

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI  
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

---



# BÀI TẬP MÔN HỌC

## HỆ ĐIỀU HÀNH

Nguyễn Duy Thành-CNTT1 20102737

Giáo viên :

**Phạm Đăng Hải**

HÀ NỘI

Ngày 12 tháng 5 năm 2012

# Mục lục

<b>1</b>	<b>Lời nói đầu</b>	<b>3</b>
<b>2</b>	<b>Chương trình shell đơn giản (Tiny Shell) cho POSIX</b>	<b>4</b>
2.1	Giới thiệu . . . . .	4
2.1.1	Tinyshell . . . . .	4
2.1.2	Hướng dẫn biên dịch và cài đặt . . . . .	4
2.2	Khởi động chương trình . . . . .	5
2.3	Gọi lệnh . . . . .	5
2.4	Lệnh nâng cao . . . . .	6
2.5	Job control . . . . .	6
2.5.1	Background/foreground . . . . .	6
2.5.2	Liệt kê các tiến trình nền . . . . .	7
2.5.3	Chuyển từ foreground sang background . . . . .	8
2.5.4	Chuyển tiến trình sang foreground . . . . .	9
2.6	Một số tiện ích khi gõ lệnh . . . . .	9
2.6.1	Tab-completion . . . . .	9
2.6.2	Lịch sử - History . . . . .	9
2.6.3	Chú thích - comment . . . . .	10
2.7	Xử lý theo lô - batch processing . . . . .	10
2.8	Gọi Tiny Shell từ các shell khác . . . . .	11
<b>3</b>	<b>Một số bài tập môn Hệ Điều Hành</b>	<b>12</b>
3.1	Bài toán Producer - Consumer . . . . .	12
3.1.1	Trường hợp Producer/Consumer là các tiến trình (Process) . . . . .	12
3.1.2	Trường hợp Producer/Consumer là các luồng (Thread) . . . . .	16

# Chương 1

## Lời nói đầu

Sinh viên thực hiện:

Nguyễn Duy Thành

# Chương 2

## Chương trình shell đơn giản (Tiny Shell) cho POSIX

:

### 2.1 Giới thiệu

#### 2.1.1 Tinyshell

Chương trình TinyShell là một *shell* dành cho Linux và các tương thích POSIX, có một số tính năng cơ bản như:

- Gọi, thực hiện chương trình ngoài,
- Thực hiện các kịch bản (*script*),
- Liệt kê các lệnh đã gọi (*history*),
- Gọi lại các lệnh trong *history* mà không cần phải gõ lại (*history expansion*),
- Quản lý tác vụ (*Job control*),
- Làm việc với biến môi trường (*Environment variable*),
- Tính toán một số phép tính số nguyên đơn giản
- Tab-completion
- Globbing
- ...

Chương trình được viết bằng ngôn ngữ C++, theo chuẩn C++11.

#### 2.1.2 Hướng dẫn biên dịch và cài đặt

Yêu cầu:

- Hệ điều hành: tương thích POSIX, với MS-Windows có thể cài CygWin để hỗ trợ POSIX,

- `cmake`  $\geq$  2.6: để sinh `makefile`. Download ở <http://www.cmake.org/cmake/resources/software.html>. Với Linux có thể cài đặt qua các `package manager`
- thư viện GNU `readline`, với hầu hết cả Linux distro hiện nay thì thư viện này đã được cài đặt sẵn,
- trình dịch: tốt nhất là `gcc 4.7`, với các trình dịch khác cần phải sửa file `CMakeLists.txt`.

Biên dịch:

1. **Sinh Makefile:** vào thư mục gốc của mã nguồn `Tinynshell`, gõ lệnh:

```
cmake CMakeLists.txt
```

Nếu có lỗi xảy ra thì có nghĩa là một số thư viện cần thiết chưa được cài đặt.

2. **Biên dịch,** gõ lệnh:

```
make
```

file thực thi `tinynshell` sẽ được sinh ra trong thư mục gốc của thư mục mã nguồn.

## 2.2 Khởi động chương trình

Mở chương trình, gõ lệnh:

```
./tinynshell
```

Từ `bash shell`(mặc định trên Linux) sẽ chuyển sang `tinynshell`. Hình 2.1. Dấu nhắc lệnh của `Tiny Shell` có dạng `<username>:<current working directory $`, nếu `username` có là `root` thì kết thúc dấu nhắc lệnh là `#`.

