

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 hướng dẫn :

Phạm Đăng Hải

HÀ NỘI

Ngày 13 tháng 5 năm 2012

Mục lục

1	Lời nói đầu	4
2	Chương trình shell đơn giản (Tiny Shell) cho POSIX	5
2.1	Giới thiệu	5
2.1.1	Tinyshell	5
2.1.2	Hướng dẫn cài đặt	5
2.2	Khởi động chương trình	6
2.3	Gọi lệnh	6
2.4	Lệnh nâng cao	7
2.5	Job control	8
2.5.1	Background/foreground	8
2.5.2	Liệt kê các tiến trình nền	9
2.5.3	Chuyển từ foreground sang background	9
2.5.4	Chuyển tiến trình sang foreground	10
2.5.5	Kết thúc một tiến trình	11
2.6	Một số tiện ích khi gõ lệnh	11
2.6.1	Tab-completion	11
2.6.2	Lịch sử - History	11
2.6.3	Chú thích - comment	12
2.7	Các lệnh có sẵn của Tiny Shell	12
2.7.1	Thay đổi thư mục hiện tại - lệnh cd	13
2.7.2	Xem thông tin về các lệnh khác - lệnh help	13
2.8	Xử lý theo lô - batch processing	13
2.9	Gọi Tiny Shell từ các shell khác	14
3	Một số bài tập môn Hệ Điều Hành	15
3.1	Bài toán Producer - Consumer	16
3.1.1	Trường hợp Producer/Consumer là các tiến trình (Process)	16
3.1.2	Trường hợp Producer/Consumer là các luồng (Thread)	20
3.2	Chương trình chat	22
3.2.1	Chương trình chat console	22
3.2.2	Chương trình chat bằng giao diện đồ hoạ	26
3.3	Bài toán người đọc - người viết (ReaderWriter)	28
3.3.1	Phát biểu bài toán	28
3.3.2	Cách giải đề xuất	28
3.3.3	Cách giải tốt hơn	29
3.3.4	Mã nguồn	30
3.3.5	Kết quả chạy thử	32
3.4	Đọc MBR	34

3.4.1	Mã nguồn	34
3.4.2	Kết quả	38
4	Tài liệu tham khảo	40

Chương 1

Lời nói đầu

Trong bản báo cáo này, em sẽ trình bày về chương trình shell đơn giản **Tiny Shell** và một số bài tập môn hệ điều hành khác. Những bài tập khác không trình bày ở đây nhưng vẫn có code ở file đính kèm.

Những chương trình C/C++ kèm theo được biên dịch trên hệ điều hành Linux 64 bit nên có thể phải biên dịch lại nếu không tương thích.

Dù đã rất cố gắng nhưng do khả năng còn hạn chế nên không thể không có sai sót, rất mong được thầy tận tình chỉ bảo. Em xin chân thành cảm ơn!

Sinh viên thực hiện: Nguyễn Duy Thành

Email:

boss14420@gmail.com

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 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,
- trình dịch: tốt nhất là `gcc` \geq 4.6, với các trình dịch khác cần phải sửa file `CMakeLists.txt`.

Với các Linux distro họ Debian (Debian, Ubuntu, Mint, ...): Cài đặt một trong những gói phần mềm (file .deb) đi kèm, tùy loại kiến trúc là 32bit và 64bit. Ví dụ Ubuntu 32bit, nhấn đúp chuột vào file `tinysHELL_0.1-1_i386.deb` để vào chương trình cài đặt gói, hoặc dùng lệnh:

```
1 $ sudo dpkg -i tinysHELL_0.1-1_i386.deb
```

Với các hệ điều hành khác: Cần phải biên dịch lại từ mã nguồn. Các bước tiến hành như sau:

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

```
1 $ 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 và cài đặt,** gõ lệnh:

```
1 $ make
2 $ sudo make install # can quyên root
```

file thực thi `tinysHELL` sẽ được tạo ra và sao chép vào thư mục `/usr/bin`.

2.2 Khởi động chương trình

Mở chương trình, gõ lệnh:

```
1 $ tinysHELL
```

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

Để kết thúc Tiny Shell, dùng tổ hợp phím `Ctrl+D` hoặc gõ lệnh `quit`.

2.3 Gõ lệnh

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

```
1 $ ls -liah # thuc thi chuong trinh ngoai
2 $ ./abc.py # thuc thi file script python
```

Chú ý:

```
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 $
1 Bash1 2 Bash2 3 Bash3 4 Bash4 5 Bash5 6 Bash6 7 Bash7
boss14420 0.71 0.72 0.76 1 Bash1 2 Bash2 3 Bash3 4 Bash4 5 Bash5 6 Bash6 7 Bash7 09 We
```

Hình 2.1: Tiny Shell

- 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 bad command.
- 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 permission denied. chỉ có owner của file mới được quyền thực thi.
- File script phải bắt đầu bằng 2 kí 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:

```
1 #!/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. [2]. Ví dụ:

```

1 $ # liệt kê thu mục gốc của người dùng boss14420
2 $ ls ~boss14420
3 ...
4 $ # xem biến môi trường $PATH
5 $ echo $PATH
6 ...
7 $ # Phép tính số học
8 $ echo $(( 2*3 ))
9 ...
10 $ # tạm dừng tiến trình firefox
11 $ kill -STOP 'pidof firefox'
12 ...
13 $ # liệt kê những file có phần mở rộng là .c
14 $ ls *.c
15 $ # xóa những file có dạng abc.<chữ số> trong thư mục
16 $ rm abc.[0-9]

```

(Chú ý, kí tự ‘ tương ứng với phím ở bên trái phím số 1 trên bàn phím).

2.5 Job control

Tiny Shell có một số tính năng của một Job Control shell [1].

