Systemy operacyjne
Programowanie w języku powłoki sh

[2] Programowanie w języku powłoki sh

Powłoka, interpreter poleceń angshell jest programem uruchamianym standardowo po otwarciu sesji użytkownika przez proces *login*. Powłoka jest aktywna aż do wystąpienia znaku `<EOT>`, który powoduje jej zatrzymanie i zgłoszenie tego faktu do jądra systemu.

Każdy użytkownik otrzymuje własny i odrębny egzemplarz **sh**. Program **sh** wypisuje monit $ na ekranie, dając znać o swojej gotowości do przyjęcia polecenia (komendy).

Interpreter poleceń funkcjonuje według następującego schematu:

1. wypisuje monit,
2. czeka na wprowadzenie tekstu z klawiatury,
3. analizuje wiersz polecenia i znajduje program,
4. zleca jądru wykonanie programu,
5. przyjmuje odpowiedź od jądra, ponownie wypisuje monit.

[3] Inicjalizacja powłoki

Inicjalizacja powłoki:

1. przypisywane są wartości **zmiennym powłoki**,
2. wykonywane są skrypty systemowe definiujące dalsze elementy otoczenia powłoki.

	shell	skrypty systemowe
1.	sh, ksh	.profile
2.	csh	.login, .cshrc

Rozbudowana lista plików inicjacyjnych dla interpretera **zsh**:

1. /etc/zprofile
2. /etc/zlogin
3. /etc/zshrc
4. /etc/zshenv

... odpowiedniki kropkowe powyższych plików w katalogu domowym.

[4] **Użytkownicy w systemie Unix**
Użytkownicy w systemie Unix

- superużytkownik (ang. *superuser, root*),
- pozostali użytkownicy.

Użytkownik w systemie, plik /etc/passwd:

- nazwa użytkownika,
- hasło,
- uid (ang. *user identification*),
- gid (ang. *group identification*),
- informacja o użytkowniku,
- katalog domowy użytkownika,
- nazwa powłoki inicjowanej po otwarciu sesji.

```
tnowak:hasło zakodowane:201:50::/usr/tnowak:/bin/sh
```

[5] **Grupy użytkowników**
Grupa w systemie, plik /etc/group:

- nazwa grupy,
- hasło grupy,
- numer grupy,
- lista użytkowników należących do grupy.

```
wheel::10:tnowak,tkruk
```

Inne zagadnienia:

- prawa dostępu do pliku (-rwxr-xr-x, 0755, komenda chmod),
- bity SUID, SGID (-r-s–x–x, np. komenda passwd),
- plik /etc/shadow.

[6] **Zmienne powłoki**
Wśród zmiennych powłoki wyróżniamy:

- **zmienne predefiniowane** z wartością jawnie przypisywaną,

- **parametry powłoki**, w których gromadzone są informacje dotyczące nazwy i argumentów aktualnie wywoływanej komendy.

Przykładowe zmienne powłoki *sh*:

HOME katalog standardowy dla komendy cd,

IFS (ang. *Internal Field Separators*) znaki separatorów rozdzielających elementy składniowe występujące w linii komendy,

MAIL skrzynka pocztowa z ustawionym powiadamianiem,

PATH lista katalogów, w których poszukiwane będą pliki wywoływanych komend,

PS1 (ang. *Prompt String 1*) pierwszy znak zachęty powłoki, standardowo **$**,

PS2 drugi znak zachęty powłoki, standardowo >,

SHELL domyślny program do wywoływania jako podpowłoka,

TERM rodzaj terminala, identyfikuje zestaw sekwencji sterujących właściwych dla danego terminal (np. *ansi*, *vt100*, *xterm*),

[7] **Polecenia**

Postać polecenia

```
$ nazwa_polecenia arg1 ... arg9
$ echo $PATH
```

Polecenia wbudowane

```
$ PATH=$PATH:/usr/local/bin
$ export PATH
```

- bezparametrowe **set** wyświetla wartości wszystkich zmiennych środowiska
- bezparametrowe **export** wyświetla wartości wszystkich eksportowanych zmiennych środowiska

