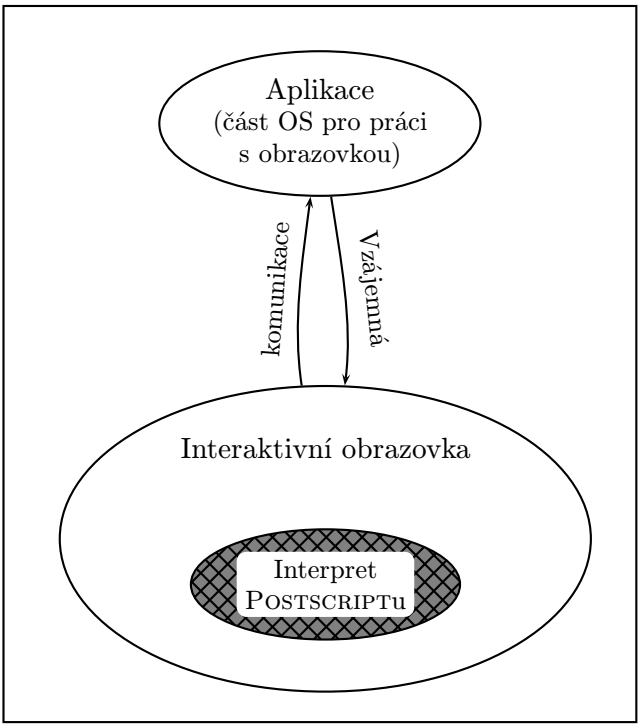
Makra pro práci s PostScriptem

Jednou z možností, jak využívat možností jazyka PostScript bez nutnosti jeho zvládnutí jsou různé balíky (LA)TEXových maker. Tato makra umožňují generování výše zmíněných příkazů `\special` pomocí standardního mechanismu vhodně volených maker.

Makra `pstricks`

Nejrozšířenějším takovým balíkem je balík s názvem `pstricks`. Autorem tohoto bohatě dokumentovaného balíku je Timothy van Zandt a jeho distribuční verzi `pstricks.zip` lze nalézt na `ftp.muni.cz` v adresáři `pub/tex/styles/tvz`.

Jako příklad používání těchto maker uveďme kód pro obrázek na obálce tohoto čísla:



Obr. 3. Model práce s Display PostScriptem

„Bomba“

```
\psset{unit=7pt}
\begin{pspicture}(-4,-4)(4,4)
 \pscircle[fillcolor=gray,fillstyle=solid]{4}
 \rput{45}{\scaleboxto(6,0){PS\kern-1pt}}
 \psellipse*(0,3.8)(1,.2)
 \psellipse*(0,5.5)(1,.2)
 \psframe*(-1,3.8)(1,5.5)
 \psbezier(0,5.5)(0,7)(1,8)(3.5,8)
 \multido{\r=-0+40}{9}
  {\rput{\r}(3.5,8){\psline(0,0)(.4,0)}}
\end{pspicture}}}}
```

Nadpis

```
\psset{linestyle=none}\LARGE
\pstextpath[c]{\psarcn(0,0){64pt}{180}{0}}%
      {Petr Sojka}
\pstextpath[c]{\psarc(0,0){64pt}{180}{0}}%
      {Grafika v~\TeX u (3)}
```

Pomocí těchto maker byly též vytvořeny obrázky 2 a 3.[4])

Makra pro vkládání EPS

Častá je situace, kdy obrázek vytvoříme specializovaným systémem v PostScriptu (jako tzv. Encapsulated PostScript, obsahující v poznámce rozměry obrázku) a chceme ho vložit do TEXového dokumentu. Maker toto umožňujících je celá řada, jmenujme alespoň `epsf.sty` či `psfig.sty`, obsažené většinou v distribuci PostScriptových ovladačů. Na místě v TEXovém dokumentu, kde chceme obrázek umístit, vyvoláme obvykle makro s názvem souboru obsahujícím obrázek, případně se specifikací, na jakou výšku a šířku se má obrázek deformovat vůči standardním rozměrům.

Prohlížení PostScriptových souborů

Nejrozšířenější prohlížeče postscriptových souborů na obrazovce jsou produkty projektu GNU Ghostscript a Ghostview, pod OS Unix jich existuje ještě několik (Pageview, Sunview). Tyto interprety jsou náročné na operační paměť; Ghostscript se dá provozovat i na počítačích PC pod operačním systémem MS DOS. Tímto programem můžete také konvertovat PostScriptové soubory do tvaru vhodného pro tisk na různých nePostScriptových tiskárnách (jehličkových, laserových). Ghostscript najdete v `ftp.muni.cz` v adresáři `pub/tex/gs`. Ghostview je verze určená pro prohlížení PostScriptu na obrazovce s možnostmi obvyklými u TEXových ovladačů jako stránkování, zooming apod.

[4]) Tento způsob je většinou rychlejší i bezpečnější, neboť při programování v „čistém“ PostScriptu je třeba dbát velké obezřetnosti a opatrnosti a ladící možnosti jsou omezené.

Závěr

Jazyk PostScript rozhodně zaslouží pozornost a při koupi např. laserové tiskárny byste na něj neměli zapomenout. Pomocí vhodných maker Vám umožní jednoduše integrovat grafiku do Vašich TEXových dokumentů.

Další informace o PostScriptu můžete najít v referenční příručce [3], učebnici jazyka [4], knize o programovacích technikách jazyka PostScript [2] či popisu formátu písem AdobeType 1 [1].

S přáním, aby se Vám dařilo vytvářet grafiku ve Vašich TEXových dokumentech na alespoň takové úrovni, na jaké jsou obsahově, se s Vámi loučí autor.

Literatura

[1] Adobe Type 1 Font Format, Addison-Wesley

[2] PostScript Language Program Design, Addison-Wesley 1990, ISBN 0-201-14396-8

[3] PostScript Language Reference Manual, Addison-Wesley 1990, ISBN 0-201-18127-4

[4] PostScript Language Tutorial and Cookbook, Addison-Wesley 1990, ISBN 0-201-10179-3

Malý dodatek pro příznivce METAFONTu (*Karel Horák*)

Ilustrační obrázek 1 mě zaujal natolik, že jsem se pokoušel porozumět jeho PostScriptovému zápisu v naději, že se něčemu přiučím. Musím se však přiznat, že nebýt samotného obrázku, asi bych se daleko nedostal. Nedá mi, abych neuvedl, jak takový obrázek jednoduše vytvořit pomocí METAFONTu:

```
beginchar(1,70mm#,70mm#,0);
z0=(w/2,h/2);
blpen d1;
 draw fullcircle scaled w shifted z0;
for j=0 upto 3:
 for k=2 upto 10: fill (fullcircle scaled (.1k*w)
                shifted(.05k*w,.5h)) rotatedaround(z0,90j);
                 cull currentpicture keeping (1,1); endfor
endfor endchar;
```

Kouzla s programem MNU

Petr Olšák

Do nového balíku CSTEXu byl zahrnut program MNU, který řídí režim nabídek. Nabídka je za pomocí tohoto programu plně konfigurovatelná. V rámci „reklamací a připomínek“ k CSTEXu zaznělo několik návrhů na mírnou úpravu konfigurace nabídek. Rozhodl jsem se žádnému z těchto návrhů nevyhovět a do inovované verze změny v této oblasti nezanášet. Místo toho chci v tomto článku ukázat, jak se takové změny dělají. Každý si je může udělat sám podle svých představ. Začnu nejprve popisem vývoje (nebo spíše „nevývoje“) vlastního programu MNU.