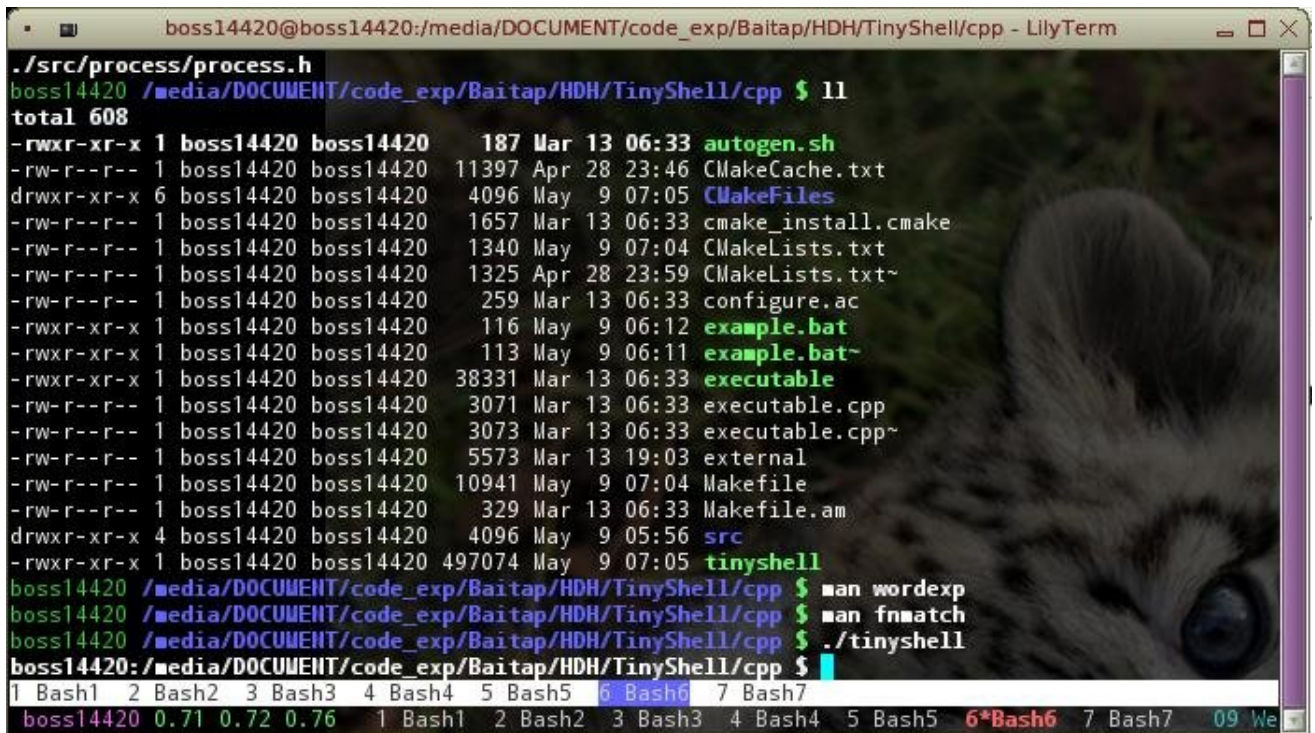
## 2.3 Gọi lệnh

Với `Tiny Shell`, ta có thể gõ lệnh bình thường như các `shell` khác:

```
$ ls -liah # thực thi chương trình ngoài
$ ./abc.py # thực thi file script python
```

Chú ý:

- Nếu ghi đường dẫn (tương đối hoặc tuyệt đối) thì `Tiny Shell` sẽ tìm kiếm file thực thi trong các thư mục được lưu trong biến môi trường `PATH`. VD: với `PATH = /usr/bin/ : /bin : /usr/local/bin` thì `Tiny Shell` sẽ tìm kiếm file thực thi trong các thư mục `/usr/bin/`, `/bin`, `/usr/local/bin`.
- Nếu không tìm thấy file thực thi thì `Tiny Shell` sẽ báo lỗi `badcommand`.



```
boss14420@boss14420:/media/DOCUMENT/code_exp/Baitap/HDH/TinyShell/cpp - LilyTerm
./src/process/process.h
boss14420 /media/DOCUMENT/code_exp/Baitap/HDH/TinyShell/cpp $ ll
total 608
-rwxr-xr-x 1 boss14420 boss14420 187 Mar 13 06:33 autogen.sh
-rw-r--r-- 1 boss14420 boss14420 11397 Apr 28 23:46 CMakeCache.txt
drwxr-xr-x 6 boss14420 boss14420 4096 May 9 07:05 CMakeFiles
-rw-r--r-- 1 boss14420 boss14420 1657 Mar 13 06:33 cmake_install.cmake
-rw-r--r-- 1 boss14420 boss14420 1340 May 9 07:04 CMakeLists.txt
-rw-r--r-- 1 boss14420 boss14420 1325 Apr 28 23:59 CMakeLists.txt~
-rw-r--r-- 1 boss14420 boss14420 259 Mar 13 06:33 configure.ac
-rwxr-xr-x 1 boss14420 boss14420 116 May 9 06:12 example.bat
-rwxr-xr-x 1 boss14420 boss14420 113 May 9 06:11 example.bat~
-rwxr-xr-x 1 boss14420 boss14420 38331 Mar 13 06:33 executable
-rw-r--r-- 1 boss14420 boss14420 3071 Mar 13 06:33 executable.cpp
-rw-r--r-- 1 boss14420 boss14420 3073 Mar 13 06:33 executable.cpp~
-rw-r--r-- 1 boss14420 boss14420 5573 Mar 13 19:03 external
-rw-r--r-- 1 boss14420 boss14420 10941 May 9 07:04 Makefile
-rw-r--r-- 1 boss14420 boss14420 329 Mar 13 06:33 Makefile.am
drwxr-xr-x 4 boss14420 boss14420 4096 May 9 05:56 src
-rwxr-xr-x 1 boss14420 boss14420 497074 May 9 07:05 tinysHELL
boss14420 /media/DOCUMENT/code_exp/Baitap/HDH/TinyShell/cpp $ man wordexp
boss14420 /media/DOCUMENT/code_exp/Baitap/HDH/TinyShell/cpp $ man fnmatch
boss14420 /media/DOCUMENT/code_exp/Baitap/HDH/TinyShell/cpp $ ./tinysHELL
boss14420:/media/DOCUMENT/code_exp/Baitap/HDH/TinyShell/cpp $
```

Hình 2.1: Tiny Shell

- Chỉ có những file có execute permission mới có quyền thực thi. VD, với file có permission/mode là 422 (`--x--w--w--`) thì chỉ có owner của file mới được thực thi. Những người dùng khác nếu thực thi file này sẽ bị báo lỗi *permissiondenied*. chỉ có owner của file mới được quyền thực thi.
- File script phải bắt đầu bằng 2 ký tự *sha-bang* (`#!`), tiếp theo là câu lệnh (có thể có cả tham số) để thực hiện chương trình đó. Chẳng hạn, một python script phải có dòng đầu tiên là:

```
#!/usr/bin/env python
```

Để bắt buộc chương trình kết thúc, dùng tổ hợp phím `Ctrl+C` (một số chương trình có thể bỏ qua yêu cầu này).

Để tạm dừng chương trình, dùng tổ hợp phím `Ctrl+Z`. Xem thêm ở mục 2.5.

## 2.4 Lệnh nâng cao

Tiny Shell sử dụng hàm `wordexp` của `libc` nên có thể thực hiện được một số thay thế các từ mà người dùng nhập vào tương tự `bash shell`. LCWE. Ví dụ:

```
$ # liệt kê thư mục gốc của người dùng boss14420
$ ls ~boss14420
...
$ # xem biến môi trường $PATH
$ echo $PATH
```

```

...
$ # Phép tính số học
$ echo $(( 2*3 ))
...
$ # liệt kê những file có phần mở rộng là .c
$ ls *.c
$ # xoá những file có dạng abc.<chữ số> trong thư mục
$ rm abc.[0-9]

```

Xem thêm ở phần [Word expansion](#) của `libc manual`.

## 2.5 Job control

Tiny Shell có một số tính năng của một Job Control shell LCJC.

### 2.5.1 Background/foreground

Một chương trình được gọi từ Tiny Shell có thể thực thi theo hai chế độ : chế độ hiện (foreground) và chế độ nền background. Ở chế độ hiện thì chương trình được gọi sẽ nhận dữ liệu từ `stdin` (thay vì Tiny Shell), tức là người dùng chỉ có thể giao tiếp với chương trình chứ không thể giao tiếp với Tiny Shell được nữa. Còn ở chế độ nền thì người dùng có thể giao tiếp với Tiny Shell trong khi chương trình đang chạy, chương trình không thể nhận dữ liệu từ người dùng nhưng vẫn có thể xuất dữ liệu ra ngoài. Để chạy chương trình ở chế độ nền thì thêm `&` ở cuối câu lệnh. Ví dụ:

```

$ ./external
Sleeping...
Waked!
$
$ ./external &
$ Sleeping...

$ ls
autogen.sh          CMakeFiles          CMakeLists.txt     configure.ac        example.bat~
executable.cpp       external             Makefile.am         tinyshell           CMakeCache.txt
cmake_install.cmake CMakeLists.txt~     example.bat         executable          executable.cpp~
Makefile             src

$ Waked!

[1]      Done          ./external
$