[8] **Parametry powłoki**

Parametry powłoki

$0 nazwa wywołanej komendy (cmd)

$1 pierwszy argument (parametr) wywołania

$2 drugi argument (parametr) wywołania

$9 dziewiąty argument (parametr) wywołania

$* argumenty jako jeden łańcuch znaków `"$*" = "$1 $2 .."`

$@ argumenty jako osbne łańcuchy znaków `"$@" = "$1" "$2" ..`

$# liczba argumentów przekazanych przy wywołaniu lub przez set,

$? stan końcowy (ang. *exit status*) ostatnio wykonywanej komendy,

$$ numer procesu aktualnie wykonywanej powłoki,

$! numer procesu ostatnio wykonywanego procesu w tle.

$0-9 także: opcje przypisane powłoce przy wywołaniu lub przez set,

[9] **Metaznaki**

Podczas opracowywania nazw plików oraz przy grupowaniu komend w większe całości stosuje się znaki o specjalnym znaczeniu dla interpretera nazywane **metaznakami**.

```
*          dowolny łańcuch znaków nie zawierający "/",
?          jeden dowolny znak,
[ ]        każdy pojedyńczy znak ze zbioru zamkniętego w te nawiasy,
[...-...] jak [ ], w zakresie od pierwszego do ostatniego podanego znaku,
[!..-...] w zakresie wszystkich oprócz od pierwszego do ostatniego znaku,
#          komentarz,
\          (back slash) przywraca poprzedzonemu metaznakowi jego normalne
           literalne znaczenie,
$          wartość zmiennej,
;          koniec komendy,
` `        łańcuch w znakach akcentu jest wykonywany jak komenda,
' '        klamrowanie apostrofami jednostki tekstu powoduje uniknięcie
           jakichkolwiek podstawień (substytucji),
" "        klamrowanie cudzysłowem jednostki tekstu powoduje uniknięcie
           wszelkich  podstawień za wyjątkiem:  $  ` `  \
```

[10] **Interpretacja komend**

Interpretacja komend przez powłokę sh odbywa się w następujący sposób:

1. wprowadzenie tekstu polecenia (ciągu znaków),
2. podzielenie ciągu znaków na ciąg słów w oparciu o zawarte w IFS seperatory,

3. substytucja 1: zastępowanie zmiennych powłoki tzn. zastępowanie metawyrażeń o postaci ${słowo} ciągami znaków zawartymi w zmiennych wyspecyfikowanych przez słowo np.

```
$ b=/usr/user
$ ls -l prog.* > ${b}3
```

4. substytucja 2: rozszerzanie parametrów tzn. rozszerzenia słów zawierających metaznaki * ? [ ] na odpowiednie nazwy plików w katalogu aktualnym,

5. substytucja 3: interpretacja łańcucha ujętego w znaki akcentu ‘ ‘ jako komendy i jej wykonanie.

[11] Grupowanie

- przyjęto konwencję, że argumenty które nie są nazwami plików należy poprzedzać znakiem minus –.
- komendy mogą być grupowane w nawiasy:
  - nawiasy okrągłe (ciąg-komend) służą do grupowania komend, które będą wykonywane jako samodzielny proces. Proces ten może być również wykonywany w tle (&).
  - nawiasy klamrowe { ciąg-komend; } służą do grupowania komend, które będą normalnie wykonywane w ramach bieżącego procesu.
- końcem komendy są następujące znaki: `<NL>` `;` `&`

[12] Przeadresowywanie wejścia/ wyjścia

Po otwarciu sesji do otoczenia użytkownika należą następujące pliki:

- wejście standardowe (**stdin**) - strumień 0,
- wyjście standardowe (**stdout**) - strumień 1,
- standardowe wyjście błędów (**stderr**) - strumień 2.