Zhruba před dvěma lety jsem při pohledu na nabídkový program, který byl tehdy zahrnut do instalace TEXu, došel k závěru, že by stálo za to napsat něco jiného. Nejprve jsem si zformuloval, co vše by program měl umět – tj. napsal jsem dokumentaci k neexistujícímu programu.[1]) Pak jsem věc konzultoval s tehdejším výborem CSTUGu a po předběžném souhlasu, že by takový program mohl být užitečný, jsem se pustil do díla. Tak vznikl program MNU. S první verzí jsem byl hotov do začátku EuroTEXu 92, kde jsem ji též zveřejnil. Pak jsem ještě zpracoval některé návrhy a připomínky a výsledkem byla verze z ledna r. 1993, kterou už považuji za definitivní. Tato verze je zařazena do balíku CSTEX.

Za celý rok provozu programu MNU v nové instalaci CSTEXu jsem nezaznamenal k tomuto programu jedinou připomínku nebo reklamaci a to mě vede k rozhodnutí dále program pokud možno neměnit. Můj názor je, že velkým přínosem každého programu je jeho stálost (viz například program TEX), protože uživatel se nemusí topit v džungli nejrůznějších čísel verzí a být neustále ve střehu, zda nepřišel náhodou o nejnovější verzi a zda ty nové verze ještě bude umět používat.

Samozřejmě svědomí mi nedovolí nechat v programu objevené chyby. Ačkoli jsem nedostal zprávu o nalezení chyby od žádného uživatele, sám jsem za ten rok jednu chybu objevil. Jedná se o nesprávnou lokalizaci aktivní editované položky v případě, že se tato položka nalézá v okně

[1]) Musím přiznat, že tento obrácený postup – nejprve dokumentace a potom teprve program – se mi osvědčil i při tvorbě jiného software, a všem jej vřele doporučuji. Je-li totiž jasně zformulovaná myšlenka, je pak další programovací práce většinou záležitostí pro „cvičenou programovací opici“.

bez rámu. Při opravě této chyby jsem program obohatil o jednu maličkost – schopnost provést definovanou akci po daném časovém limitu „nic-nedělání“ (dá se nakonfigurovat například zhasnutí obrazovky po daném čase). Inovovaný program jsem zařadil do CSTEXu 94 a věřím, že to byly skutečně poslední úpravy programu.

Program MNU má na rozdíl od programu TEX omezenou životnost. Program totiž ryze účelově rozšiřuje možnosti „dávkového programování“ pod operačním systémem DOS. Ačkoli je DOS velmi rozšířen, nevěštím mu dlouhou budoucnost. Větší vyhlídky do budoucna mají systémy, v jejichž jádru je řešena problematika paralelního zpracování procesů.[2]) V současné době jsou už i stolní počítače pro tyto účely dostatečně výkonné. Pro uživatele TEXu jsou systémy s více procesy přímo nutností, neboť vypisovat si poznámky na papír v okamžiku, kdy mám na obrazovce „View“, abych mohl tyto poznámky použít hned, jakmile se vrátím do editoru, to je metoda, které se budeme zanedlouho (doufám) pouze smát. V DOSu přitom není příliš mnoho jiných možností.[3])

Dnes bych program MNU asi naprogramoval jinak a možná lépe (například jazyk konfiguračního souboru by nebyl tak kryptický), ale z důvodů uvedených výše nebudu program dále rozvíjet. DOS ale zatím u nás vládne, a proto si myslím, že některé triky, které se dají s programem MNU a s DOSem provádět a které uvádím v tomto článku, může ještě plno lidí využít.

Program MNU je „in public domain“. Můžete jej získat ve dvou podobách (v obou případech se jedná o naprosto stejný program). První možností je získat jej jako samostatný balík (ftp z archívu `ftp.tex.ac.uk`, adresář `support`). Zde najdete anglickou dokumentaci a dva příklady použití programu. Oba příklady pracují pouze „teoreticky“, tj. v místě volání zvoleného programu z nabídky se ve skutečnosti neuskuteční nic. První příklad je jednoduchý a druhý demonstruje konfiguraci CSTEXu.

Program můžete také získat jako součást CSTEXu. Naleznete zde českou dokumentaci k programu (v souboru `\doc\programs\mnu.tex`) a rovněž tu najdete dva příklady – v tomto případě ovšem plně funkční. Prvním příkladem je samotná konfigurace nabídek v CSTEXu a druhým instalační program umožňující pohodlnou instalaci CSTEXu z disket. Dokumentace

[2]) Nemám na mysli DOSovskou nadstavbu WINDOWS. Ta není příliš vydařeným pokusem dát DOSu náplast ve formě vize z paralelního zpracování procesů. Přitom zůstává jen u té vize a skutek utek'.

[3]) Ani rezidentní editory typu Side Kick apod. příliš nepomohou. Jsou s nimi spíš problémy – motají grafiku prohlížeče a paměť počítače a jejich užitečnost je sporná.

k prvnímu příkladu je uložena v souboru `\doc\cfg.doc` a k druhému příkladu v souboru `install.doc` na první instalační disketě. Nechci zde opakovat, co je řečeno v dokumentaci. Proto se raději věnujme jednotlivým ukázkám. Začneme tím nejjednodušším.

Jak přidat novou položku do systému nabídek?

Z veřejných archívů jsme například získali program Ghostscript a chceme ho zařadit do systému nabídek CSTEXu tak, aby se jeho vyvoláním spustila grafická interpretace PostScriptového souboru na obrazovce. Položku nazveme třeba `GS-View` a zařadíme ji do okénka `Others`. Nejprve provedeme změny v souboru `\cfg\ram\cfg.mnu`:

```
^131  132  133  134  135  136  [37]  ; ******* Others *******
~param {31, 16, 14, 8}
| DVI-O^ut    |
| DVIP^S      |
| MakeI^ndex  |
| B^ibTeX     |
| T^eXCad     |
| G^S-View    |
```

Pro položku `GS-View` jsme zvolili číslo 136 (viz první řádek) a výšku okénka `Others` jsme o jedničku zvětšili (viz druhý řádek, číslo 8). Text nové položky je napsán na posledním řádku. Volba čísla položky je trošku problematická záležitost. Je třeba nejprve zjistit, zda zvolené číslo nekoliduje s již existujícími položkami – seznam všech položek najdeme v souboru `cfg.doc`. Pokud chceme s číslem položky pracovat v řídicí dávce prostřednictvím error-kódu, jsme navíc omezeni číslem 255. Jistou nevýhodou programu MNU tedy je, že nelze pro položky použít symbolických názvů.

Při vyvolání programu MNU na změněný soubor `cfg.mnu` se změní okénko `Others`. Především je o jeden řádek větší a v tomto řádku se nabízí položka `GS-View`. Po jejím potvrzení vrátí program MNU dávce error-kód číslo 136. Pokud jsme v řídicích dávkách neudělali žádné změny, vyvolá se program stejný, jako při volbě položky `TeXCad` (error-kód nejbližší nižší; zde č. 135). K tomu, abychom spustili jiný program, potřebujeme ještě editovat dávku `\cfg\others.bat`:

```
if errorlevel 136 goto gsview
if errorlevel 135 goto texcad
if errorlevel 134 goto bibtex
...
:gsview
  %TEXDIR%\gs -I%TEXDIR%\gslib %MAIN%.ps > gs.mes
  goto end
```

V této dávce jsme přidali skok na návěští `:gsview` (první řádek dávky) a pak jsme přidali řádky, týkající se našeho nového programu. Zde se konkrétně předpokládá, že program `gs.exe` je uložen v adresáři `%TEXDIR%`, tj. v „hlavním“ adresáři CSTEXu. Knihovní soubory pro program jsou v podadresáři `gslib` a vstupem pro program je soubor s názvem hlavního souboru a s příponou `ps`. Přesměrování textového výstupu je provedeno proto, že program jinak hloupě míchá grafický výstup se svými textovými poznámkami a je to rušivé.