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:

```

1 $ ./external
2 Sleeping...
3 Waked!
4 $
5 $ ./external &
6 $ Sleeping...
7
8 $ ls
9 autogen.sh          CMakeFiles          CMakeLists.txt      configure.ac
10 example.bat~        executable.cpp       external             Makefile.am
11 tinysHELL           CMakeCache.txt      cmake_install.cmake
12 CMakeLists.txt~     example.bat          executable           executable.cpp~
13 Makefile            src
14 $ Waked!
15

```



```

16 [1]      Done      ./external
17 $

```

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:

```

1 [<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:

```

1 $ vim
2 [1]+  Stopped      vim
3 $ less example.bat
4 [2]+  Stopped      less example.bat
5 $ ./external 6 &
6 $ Sleeping...
7
8 $ jobs
9 [1]-  Stopped      vim
10 [2]+  Stopped      less example.bat
11 [3]   Running      ./external 6
12 $

```

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:

```
1 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:

```
1 $ mpg123 "abcde.mp3"
2 High Performance MPEG 1.0/2.0/2.5 Audio Player for Layers 1, 2 and 3
3 version 1.14.1; written and copyright by Michael Hipp and others
4 free software (LGPL/GPL) without any warranty but with best wishes
5
6 Playing MPEG stream 1 of 1: abcde.mp3 ...
7
8 MPEG 1.0 layer III, 192 kbit/s, 44100 Hz joint-stereo
9 Title:      dam Ma (Remix)           Artist: DJ
10 Comment:   NhacCuaTui.Com - Nghe Nhac Moi Luc Moi Noi
11 Album:     NhacCuaTui.Com
12 Year:      2011                     Genre:  The Loai Khac
13 ^Z[2]+  Stopped          mpg123 "abcde.mp3"
14 $ bg
15 $ # lam cong viec khac trong khi mpg123 van tiep 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

```
1 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:

```
1 $ # dung vim de soan thao ma nguon
2 $ vim source.c
3 [1]+  Stopped          vim source.c
4 $ # tam dung de bien dich va chay thu
5 $ gcc source.c -o source -Wall -O2
6 $ ./source
7 ...
8 $ # tiep tục sửa file source.c
9 $ fg
```

2.5.5 Kết thúc một tiến trình

Tiny Shell cung cấp lệnh `kill` dùng để gửi một tín hiệu (signal) đến một tiến trình. Chức năng của lệnh `kill` giống như lệnh `kill` của GNU Binutils nhưng có thêm tính năng gửi signal đến các tiến trình nền của Tiny Shell dựa trên `job_id` (bằng cách thêm % vào trước chỉ số) thay vì `pid`. Ví dụ:

```
1 $ ./external 10 &
2 $ Sleeping...
3
4 $ jobs
5 [1]      Running      ./external 10
6 $ kill %1
7 [1]      Done         ./external 10
8 $
9 $ # Tiếp tục tiến trình khác
10 $ kill -CONT 'pidof firefox'
11 $
```

Thông tin về cú pháp của lệnh `kill` xem thêm ở manual (dùng lệnh `man 1 kill`).

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:

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 kí 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).

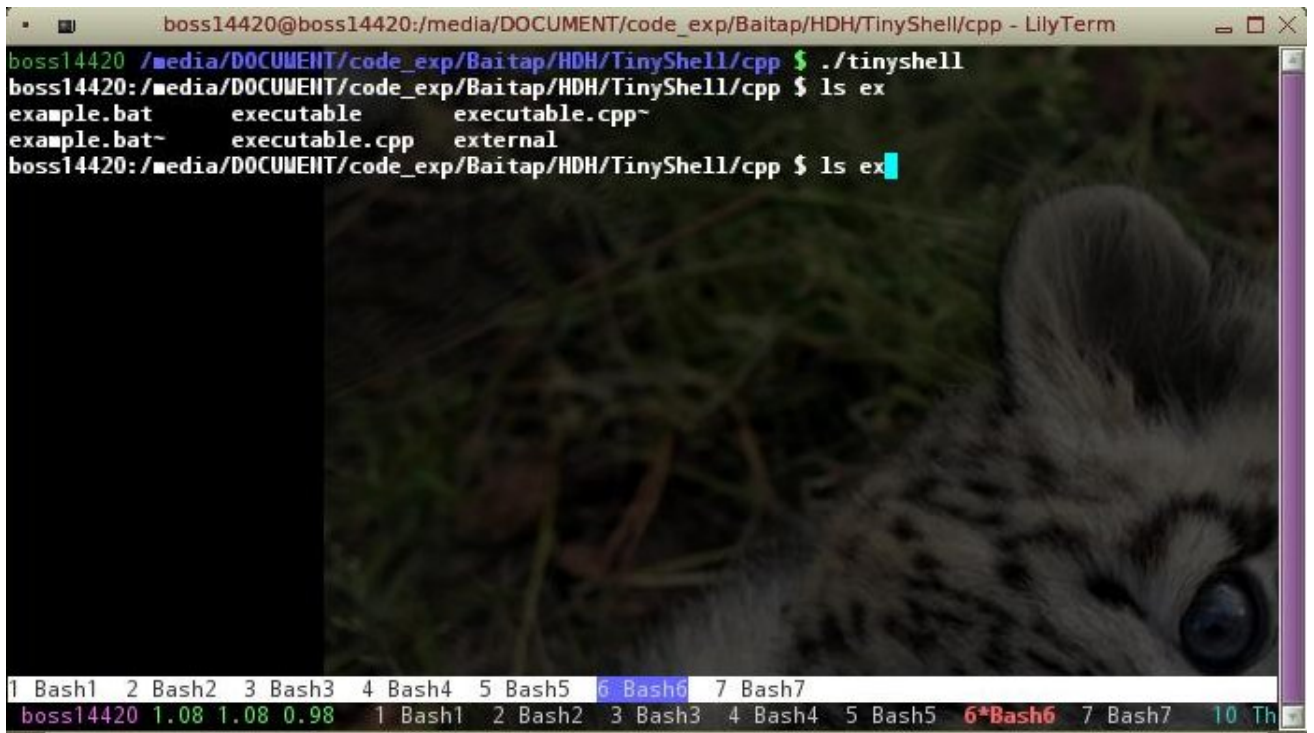
2.6.2 Lịch sử - History

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`

```
1 $ ls
2 ...
3 $ shfdjskj
4 $ history
5 1. ls
6 2. shfdjskj
7 $
```

- Dùng các phím mũi tên lên/xuống để thay câu lệnh đang gõ dở bằng các câu lệnh trước/sau.



