IPC w TEXu

Piotr Bolek
P.Bolek@gust.org.pl

Streszczenie

Tradycyjny sposób komunikacji TEXa ze światem zewnętrznym sprowadza się do możliwości prostego wczytywania informacji z plików tekstowych `\input` i `\read`, zapisywania danych do plików `\write`, operacji zapisywania efektów przetwarzania w pliku `.dvi` oraz rejestru przebiegu przetwarzania w pliku `.log`. Dostępne są rozszerzenia (`-shell-escape`) oraz możliwość zapisywania logów, danych `dvi` i `\write` do plików FIFO. Tekst jest próbą prezentacji pomysłów na wykorzystanie komunikacji międzyprocesowej w TEXu oraz związanych z tym ograniczeń i niebezpieczeństw.

Wprowadzenie

TEX został wymyślony ćwierć wieku temu. Od tego czasu jest używany w zasadniczo niezmienionej postaci. Oczywiście z upływem czasu TEX musiał dostosować się do rozwijającego się dookoła niego świata. Kolory, grafika, fonty wektorowe, PostScript, PDF – wszystko to jest dostępne. Ale TEX wciąż jest programem wsadowym, a sposób jego komunikowania się ze światem zewnętrznym jest ograniczony. Sprowadza się w praktyce do możliwości wczytywania plików tekstowych (polecenia `\input`, `\read`) i zapisywania pliku wyjściowego `dvi` albo `pdf` (polecenie `\shipout`), pliku `log` (na którego zawartość można w różny sposób wpływać) i zapisywania plików tekstowych (polecenie `\write`).

Nie da się TEXa uruchomić w potoku „karmiąc” danymi na standardowym wejściu i odbierając wyniki ze standardowego wyjścia.

Taki sposób przetwarzania jest wystarczający do przetwarzania wsadowego, ale nie pozwala skorzystać z nowych możliwości, jakie pojawiły się przez ostatnie 25 lat. W szczególności nie da się w czasie działania TEXa przesłać danych do zewnętrznego przetworzenia (np. za pomocą innego języka czy narzędzia) i wczytać wyników.

Oczywiście zawsze można (a czasami i trzeba, bo inaczej się nie da) poradzić sobie wieloprzebiegowo. Bez gruntownego przerobienia TEXa nie sposób np. zrealizować odsyłaczy, w których odwołanie następuje przed wystąpieniem elementu, do którego się odwołujemy. Rozwiązania wieloprzebiegowe są oczywiście bardzo skuteczne i w większości przypadków dają możliwość osiągnięcia pożądanego efektu.

TEX jednak się zmienia, wprowadzane są zmiany i rozszerzenia. Obecnie można już w TEXu wywoływać zewnętrzne polecenia systemowe, zapisywać dane do plików FIFO i czytać z takich plików.

Dostępne rozszerzenia, tj. polecenie `\write18` (opcja `--shell-escape`) oraz możliwość zapisywania logów, danych `dvi` i wyników polecenia `\write` do plików FIFO, poprawiają nieco sytuację, wyprowadzając TEXa ze świata jednozadaniowego i pozwalają skorzystać z możliwości współczesnych wielozadaniowych systemów operacyjnych.

W tym tekście przedstawię przegląd tradycyjnych możliwości TEXa związanych z poleceniami wejścia-wyjścia oraz dodanych później i dostępnych obecnie rozszerzeń. Ponieważ dostępne w TEXu mechanizmy komunikacji międzyprocesowej wykorzystują notację i standardowe polecenia wejścia-wyjścia, to zaczniemy od przypomnienia jak działają polecenia czytania i zapisywania danych w TEXu.

Tradycyjne operacje wejścia-wyjścia

Polecenie `\input` – wczytywanie plików. Polecenie `\input` jest wbudowanym poleceniem TEXa, powodującym wczytanie w miejscu wystąpienia podanego jako argument pliku. Uwaga – oryginalne, TEXowe niskopoziomowe polecenie `\input` nie jest dokładnie tym samym co polecenie `\input` dostępne w LATEXu. LATEX przedefiniowuje je, pozwalając w szczególności na podanie argumentu, czyli nazwy pliku w nawiasach klamrowych. Domyślnym rozszerzeniem pliku wejściowego jest `.tex`.