V našem příkladě jsme využili toho, že v hlavní řídicí dávce `texbat.bat` je zaneseno vyvolání dávky `others.bat` v případě, že je errorkód v intervalu $\langle 131, 140\rangle$ (viz příkaz `if errorlevel 131 goto others` v dávce `texbat.bat`). V případě, že jsme zvolili jiné číslo položky, musíme většinou editovat přímo dávku `texbat.bat`.

Přidáme nové okénko

Rozhodli jsme se například usnadnit uživatelům „útěk“ z nabídkového systému přidáním položky `Quit` do nejčastěji používaného okénka `TeX`. Uživatel nyní může rovnou odejít zmáčknutím klávesy Q. Aby to ale neměl tak jednoduché, nechť se po volbě této položky zobrazí nové okénko, které nabídne dvě možnosti – buď odejít z TEXovského systému za současného smazání „nepotřebných“ souborů, (tj. `dvi`, `bak`, `aux` apod.), nebo odejít bez mazání těchto souborů. V souboru `cfg.mnu` provedeme úpravy následujícím způsobem (srovnejte s originálním obsahem souboru):

```
^31, 32, 33, 34, 34, 35, 36, 37, 38, 39, 30, 330 [3] <21, 41>; *TeX menu*
~param  {19, 7, 15, 16}
| E^dit        |
| [%FFMT%]     |
| V^iew        |
...
| C^StoCS...   |{141}
```

```
| Q^uit...     |{331}

^331 332 [330] ; **** Quit menu ****
~param  {31, 20, 23, 4}
| Q^uit plus clear dvi |{$25}
| Quit -- L^eave dvi   |{$29}
```

Na prvním řádku ukázky jsme přidali číslo 330, což bude číslo nové položky. Na druhém řádku jsme zvětšili výšku okénka `TeX` na 16 a dále jsme přidali text nové položky, viz řádek `Q^uit`. Na tomto řádku je vidět, že volba položky `Quit` neukončí systém nabídek, ale způsobí otevření okna s položkou 331, přičemž tato položka bude aktivní.

Okénko s položkou 331 jsme definovali na posledních čtyřech řádcích ukázky. Nejprve jsme napsali řádek s čísly položek: 331 a 332. Na tomto řádku je ještě v hranaté závorce číslo položky, kam se program vrátí v případě stisku klávesy `Esc`. V našem případě se jedná o položku, z níž je naše okénko vyvoláno. To má svoji logiku, ovšem dají se udělat věci, které logiku nemají.

Nové okénko obsahuje dvě položky s textem `Quit plus clear dvi` a `Quit -- Leave dvi`. První položka vrací kód 25 (viz `$25`), což zatím dávka `texbat.bat` neumí správně zpracovat a budeme se tím za chvíli zabývat. Volbou druhé položky se ukončí program MNU s kódem 29, což je kód, který odpovídá požadavku ukončení systému nabídek (viz soubor `texbat.bat`).

Poznamenejme ještě, že je velice výhodné měnit konfigurační soubor MNU jakoby rovnou ze systému nabídek voláním editoru. Po ukončení editoru okamžitě vidíme všechny spáchané změny, takže výsledný vzhled a umístění okének můžeme průběžně „ladit“. Takové úpravy musíme ale dělat v kopii souboru `cfg.mnu`, kterou obvykle najdeme na RAM disku. Právě tato kopie je totiž „v akci“. Po ukončení práce pak nesmíme zapomenout naše změny uložit do souboru na fyzickém disku, protože na RAM disku nám po vypnutí počítače nebudou příliš platné. Navíc, pokud jsme konfigurovali CSTEX v době instalace tak, že po ukončení systému nabídek se vymažou pomocné soubory z RAM disku, musíme být nanejvýš opatrní a systém nabídek neukončovat před uložením změn na disk.

Nyní ukončíme naši práci editováním souboru `texbat.bat`. Změny mohou vypadat následovně:

```
...
if errorlevel  26 goto shell
if errorlevel  25 goto clearq
if errorlevel  24 goto clear
...
:clearq
  for %%p in (%CLFILES%) do del %MAIN%.%%p
  if exist %WORK%.bak del %WORK%.bak
  goto quit
```

V této ukázce jsme zavedli mezi příkazy `if errorlevel` odskok na návěští `clearq` při kódu 25. V místě tohoto návěští je realizováno mazání souborů s příponami obsaženými v proměnné `CLFILES`. Je vhodné do inicializačního souboru `texset.bat` zařadit naplnění této proměnné třeba takto:

```
set CLFILES=dvi bak aux
```

a zařadit obsah proměnné `CLFILES` do okna `Envir`, aby si rozsah a množství mazaných souborů mohl uživatel sám nastavit. Výšku okénka `Envir` je proto nutné zvětšit o jeden řádek a přidat tam editovatelnou položku. Jak se to dělá, nebudeme znova opakovat.

Poznamenejme ještě, že chceme-li být důslední, měli bychom se zabývat i texty helpů pro naše nové položky. Například pro všechny tři nové položky 330, 331 a 332 připravíme společný text helpu a zařadíme jej někam na konec souboru `mnu.cfg`. Takový text může vypadat následovně:

```
^(330, 331, 332)
 Quit plus clear dvi
 ===================
 Po volbě položky Quit budete ještě tázáni, zda si přejete smazat
 nepotřebné soubory, či nikoli. Za nepotřebné se považují soubory
 [%MAIN%] s příponami: [%CLFILES%].
 Množinu těchto přípon lze měnit pomocí | environment proměnné |{\1163}
 CLFILES.
```

Poznamenejme ještě, že zde máme „proměnlivý“ obsah helpu v závislosti na obsahu proměnné `CLFILES` a `MAIN`. To je celkem sympatická vlastnost programu MNU. Také zde vidíte odkaz na další okno helpu s položkou 1163.

Barevná nebo černobílá nabídka

Smyslem tohoto článku není opakovat to, co bylo řečeno v dokumentaci k programu MNU nebo co je předvedeno ve stávající konfiguraci CSTEXu. Více bych se chtěl zaměřit na triky, které možná nejsou při povrchním přečtení dokumentace a příručky DOSu zcela patrné. Tento článek bych přirovnal ke kapitole „Dirty Tricks“ v *TEXbook*u. Jen jsem opomněl vložit na toto místo příslušné množství „dangerous bendů“.

Zajímavým příkladem může být vlastnost nabídky „dynamicky“ měnit svůj vzhled v závislosti na provedené volbě. Třeba položka `TeX` v okně `TeX` má vzhled závislý na zrovna zvoleném formátu (tj. `PlainTeX`, `LaTeX` apod.). Je to zařízeno tak, že v místě textu položky je požadavek substituce textu ve tvaru `[%...]`. V tomto případě se text nahrazuje hodnotou proměnné DOSu.