```

### 2.5.2 Liệt kê các tiến trình nền

Dùng lệnh `jobs` để liệt kê các tiến trình nền đang chạy. Nội dung kết quả gồm nhiều dòng, một dòng tương ứng một tiến trình nền, có dạng:

[<id>]<default>                      <status>                      <command>

Trong đó:

- <id> là chỉ số của tiến trình, dùng để phân biệt với các tiến trình khác,
- Nếu <default> là '+', tiến trình sẽ là tiến trình mặc định cho các lệnh **fg**, **bg** (sẽ nói phần tiếp theo). Nếu <default> là '-', tiến trình sẽ trở thành mặc định khi tiến trình mặc định kết thúc. Chỉ có tối đa một tiến trình '+' và một tiến trình '-'.
- <status> là trạng thái của tiến trình. Có hai trạng thái là **Running** có nghĩa là tiến trình đang chạy, **Stopped** có nghĩa là tiến trình đã bị dừng lại do người dùng gửi tín hiệu dừng **STOP signal** đến tiến trình bằng tổ hợp phím **Ctrl+Z**,
- <command> là câu lệnh shell.

Ví dụ:

```
$ vim
[1]+  Stopped                  vim
$ less example.bat
[2]+  Stopped                  less example.bat
$ ./external 6 &
$ Sleeping...

$ jobs
[1]-  Stopped                  vim
[2]+  Stopped                  less example.bat
[3]   Running                  ./external 6
$
```

### 2.5.3 Chuyển từ foreground sang background

Đôi khi có những chương trình cần thời gian thực hiện dài (ví dụ chương trình tính toán, chương trình chơi nhạc) mà cần một số dữ liệu đầu vào do người dùng nhập. Rõ ràng không thể bắt đầu chương trình ở chế độ **background** (vì cần nhập dữ liệu) và cũng không thể đợi chương trình chạy xong mới tiếp tục làm việc với **Tiny Shell**. Hay có nhiều chương trình ta muốn chạy ở chế độ **background** nhưng lại quên thêm dấu **&** ở cuối câu lệnh, ta cũng không thể chờ chương trình chạy xong hoặc tắt chương trình để chạy lại.

Với những trường hợp như trên, ta có thể chuyển tiến trình từ **foreground** sang **background** lúc cần thiết, như vậy chương trình vẫn chạy mà ta vẫn có thể làm việc khác.

Ta thực hiện như sau:

Vì lúc này người dùng không thể tương tác với **Tiny Shell** nên trước hết phải cho tiến trình tạm dừng bằng tổ hợp phím **Ctrl+Z**.

Sau đó, đưa tiến trình về chế độ **background** và tiếp tục tiến trình, dùng lệnh **bg**. Cú pháp lệnh **bg** như sau:

```
bg [job_id ...]
```



Trong đó, `job_id` là danh sách chỉ số của các tiến trình (đã bị tạm dừng) muốn đưa về chế độ nền. Nếu không chỉ ra `job_id` thì tiến trình được chọn sẽ là tiến trình mặc định (tức tiến trình có kèm theo dấu '+' trong kết quả liệt kê bằng lệnh `jobs`). Một tiến trình sẽ trở thành tiến trình mặc định khi nó là tiến trình gần nhất bị tạm dừng. Ví dụ:

```
$ mpg123 "abcde.mp3"
High Performance MPEG 1.0/2.0/2.5 Audio Player for Layers 1, 2 and 3
version 1.14.1; written and copyright by Michael Hipp and others
free software (LGPL/GPL) without any warranty but with best wishes

Playing MPEG stream 1 of 1: abcde.mp3 ...

MPEG 1.0 layer III, 192 kbit/s, 44100 Hz joint-stereo
Title:      dam Ma (Remix)                Artist: DJ
Comment:    NhạcCuaTui.Com - Nghe Nhạc Moi Luc Moi Noi
Album:      NhạcCuaTui.Com
Year:       2011                          Genre:  The Loai Khac
^Z[2]+  Stopped          mpg123 "abcde.mp3"
$ bg
$ # làm công việc khác trong khi mpg123 vẫn tiếp tục chơi nhạc
```