Hình 2.2: Tab-completion

- 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:

```

1 $ # Day la mot comment
2 $ echo " day # khong phai la mot comment "
3 day # khong phai la mot comment
4 $

```

2.7 Các lệnh có sẵn của Tiny Shell

Ngoài các lệnh `kill`, `history`, `fg`, `bg`, `jobs` đã đề cập, Tiny Shell còn tích hợp một số lệnh khác:

2.7.1 Thay đổi thư mục hiện tại - lệnh cd

Cú pháp lệnh cd như sau:

```
1 cd [new_dir]
```

`new_dir` là thư mục sẽ chuyển đến.

Trong đó:

- Nếu không chỉ ra `new_dir` thì sẽ chuyển đến thư mục gốc của người dùng hiện tại,
- Nếu `new_dir` là `-` thì thư mục được chuyển tới là thư mục trước đó (thư mục trước khi chuyển đến thư mục hiện tại),
- Nếu `new_dir` là `..` thì chuyển đến thư mục cha của thư mục hiện tại

Ví dụ:

```
1 boss14420:/media/DATA/Music $ cd ../Video
2 boss14420:/media/DATA/Video $
3 boss14420:/media/DATA/Video $ # Ve thu muc goc
4 boss14420:/media/DATA/Video $ cd
5 boss14420:/home/boss14420 $
6 boss14420:/home/boss14420 $ # Den thu muc cha
7 boss14420:/home/boss14420 $ cd ..
8 boss14420:/home $
9 boss14420:/home $ # tro ve thuc muc goc
10 boss14420:/home $ cd -
11 boss14420:/home/boss14420 $
```

2.7.2 Xem thông tin về các lệnh khác - lệnh help

Để xem danh sách tất cả các lệnh tích hợp sẵn của Tiny Shell, dùng lệnh `help`.

Để xem thông tin về một lệnh nào đó (ví dụ `cd`), dùng lệnh:

```
1 help cd
```

2.8 Xử lý 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`:

```
1 #!/usr/bin/tinysHELL -f
2 # luon phai bat dau bang dong nhu tren
3 ls -l
4 # build
5 $ gcc abc.c -o abc
```

Thực thi file này:

```
1 $ # them quyền thực thi
2 $ chmod +x example.tsh
3 $ # thực thi như một file script bình thường
4 $ ./example.tsh
5 $
6 $ # hoặc có thể gọi qua Tiny Shell, bằng cách này thì không cần file có quyền
7 $ # thực thi
8 $ tinysHELL -f example.tsh
```

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

2.9 Gọi Tiny Shell từ các shell khác

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

```
1 $ tinysHELL -c "<command>"
2 $
3 $ # học
4 $ tinysHELL --command="<command>"
5 $
```

Để thực thi file script:

```
1 $ tinysHELL -f "<file>"
2 $
3 $ # hoặc
4 $ tinysHELL --script="<file>"
5 $
```

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)** [3, p. 594]. 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** vào 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)

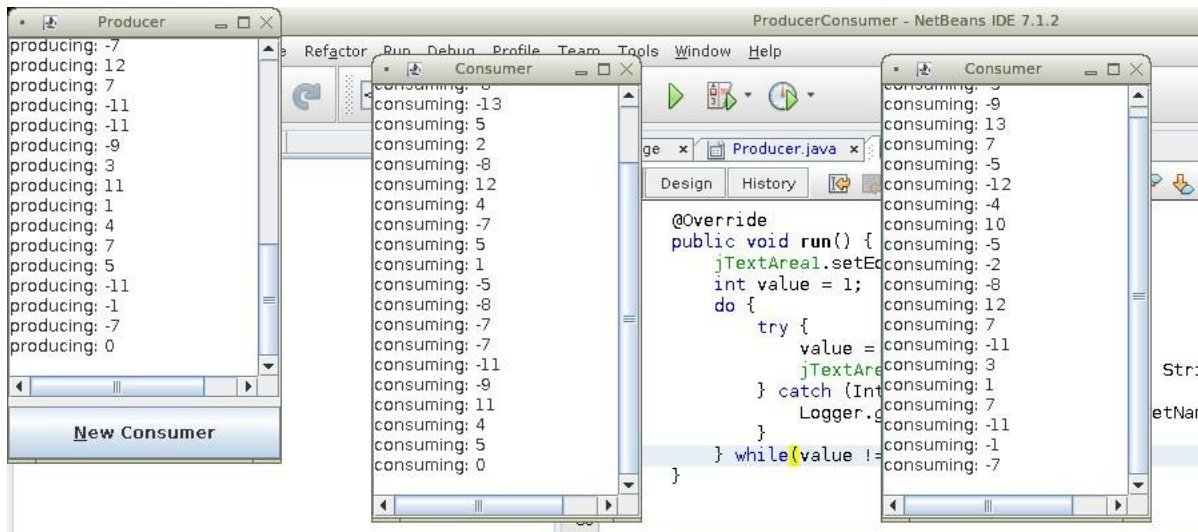
Với trường hợp này, ta dùng `Blocking queue` để giải quyết bài toán.

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`. [4]

Cài đặt

Đây là thành phần thực hiện chính của luồng `Producer`:



Hình 3.1: Producer-Consumer

```

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

Hình 3.1 là kết quả chạy thử chương trình với 1 producer và 2 consumer.

3.2 Chương trình chat

Ở đây, ta sẽ tạo hai tiến trình trao đổi các đoạn `text` ngắn (dưới 255 kí tự) với nhau. Ban đầu, một tiến trình đóng vai trò là `server` có nhiệm vụ lắng nghe yêu cầu kết nối `quasocket` của tiến trình `client`. Sau khi kết nối được thiết lập, hai tiến trình bắt đầu chat, lúc này vai trò của chúng bình đẳng với nhau.

3.2.1 Chương trình chat console

Mã nguồn

Mỗi tiến trình gồm 4 luồng, trong đó có 2 luồng chính: một luồng `Writer` dùng để nhận dữ liệu từ người dùng và gửi qua `socket` và một luồng `Reader` dùng để lắng nghe và nhận câu hội thoại từ `socket`.