Další možností je nabídnout programu MNU různé konfigurační soubory. Tuto možnost si předvedeme podrobněji. V souboru `cfg.zip` na první instalační disketě najdete dva skoro stejné konfigurační soubory. Jeden se jmenuje `cfg.mnu` a konfiguruje barevnou nabídku pro CSTEX, zatímco druhý se jmenuje `cfgmono.mnu` a konfiguruje černobílou (ale jinak stejnou) nabídku. V instalačním programu je otázka, zda se má nabídka chovat barevně nebo černobíle, a na základě toho se jeden z těchto dvou souborů použije jako definitivní.

Zkusme použít oba soubory naráz. Přidejme do prvního souboru na nějaké vhodné místo položku `Mono` ve tvaru

```
| Mono  |{"set CFGMNU=cfgmono.mnu">2}
```

V řídicí dávce nahraďme ve volání programu MNU název konfiguračního souboru hodnotou proměnné `CFGMNU` takto:

```
%RAM%\mnu %RAM%\%CFGMNU% %RAM%\envir.bat %RAM%\dos.bat %RAM%\mfbat.bat
call %RAM%\envir
```

a postarejme se o to, aby po volbě položky `Mono` dávka pouze načetla soubor `envir.bat` a dále se vrátila na volání menu. Pokud v souboru `cfgmono.mnu` vytvoříme analogickou položku s názvem `Color`, můžeme celý zázrak vyzkoušet. Nesmíme ale zapomenout v souboru `texset.bat` definovat implicitní hodnotu proměnné `CFGMNU` například jako `cfg.mnu`.

Po startu nám naskočí barevná nabídka obsahující položku `Mono`. Po volbě této položky se ihned celá nabídka stane pouze černobílou a bude

obsahovat položku `Color`. Po volbě této položky zase dostane nabídka barvu.

Tento příklad s barvami není příliš efektivní. Potřebujeme mít totiž v akci dva skoro stejné soubory. Navíc, pokud bychom měnili strukturu nabídek, musíme tak učinit dvojmo. Asi lepší řešení je použít pouze v místech, kde se zmíněné dva soubory liší, substituční konstrukce typu `[%...]`, přitom zde bude asi vhodné substituovat krátkými soubory.

Podobně můžeme střídat třeba jazyky. Vytvořme si soubor s názvem `cfgcz.mnu`, který bude překladem souboru `cfg.mnu` do češtiny. Zařadíme si na vhodná místa položky `Anglicky` a `Czech` a můžeme se bavit tím, jak bude uživatel zmaten, když se texty všech položek jedním rázem změní.

Uživatelské konfigurační soubory trochu jinak

Zaměřme se na problém uživatelských konfiguračních souborů, které mají v $\mathcal{CS}$TEXu název `texcfg.bat`. Příkazy uvedené v takovém souboru se provedou na konci inicializační dávky `texset.bat`. Například příkazy typu `set` v konfiguračním souboru mají vyšší prioritu než tytéž příkazy `set` nastavující „společnou konfiguraci“ v inicializační dávce. Současný návrh systému nabídek umožňuje sice automaticky vytvořit soubor `texcfg.bat`, ale ukládají se do něj *všechny* hodnoty proměnných DOSu. To má plno nepříjemných důsledků. Například pokud uživatel zkopíruje obsah svého adresáře do jiného počítače (včetně souboru `texcfg.bat`), pak se může stát, že bude mít nepředstavitelné problémy. V tomto jiném počítači může například být jinak nastavena hodnota `%COMSPEC%`, nebo třeba vlastní TEX může být v jiném adresáři, takže se liší hodnota proměnné `TEXDIR`. Také se v souboru `texcfg.bat` ukládá hodnota proměnné `MAIN` a `WORK` a uživatel se pak může strašně divit, že hodnoty těchto proměnných přestávají být závislé na obsahu příkazového řádku, kterým je nabídkový systém vyvolán.

Pokusíme se předvést možnost, jak automaticky vytvářet uživatelské konfigurační soubory, které obsahují nastavení *pouze některých* zvolených proměnných DOSu. Chtěl bych poznamenat, že při vývoji nabídkového systému pro $\mathcal{CS}$TEX jsem měl toto řešení původně navrženo, ale pak jsem od něj ustoupil, protože tyto „statické“ soubory `texcfg.bat` samozřejmě nebudou obsahovat důležité hodnoty proměnných DOSu, které souvisejí s novými programy nezařazenými do původního balíku.

Podívejme se nejprve, jakým způsobem se soubor `texcfg.bat` vytváří ve stávající instalaci CSTEXu. Při volbě položky `Save` se v dávce `texbat.bat` provedou následující příkazy:

```
:save
  set > %RAM%\file.mnu
  echo echo **** CONFIGURATION from file %MARK%.bat **** > %MARK%.bat
  %TEXDIR%\dupcent %RAM%\file.mnu set >> %MARK%.bat
  echo set MNU=31 >> %MARK%.bat
  del %RAM%\file.mnu
  set mnu=65
  goto menu
```

Zkusme si to vysvětlit. Nejprve se uloží výpis všech proměnných DOSu ve formátu `NAZEV=hodnota` do pomocného souboru `file.mnu`. Pak se začne vytvářet vlastní soubor `%MARK%.bat`. První řádek tohoto souboru bude obsahovat text

```
echo **** CONFIGURATION from file texcfg.bat ****
```

a další řádky budou obsahovat postupně všechny použité proměnné DOSu ve formátu `set NAZEV=hodnota`. Je zde použit program `dupcent.exe`, který čte pracovní soubor `file.mnu` a před každý řádek přidá slovo `set` a navíc duplikuje případná procenta uvnitř hodnot proměnných DOSu. Poslední řádek v souboru `%MARK%.bat` bude obsahovat příkaz `set MNU=31`, tj. systém nabídek po zpětném načtení tohoto souboru naskočí na položku s číslem 31, což je položka `Edit`.

Pokud změníme výše uvedené příkazy v dávce `texbat.bat` následujícím způsobem, dostáváme „statické“ konfigurační soubory:

```
:save
  %RAM%\mnu %TEXDIR%\cfg\maskcfg.mnu > %RAM%\file.mnu
  %TEXDIR%\dupcent %RAM%\file.mnu > %MARK%.bat
  del %RAM%\file.mnu
  set mnu=65
  goto menu
```

Program MNU je zde použit jako filtr, který čte soubor `maskcfg.mnu` a na základě něj vytváří pracovní soubor `file.mnu`. V tomto pracovním souboru ještě zdvojíme procenta programem `dupcent.exe` (tentokrát nepřidáváme před řádky slovo `set`) a výsledek uložíme do souboru s názvem `%MARK%.bat`. Obsah výchozího souboru `maskcfg.mnu` může být třeba následující:

```
***** Mask for configuration files *.bat ****
~copy
echo **** CONFIGURATION from file [%MARK%].bat ****
set EDIT=[%EDIT%]
set EMTEXED=[%EMTEXED%]
set SHELL=[%SHELL%]
...
set MFINPUT=[%MFINPUT%]
```

Příkaz `~copy` nastavuje program MNU do „kopírovacího režimu“ (viz dokumentace `mnu.tex`, str. 6 dole), tj. program lze použít jako filtr, přičemž se aplikuje náhrada textu podle pravidel pro konfigurační soubory programu MNU. Výstupem z tohoto filtru bude soubor třeba s tímto obsahem (to samozřejmě závisí na aktuálním stavu proměnných DOSu):

```
echo **** CONFIGURATION from file texcfg.bat ****
set EDIT=call e:\textmp\qedit
set EMTEXED=call e:\textmp\qedit %2 %3 -n%1
set SHELL=c:\nc\nc
...
set MFINPUT=c:\emtex\mfinput;.
```

Program `dupcent` ještě zdvojí procenta, takže ve výsledném souboru `texcfg.bat` máme např. řádek:

```
set EMTEXED=call e:\textmp\qedit %%2 %%3 -n%%1
```

Pokud se někdy zpětně vyvolá soubor `texcfg.bat`, pak se zdvojená procenta interpretují jako jedno „obyčejné“ procento a vše bude pracovat tak, jak chceme.