Działanie tego polecenia jest oczywiste – TEX wczytuje podany jako argument plik i wykonuje wszystkie zawarte w nim polecenia i składa znajdujący się tam tekst do napotkania końca pliku, polecenia `\endinput`, kończącego czytanie tego pliku w miejscu wystąpienia tego polecenia albo polecenia `\end` kończącego całkowiecie bieżące przetwarzanie.

Polecenie `\read` – czytanie rekordów. W TEXu można także otwierać i czytać dane z 16 plików jednocześnie. Plik otwiera się poleceniem `\openin`:

```
\openin <nr>=<nazwa pliku>
```

gdzie <nr> jest numerem strumienia wejściowego.

Kolejne wiersze z pliku odczytuje się poleceniem \read:

```
\read <nr> to <polecenie>
```

gdzie <nr> jest numerem strumienia wejściowego, a to <polecenie> jest definicją zawierającą dane z wiersza (lub wierszy) wczytanego z pliku. Polecenie \read może odczytać więcej niż jeden wiersz z pliku, ponieważ kiedy wczytywane dane zawierają nawiasy klamrowe, to TEX czyta dane z pliku, aż do momentu sparowania nawiasów[1].

Jeśli z podanym numerem strumienia nie jest związany żaden otwarty plik albo numer jest spoza zakresu od 0 do 15, to TEX próbuje odczytać dane z terminala, zgłaszając znak zachęty.

Zamiast numerów strumieni można wykorzystywać nazwy symboliczne. Nowy strumień wejściowy tworzymy poleceniem \newread np.:

```
\newread\wejscie
```

definiuje nowy strumień wejściowy, który możemy potem związać z plikiem np.:

```
\openin\wejscie=test.tex
```

Poniżej jest przykład pliku wejściowego, który wykorzystamy do zaprezentowania działania polecenia \read. Zawiera on zarówno definicje jak i specjalnie dobrane nawiasy klamrowe, tak żeby zaprezentować sposób działania polecenia.

```
\noindent Wczytywanie {danych poleceniem
\def\tmp{\texttt{\string\read,
% Tutaj wiersz komentarza
\string\openin, % A tutaj komentarz w wierszu
\string\newread}}\tmp
\halign{\strut
\hfill\texttt{#} \vrule&\ #\hfill\cr
\string\openin& Otwarcie pliku\cr
\string\newread & Definicja strumienia\cr
\string\read & Wczytanie danych\cr
\string\ifeof & Koniec pliku?\cr
\string\closein&Zamknięcie pliku\cr}}} KONIEC
{Słowo ,,\texttt{KONIEC}'' nie będzie
wczytane bo ostatni nawias przed nim
jest} nadmiarowy
```

Po wykonaniu sekwencji poleceń, zawierającej dwukrotne wywołanie polecenia \read

```
\newread\pliktestowy
\openin\pliktestowy=test.tex
\read\pliktestowy to \testread
\testread
\read\pliktestowy to \testread
\testread
```

uzyskamy efekt jak poniżej (zakładając, że jest zdefiniowane jednoargumentowe polecenie \texttt i powoduje ono złożenie tekstu pismem maszynowym):[2]

Wczytywanie danych poleceniem \read, \openin, \newread

\openin	Otwarcie pliku
\newread	Definicja strumienia
\read	Wczytanie danych
\ifeof	Koniec pliku?
\closein	Zamknięcie pliku

Słowo „KONIEC" nie będzie wczytane bo ostatni nawias przed nim jest nadmiarowy

---

Jednym poleceniem \read zostały do polecenia \testread przypisane wiersze od pierwszego, aż do wiersza zawierającego słowo „KONIEC". Ponieważ jednak przed tym słowem znajduje się nawias zamykający, któremu nie odpowiada nawias otwierający, to polecenie \read zakończy czytanie w tym miejscu.

Zwróćmy uwagę, że gdybyśmy przykładowy plik testowy próbowali włączyć poleceniem \input, to TEX zgłosiłby błąd, wykrywając nadmiarowy zamykający nawias klamrowy.