#### 2.5.4 Chuyển tiến trình sang foreground

Ngược lại với phần trên, giả sử có những chương trình đang chạy ở **background** hoặc chương trình đang tạm dừng muốn chuyển về **foreground**. Ta có câu lệnh `fg`

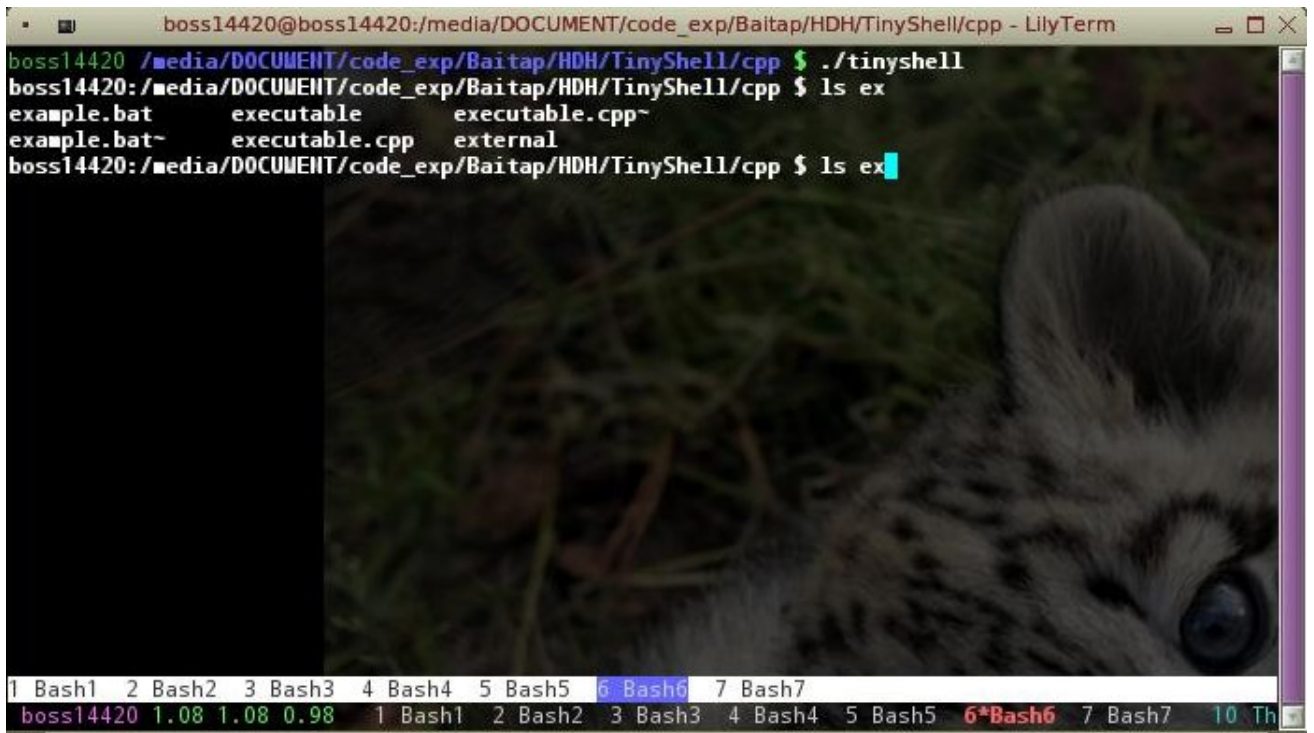
```
fg [job_id]
```

Trong đó `job_id` là chỉ số của tiến trình muốn chuyển về **foreground**. Nếu không chỉ ra `job_id` thì chọn tiến trình mặc định (tương tự lệnh `fg`). Ví dụ:

```
$ # dùng vim để soạn thảo mã nguồn
$ vim source.c
[1]+  Stopped          vim source.c
$ # tạm dừng để biên dịch và chạy thử
$ gcc source.c -o source -Wall -O2
$ ./source
...
$ # tiếp tục sửa file source.c
$ fg
```

## 2.6 Một số tiện ích khi gõ lệnh

Do **Tiny Shell** giao tiếp với người dùng chủ yếu qua lệnh nên có một số tính năng để công việc này nhẹ nhàng hơn:



Hình 2.2: Tab-completion

### 2.6.1 Tab-completion

Khi lệnh có chứa tên một đường dẫn có thực hoặc một file có trong thư mục hiện tại thì ta chỉ cần gõ một số ký tự đầu và sau đó nhấn phím Tab 2 lần, Tiny Shell sẽ tự động hoàn thành. Nếu có nhiều file có tên bắt đầu trùng với các ký tự đã gõ thì những tên file đó sẽ được in ra (Hình 2.2).