Je tedy vidět, že obsah uživatelských konfiguračních souborů typu `texcfg.bat` můžeme mít zcela pod kontrolou. Tento obsah určíme pomocí souborů typu `maskcfg.mnu`. Můžeme mít samozřejmě takových „maskových souborů“ více; například roztřídíme proměnné DOSu do tematických okruhů. Nabídku týkající se možnosti uložení hodnot proměnných do souboru pak musíme příslušným způsobem rozšířit. Uživatel si může vybrat, zda uloží třeba stav proměnných týkajících se editoru a shellu, nebo stav proměnných týkajících se třeba TEXu, ovladačů, nebo názvů pracovního a hlavního souboru.

Staré menu pomocí nového MNU

Následující příklad má dokumentovat flexibilitu programu MNU. Ukážeme, že starý známý systém nabídek z předchozích verzí CSTEXu lze pomocí programu MNU snadno simulovat. Užitečnost tohoto příkladu se může zdát sporná. Je třeba si ale uvědomit, že někdy je velmi nebezpečné dát lidem k dispozici jiný systém nabídek, třebaže má více možností. Už jiný vzhled systému nabídek na obrazovce dokáže mnohé uživatele odradit. Lidé se začnou rozčilovat, že se musí učit *zase* něco jiného. Proto například O. Ulrych raději poněkud vylepšil staré menu, než aby uživatelům na svém pracovišti nabídl nový systém nabídek z CSTEXu. Chápu ho – uživatelé jsou totiž velmi konzervativní.

Pokud si vzpomínám, tak staré menu nabízelo vpravo od názvu položky jakési editační pole, kde se psaly parametry pro jednotlivé programy. Myslím si, že tomu nejvíce odpovídá následující konfigurační soubor (označme ho např. `oldmenu.mnu`).

```
******* Staré menu pomocí MNU *******
~start {!>2}
~final {"set MNU=#">2}
^1 2 3 4 5 6 7 8 9
~param {30, 1, 50, 15, 1, 7, 7, 7, 112, 112}
                CSTeX - menu
================================================
▷ Edit   ◁ |#[%EDITPAR%37]|{"set EDITPAR=#">%2}
▷ TeX    ◁ |#[%TEXPAR%37]|{"set TEXPAR=#">%2}
▷ View   ◁ |#[%VIEWPAR%37]|{"set VIEWPAR=#">%2}
================================================
▷ Vlnka  ◁ |#[%TIEPAR%37]|{"set TIEPAR=#">%2}
▷ Spell  ◁ |#[%SPELLPAR%37]|{"set SPELLPAR=#">%2}
▷ Matrix ◁ |#[%MATRPAR%37]|{"set MATRPAR=#">%2}
▷ Laser  ◁ |#[%LASERPAR%37]|{"set LASERPAR=#">%2}
================================================
▷ DOS    ◁ |#[%DOSPAR%37]|{"set DOSPAR=#">%2}
▷ Quit   ◁ |#[%QUITPAR%37]|{"set QUITPAR=#">%2}
```

Symboly ▷ a ◁ zde označují znaky s kódem 16 a 17, což se na obrazovce jeví jako trojúhelníčky. Menu vyvoláme například dávkou `oldtex.bat`, která obsahuje tyto příkazy:

```
@echo off
if '%1==' goto err
%COMSPEC% /e:1300 /c oldmenu %1
goto end
:err
echo Chybí název souboru (bez přípony) jako parametr.
:end
```

Takovou věc asi znáte z CSTEXu. Rozšiřuje se zde prostor pro proměnné DOSu a vyvolá se další dávka, zde nazvaná `oldmenu.bat`. Tato dávka vypadá následovně (místo teček si doplňte analogicky podobné příkazy pro další položky):

```
@echo off
set EDITPAR=%1.tex
set TEXPAR=%1.tex
set VIEWPAR=@scr.cnf %1.dvi
...
set QUITPAR=echo ahoj!
set MNU=1

:menu
mnu oldmenu.mnu envir.bat
call envir
if errorlevel 9 goto quit
if errorlevel 8 goto dos
...
if errorlevel 1 goto edit
pause > nul
goto menu

:edit
  call edit %EDITPAR%
  goto menu
  set MNU=2
:tex
  ...
:dos
  %DOSPAR%
  goto menu
:quit
  %QUITPAR%
```

Poznamenejme, že zde nejsou řešeny zdaleka všechny problémy související s TEXem. Například zde nenajdete nastavení některých proměnných DOSu, které jsou typické pro emTEX. Berte to proto jako příklad

konfigurace nabídky. Pokud ji budete chtít skutečně použít, bude potřeba ji v některých ohledech upravit.

Nyní si shrneme, jak se takto konfigurovaný systém nabídek chová. Vidíme, že všechny položky v menu mají typ editovatelné položky. Pokud se zmáčkne `Enter` (nezáleží na tom, zda se editovalo či nikoli), potvrdí se hodnota položky (tj. parametry volaného programu) a navíc se požadovaný program vyvolá. Položku volíme šipkami a jakmile začneme psát, ukládá se text v editačním poli do parametrů programu. Klávesa `Esc` má dva významy. Pokud se už začalo editovat, pak vrací editační pole do původního stavu, jinak skrývá celé okénko s nabídkou. V tomto případě totiž program MNU ukončí činnost s kódem 0 a v dávce `oldmenu.bat` se provede příkaz `pause > nul`, tj., čeká se na libovolnou klávesu, aby se proces mohl vrátit zpět do nabídky. Myslím, že tak nějak to v tom starém menu fungovalo.

Nevýhoda takovéto konstrukce nabídek je v tom, že nám nedovoluje používat zvýrazněná písmena v textu položek pro rychlý výběr položky. Jakmile totiž zmáčkneme písmeno, nabídkový systém to interpretuje jako požadavek na změnu parametrů programu a odstartuje editaci v editačním poli. Nevzpomínám si, že by ve starém menu šlo volit položku rovnou písmenem, takže se domnívám, že to pracuje přesně tak, jak byli uživatelé starého menu zvyklí. Jednu věc programem MNU simulovat nelze. Jde o to, že při startu editace se celé editační pole zaplnilo jakýmisi nesmyslnými znaky vypadajícími přibližně takto: ≪. Tato skutečnost ale spíše rušila a mátla, takže z toho, že to nelze simulovat, nemusíme být smutní.

Mezi znaky, které by se neměly objevit v textu parametrů, patří `|`, `=`, `<` a `>`. Tyto znaky totiž mají v DOSu speciální význam a z těchto důvodů jimi nelze naplnit proměnnou DOSu. Tuto věc můžeme obejít použitím pomocných souborů. Stačí se inspirovat řešením položky DOS v novém systému nabídek CSTEXu.

Namítnete ještě, že čáry oddělující jednotlivá pole v původním programu pěkně navazovaly na vnější rámeček. Pokud chcete, můžete toho pomocí MNU snadno dosáhnout. Stačí volit v příkazu `~param` velikost rámečku rovnou nule a rámeček si do konfiguračního souboru `oldmenu.mnu` zanést explicitně pomocí semigrafických symbolů včetně všech spojujících mezičar. Pokusíte-li se ale o spojení dvojité čáry s jednoduchou, nebude vám to fungovat v kódování PC Latin2, protože tam požadované semigrafické znaky nejsou.