Do sprawdzenia czy doszliśmy do końca pliku służy polecenie warunkowe \ifeof. Polecenie to może się także przydać do sprawdzenia czy plik, który chcemy otworzyć istnieje:

```
\openin\plik=abc.tex
\ifeof\lastpagerd
   <pliku nie ma>
\else <plik istnieje>
\fi
```

Po zakończeniu pracy z plikiem zamykamy go poleceniem \closein.

Polecenie \write. Polecenie \write służy do zapisywania danych do plików wyjściowych, innych niż domyślnie tworzony plik .dvi i .pdf. Z poleceniem tym związane są oczywiście polecenia \newwrite, \openout i \closeout

Zdefiniowanie nowego strumienia wyjściowego oraz otwarcie pliku ma składnię podobną jak przy czytaniu np.:

[1] Uwaga: Jeśli nawiasów zamykających jest więcej niż otwierających, to TEX wczytuje dane tylko do pierwszego „nadmiarowego" nawiasu zamykającego. Do wczytywanych danych stosują się reguły analizowania tekstu – w szczególności uwzględniane są np. komentarze, mogą być umieszczone definicje makr, z tym, że należy pamiętać o grupowaniu i lokalności definicji względem grup.

[2] Dodatkowo w tym konkretnym przykładzie trzeba na chwilę przedefiniować polecenie \newread (np. za pomocą \let\newread=\relax, ponieważ w plain TEXu jest ono zdefiniowane jako \outer

```
\newwrite\toc
\openout\toc=\jobname.toc
```

Polecenie to otwiera do zapisu plik o nazwie takiej samej jak główny plik, czyli plik podany w wywołaniu TeXa i rozszerzeniem `.toc`.

Polecenie `\write` wygląda następująco:

```
\write<nr>{<tekst zbalansowany>}
```

`<Nr>` to numer strumienia (albo nazwa symboliczna strumienia) a `<tekst zbalansowany>` to ciąg znaków (mogący zawierać polecenia, które będą rozwinięte przed zapisaniem) z taką samą liczbą otwartych i zamkniętych nawiasów klamrowych. Polecenie `\write` zapisuje podany tekst jako jeden wiersz. Chcąc zapisać kilka wierszy należy wywołać polecenie `\write` kilka razy. Nie jest to cała prawda – zainteresowanych tym jak jest naprawdę zachęcam do sięgnięcia do TeXbook-a.

Normalnie polecenia `\write` (a także `\openout` i `\closeout`) nie są wykonywane wtedy, gdy TeX je znajdzie w tekście wejściowym, ale z opóźnieniem, w momencie, kiedy pudełko, w którym znalazła się operacja `\write` zostaje zapisane do pliku `.dvi` (albo `.pdf`). Działanie takie jest uzasadnione tym, że dla znalezienia miejsca przełamania strony TeX musi przeczytać i złożyć więcej materiału niż mieści się na stronie, a polecenie `\write` jest wykorzystywane do realizacji odsyłaczy w tekście i w momencie zapisu musi być znany ostateczny numer strony, na której wyląduje dany materiał.

Możliwe jest natychmiastowe wykonanie poleceń związanych z zapisywaniem danych. Jeśli takiego działania oczekujemy, to powinniśmy właściwe polecenie wyjściowe poprzedzić przedrostkiem `\immediate` np.

```
\immediate\write\plikwy{\number\count0}
```

zapisuje wartość licznika nr 0 (czyli numeru jeszcze nie zapisanej strony), jaka była aktualna w momencie napotkania tego polecenia. Wymuszanie natychmiastowego wykonania poleceń wyjściowych jest konieczne, jeśli chcemy skorzystać z rozszerzeń polecenia `\write` do wywoływania zewnętrznych poleceń.