```
1  /*
2   * To change this template, choose Tools / Templates
3   * and open the template in the editor.
4   */
5  package chatconsole2;
6
7  /**
8   *
9   * @author boss14420
10  */
11  import java.io.*;
12  import java.net.ServerSocket;
13  import java.net.Socket;
14  import java.util.Scanner;
15  import java.util.logging.Level;
16  import java.util.logging.Logger;
17
18  class ChatConsole2 extends Thread {
19      //SocketChannel sChannel;
20
21      Socket rSocket, wSocket;
22      Reader reader;
23      Writer writer;
24
25      ChatConsole2(Socket rSocket, Socket wSocket) {
26          this.rSocket = rSocket;
27          this.wSocket = wSocket;
28          try {
29              reader = new Reader(rSocket.getInputStream());
30              writer = new Writer(wSocket.getOutputStream());
31          } catch (Exception ex) {
32              Logger.getLogger(ChatConsole2.class.getName())
33                  .log(Level.SEVERE, null, ex);
34          }
35      }
```

```

36     }
37
38     @Override
39     public void run() {
40         reader.start();
41         writer.start();
42
43         try {
44             reader.join();
45             writer.join();
46         } catch (Exception ex) {
47             Logger.getLogger(ChatConsole2.class.getName())
48                 .log(Level.SEVERE, null, ex);
49         }
50     }
51
52     private class Reader extends Thread {
53
54         ObjectInputStream ois;
55         InputStream is;
56
57         public Reader(ObjectInputStream ois) {
58             this.ois = ois;
59         }
60
61         private Reader(InputStream inputStream) {
62             is = inputStream;
63         }
64
65         @Override
66         public void run() {
67             String message = "";
68             byte[] cbuf;
69             int len;
70             do {
71                 try {
72                     //message = (String) ois.readObject();
73                     len = is.read();
74                     cbuf = new byte[len];
75                     is.read(cbuf, 0, len);
76
77                     message = new String(cbuf);
78                     System.out.println("Received : " + message);
79                 } catch (IOException ex) {
80                     Logger.getLogger(Reader.class.getName())
81                         .log(Level.SEVERE, null, ex);
82                     if (rSocket.isClosed()) {
83                         System.exit(-1);
84                     }

```

```

85         }
86
87         } while (!message.equals("Quit!"));
88         System.exit(0);
89     }
90 }
91
92 private class Writer extends Thread {
93
94     ObjectOutputStream oos;
95     OutputStream os;
96
97     public Writer(ObjectOutputStream oos) {
98         this.oos = oos;
99     }
100
101     private Writer(OutputStream outputStream) {
102         os = outputStream;
103     }
104
105     @Override
106     public void run() {
107         Scanner scanner;
108         try {
109             scanner = new Scanner(System.in);
110         } catch (Exception ex) {
111             Logger.getLogger(Writer.class.getName())
112                 .log(Level.SEVERE, null, ex);
113             return;
114         }
115
116         String message = "";
117         byte[] cbuf;
118         do {
119             try {
120                 message = scanner.nextLine();
121                 cbuf = message.getBytes();
122                 //oos.writeObject(message);
123
124                 os.write(cbuf.length);
125                 os.write(cbuf, 0, cbuf.length);
126
127                 System.out.println("Sent : " + message);
128             } catch (Exception ex) {
129                 Logger.getLogger(Writer.class.getName())
130                     .log(Level.SEVERE, null, ex);
131                 if (wSocket.isClosed()) {
132                     System.exit(-1);
133                 }

```



```

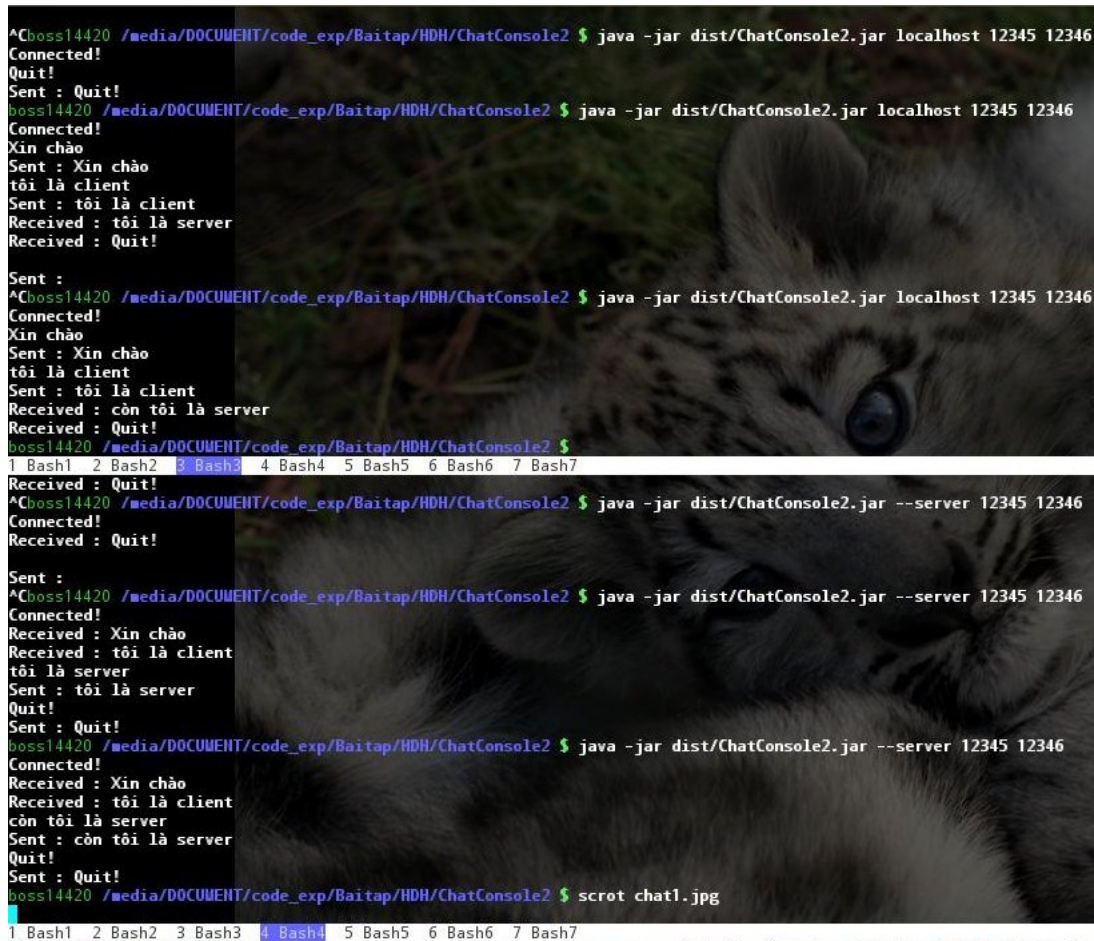
134         }
135     } while (!message.equals("Quit!"));
136     System.exit(0);
137 }
138 }
139
140 public static void main(String[] args) {
141     try {
142         //SocketChannel sc;
143         Socket rSocket, wSocket;
144         if (args[0].equals("--server")) {
145             int rport = Integer.parseInt(args[1]);
146             int wport = Integer.parseInt(args[2]);
147
148             ServerSocket rsSocket = new ServerSocket(rport);
149             ServerSocket wsSocket = new ServerSocket(wport);
150
151             rSocket = rsSocket.accept();
152             wSocket = wsSocket.accept();
153
154         } else {
155             String hostname = args[0];
156             int rport = Integer.parseInt(args[2]);
157             int wport = Integer.parseInt(args[1]);
158
159             rSocket = new Socket(hostname, rport);
160             wSocket = new Socket(hostname, wport);
161         }
162
163         System.out.println("Connected!");
164
165         ChatConsole2 c = new ChatConsole2(rSocket, wSocket);
166         c.start();
167
168         c.join();
169         rSocket.close();
170         wSocket.close();
171
172     } catch (Exception ex) {
173         ex.printStackTrace();
174     }
175 }
176 }

```

Chạy chương trình

Ở thư mục gốc của project ChatConsole2, gõ lệnh:

- Khởi động server