Od jednoduchých formulářů k databázím

Uděláme si „uživatelské rozhraní“ pomocí programu MNU, které umožní snadno vkládat údaje, které je třeba často vyplňovat do stanoveného formuláře. Vlastní formulář nakonec vysází TEX. Uživatel přitom vůbec nemusí vědět, že používá TEX.

Nechť je celá procedura přípravy formuláře vyvolána z DOSu dávkou `form.bat`. Ta bude obsahovat pouze řádek, kterým se zvětší prostor pro proměnné DOSu a vyvolá se řídicí dávka (podobně, jako v dávce `oldtex.bat` z předchozího odstavce). Řídicí dávka se může jmenovat třeba `formbat.bat` a její obsah je následující:

```
@echo off
keybcs
set JMENO=Ferdinand
set PRIJMENI=Mravenec
set ULICE=Ondřeje Sekory 13
set POVOLANI=Práce všeho druhu
:menu
  set MNU=1
  mnu form.mnu envir.bat
  if errorlevel 1 goto quit
  call envir
  mnu masktex.mnu > form.tex
  call texrun form.tex
  call view form.dvi
  set MNU=10
  mnu form.mnu envir.bat
  if errorlevel 2 goto menu
  if errorlevel 1 goto quit
  call print form.dvi
:quit
  del envir.bat
  keybcs -u
```

Popišme si její funkci. Nejprve se vyvolá ovladač pro českou (slovenskou) klávesnici. Předpokládáme, že ovladač obrazovky je řešen globálně (např. v `autoexec`u). Pak se naplní proměnné DOSu, které odpovídají jednotlivým položkám formuláře, jistými „výchozími hodnotami“. Pak se poprvé spustí program MNU s výchozí položkou č. 1. V tomto místě provede uživatel případné změny výchozích hodnot proměnných, které se pak aktualizují voláním přechodné dávky `envir.bat`. Poté se pomocí programu MNU jako filtru vytvoří zdrojový soubor TEXu s názvem `form.tex`. Výchozí soubor `masktex.mnu` může vypadat třeba takto:

```
~copy
\input makra
\JMENO    {[%JMENO%]}
\PRIJMENI {[%PRIJMENI%]}
\ULICE    {[%ULICE%]}
\POVOLANI {[%POVOLANI%]}
\makeform
```

Pokud například uživatel změnil pouze první dva údaje, pak po provedení příkazu `mnu masktex.mnu > form.tex` dostáváme soubor `form.tex` třeba ve tvaru:

```
\input makra
\JMENO    {Petr}
\PRIJMENI {Olšák}
\ULICE    {Ondřeje Sekory 13}
\POVOLANI {Práce všeho druhu}
\makeform
```

Jak vypadá soubor `makra.tex` zde nebudeme uvádět, protože to závisí na vlastním formuláři a na tom, v jaké podobě ho chceme vytisknout. Pokračujme v trasování dávky `formbat.bat`. Po vytvoření souboru `form.tex` se tento soubor zpracuje TEXem a hned se spustí prohlížeč. Po ukončení prohlížeče se program MNU zeptá uživatele, zda to chce vytisknout nebo opravit (položka 10). Další věci v dávce `formbat.bat` jsou už zřejmé.

Nakonec si uvedeme, jak vypadá soubor `form.mnu`, v němž konfigurujeme vlastní uživatelské rozhraní. Může mít třeba tento obsah:

```
******* Uživatelské rozhraní "formulář" pomocí programu MNU *******
~start {!>2}
^1 2 3 4 5 6
~param {5, 2, 72, 13}
               Formulář pro Velkého Byrokrata
               ==============================
 Následující systém nabídek Vám umožní formulář vyplnit a~vytisknout
------------------------------------------------------------------------
 Jméno:    |#[%JMENO%64]|{   "set JMENO=#">%2,   "#">JMENO,   2}
 Příjmení: |#[%PRIJMENI%64]|{"set PRIJMENI=#">%2,"#">PRIJMENI,3}
 Ulice:    |#[%ULICE%64]|{   "set ULICE=#">%2,   "#">ULICE,   4}
 Povolání: |#[%POVOLANI%64]|{"set POVOLANI=#">%2,"#">POVOLANI,5}

              | OK    |{$0} (Start zpracování)
              | Konec |{$1} (Nechci žádný formulář)
```

```
^10
~param {5, 15, 72, 5}
 | Vytisknout |{$0} (Formulář se vytiskne na tiskárně, připravte ji)
 | Vrátit     |{$2} (Návrat do systému nabídek, provedu opravy)
 | Konec      |{$1} (Rozmyslel jsem si to, nechci žádný formulář)

^0
~param {20, 20, 42, 4}
 | Konec |{$1} (Nechci žádný formulář)
 | Dále  |{\[%MNU%]} (Pokračuji)
```

Zde samozřejmě záleží na naši tvořivosti a fantazii, ale také na obsahu skutečného formuláře. Vše můžeme rozvést do více okének a přidat systém nápověd (helpů). Uvedená ukázka je pokud možno co nejjednodušší. Všimneme si některých triků. Za prvé v jediném souboru je řešen vzhled nabídky pro obě volání programu mnu (jak s výchozí položkou 1, tak s položkou 10). Také je zde konfigurována reakce na klávesu `Esc` (položka 0), přitom položka `Dále` způsobí skok na položku 1 nebo 10 v závislosti na současném obsahu proměnné MNU.

Program MNU neumožní editovat položku většího rozsahu, než je jeden řádek. Pokud potřebujeme rozvést některou položku do více řádků, musíme ji buď rozdělit do samostatných řádků, nebo vyvolat na vyplnění takové položky externí editor. První případ použijeme jen výjimečně, protože zde narazíme na omezené editovací možnosti řádkového editoru vestavěného do programu MNU. Stručně řečeno, řádkový editor aplikovaný opakovaně na řádky umístěné pod sebou zdaleka nevytvoří komfort celoobrazovkového editování. Je skutečně lepší pro vyplnění obsáhlých položek vyvolat externí editor. V prvních několika řádcích „předzpracovaného textu“ může být komentář, který například říká, kam se má text napsat a jak se z editoru odchází. Do souboru `form.tex` pak můžeme zařadit vytvořený text buď TEXovským `\input`, nebo jej zařadíme fyzicky v souboru `masktex.mnu` pomocí substituční konstrukce. Program MNU totiž umí expandovat do daného místa obsah jiného souboru.

Nedávno jsem měl možnost vyzkoušet program W94, který vytvořil ve výpočetním centru ČVUT pan ing. Strejc. Program má usnadnit uživatelům vytvoření LATEXovského souboru, který obsahuje příspěvek do sborníku. Autor programu totiž každoročně formátuje sborník výsledků odborné činnosti na naší škole pomocí LATEXu. Nejvíce práce přitom má s vytvořením hlaviček k příspěvkům tak, aby to bylo podřízeno jednotnému formátu sborníku. Navíc mu příspěvky docházejí v rozličných

podobách (napsané v různých programech typu `word-`*cokoli* apod.). Rozhodl se proto distribuovat všem přispěvatelům prográmek W94 společně se stylem pro sborník a značně zúženým a jednoúčelově upraveným balíkem TEXu. Prográmek přitom řeší naprosto analogickou věc, jaká byla předvedena zde. Vytváří uživatelsky přítulné rozhraní mezi autorem příspěvku a vlastním LATEXovým souborem. Nejprve uživatel vloží prostřednictvím okének text názvu příspěvku, jména autora apod. Jedná se vlastně o strukturu hlavičky příspěvku. Potom se vyvolá editor a je třeba napsat vlastní tělo příspěvku. Vše je doprovázeno velkým množstvím příkladů. Z předchozí ukázky vidíme, že by mohl být systém W94 vytvořen pomocí programu MNU.