Wywoływanie poleceń systemowych `\write18`. Możliwość wywoływania poleceń systemowych została dodana do TeXa ponad 10 lat temu. Nie została utworzona żadna specjalna składnia, polecenie systemowe wywołuje się po prostu zapisując do specjalnego strumienia wyjściowego nr 18. Mechanizm ten w obecnych implementacjach TeXa opartych na `web2c` włącza się opcją `--shell-escape` albo ustawieniem odpowiedniej zmiennej w pliku `texmf.cnf`.

```
\immediate\write18{pstree -A > tmp18.tex}
```

Powyższy przykład prezentuje wywołanie polecenia `pstree` i zapisanie wyników jego działania do pliku `tmp18.tex`. Robi się tak po to, żeby można było skorzystać z efektów działania wykonanego polecenia, ponieważ standardowe wyjście wywoływanego polecenia jest zapisywane do pliku `.log` i na terminal. Poniżej przykładowy wynik wywołania powyższego polecenia wstawiony z wykorzystaniem LaTeXowego pakietu `verbatim`.

```
init-+-acpid
     |-6*[agetty]
     |-bdflush
     |-cardmgr
     |-devfsd
     |-gdm---gdm-+-X
     |           `-openbox-+-ascpu
     |                     |-2*[ssh-agent]
     |                     |-wmCalClock
     |-klogd
     |-kupdated
     |-master-+-pickup
     |        `-qmgr
     |-svscan-+-supervise---dnscache
     |        `-supervise---multilog
     |-syslogd
     |-wine-preloader---11*[wine-preloader]
     |-wineserver
     `-xfs
```

Pokazany przykład wykorzystania mechanizmu nie jest specjalnie ciekawy. Z bardziej użytecznym zastosowaniem można spotkać się w przypadku ConTeXt-a, przy tworzeniu grafiki metapostowej definiowanej w pliku TeXowym. ConTeXt wykorzystuje mechanizm `\write18` do wywołania programu `mpost` przetwarzającego pliki wejściowe wygenerowane przez TeXa w czasie działania.

Mechanizm ten można także wykorzystać do realizacji kalendarza. W tym celu wykorzystujemy linuksowe polecenie `cal`, którego wyjście jest przekształcane z wykorzystaniem języka Perl, a całość jest obudowana pakietem LaTeXowym wywołującym skrypt perlowy z parametrami przekazanymi z dokumentu TeXowego. Przykładowy efekt działania widać poniżej.

STYCZEŃ

Po	Wt	Śr	Cz	Pi	So	Ni
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

LUTY

Po	Wt	Śr	Cz	Pi	So	Ni
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28						

MARZEC

Po	Wt	Śr	Cz	Pi	So	Ni
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

Część TeXowa związana z generowanie kalendarza jest bardzo prosta: Każdy miesiąc jest składany po wywołaniu polecenia `\kalendarz`, np:.

```
\kalendarz{width=3cm, size=6pt, height=2cm,
                               month=1,  year=2005}
```

Zdefiniowane z wykorzystaniem pakietu *keyval* opcje są przekazywane jako argumenty do skryptu generującego kalendarz. Część perlowa jest nieco bardziej skomplikowana, ale jej działanie sprowadza się do wywołania polecenia systemowego `cal` z przekazanym z TEXa miesiącem oraz rokiem i „otagowaniu" wyjścia z tego polecenia.

Pliki FIFO – prawdziwe IPC. W omawianych do tej pory przykładach komunikacja odbywa się „jednorazowo": TEX zapisuje dane do pliku zewnętrznego, po czym wywołuje zewnętrzny program, który przetwarza je, zapisując wyniki do kolejnego pliku, wczytywanego w końcu znów przez TEXa. Schemat taki można zrealizować bez mechanizmu wywoływania zewnętrznych poleceń, zapisując dane pośrednie w plikach przetwarzanych przez polecenia zewnętrzne między dwoma kolejnymi wywołaniami TEXa.

Prawdziwie ciekawe rzeczy można robić, kiedy TEX i program zewnętrzny współpracują bardziej ściśle, działając w tym samym czasie. Pliki wczytywane przez TEXa albo otwierane przez niego do zapisu nie muszą być zwykłymi plikami. Mogą to być pliki FIFO, czyli potoki, umożliwiające dwukierunkową komunikację działających jednocześnie procesów. Schemat działania w takim wypadku wygląda jak klasyczny przypadek klient-serwer, gdzie klientem jest TEX, a serwerem np. perlowy program zarządzający danymi na potrzeby TEXa.