```
^Cboss14420 /media/DOCUMENT/code_exp/Baitap/HDH/ChatConsole2 $ java -jar dist/ChatConsole2.jar localhost 12345 12346
Connected!
Quit!
Sent : Quit!
boss14420 /media/DOCUMENT/code_exp/Baitap/HDH/ChatConsole2 $ java -jar dist/ChatConsole2.jar localhost 12345 12346
Connected!
Xin chào
Sent : Xin chào
tôi là client
Sent : tôi là client
Received : tôi là server
Received : Quit!

Sent :
^Cboss14420 /media/DOCUMENT/code_exp/Baitap/HDH/ChatConsole2 $ java -jar dist/ChatConsole2.jar localhost 12345 12346
Connected!
Xin chào
Sent : Xin chào
tôi là client
Sent : tôi là client
Received : còn tôi là server
Received : Quit!
boss14420 /media/DOCUMENT/code_exp/Baitap/HDH/ChatConsole2 $
1 Bash1 2 Bash2 3 Bash3 4 Bash4 5 Bash5 6 Bash6 7 Bash7
Received : Quit!
^Cboss14420 /media/DOCUMENT/code_exp/Baitap/HDH/ChatConsole2 $ java -jar dist/ChatConsole2.jar --server 12345 12346
Connected!
Sent : Quit!

Sent :
^Cboss14420 /media/DOCUMENT/code_exp/Baitap/HDH/ChatConsole2 $ java -jar dist/ChatConsole2.jar --server 12345 12346
Connected!
Received : Xin chào
Received : tôi là client
tôi là server
Sent : tôi là server
Quit!
Sent : Quit!
boss14420 /media/DOCUMENT/code_exp/Baitap/HDH/ChatConsole2 $ java -jar dist/ChatConsole2.jar --server 12345 12346
Connected!
Received : Xin chào
Received : tôi là client
còn tôi là server
Sent : còn tôi là server
Quit!
Sent : Quit!
boss14420 /media/DOCUMENT/code_exp/Baitap/HDH/ChatConsole2 $ scrot chat1.jpg
1 Bash1 2 Bash2 3 Bash3 4 Bash4 5 Bash5 6 Bash6 7 Bash7
```

Hình 3.2: Chat console

```
1 $ java -jar --server <read-port> <write-port>
```

Trong đó <read-port> là cổng (port) để server nhận dữ liệu, <write-port> là cổng để gửi dữ liệu.

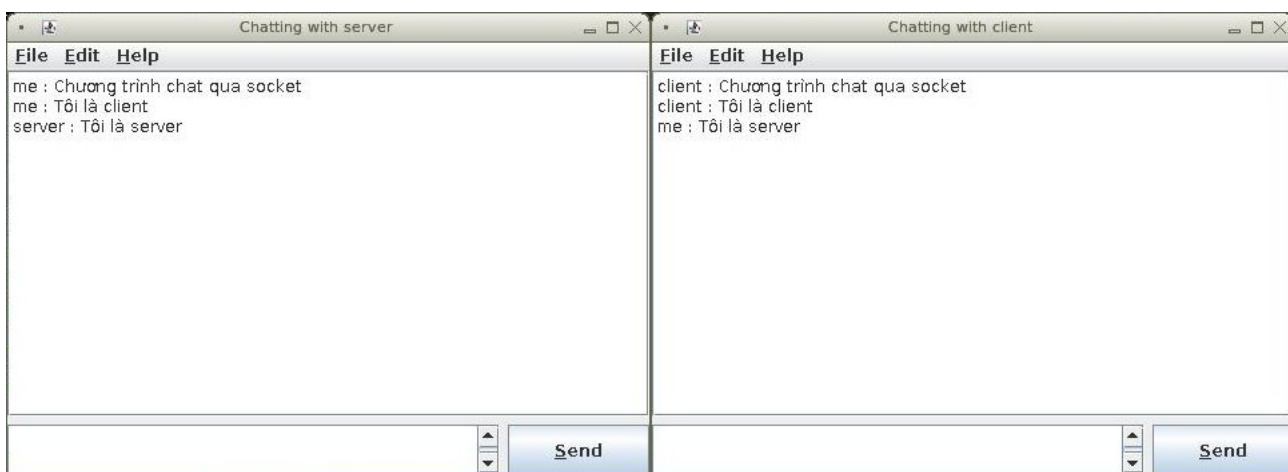
- Khởi động client

