I/O Multiplex - poll()

Trong bài trước, chúng ta đã tìm hiểu về mô hình multiplex I/O và system call select() được dùng rất phổ biến trong lập trình hệ thống Linux. Bài này sẽ giới thiệu một system call khác cũng rất phổ biến là poll() và so sánh ưu nhược điểm giữa hai system call quan trọng này.

System call poll()

Về mặt chức năng, poll() có khả năng thực hiện đầy đủ công việc của system call select(). Sự khác nhau nằm ở cách xây dựng và cài đặt các mô tả file cần theo dõi. Với select() thì chúng ta có 3 tập hợp mô tả file (fd_set) là readfds, writefds và exceptfds; trong khi với poll() thì chúng ta có một danh sách của các mô tả file, mỗi mô tả file đi kèm với tập hợp các sự kiện mà ta quan tâm.

Prototype của poll() như sau:

```
#include <poll.h>
int poll(struct pollfd fds[], nfds_t nfds, int timeout);
   /*Trả về số mô tả file sẵn sàng, hoặc 0 nếu timeout, hoặc –1 nếu lỗi*/
```

Đối số fds[] là một mảng của cấu trúc pollfd, cấu trúc này gồm 1 mô tả file và tập hợp các sự kiện chúng ta quan tâm với mô tả file đó, Đối số nfds là độ lớn của mảng fds[] chỉ số mô tả file mà chúng ta cần theo dõi (kiểu dữ liệu nfds_t thực ra là kiểu unsigned integer trong Linux) và timeout là thời gian block tối đa của system call poll().

Cấu trúc pollfd của mảng fds[] được định nghĩa như sau:

```
struct pollfd {
    int fd; /* mô tả file cần theo dõi */
    short events; /* bit mask các sự kiện theo dõi */
    short revents; /*bit mask các sự kiện trả về*/
};
```

Nhìn vào cấu trúc pollfd chúng ta có thể dễ hình dung về cách sử dụng poll(): với mỗi file cần theo dõi, chỉ cần truyền mô tả file và tham số fd và sự kiện chúng ta quan tâm vào trường events, sau khi poll() return thì đọc kết quả ở trường revents của mỗi file.

Hai trường events và revents có kiểu bit mask dùng để người dùng cài đặt sự kiện mà chúng ta muốn theo dõi (events) và để kernel trả về sự kiện xảy ra (revents) của mô tả file fd. Các bit sự kiện của event và revent có thể dưới đây:

Bit	event	revent	Ý nghĩa
POLLIN	ü	ü	Có dữ liệu có thể đọc
POLLRDNORM	ü	ü	Bằng với POLLIN
POLLRDBAND	ü	ü	Có dữ liệu ưu tiên có thể đọc (không dùng trong Linux)

POLLPRI	ü	ü	Có dữ liệu độ ưu tiên cao có thể đọc
POLLRDHUP	ü	ü	Socket bị đóng ở đầu kết nối kia
POLLOUT	ü	ü	Có thể ghi dữ liệu vào không block
POLLWRNORM	ü	ü	Bằng với POLLOUT
POLLWRBAND	ü	ü	Dữ liệu ưu tiên có thể ghi không block
POLLERR		ü	Có lỗi xảy ra với mô tả file
POLLHUP		ü	Hangup xảy ra với mô tả file
POLLNVAL		ü	Mô tả file không được mở

Có thể bạn sẽ nghĩ các bit của poll() khó hiểu so với dùng select() thì trong thực tế chúng ta thường chỉ dùng một số bit đơn giản. Có thể hiểu cách dùng đơn giản như sau: POLLIN | POLLPRI bằng với sự kiện đọc (readfds) và POLLOUT | POLLWRBAND bằng với sự kiện ghi (writefds) của select().

Đối số timeout (int) quy định thời gian block theo đơn vị mili giây của system call poll() như sau:

- **Nếu timeout == -1: block không giới hạn cho đến khi một trong các mô tả file sẵn sàng hoặc có signal xảy ra.**
- **Nếu timeout == 0: không block, kiểm tra luôn xem các mô tả file sẵn sàng chưa và trả về kết quả ngay.**
- **Nếu timeout > 0: block tối đa thời gian bằng giá trị của timeout.**

System call poll() nếu thành công sẽ trả về số mô tả file sẵn sàng với kết quả được kernel ghi vào trường revent tương ứng của cấu trúc chứa mô tả file đó. Nếu sau thời gian timeout chưa có mô tả file nào sẵn sàng thì poll() trả về 0. Nếu có lỗi xảy ra thì poll() trả về -1 và ghi mã lỗi vào biến errno.