### 2.6.2 Lịch sử - History

Mỗi khi người dùng thực thi một lệnh bằng Tiny Shell (có thể lỗi) thì lệnh đó được lưu vào history. Người dùng sau này có thể gọi lại các lệnh đã gõ mà không cần phải gõ lại tất cả.

- Để hiển thị tất cả các câu lệnh đã thực thi, dùng lệnh `history`

```
$ ls
...
$ shfdjskj
$ history
1. ls
2. shfdjskj
$
```

- Dùng các phím mũi tên lên/xuống để thay câu lệnh đang gõ dở bằng các câu lệnh trước/sau.
- Để thực hiện câu lệnh thứ  $n$  trong history, dùng lệnh `!n`

- Để thực hiện câu lệnh thứ  $n$  tính từ câu lệnh hiện tại, ta dùng lệnh `!-n`. Ví dụ, để thực hiện câu lệnh gần nhất, ta dùng lệnh `!-1`.
- Nếu không nhớ thứ tự các câu lệnh, vẫn có thể gọi lại mà chỉ cần gõ một vài kí tự đầu. Lệnh `!<string>` sẽ thực thi câu lệnh gần nhất bắt đầu bằng `<string>`. Với ví dụ trên, lệnh `!h` tương đương gọi lại lệnh liệt kê lịch sử.
- ...

### 2.6.3 Chú thích - comment

Tất cả những kí tự kể từ `#` (không nằm trong dấu nháy đơn hoặc nháy kép) cho đến kết thúc dòng đều được Tiny Shell bỏ qua:

```
$ # Đây là một comment
$ echo " đây # không phải là một comment "
đây # không phải là một comment
$
```

## 2.7 Xử lí theo lô - batch processing

Thay vì gõ từng dòng lệnh, người dùng có thể xử thực thi một lúc nhiều lệnh khác nhau, những lệnh này được lưu vào trong một file script.

File script của Tiny Shell có cấu trúc tương tự các shell script khác: các lệnh được viết trên một dòng, có thể bao gồm các comment, có thể chạy nền hoặc ẩn. Ví dụ, file `example.tsh`:

```
#!/path/to/tinysHELL -f
# luôn phải bắt đầu bằng dòng như trên, trong đó /path/to/tinysHELL là đường
# dẫn đến file thực thi của Tiny Shell
ls -l
# build
$ gcc abc.c -o abc
```

Thực thi file này:

```
$ # thêm quyền thực thi
$ chmod +x example.tsh
$ # thực thi như một file script bình thường
$ ./example.tsh
$
$ # hoặc có thể gọi qua Tiny Shell, bằng cách này thì không cần file có quyền
$ # thực thi
$ tinysHELL -f example.tsh
```

File script Tiny Shell có thể thực thi trên các shell khác.

## 2.8 Gọi Tiny Shell từ các shell khác

Để thực thi một câu lệnh bằng Tiny Shell từ một shell khác, dùng lệnh:

```
$ /path/to/tinysHELL -c "<câu lệnh>"
$
$ # hoặc
$ /path/to/tinysHELL --command="<câu lệnh>"
$
```

Để thực thi file script:

```
$ /path/to/tinysHELL -f "<file>"
$
$ # hoặc
$ /path/to/tinysHELL --script="<file>"
$
```

Có thể gọi Tiny Shell ngay từ chính Tiny Shell.

# Chương 3

## Một số bài tập môn Hệ Điều Hành

### 3.1 Bài toán Producer – Consumer

Hệ thống gồm 2 tác vụ:

- **Producer** sản xuất ra các sản phẩm
- **Consumer** tiêu thụ các sản phẩm được sản xuất ra.

Vấn đề ở đây là phải có cơ chế để điều độ các tiến trình sản xuất/tiêu thụ sao cho các tiến trình tiêu thụ chỉ tiêu thụ khi đã có sản phẩm đã được sản xuất ra.

Trong các ví dụ minh họa dưới đây, **Producer** sẽ tạo ra một số nguyên ngẫu nhiên, **Consumer** sẽ in ra số nguyên đó.