```
1 $ java -jar <hostname> <read-port> <write-port>
```

Trong đó hostname là địa chỉ của máy chạy tiến trình server, <read-port> <write-port> giống như trên.

3.2.2 Chương trình chat bằng giao diện đồ hoạ

Cách sử dụng tương tự. Project của chương trình nằm trong thư mục Chat.



Hình 3.3: Chat GUI

3.3 Bài toán người đọc - người viết (ReaderWriter)

3.3.1 Phát biểu bài toán

- Nhiều tiến trình (Readers) cùng truy nhập một cơ sở dữ liệu (CSDL)
- Một số tiến trình (Writers) cập nhật cơ sở dữ liệu
- Cho phép số lượng tùy ý các tiến trình Readers cùng truy nhập CSDL
 - Đang tồn tại một tiến trình Reader truy cập CSDL, mọi tiến trình Readers khác mới xuất hiện đều được truy cập CSDL (Tiến trình Writers phải xếp hàng chờ đợi)
- Chỉ cho phép một tiến trình Writers cập nhật CSDL tại một thời điểm.
- Vấn đề không tương dụng. Các tiến trình ở trong đoạn găng mà không bị ngắt

[5]

3.3.2 Cách giải đề xuất

- với tiến trình Writer, nếu muốn vào đoạn găng để đọc CSDL thì nó cần phải kiểm tra có Reader hoặc Writer nào đó đang ở trong đoạn găng hay không. Nếu có thì nó cần phải bị block cho đến khi các tiến trình khi ra khỏi đoạn găng,
- tiến trình Reader muốn vào đọc CSDL không cần quan tâm có bao nhiêu tiến trình khác cũng đang đọc mà chỉ cần quan tâm đến có tiến trình khác đang ghi CSDL. Do đó, trước khi vào đoạn găng, tiến trình Reader cần phải kiểm tra có tiến trình Writer đang ở trong đoạn găng hay không, nếu không nó sẽ bị block cho đến khi tiến trình Writer ra khỏi đoạn găng. Khi Reader ra khỏi đoạn găng, nó cần kiểm tra xem có Reader khác đang ở trong đoạn găng hay không, nếu không có thì đây đã là Reader cuối cùng, các Writer có thể vào đoạn găng lúc này.

```
1 readcount = 0 # biến đếm số lượng reader muốn vào đoạn găng
2 def reader():
3     reader_mutex.lock() # khóa reader_mutex để cập nhật biến đếm
4     readcount = readcount+1
5     if readcount == 1:
6         # nếu là reader đầu tiên vào đoạn găng thì khóa writer
7         writer_mutex.lock()
8     reader_mutex.unlock()
9
10    # bắt đầu đoạn găng
11    ...
12    # kết thúc đoạn găng
13
14    # cập nhật lại biến đếm
15    reader_mutex.lock()
16    readcount = readcount - 1
17    if readcount == 0:
18        # Nếu là reader cuối cùng ra khỏi đoạn găng thì mở khóa cho writer
```

```

19     writer_mutex.unlock()
20     reader_muter.unlock()
21
22 def writer():
23     # khoa writer_mutex, neu thread bi block o day chung to co mot
24     # writer hoac reader nao do da khoa no truoc do
25     writer_mutex.lock()
26
27     # bat dau doan gang
28     ...
29     # ket thuc doan gang
30
31     writer_mutex.unlock()

```

Điều độ các tiến trình bằng cách như trên có thể gây ra vấn đề: tiến trình **Writer** có thể bị đói (starvation). Nguyên do là với tiến trình **Writer** muốn vào đoạn găng thì cần đợi tất cả tiến trình đang ở trong đoạn găng, trong khi tiến trình **Reader** chỉ cần phải đợi tiến trình **Writer**. Điều này có thể dẫn tới việc một tiến trình **Writer** phải đợi rất nhiều **Reader** đang ở trong đoạn găng, và trong khi chờ đợi nếu có các **Reader** khác xuất hiện thì thời gian chờ đợi lại tăng thêm.

3.3.3 Cách giải tốt hơn

Để giải quyết vấn đề ở trên, ta có thể thay đổi chiến lược điều độ tiến trình như sau:

- Nếu tiến trình đang ở trong đoạn găng là **reader** thì các tiến trình **writer** đến sau vẫn phải đợi, nhưng nó được ưu tiên hơn các tiến **reader** đến sau nó. Tức là kể từ khi tiến trình **writer** thứ nhất vào hàng đợi thì tiến trình **reader** đến sau không được vào đoạn găng như trước nữa mà phải đợi tiến trình **writer** này vào và sau đó ra khỏi đoạn găng.

Như vậy, một tiến trình **writer** chỉ phải đợi các **reader** và **writer** đến trước nó chứ không phải đợi các **reader** đến sau như trước cách làm trước nữa.

Để thực hiện điều này, ta dùng một biến mutex **turnstile**.

- Khi một **reader** muốn vào đoạn găng, nó khoá và mở **turnstile** ngay lập tức. Nếu không bị block thì tức là chưa có **writer** nào đang ở trong đoạn găng hoặc hàng đợi, các **reader** hoặc **writer** đến sau có thể khoá **turnstile**. Ngược lại, tiến trình này phải đợi cho đến khi **writer** kết thúc đoạn găng mới tiếp tục công việc tiếp theo được,
- Khi một **writer** muốn vào đoạn găng, nó khoá **turnstile** và thực hiện công việc như trước. Khi ra khỏi đoạn găng thì mở khoá **turnstile** để các tiến trình khác có thể tiếp tục.