Znakami przeadresowywania są:

`> plik`	przekieruj stdout do pliku
`>> plik`	dopisz stdout do pliku
`< plik`	przekieruj stdin z pliku
`<< EOT`	czytanie tekstu z stdin w trybie bezpośrednim, aż do wystąpienia słowa EOT
`n > plik`	przekierowanie wyjścia strumienia o deskryptorze n do pliku,
`n >> plik`	dopisanie przekierowania wyjścia strumienia do pliku,
`n>&m`	przekierowanie wyjścia strumienia n do wyjścia strumienia m,
`n<&m`	przekierowanie wejścia strumienia n do wejścia strumienia m.

[13]

Procedury powłoki (skrypty)

Komendy powłoki zgrupowane w zwykłym pliku tekstowym mogą być wykonane poprzez:

```
$ sh [opcje] plik_z_komendami [arg ...]
```

Po nadaniu plikowi zawierającemu komendy, atrybutu wykonywalności, komendą **chmod**, np.:

```
$ chmod +x plik_z_cmd
```

można go wykonać jak komendę, bez podawania sh przed jego nazwą.

```
$ plik_z_komendami arg ...
```

[14] **Struktury sterujące**

- do sterowania przebiegiem procedury powłoki służą instrukcje takie, jak: **if**, **for**, **while**, **until**, **case**
- możliwe jest skrócenie zapisu **if**, przy użyciu znaków:

And-if	`&&`	(gdy rezultat równy zero)
Or-if	`||`	(gdy rezultat różny od zera)

  ```
  $ cp x y   &&   vi y
  $ cp x y   ||   cp z y
  ```

- Każda komenda umieszcza w `$?` status z jakim zakończyło się jej wykonanie. Status 0 oznacza pomyślne zakończenie działania procesu. Status niezerowy oznacza wystąpienie błędu podczas wykonywania się komendy.

[15] **Instrukcja if**

- ogólny zapis jest następujący:

  ```
  if ciąg_komend_1
      then ciąg_komend_2
      {else ciąg_komend_3}
  fi
  ```

- przykład

  ```
  if cc -c p.c
  then
    ld p.o
  else
    echo "compilation error" 1>&2
  fi
  ```

[16] **Instrukcja case**

- ogólny zapis jest następujący:

```
case słowo in
  wzór_1) lista_komend_1;;
  wzór_2) lista_komend_2;;
  *) lista_komend_domyślnych;;
esac
```

- przykład

```
case $# in
  0) echo 'usage: man name' 1>&2; exit 2;;
```

[17] **Instrukcje iteracyjne (pętle)**

W powłoce sh instrukcje iteracyjne (pętle) występują w trzech odmianach:

- instrukcja **for**, której treść jest wykonywana jednorazowo dla każdego słowa w liście słów,
- instrukcja **while**, której treść jest wykonywana tak długo, jak długo jest spełniony warunek w while,
- instrukcja **until**, której treść jest wykonywana tak długo, aż nastąpi spełnienie warunku w until.
- można stosować instrukcje **continue** i **break**

```
#!/bin/sh
for i in /tmp /usr/tmp
do
        rm -rf $i/*
done
```

[18] **Przykłady różne**

- ```
  $ cat file.dat | while read x y z
    do
      echo $x $y $z
    done
  ```
  ```

- ```
  #!/bin/sh
  i=1
  while [ $i -le 5 ]; do
    echo $i
    i=`expr $i + 1`
  done
  ```

- ```
  $ who -r
  . ru-level 2 Aug 21 16:58 2 0 S
  $ set `who -r`
  $ echo $6
  16:58
  ```

[19] **Przykład duży**

```
#!/usr/bin/zsh
PATH=/usr/bin:/usr/local/bin:/bin
WAIT_TIME=5
. /export/home/oracle/.zshenv
#sprawdz czy jest sens go sprawdzac..
PID=`ps -ef | grep LISTENER | grep -v grep | awk -e '{print $2 }'`
if test -z "$PID"
then
        exit 0
fi
# sprawdz jak dziala
lsnrctl status >/dev/null 2>&1 &
sleep $WAIT_TIME
kill $! 2>/dev/null
res="$?"
if test "$res" != "1"
then
        kill $PID
        kill -9 $PID
        logger -p user.err Oracle LISTENER ERROR (stunned) - restarted
        lsnrctl start
fi
```