Операционни системи, лекции

Георги Георгиев (Скелета)

21 март 2018 г.

Този документ не е учебник. Това са бележки на автора, част от подготовката за лекции. Целта е да ми помагат, когато се скъса тънката нишка на мисълта.

1 Въведение

Дефиниция Няма точна дефиниция на ОС.

Неформални дефиниции:

ОС осигурява удобства за агентите, ползващи изчислителната система (хора, други ИС, вътрешни компоненти на ИС).

ОС осигурява абстракции и изолация.

[Таненбаум] providing application programmers (and application programs, naturally) a clean abstract set of resources instead of the messy hardware ones and managing these hardware resources.

[http://www.toves.org/books/os/] ОС осигурява удобни абстракции за сложните компютърни ресурси, хардуерна съвместимост, сигурност.

[http://www.linfo.org/operating_system.html] An operating system is a collection of programs that manages all the other programs (i.e, application programs) in a computer as well as the allocation and use of hardware resources such as the CPU (central processing unit), memory and the hard disk drive (HDD).

Основни понятия (абстракции):

периферни устройства, драйвери (хардуерна съвместимост)

файлове, имена (разделяне на пространството, идентификация на ресурсите)

процес, комуникационен канал (обработка и обмен на информацията, многозадачност, времеделене)

права за достъп, изолация (защита на данните и информационните процеси)

Структура, интерфейси:

Потребителски интерфейс: GUI, shell

Програмен интерфейс: API

ОС в тесен смисъл: kernel – осигурява основните абстракции и съвместната им работа

ОС в широк смисъл: kernel + обвивки – осигурява стандартни инструменти за работа

Литература:

The Mythical Man-Month
Essays on Software Engineering
Frederick P. Brooks, Jr.
University of North Carolina, Chapel Hill

The Unix-HATERS Handbook
Published by IDG Books Worldwide, Inc.

2 Увод в Unix

Стил Unix Основни принципи за програмите и интерфейса:

Doug McIlroy (изобретил pipeline през 1973, реализиран от Ken Thompson):
Write programs that do one thing and do it well.
Write programs to work together.
Write programs to handle text streams, because that is a universal interface.

Ортогоналност на инструментите (и на системните извиквания).

Предпочитане на текстов формат – вход-изхода е чист от съобщения и няма интерактивна работа (диалог с потребителя). Съобщенията са в stderr, а ако правим диалог, добавяме опции или пишем програма-обвивка.

Ранна философия на Unix – малки и прости инструменти, които се комбинират.

Използване на абстракциите на ОС – процеси, комуникационни канали, файлове.

Изграждане на технологични решения и програми за фиксирането им.

Средство за този инженерен подход – Unix shell.

Сравнение GUI – shell

GUI е потребителски – приложения, документи, скриване на абстракциите.

shell е програмистки – достъп до абстракциите и ресурсите, нужда от квалификация. Улеснява комбинирането на инструменти, конструиране, тестване и поддръжка на сложни решения.

2.1 Въведение в shell

(Linux dash/bash).

Разглеждане на ресурсите (coreutils):

(1) файлова система (Unix name space), структура, монтиране, mount, df, du, ls, mkdir, mv, cp, rm

(2) процеси, наследяване, ps, top

(3) документация – man

Прости инструменти: cat echo cut grep find sort wc tar
Пример: find /etc -name *.sh|wc -l
Учебник: /usr/share/doc/bash/bash*.pdf

Литература:

https://en.wikipedia.org/wiki/Unix
https://en.wikipedia.org/wiki/Unix_philosophy
https://en.wikipedia.org/wiki/Pipeline_(Unix)
критика на интерфейса на Unix:
Norman, Don (1981). "The truth about Unix: The user interface is horrid"

2.2 Процеси и наследяване в Unix

Процесите в Unix са представени чрез дърво с ребра родител–наследник.

Процесът работи в обкръжение, стартираната програма наследява от родителя няколко неща:
(1) обкръжение (вижда се с команда env) – списък от низове от вида name=value.
(2) аргументи – името на извиканата програма, аргументи и опции.
(3) отворени файлови дескриптори (комуникационните канали на родителя).

Системни извиквания за пораждане на процеси:
fork() – създава точно копие на процеса.
execve(filename, argv[], envp[]) – заменя процеса с нов.