```

1 def reader():
2     turnstile.lock()
3     turnstile.unlock();
4
5     ...
6
7 def writer():

```

```

8      turnstile.lock()
9
10     ...
11
12     turnstile.unlock()

```

3.3.4 Mã nguồn

```

1  /*
2  * =====
3  *
4  *      Filename:  readerwriter.cc
5  *
6  *      Description:
7  *
8  *          Version:  1.0
9  *          Created:  03/04/2012 05:17:47 PM
10 *          Revision:  none
11 *          Compiler:  gcc
12 *
13 *          Author:   BOSS14420 (boss14420), boss14420@gmail.com
14 *          Company:
15 *
16 * =====
17 */
18
19 #include <iostream>
20 #include <thread>
21 #include <chrono>
22 #include <mutex>
23 #include <condition_variable>
24 #include <random>
25 #include <deque>
26
27 int readcount = 0, waittime = 0;
28 std::mutex writer_mutex, reader_mutex, turnstile;
29 std::uniform_int_distribution<int> uid(1, 6);
30 std::random_device rd;
31
32 void reader_routine(int id) {
33     std::chrono::seconds read_time(uid(rd));
34
35     // reader thread is blocked if a prev writer thread lock this mutex
36     turnstile.lock();
37     turnstile.unlock();
38
39     reader_mutex.lock();
40     if(1 == ++readcount)
41         writer_mutex.lock();

```

```

42     reader_mutex.unlock();
43
44     std::cout << "reader #" << id << " entered" << std::endl;
45     std::this_thread::sleep_for(read_time);
46     std::cout << "reader #" << id << " exit" << std::endl;
47
48     reader_mutex.lock();
49     if(0 == --readcount)
50         writer_mutex.unlock();
51     reader_mutex.unlock();
52 }
53
54 void writer_routine(int id) {
55     std::chrono::seconds write_time(uid(rd));
56
57     turnstile.lock();
58     writer_mutex.lock();
59     std::cout << "writer #" << id << " entered" << std::endl;
60     std::this_thread::sleep_for(write_time);
61     std::cout << "writer #" << id << " exit" << std::endl;
62     turnstile.unlock();
63
64     writer_mutex.unlock();
65 }
66
67 int main() {
68     int reader_count = -1, writer_count = -1;
69     std::chrono::milliseconds sleep_time(400);
70
71     std::deque<std::thread> thrds;
72
73     while(1) {
74         if(uid(rd) % 2 == 0)
75             thrds.emplace_back(reader_routine, ++reader_count);
76         else
77             thrds.emplace_back(writer_routine, ++writer_count);
78
79         std::this_thread::sleep_for(sleep_time);
80     }
81
82     for(std::thread& th : thrds)
83         th.join();
84
85     return 0;
86 }

```

Biên dịch:

```
1 $ g++ readerwritex.cc -o readerwriter-cc -std=gnu++0x -pthread
```

3.3.5 Kết quả chạy thử

```
1 reader #0 entered
2 reader #0 exit
3 writer #0 entered
4 writer #0 exit
5 reader #1 entered
6 reader #1 exit
7 writer #1 entered
8 writer #1 exit
9 reader #2 entered
10 reader #3 entered
11 reader #4 entered
12 reader #4 exit
13 reader #2 exit
14 reader #3 exit
15 writer #2 entered
16 writer #2 exit
17 writer #3 entered
18 writer #3 exit
19 writer #4 entered
20 writer #4 exit
21 reader #5 entered
22 reader #6 entered
23 reader #5 exit
24 reader #6 exit
25 writer #5 entered
26 writer #5 exit
27 reader #7 entered
28 reader #8 entered
29 reader #9 entered
30 reader #10 entered
31 reader #8 exit
32 reader #7 exit
33 reader #10 exit
34 reader #9 exit
35 writer #6 entered
36 writer #6 exit
37 reader #11 entered
38 reader #11 exit
39 writer #7 entered
40 writer #7 exit
41 reader #12 entered
42 reader #12 exit
43 writer #8 entered
44 writer #8 exit
45 writer #9 entered
46 writer #9 exit
47 reader #13 entered
```


Ta thấy số lượng `reader` và `writer` chênh lệch nhau không quá lớn, nạn đói đã được giải quyết khá hiệu quả.

3.4 Đọc MBR

3.4.1 Mã nguồn

```
1  /*
2  * =====
3  *
4  *     Filename:  readmbr.c
5  *
6  *     Description:
7  *
8  *     Version:   1.0
9  *     Created:   04/17/2012 08:05:39 PM
10 *     Revision:  none
11 *     Compiler:  gcc
12 *
13 *     Author:    BOSS14420 (boss14420), boss14420@gmail.com
14 *     Company:
15 *
16 * =====
17 */
18
19 #include <stdint.h>
20 #include <stdio.h>
21
22 void read_mbr(FILE *file);
23 char const* part_type(uint8_t code);
24 void print_part_info(uint8_t const*, uint32_t*);
25 void print_header();
26
27 int main(int argc, char *argv[]) {
28     if(argc >= 2) {
29         for(; argc > 1; --argc) {
30             FILE *f = fopen(argv[argc-1], "r");
31             if(f) {
32                 printf("Disk%s:\n\n", argv[argc-1]);
33                 read_mbr(f);
34                 fclose(f);
35             } else
36                 perror(argv[argc-1]);
37         }
38     }
39
40     return 0;
41 }
42
43 char const* part_type(uint8_t code) {
44     switch(code) {
45         case 0x83:
46             return "Linux";
```

```

47     case 0x82:
48         return "Linux swap";
49     case 0xAF:
50         return "HFS/HFS+";
51     case 0x05:
52     case 0x0F:
53     case 0x85:
54         return "Extended";
55     default:
56         return NULL;
57 }
58 }
59
60 void print_header() {
61     puts("+-----+-----+-----+-----+");
62     puts("      |-----+-----+");
63     puts("|      |      Begin      |      |      End      |");
64     puts("      | Relative |      Number |");
65     puts("|Active+---+-----+---+      Type      +---+-----+");
66     puts("      +---+      +      Of      +");
67     puts("|      |Hdr| Cylinder |Sct|      |Hdr| Cylinder |");
68     puts("      |Sct| Sector |      Sector |");
69     puts("+-----+-----+-----+-----+");
70     puts("      +-----+-----+");
71 }
72
73 void read_mbr(FILE *file) {
74     uint8_t mbr[512];
75     fread(mbr, 512, 1, file);
76
77     uint8_t *part_table = mbr + 446;
78     uint32_t extended_sector = 0;
79
80     print_header();
81     print_part_info(part_table, &extended_sector);
82     print_part_info(part_table+16, &extended_sector);
83     print_part_info(part_table+32, &extended_sector);
84     print_part_info(part_table+48, &extended_sector);
85     printf("+-----+-----+-----+-----+");
86     printf("      +-----+-----+-----+-----+\n\n");
87
88     uint32_t first_extended_sector = extended_sector;
89     if(extended_sector) {
90         printf("Extended partition table:\n");
91         extended_sector = 0;
92         do {
93             fseek(file, ((uint64_t)extended_sector
94                 + first_extended_sector) << 9, SEEK_SET);
95             extended_sector = 0;

```