- używając np. mechanizmu `\write18` tworzymy pliki FIFO wykorzystywane do komunikacji,
- uruchamiamy w tle zewnętrzny program (np. skrypt perlowy) oczekujący na dane od TEXa w pliku FIFO.
- jednym plikiem FIFO wysyłamy dane do przetworzenia na zewnątrz TEXa, a wyniki odbieramy drugim plikiem FIFO.
- przy końcu przetwarzania musimy zadbać o ubicie programu zewnętrznego, chyba że przechowuje on dane niezbędne w różnych dokumentach, czy kolejnych przetwarzaniach tego samego dokumentu.

Stosując taki sposób wspierania TEXa zewnętrznymi programami należy jednak mieć na uwadze synchronizację, która staje się bardzo istotna. Bardzo łatwo doprowadzić do zakleszczenia TEXa i jego współpracownika. W praktyce okazuje się, że najlepiej po prostu otwierać plik FIFO wyłącznie na czas zapisania czy odczytania danych i zamykać go zaraz po wykonaniu operacji. Taki sposób zapewnia poprawną komunikację bez zakleszczania.

Używając tej techniki i wykorzystując perlowy moduł `Spreadsheet::Perl` można zrealizować przeliczanie wartości w tabelach TEXowych (np. sumowanie wierszy czy kolumn). Możliwe jest oczywiście także zaimplementowanie w samym TEXu, ale mimo komplikacji związanych z synchronizacją, łatwiej to zrealizować w sposób hybrydowy. Tym bardziej, że pozwala to na łatwe przenoszenie wartości między tabelami, pod warunkiem, że liczymy zawsze „w przód" i w tabelach występujących wcześniej w dokumencie nie wykorzystujemy obliczeń z tabel późniejszych. W takim wypadku nie obejdzie się bez podejścia wieloprzebiegowego.

Schemat działania takiego kalkulatora TEXowo-perlowego wygląda tak:

- utworzenie plików FIFO do komunikacji,
- wywołanie perlowego „demona" obsługującego obliczenia,
- wysyłanie danych i pobieranie wyników z TEXa,
- zakończenie pracy „demona".

Definicja tabeli z kalkulacją wygląda np. tak:

```
\pss \noalign{\hrule} \vrule
\pleft{#}&\pcenter{#}&\pright{#} \vrule\cr
1000 & 5 & formula $ss{C3}/$ss{B1}\cr
1000 & 8 & formula $ss{A2}+$ss{B2}\cr
2000 & 8 & formula $ss{A3}*$ss{B3}\cr
formula $ss->Sum("A1:A3") & %$
formula $ss->Sum("B1:B3") & %$
formula $ss->Sum("C1:C3") \cr %$
\noalign{\hrule} \endpss
```

A po przetworzeniu tak:

1000	5	3200
1000	8	1008
2000	8	16000
4000	21	20208

W wyrażeniach można korzystać z dowolnych wyrażeń matematycznych dostępnych w perlu. Można też odwoływać się do wartości dowolnych rubryk, także tych wyliczanych. Adresowanie i notacja jest zgodna z opisaną w dokumentacji do perlowego modułu `Spreadsheet::Perl`.

Zakończenie

Mimo egzotycznej, jak na dzisiejsze czasy, składni TEX zapewnia dosyć szerokie możliwości realizacji operacji wejścia-wyjścia. Możliwość uruchamiania zewnętrznych poleceń systemu operacyjnego oraz wykonywania operacji zapisu i odczytu w plikach FIFO pozwala na tworzenie makr wykorzystujących inne niż TEX języki i narzędzia. Nie jest to rewolucja, ponieważ większość tego co można w ten sposób osiągnąć da się także zrealizować korzystając z tradycyjnego TEXowego języka makr. Nowe możliwości dają jednak większą elastyczność i pozwalają na tworzenie pakietów wykorzystujących „normalne" języki programowania takie jak Perl, czy Python.