Файлов дескриптор – указател към единия край на комуникационен канал (stream, поток от байтове). Основни видове канали в Unix:
pipe – еднопосочна тръба, свързваща процеси.
file_stream – тръба между процес и обикновен файл или периферно устройство.
connection – двупосочна тръба, свързваща процеси.
socket – крайна точка за изграждане на конекции, генератор на конекции.

Не е много ясно ако няколко процеса ползват общ край на канал, кои атрибути на канала са общи и кои локални (man open, dup).

2.3 Управление на процесите в bash

Няколко специални символа в bash управляват изпълнението на процеси – >, <, >>, |, & и ;.

Изпълнение на команда:

Примерна програма на C, имитираща изпълнение на команда:

```
int pid;
pid=fork();
if (pid>0) waitpid(pid,...);
if (pid==0) execvp(args[0], args);
if (pid<0) perror("fork");
```

Пренасочване на вход/изхода

`cmd > filename` пренасочва изхода.
`cmd < filename` пренасочва входа.
`cmd >> filename` пренасочва изхода, като дописва във файла (`O_APPEND`).

Пример на C за пренасочване на изхода към файл "demo.txt":

```
int fd, pid;
pid=fork();
if (pid>0) waitpid(pid,...);
if (pid==0) {
 fd=open("demo.txt",O_WRONLY|O_CREAT,...);
 close(STDOUT_FILENO);
 dup(fd);
 close(fd);
 execvp(args[0], args);
}
if (pid<0) perror("fork");
```

Конвейри:

`cmd1 | cmd2 | cmd3` ... изпълнява паралелно няколко команди, като подава изхода на всяка като вход на следващата.

Пример на C за изпълнение на конвейр "`cmd1 | cmd2`":

```
int fd[2], pid, pid1;
pid=fork();
if (pid>0) waitpid(pid,...);
if (pid==0) {   /* Child shell */
 pipe(fd);      /* Create pipe */
 pid1=fork();
 if (pid1==0) { /* process for cmd1 */
  close(fd[0]);  /* Close unused read end */
  dup2(fd[1],STDOUT_FILENO);
  close(fd[1]);  /* Close duplicated write end */
  execvp("cmd1", args_cmd1);
 }
 if (pid1>0) { /* process for cmd2 */
  close(fd[1]);  /* Close unused write end */
  dup2(fd[0],STDIN_FILENO);
  close(fd[0]);  /* Close duplicated read end */
  execvp("cmd2", args_cmd2);
 }
}
if (pid<0) perror("fork");
```

Изпълнение на команда във фонов режим (background):

`cmd1 &` стартира команда за паралелно изпълнение с bash.

Литература:

https://brennan.io/2015/01/16/write-a-shell-in-c/
http://www.teach.cs.toronto.edu/ ajr/209/notes/procfiles/pipe-example.c

2.4 bash като език

bash е интерпретатор, чете от 3 възможни източника – конзола, файл или низ, ако се вика с 'bash -c'.

bash прави синтактически анализ на интерпретирания текст. Разделя го на команди, всяка команда е поредица от думи. За всяка дума се прави оценка (заместване), след това се правят пренасочванията и командата се изпълнява.

Подтискане на оценяването (quoting)

Символът \ подтиска оценяването на следващия го символ. Ако е в края на реда, изтрива от входния поток символа за нов ред, все едно залепя текущия и следващия ред.

Простите кавички 'текст' подтискат напълно оценяването в оградения текст.

Двойните кавички "текст" подтискат частично оценяването в оградения текст. Не се подтиска оценяването на '$' и '`'.

Примери:

```
$ echo 123\&45
123&45
$ x=123
$ echo 'x=$x'
x=$x
$ echo "x=$x"
x=123
$ echo "`whoami`"
skelet
```

Оценяване на думите, заместване (shell expansions)

При оценяването една дума може да се замести с няколко нови. Оценяването се предизвиква от специален символ или синтактична конструкция.

Има няколко вида оценяване, те се извършват в определен ред, като резултатът от всяко оценяване се подлага на следващото:

- Заместване на списък в големи скоби:
  ```
  $ echo L3{a,b,c}456
  L3a456 L3b456 L3c456
  ```
- Заместване на ~ с домашната директория.
- Изчисляване на стойност. Започва с $ и има няколко форми:
  - $name или ${name} връща стойността на променливата name.

