linux新增的系统调用eventfd

主要完成进程或线程间通信

int eventfd(unsigned int initval, int flags);

Initval：初始化计数器值，该值（暂时取名为A）保存在内核。

函数返回一个文件描述符，与打开的其他文件一样，可以进行读写操作。

Read：如果计数器A的值不为0时，读取成功，获得到该值。

如果A的值为0

非阻塞模式时，会直接返回失败，并把error置为EINVAL

如果为阻塞模式，一直会阻塞到A为非0为止。

Write：将缓冲区写入的8字节整形值加到内核计数器上，即会增加8字节的整数在计数器A上，如果A的值达到0xfffffffffffffffe时，就会阻塞（在阻塞模式下），直到A的值被read。阻塞和非阻塞情况同上面read一样。

**[cpp]** [view plaincopy](http://blog.csdn.net/chdhust/article/details/9270121)

1. #include <sys/eventfd.h>
2. #include <unistd.h>
3. #include <stdlib.h>
4. #include <stdio.h>
5. #include <stdint.h>             /\* Definition of uint64\_t \*/
7. #define handle\_error(msg) \
8. **do** { perror(msg); exit(EXIT\_FAILURE); } **while** (0)
10. **int**
11. main(**int** argc, **char** \*argv[])
12. {
13. uint64\_t u;
15. **int** efd = eventfd(10, 0);
16. **if** (efd == -1)
17. handle\_error("eventfd");
19. **int** ret = fork();
20. **if**(ret == 0)
21. {
22. **for** (**int** j = 1; j < argc; j++) {
23. printf("Child writing %s to efd\n", argv[j]);
24. u = atoll(argv[j]);
25. ssize\_t s = write(efd, &u, **sizeof**(uint64\_t));
26. **if** (s != **sizeof**(uint64\_t))
27. handle\_error("write");
28. }
29. printf("Child completed write loop\n");
31. exit(EXIT\_SUCCESS);
32. }
33. **else**
34. {
35. sleep(2);
37. ssize\_t s = read(efd, &u, **sizeof**(uint64\_t));
38. **if** (s != **sizeof**(uint64\_t))
39. handle\_error("read");
40. printf("Parent read %llu from efd\n",(unsigned **long** **long**)u);
41. exit(EXIT\_SUCCESS);
42. }
43. }
44. ./eventfd 10 20 30
45. Child writing 10 to efd
46. Child writing 20 to efd
47. Child writing 30 to efd
48. Child completed write loop
49. Parent read 70 from efd

命令行传入的是10、20、30其和应为60，为啥读取的是70呢？请看15行调用eventfd时第一个参数是10，这个参数是创建eventfd时初始化计数器的值。