Komplikovanější je případ, kdy potřebujeme zpracovávat více formulářů současně. Každý formulář je vlastně jeden údaj v jakési databázi. Je nesmyslné, aby se všechny formuláře zpracovávaly naráz v proměnných DOSu – bude nám stačit, když tam budeme mít obsah všech položek jednoho formuláře. V takovém případě už nevystačíme jen s možnostmi programu MNU a DOSu, ale budeme muset vytvořit jednoduché podpůrné prográmky, které umožní například vybrat z databáze jeden formulář a hodnoty jeho položek uložit do proměnných DOSu.

Z mého laického pohledu (s databázemi nemám nic společného) zabezpečuje databázový systém tři druhy služeb. Za prvé vytváří přítulné rozhraní mezi uživatelem, který vkládá nové údaje, a vlastním databázovým souborem. To se dá vesměs řešit programem MNU. Za druhé umožní tisk údajů v nejrůznějších formátech. V tom je zase nepřekonatelný TEX nejen možností definovat jakékoli výstupní formáty pomocí maker, ale samozřejmě též kvalitou tisku. Třetí funkcí databázových systémů jsou nejrůznější třídící a vyhledávací algoritmy, pracující nad databázovým souborem. Aby byly tyto algoritmy dostatečně výkonné, je většinou formát databázového souboru netriviální. Pokud ovšem pracujeme jen s malými databázemi, postačí nám databázový soubor v obyčejném textovém formátu a můžeme si naprogramovat jednoduché vyhledávací prográmky (v Céčku se jedná o pár desítek řádků). Můžeme si tak vytvořit „na míru“ databázový systém za pomocí programu MNU a TEXu. Tento systém sice pracuje pomalu a jen s malými a ne příliš složitými databázemi, zato komfort vkládání údajů je dostatečný a výstup na tiskárnu je bezkonkurenční. Navíc je to celé zadarmo.

Příklad tohoto typu dnes nebudu uvádět. Chystám se ale v nejbližší době řešit problém vyplňování školních vysvědčení za pomocí TEXu. Přitom se využije buď výstup z databázového systému, který má škola už

zakoupený, nebo se pro vkládání údajů použije program MNU. Pokud při řešení tohoto úkolu narazím na nějaké zajímavosti, zmíním se o tom někdy příště na stránkách tohoto časopisu.

Okno pro přihlášení uživatele do sítě

Na naší katedře používáme pro připojení do lokální sítě systém PC NFS, který umožňuje přihlášení do sítě pouze z řádky DOSu (příkazem `net init` nebo `net login`. Pomocí programu MNU jsem zvýšil bezpečnost sítě i uživatelský komfort při přihlašování. V této ukázce je plno „špinavých triků“. Kdo nemá takové postupy rád, nechť tento odstavec přeskočí. Také je to ukázka, která nemá nic společného s TEXem. Má pouze dokumentovat další možné využití programu, který se čtenářům dává k dispozici jako součást CSTEXu.

Začneme konfiguračním souborem pro program MNU. Jeho název je `log.mnu` a zde je jeho obsah:

```
******* Login promtp *******
~timeout {20, *"scrblank">0}
~mouse {65535, 0}
^1
~param {5, 2, 72, 16, 1, 112, 30, 30, 14, 14}
     Přihlášení do lokální sítě na katedře matematiky FEL ČVUT
     ==========================================================

 Uživatelské jméno (user): |#        |{   "#">USER, 2}

 Heslo (password):

    Po přihlášení do sítě máte k dispozici lokální disky:
         K: ... s uživatelskými soubory
         M: ... s chráněným softwarem
    a na síť připojené tiskárny:
         LPT1 ... jehličková (Epson)
         LPT2 ... laserová
^2 [1]
~param {26, 8, 20, 1, 0, 0, 0, 0, 0}
|#                   |{*"trylogin [%USER%] #">0}

^10 [1]
~param {20, 14, 40, 7, 2, 78} " !! CHYBA !! " ;

          Chybné přihlášení.
     Údaje nebyly vloženy správně.
           Zmáčkněte  Esc.
```

```
^0 [1]
~param {30, 14, 21, 3, 2, 78} " ?? Únik ?? "
   Zmáčkněte Esc
```

Celá procedura se vyvolá dávkou `login.bat` s tímto obsahem:

```
@echo off
call initnet
set MNU=1
:znova
mnu log.mnu
set MNU=10
if not exist OK.bat goto znova
call OK
del OK.bat
```

Po uvedení síťových programů do provozu (dávka `initnet.bat`) naskočí program MNU na položku s číslem 1. Zde uživatel edituje své jméno. Po zmáčknutí `Enter` se objeví v místě pro napsání hesla černé okénko o výšce jednoho řádku, obsahující „neviditelnou“ editovatelnou položku (s číslem 2). Její neviditelnost je dána tím, že má barevné atributy rovny nule, tj. černý text na černém pozadí. Tato položka musí být popsána v samostatném okně, protože barevné atributy pro položky jsou společné pro celé okno; přitom my chceme text položky `user` vidět, zatímco `password` nikoli.

Toto řešení sice nezobrazuje v místě hesla vůbec nic (tedy ani žádné hvězdičky či čtverečky), ale myslím si, že to je spíš k dobru věci – zvyšuje to bezpečnost uživatelských hesel.

Po zadání hesla se vyvolá *zevnitř* (z programu MNU) dávka `trylogin.bat`. Tato dávka obdrží dva parametry. Jedním je uživatelské jméno a druhým je heslo. Dávka zavolá přihlašovací program sítě s těmito dvěma parametry a vytvoří soubor `OK.bat` v případě, že bylo přihlášení úspěšné. Proč voláme dávku zevnitř programu MNU? Je to proto, že informaci o obsahu hesla nejsme schopni předat řídicí dávce jinak, než v souborech typu `envir.bat`. Potulování hesel po souborech (třebaže smazaných) by ale mohlo oslabit jejich bezpečnost.

Zpětnou informaci o úspěšnosti přihlášení zase nelze přenést jinak, než existencí souboru. Navíc nyní musíme program MNU opustit, protože v programu samotném nemůžeme zařídit větvení výpočtu závislé

na existenci souboru.[4]) V hlavní dávce se v případě, že neexistuje soubor `OK.bat` přejde znova na volání programu MNU, nyní ale má výchozí položka číslo 10. Na této položce nám program MNU vynadá a umožní po klávese `Esc` přihlašovací proceduru opakovat.

Pokud je přihlášení úspěšné, řídicí proces se ukončí vyvoláním dávky `OK.bat`, která obsahuje jediný řádek ve tvaru např. `set USER=vopicka`. Po ukončení přihlašovací procedury máme tedy v proměnné `USER` podchycené uživatelské jméno. To je užitečné pro další dávky, které mohou s touto hodnotou pracovat. Samotný PC NFS software nám takovou věc neumožní.