#### 3.1.1 Trường hợp Producer/Consumer là các tiến trình (Process)

##### Message queue

Để giải quyết bài toán này, ta sử dụng **Message queue** của **System V API**.

**Message queue** là một bộ phận của **System V IPC (Inter-Process Communication) BLP**. Nó cho phép hai tiến trình không liên quan đến nhau có thể trao đổi những khối dữ liệu (**Message**) có cấu trúc cho nhau. Các **Message** này được nhận và gửi vào một hàng đợi do Hệ điều hành quản lý, mỗi hàng đợi có một chỉ số riêng gọi là **key** để nhận dạng. Số lượng các **Message** trong một hàng đợi là hạn chế, do đó tiến trình nào nếu gửi **Message** và hàng đợi khi hàng đợi đó đầy thì nó sẽ bị **block** một cách tự động bởi Hệ điều hành cho đến khi gửi được dữ liệu. Tương tự, tiến trình nào gọi lệnh nhận dữ liệu khi hàng đợi rỗng cũng bị **block** cho đến khi một tiến trình nào đó gửi **Message** vào hàng đợi.

Cấu trúc của một **Message** như sau:

```
1  struct message
2  {
3      long int message_type;
4      /* User-defined data */
5      ...
6  };
```

Trong đó trường đầu tiên luôn có kiểu **long int** là số nguyên chỉ tên kiểu **Message**, do đó 2 tiến trình có thể trao đổi nhiều loại **Message** khác nhau. Phần sau là tùy ý do người dùng định nghĩa.

Chi tiết về **Message queue**, xem ở trang manual của các hàm **msgget**, **msgctl**, **msgrcv**, **msgsnd**.

## Mã nguồn

Gồm 3 file: message.h, producer.h, consumer.h và Makefile.

```
1  /*
2  * =====
3  *
4  *      Filename:  message.h
5  *
6  *      Description:  Pre-defined constant
7  *
8  *      Version:  1.0
9  *      Created:  02/16/2012 08:23:44 PM
10 *      Revision:  none
11 *      Compiler:  gcc
12 *
13 *      Author:  BOSS14420 (boss14420), boss14420@gmail.com
14 *      Company:
15 *
16 * =====
17 */
18
19
20 #define KEY 1234
21 #define MAX 20
22
23 struct msg {
24     long int type;
25     int data;
26 };
```

```
1  /*
2  * =====
3  *
4  *      Filename:  producer.c
5  *
6  *      Description:  Producer
7  *
8  *      Version:  1.0
9  *      Created:  02/16/2012 08:12:56 PM
10 *      Revision:  none
11 *      Compiler:  gcc
12 *
13 *      Author:  BOSS14420 (boss14420), boss14420@gmail.com
14 *      Company:
15 *
16 * =====
17 */
18
19 #include <unistd.h>
```

```

20 #include <stdlib.h>
21 #include <stdio.h>
22 #include <time.h>
23 #include <errno.h>
24
25 #include <sys/msg.h>
26 #include <signal.h>
27
28 #include "message.h"
29
30 int msgid;
31
32 void signal_handler(int sig) {
33     msgctl(msgid, IPC_RMID, NULL);
34     fprintf(stderr, "Terminated!\n");
35     exit(EXIT_FAILURE);
36 }
37
38 int main() {
39     int current, sleeptime;
40
41     struct msg msg_to_snd;
42     msg_to_snd.type = 1;
43
44     msgid = msgget((key_t)KEY, 0666 | IPC_CREAT);
45
46     signal(SIGTERM, signal_handler);
47     signal(SIGINT, signal_handler);
48
49     srand(time(NULL));
50
51     do {
52         current = rand() % MAX;
53         sleeptime = rand() % 4;
54
55         printf("Product : %d\n", current);
56         msg_to_snd.data = current;
57         if(msgsnd(msgid, &msg_to_snd, sizeof(int), 0) == -1) {
58             perror("msgsnd: ");
59             msgctl(msgid, IPC_RMID, NULL);
60             exit(EXIT_FAILURE);
61         }
62         printf("sleeping...\n");
63         sleep(sleeptime);
64     } while(current);
65     printf("Exit.\n");
66     msgctl(msgid, IPC_RMID, NULL);
67
68     return EXIT_SUCCESS;