Ví dụ

Tương tự như ví dụ của bài trước về sử dụng select(), ví dụ lần này mục đích cũng là minh họa về cách sử dụng system call poll(). Chúng ta sẽ kiểm tra nếu mô tả file stdin và stdout có sẵn sàng để đọc và ghi thì sẽ in log ra màn hình, thời gian block timeout là 5 giây.

```
#include <stdio.h>
```

```c
#include <unistd.h>
#include <sys/poll.h>

#define TIMEOUT 5  /*timeout 5 giây*/

int main (void)
{
	struct pollfd fds[2];
	int ret = -1;

	/* Kiểm tra stdin sẵn sàng đọc */
	fds[0].fd = STDIN_FILENO;
	fds[0].events = POLLIN;

	/* kiểm tra stdout sẵn sàng ghi */
	fds[1].fd = STDOUT_FILENO;
	fds[1].events = POLLOUT;

	ret = poll(fds, 2, TIMEOUT * 1000);

	if (-1 == ret)
	{
		perror("poll() error!");
		return -1;
	}
	if (0 == ret)
	{
		printf ("poll() timeout sau %d giay.\n", TIMEOUT);
		return 0;
	}

	/*Kiem tra revents cho stdin va stdout*/
	if (fds[0].revents & POLLIN)
	{
		printf("stdin san sang doc\n");
	}
	if (fds[1].revents & POLLOUT)
	{
		printf("stdout san sang ghi\n");
	}

	return 0;
}
```

Sau khi compile, và chạy chương trình, bạn sẽ thấy kết quả in ra là stdout sẵn sàng ghi ngay (vì stdout lúc này không bận nên có thể ghi vào luôn mà không cần block), stdin chưa có dữ liệu vào nên chưa báo sẵn sàng đọc.

```
minhlv@vimentor:~/work/code$ vi poll.c
minhlv@vimentor:~/work/code$ gcc -o poll poll.c
minhlv@vimentor:~/work/code$ ./poll
stdout san sang ghi
```

Bây giờ, bạn chạy chương trình cùng với việc redirect một file vào chương trình, lúc này stdin của chương trình sẽ tiếp nhận file, do đó stdin đẽ có dữ liệu sẵn sàng đọc và kết quả như sau:

```
minhlv@vimentor:~/work/code$ vi poll.c
minhlv@vimentor:~/work/code$ gcc -o poll poll.c
minhlv@vimentor:~/work/code$ ./poll < hello.txt
stdin san sang doc
stdout san sang ghi
minhlv@vimentor:~/work/code$
```

So sánh select() và poll()

Về mặt triển khai ở tầng Linux kernel, cả select() và poll() đều được kernel gọi đến các poll routines. Lưu ý rằng poll routines này nằm ở dưới kernel và không liên quan gì đến system call poll() gọi trên tầng user. Có thể hiểu đơn giản poll routine là một chuỗi hành động của kernel tương ứng với mỗi file, mỗi poll routine sẽ xử lý và trả về thông tin của một mô tả file xem nó đã sẵn sàng để đọc hoặc ghi chưa. Vì vậy, bản chất của select() và poll() đều là gọi poll routine cho mỗi mô tả file mà hai system call này truyền xuống kernel. Sự khác nhau chỉ nằm ở cách truyền mô tả file và sự kiện quan tâm của mỗi file xuống kernel mà thôi.

Mặc dù cả select() và poll() đều cung cấp cùng một chức năng, nhưng poll() có hiệu năng (performance) cao hơn select(). Lý do là khi select() truyền tập hợp các mô tả file (fd_set) và số nguyên nfds là số mô tả file lớn nhất cộng thêm 1, kernel sẽ phải làm việc với tất cả các mô tả file từ 0 đến số lớn nhất nfds và kiểm tra xem chính xác các mô tả file nào sẽ được theo dõi. Giả sử nếu bạn chỉ muốn theo dõi mô tả file fd là 1000 thì kernel vẫn phải quét 1001 mô tả file (từ 0 đến 1000) để xem mô tả file nào được theo dõi (1001 poll routine). Với poll() thì khác, bạn chỉ cần truyền cấu trúc với chính xác mô tả file fd 1000 kèm với sự kiện quan tâm tương ứng xuống kernel, do vậy kernel sẽ gọi 1 poll routine để kiểm tra duy nhất mô tả file 1000.

Vì lý do trên, poll() sẽ có hiệu năng cao hơn select() rõ rệt nếu chúng ta cần theo dõi chỉ một số ít các mô tả file mà giá trị của chúng lớn. Còn trong trường hợp sau đây, hiệu năng của select() và poll() không quá cách biệt và có thể coi là giống nhau:

- **Các mô tả file cần theo dõi nhỏ (mô tả file lớn nhất có giá trị thấp), khi đó kernel không phải quét quá nhiều mô tả file khi gọi select()**
- **Cần theo dõi số lượng lớn mô tả file, nhưng giá trị các mô tả file không quá cách xa.**

Dù poll() có hiệu năng cao hơn select(), nhưng trong thực tế select() vẫn được dùng rộng rãi hơn. Nguyên nhân là select() có tính tương thích (portability) với nhiều hệ thống hơn trong khi poll() vẫn có 1 số biến thể nho nhỏ giữa các hệ thống khác nhau. Một phần nữa do thói quen sử dụng và người dùng cũng không thường xuyên phải theo dõi số lượng file lớn đến mức cần để ý hiệu năng giữ select() và poll().

Kết luận

Trong bài này, chúng ta đã tìm hiểu về system call poll() và so sánh giữa select() và poll. Cả poll() và select() đều hữu ích khi cần theo dõi nhiều mô tả file đã sẵn sàng đọc/ghi chưa. Cả hai system call này đều có khả năng tương thích cao với các hệ thống Unix/Linx và đều được quy định trong tiêu chuẩn SUSv3 và được dùng phổ biến trong lập trình Linux. Ngoài ra, poll() còn có hiệu năng cao hơn select() nếu cần theo dõi một số ít mô tả file mà giá trị mô tả file lớn.