Pro zajímavost ještě uveďme, jak vypadá dávka `trylogin.bat`, která realizuje vlastní přihlášení. Zde je její obsah:

```
@echo off
if exist OK.bat del OK.bat
if `%2==` exit
if `%1==` exit
net init %1 %2 < letter.a
if errorlevel 1 exit
echo set USER=%1 > OK.bat
```

Zde je použit ještě jeden nehezký trik. Program `net init` při neúspěšném přihlášení vydá zprávu typu `Abort, Retry, Ignore?`, přičemž my nechceme, aby uživatel měl možnost na tuto zprávu reagovat. Volbou `Ignore` by totiž mohl z přihlašování vyklouznout. Soubor `letter.a`, na který je přesměrován vstup, obsahuje písmeno `a`. Program `net init` tedy při neúspěšném přihlášení reaguje, jako by bylo řečeno `Abort`, a v takovém případě vrátí nenulový error kód, což my v zápětí testujeme.

Celý systém dávek je navržen tak, aby uživatel neměl šanci přihlášení nijak obejít. Při té příležitosti podotkněme, že program MNU je odolný proti kombinacím typu `Ctrl-Break` apod.

V konfiguračním souboru jsou dvě zajímavé globální definice. První z nich (`~timeout`) říká, že po dvaceti vteřinách klidu klávesnice se vyvolá program `scrblank`, který zhasíná obrazovku. Takový program udělá šikovný programátor za pár minut. Nemáte-li program tohoto typu po ruce, může posloužit DOSovské `cls`, ale toto řešení neskrývá kurzor. Šikovný programátor se přitom může vyřádit a udělat z obrazovky

[4]) Není to tak docela pravda; lze využít konstrukce typu `[%...]` například v definičním řádku okna a tím ovlivnit chování programu v závislosti na obsahu souboru. Není to ale příliš elegantní.

třeba „toulky vesmírem“ nebo „akvárium“ apod. Velice jednoduchý program `scrblank` jsem zařadil do CSTEXu 94 (jedná se o pár řádků v jazyce C).

Druhou definicí, na niž bych chtěl upozornit, je `~mouse`. Parametry kurzoru myši jsou nastaveny tak, aby tento kurzor nebyl vidět. Přesněji, maska AND propustí všechno (hodnota FFFFh) a maska XOR nezmění nic (hodnota 0000h). To je důležité, protože například při implicitních hodnotách kurzoru myši by šlo pohybem kurzoru v poli pro heslo demaskovat postupně jednotlivá písmena hesla.

Nakonec ještě poznamenejme, že lokalizace některých souborů (například `OK.bat`) je ve skutečných dávkách provedena samozřejmě jinde než v aktuálním adresáři. Pro stručnost jsem tyto věci nevypisoval. Pro vyšší bezpečnost proti virům a vlezlým uživatelům je navíc vše, co se týká konfigurace tohoto přihlášení, umístěno na takové části lokálního disku, která je zabezpečena proti zápisu.

Závěrem

Tímto článkem jsem chtěl naznačit, že lze program MNU použít při rozličných příležitostech. Zdaleka jsem nevyčerpal všechny možnosti programu. Pokud vás článek zaujal, můžete si přímo jednotlivé ukázky vyzkoušet. Všechny byly testovány.

Připravuji se provést nějaké úpravy v CSTEXu, týkající se specifik naší lokální sítě. Právě proto, že vše stojí na řídicích dávkách DOSu, budou se takové změny dělat celkem snadno. Do všeho je totiž vidět; nic není skryto před všetečnými zásahy tzv. „MNU-inženýrů“, kterými se můžete stát i vy. Jediné, co je skryto, je zdrojový text programu MNU (asi 1 600 řádků v jazyce C). Tento text nedám z ruky. Tím zabráním možnému šíření různých mutací se „zaručeně nejlepšími“ vylepšeními.

Ještě poslední příklad použití programu MNU. Asi před rokem jsem zařadil volání tohoto programu do `autoexec`u počítače naší paní sekretářky. Musel jsem samozřejmě pamatovat na všechny operace, které jsou na tomto počítači prováděny – od volání jednotlivých programových celků až po archivaci souborů na diskety. Tím se mi podařilo zcela izolovat uživatelku tohoto počítače od DOSovského promptu. Systém už nějakou dobu funguje a paní uživatelka si nestěžuje.

Na závěr bych chtěl popřát mnoho zdaru a trpělivosti všem odvážlivcům, kteří chtějí můj program potrápit nějakým netriviálním způsobem. Bohužel, jazyk konfiguračních souborů je dosti kryptický a mož-

nosti DOSu a programu MNU jsou dost omezené. I za těchto omezení lze ale vytvořit neuvěřitelné věci.

Pokud někdo z vás přijde při konfiguraci TEXu na zajímavé použití programu MNU, velice mě potěší, když si o tom přečtu na stránkách tohoto časopisu.

Literatura

Kernighan, B. W. – Ritchie, D. M.: *The C Programming Language*, Englewood Cliffs, Prentice-Hall 1978.

Knuth D. E.: *The Art of Computer Programming*, vol. I–III, Addison Wesley, 1973, 1981

Knuth D. E.: *The TEXbook*, vol. A of *Computers & typesetting*, Addison Wesley, 1986

Olšák P.: *Program MNU, Konfigurovatelné menu pro spouštění aplikací pod DOSem* (`mnu.tex` – dokumentace zahrnutá do CSTEXu), 1993

`cfg.doc`, `install.doc` – dokumentační soubory k příkladům použití programu MNU v CSTEXu.

MS DOS: Dokumentace k systému.

EuroTEX'94

Letošní EuroTEX se bude konat

od 26. do 30. září 1994

na severu sousedního Polska: v Sobieszewu na tichém ostrově u gdańského pobřeží.

Náklady na týdenní pobyt by neměly přesáhnout 260 $ pro dvě osoby (při ubytování v dvoulůžkovém pokoji). Naše sdružení bude samozřejmě podporovat účast svých členů na konferenci — předběžně jsme se na valné hromadě konané v sobotu 26. května 1994 dohodli na poloviční úhradě veškerých nákladů včetně cesty. K tomu je potřeba oznámit zamýšlenou účast do 26. srpna na adresu CSTUGu. Zde je rovněž možno obdržet přihlášku, jinak se můžete obrátit přímo na organizátory na elektronické adrese `eurotex@halina.univ.gda.pl`. Zatím není

znám přesný program konference ani následných kursů (předpokládají se kursy o knižní úpravě [hádal bych, že pod vedením Phila Taylora] a o LaTeXu 2_ϵ), ale všechny zájemce, kteří se nám ozvou, budeme průběžně informovat.

Karel Horák

Z obsahu příštího čísla

Petr Sojka: Virtuální fonty, accents a přátelé
Tomáš Mráz: PostScriptová písma v TeXu
Zdeněk Wagner: Záludnosti PostScriptu
Jaromír Kuben: Kreslení obrázků v TeXu pomocí `mujmfpic`

Vydalo:	Československé sdružení uživatelů TeXu vlastním nákladem jako interní publikaci
Obálka:	Bohumil Bednář
Počet výtisků:	600
Tisk a distribuce:	KOPP, Máchova 16, 370 01 České Budějovice
Adresa:	CSTUG, MÚ UK, Sokolovská 83, 186 00 Praha 8
Korespondence:	CSTUG, MÚ AV ČR, Žitná 25, 115 67 Praha 1

Podávání novinových zásilek povoleno
Ředitelstvím pošt Praha č.j. NP 320/1994 ze dne 10.2. 1994