    Втората форма е удобна когато променливата е потопена в текст. Стойността може да бъде модифицирана. *Пример:*

    ```
    $ x=my_video.webm
    $ echo $x
    my_video.webm
    $ echo ${x%.webm}.ogg
    my_video.ogg
    ```

– $(command) или ‘command‘ изпълнява команда и връща текста, който тя поражда. *Пример:*

```
$ echo $(date +%d.%m.%Y)
19.03.2018
```

– $((expr)) изчислява аритметичен израз. *Пример:*

```
$ echo $((2+3*5))
17
```

- Разширяване на имената на файлове. Ако дума съдържа *, ? или [list], тя се интерпретира като образец и се замества със списък от файлови имена, съответни на образеца.

  Образецът е частен случай на регулярен израз в смисъл на Клини:
  (1) обикновен символ съответства на себе си.
  (2) ? съответства на кой да е символ.
  (3) * съответства на кой да е низ.
  (4) [x-y] съответства на символите от x до y.
  (5) $[c_1 \dots c_k]$ съответства на изброените символи.
  (6) [!list] съответства на всички символи освен тези, описани в list. *Пример:*

```
$ echo *
counting.sh demo1.sh demo2.sh problem1.txt vi_demo.txt
$ echo *.sh
counting.sh demo1.sh demo2.sh
$ echo demo?.sh
demo1.sh demo2.sh
$ echo demo[2-5].sh
demo2.sh
$ echo demo[!2-5].sh
demo1.sh
```

Специални променливи:
$0 – име на изпълняваната програма
$1 ... $n – аргументи на изпълняваната програма
$# – брой на аргументите
$* $@ – списък от всички аргументи
$? – код на грешка от последната изпълнена програма
$$ – PID на изпълняваната програма

3 Синхронизация

Когато няколко изчислителни ресурса работят паралелно, необходимо е да синхронизират своята работа.

Терминът *синхронизация* следва да се разбира като съгласуване на общата работа, изчакване във времето – процес или хардуер, който използва резултат от друг процес или хардуер, трябва да изчака завършването на предходните задачи, за да ползва резултата.

Прекъсвания

Паралелизъм на хардуера, сигнализация между устройствата.

Превключване на процесите от входно-изходни събития.

Времеделене – timer interrupt.

Race condition, spinlock:

състезание за ресурси (*race condition*)

пример, критична секция

анализ на проблема – инварианта на структура данни

Взаимно изключване (*mutual exclusion, mutex*) – механизъм, забраняващ два процеса едновременно да изпълняват код, който променя данни в споделената структура (ресурс). Прилагането му е достатъчно условие да няма race condition, но не е необходимо (може да пипаме едновременно различни инварианти в случай, че промените по тях са независими).

Решения за защита на критичната секция (ниско ниво):
(1) алгоритми на Декер и Петерсон
(2) еднопроцесорна система – временна забрана на прекъсванията
(3) хардуерни механизми – test and set, atomic swap.
(4) многопроцесорни системи – memory barrier, подреждат във времето конвейри на различни процесори, за да се извърши атомарното заключване.

Всички тези механизми осигуряват взаимно изключване, като чакат в цикъл ресурса да се освободи (Tanenbaum: Mutual Exclusion with Busy Waiting). За тях ползваме термина *spinlock*.

Предимства на spinlock:
(1) няма смяна на контекста, ако критичната секция е кратък код това е оптималното решение.
(2) това е първичен инструмент, всеки друг разчита на него за да осигури атомарни операции.

Недостатъци на spinlock:
(1) не пести CPU-ресурс, когато няколко процеса чакат lock-а, всички те ползват CPU.
(2) не може пряко да реализира абстракция на хардуерното прекъсване и естествения паралелизъм на хардуера.
(3) spinlock следва да е непрекъсваем, иначе се получават големи забавяния.

Когато spinlock е неудобен, се ползват инструменти за синхронизация от високо ниво.

Семафор

Concurrent programming:

Литература:

Tanenbaum, Boss. "Modern Operating Systems". 4-th edition:
1.3.4 I/O Devices
2.3.1 Race Conditions
2.3.2 Critical Regions
2.3.3 Mutual Exclusion with Busy Waiting

Литература:

Tanenbaum, Boss. "Modern Operating Systems". 4-th edition:

POSIX, последна версия: http://pubs.opengroup.org/onlinepubs/9699919799/