```

69 }

```
1  /*
2  * =====
3  *
4  *      Filename:  consumer.c
5  *
6  *      Description:
7  *
8  *          Version:  1.0
9  *          Created:  02/16/2012 08:56:07 PM
10 *          Revision:  none
11 *          Compiler:  gcc
12 *
13 *          Author:  BOSS14420 (boss14420), boss14420@gmail.com
14 *          Company:
15 *
16 * =====
17 */
18
19 #include <unistd.h>
20 #include <stdlib.h>
21 #include <stdio.h>
22 #include <time.h>
23 #include <errno.h>
24
25 #include <sys/msg.h>
26
27 #include "message.h"
28
29 int main() {
30     int msgid;
31
32     struct msg msg_to_recv;
33     msg_to_recv.type = 1;
34
35     msgid = msgget((key_t)KEY, 0666 | IPC_CREAT);
36
37     do {
38         if(msgrcv(msgid, &msg_to_recv, sizeof(int), 1, 0) == -1) {
39             perror("msgrcv: ");
40             exit(EXIT_FAILURE);
41         }
42         printf("Consuming: %d\n", msg_to_recv.data);
43     } while(msg_to_recv.data);
44     printf("Exit.\n");
45
46     msgctl(msgid, IPC_RMID, 0);
47
```



```

48     return EXIT_SUCCESS;
49 }

```

```

1 all: producer consumer
2 CC = gcc
3
4 INCLUDE = .
5 CFLAGS = -Wall -O2
6
7 producer: producer.c message.h
8     $(CC) -I$(INCLUDE) $(CFLAGS) -o producer producer.c
9
10 consumer: consumer.c message.h
11     $(CC) -I$(INCLUDE) $(CFLAGS) -o consumer consumer.c

```

Biên dịch bằng cách gõ lệnh `make`.

## Chạy thử

Trên hai cửa sổ dòng lệnh khác nhau, chạy lần lượt 2 chương trình `producer`, `consumer`.

<pre> \$ ./producer Product : 6 sleeping... Product : 16 sleeping... Product : 14 sleeping... Product : 16 sleeping... Product : 19 sleeping... Product : 4 sleeping... ^CTerminated! \$ </pre>	<pre> \$ ./consumer Consuming: 6 Consuming: 16 Consuming: 14 Consuming: 16 Consuming: 19 Consuming: 4 msgrcv: : Identifier removed \$ </pre>
---	--

## Nhận xét

- Mỗi lần tạo ra một số nguyên thì tiến trình `producer` tạm dừng trong vài giây, tiến trình `consumer` theo đó cũng bị block (do chưa có số nguyên mới được tạo ra).
- Với một `producer`, có thể chạy cùng một lúc nhiều `consumer`.
- Nếu người dùng cho dừng tiến trình `producer` bằng tổ hợp phím `Ctrl+C` thì tiến trình `consumer` báo lỗi và kết thúc theo.

### 3.1.2 Trường hợp Producer/Consumer là các luồng (Thread)

Với trường hợp này, ta dùng `Blocking queue` để giải quyết bài toán.

## Blocking queue

Blocking queue là một Collection của Java API, có đầy đủ các chức năng của một queue và có thêm khả năng đồng bộ hoá giữa các luồng sử dụng nó. Cụ thể, nếu queue là rỗng thì một luồng lấy dữ liệu từ nó bằng hàm `take()` sẽ bị block cho đến khi có luồng khác đưa dữ liệu vào. Với Bounded Blocking queue thì khi đầy, những luồng đưa dữ liệu vào queue đều bị block. JavaAPIBQ

## Cài đặt

Đây là thành phần thực hiện chính của luồng Producer:

```
1  do {
2      try {
3          value = producer.nextInt() % 0xf;
4          queue.put(value);
5          jTextArea1.append("producing: " + String.valueOf(value)+"\n");
6          sleepTime = Math.abs(producer.nextInt() % 4);
7          Thread.sleep(500*sleepTime);
8      } catch (InterruptedException ex) {
9          Logger.getLogger(Producer.class.getName()).
10             log(Level.SEVERE, null, ex);
11      }
12 } while(value != 0);
```

Consumer:

```
1  int value = 1;
2  do {
3      try {
4          value = queue.take();
5          jTextArea1.append("consuming: " + String.valueOf(value) + "\n");
6      } catch (InterruptedException ex) {
7          Logger.getLogger(Consumer.class.getName()).
8             log(Level.SEVERE, null, ex);
9      }
10 } while(value != 0);
```

Mã nguồn đầy đủ của chương trình (project netbeans) trong thư mục ProducerConsumer/java.

## Chạy thử chương trình