```

96         fread(mbr, 512, 1, file);
97         part_table = mbr + 446;
98
99         //         print_header();
100         printf("+-----+-----+-----+-----"
101               "-----+-----+-----+-----\n");
102         print_part_info(part_table, &extended_sector);
103         print_part_info(part_table+16, &extended_sector);
104         printf("+-----+-----+-----+-----"
105               "-----+-----+-----+-----\n\n");
106     } while(extended_sector);
107 }
108 }
109
110 void print_part_info(uint8_t const *part_table, uint32_t *extended_sector) {
111     if(!part_table[2])
112         return;
113
114     // state
115     printf("|%3s  ", (part_table[0] == 0x80) ? "yes" : "no");
116
117     // start head
118     printf("|%3u", part_table[1]);
119
120     // start cylinder
121     printf("|%4u  ", (((unsigned)part_table[2] & 0xC0) << 2)
122            | part_table[3]);
123
124     // start sector
125     printf("|%2u ", part_table[2] & 0x3F);
126
127     // sys type
128     char const* pt = part_type(part_table[4]);
129     if(pt)
130         printf("|%-12s", pt);
131     else
132         printf("|0x%-10X", part_table[4]);
133
134     // end head
135     printf("|%3u", part_table[5]);
136
137     // end cylinder
138     printf("|%4u  ", (((unsigned)part_table[6] & 0xC0) << 2)
139            | part_table[7]);
140
141     // end sector
142     printf("|%2u ", part_table[6] & 0x3F);
143
144     // start sector (LBA)

```

```

145     uint32_t start_sector = (uint32_t)part_table[8] | (part_table[9] << 8)
146         | (part_table[10] << 16) | (part_table[11] << 24);
147     printf("|%10u", start_sector);
148     if(part_table[4] == 0x05 || part_table[4] == 0x0F
149         || part_table[4] == 0x85)
150         *extended_sector = start_sector;
151
152     // num of sector
153     uint32_t num_sector = (uint32_t)part_table[12] | (part_table[13] << 8)
154         | (part_table[14] << 16) | (part_table[15] << 24);
155     printf("|%10u|", num_sector);
156     printf("\n");
157 }

```

```
boss14420 /media/DOCUMENT/code_exp/Baitap/HDH/MBR $ ./readmbr /dev/sda
```

Active	Begin			Type	End			Relative	Number of
	Hdr	Cylinder	Sct		Hdr	Cylinder	Sct		
no	1	0	1	Linux	254	11	63	63	192717
no	0	12	62	Extended	254	1023	63	192841	520615607
yes	254	1023	63	HFS/HFS+	254	1023	63	520808448	30869504
no	254	1023	63	Linux	254	1023	63	551677952	73463808

Extended partition table:

no	1	12	1	Linux	254	1023	63	2	82027827
no	254	1023	63	Extended	254	1023	63	82027829	6008310

no	254	1023	63	Linux swap	254	1023	63	63	6008247
no	254	1023	63	Extended	254	1023	63	88036139	31246425

no	254	1023	63	Linux	254	1023	63	63	31246362
no	254	1023	63	Extended	254	1023	63	119282564	389731123

no	254	1023	63	Linux	254	1023	63	63	389731060
no	254	1023	63	Extended	254	1023	63	509013687	11601920

no	254	1023	63	Linux	254	1023	63	2048	11599872
----	-----	------	----	-------	-----	------	----	------	----------

Hình 3.4: Bảng phân vùng

3.4.2 Kết quả

- Đọc thông tin MBR của đĩa cứng laptop, gồm 1 phân vùng ext2, một phân vùng swap, 2 phân vùng ext4, 1 phân vùng reiserfs, 1 phân vùng xfs và một phân vùng hfs+ (MAC OS). Tất cả các phân vùng Linux (ext2, ext4, reiserfs, xfs) đều có mã phân vùng là 0x83. Hình 3.4
- Đọc thông tin MBR của đĩa cứng gắn ngoài (gồm 1 phân vùng NTFS và 2 phân vùng FAT32), hình 3.5

```
boss14420 /media/DOCUMENT/code_exp/Baitap/HDH/MBR $ ./readmbr /dev/sdb
```

	Begin				End			Relative	Number
Active				Type					of
	Hdr	Cylinder	Sct		Hdr	Cylinder	Sct	Sector	Sector
no	1	0	1	0x7	254	1023	63	63	510529572
no	243	1023	1	Extended	254	1023	63	510544944	114592401

Extended partition table:

yes	254	1023	63	0xB	254	1023	63	819	65529072
no	254	1023	63	Extended	254	1023	63	65529891	49062510

no	254	1023	63	0xB	254	1023	63	63	49062447
----	-----	------	----	-----	-----	------	----	----	----------

Hình 3.5: Bảng phân vùng

Chương 4

Tài liệu tham khảo

- [1] GNU Project. *GNU C Library - Job Control*, March 2012. Available at http://www.gnu.org/software/libc/manual/html_node/Job-Control.html#Job-Control.
- [2] GNU Project. *GNU C Library - Word Expansion*, March 2012. Available at http://www.gnu.org/software/libc/manual/html_node/Word-Expansion.html#Word-Expansion.
- [3] Neil Matthew and Richard Stones. *Beginning Linux Programming*. Wrox, 4 edition, 2007.
- [4] Oracle. *Java SE6 document - BlockingQueue*, 2011. Available at <http://docs.oracle.com/javase/6/docs/api/java/util/concurrent/BlockingQueue.html>.
- [5] Phạm Đăng Hải. *Chương 2 - Quản lý tiến trình*, page 130. No publisher, 2012.