1. 线程同步

a. 需要线程同步的情况：

1）同时访问同一内存空间；

2）需要指定访问同一内存空间的线程执行顺序。

b. 同步技术：互斥量Mutex和信号量Semaphore

将临界区比喻成洗手间，线程同步理解成一把锁。

为了保护个人隐私，进洗手间时锁上门，出来再打开；

如果有人使用洗手间，其他人需要在外面等待；

等待的人数可能很多，这些人需要排队进入洗手间。

2. 互斥量Mutual Exclusion，创建和销毁函数

1. #include <pthread.h>
3. **int** pthread\_mutex\_init(pthread\_mutex\_t \* mutex,**const** pthread\_mutexattr\_t \* attr);
5. **int** pthread\_mutex\_destroy(pthread\_mutex\_t \* mutex);
7. 成功返回0，失败返回其他值
8. mutex：创建/销毁互斥量时传递保存互斥量的变量地址值
9. attr：创建的互斥量属性

创建互斥量时，如果第二个参数为NULL，也可以通过宏PTHREAD\_MUTEX\_INITIALIZER创建。

但是最好还是使用函数，宏比较难检查错误。

3. 互斥量锁住、释放临界区（lock和unlock是对临界区进行操作的）

1. **int** pthread\_mutex\_lock(pthread\_mutex\_t \* mutex);
2. **int** pthread\_mutex\_unlock(pthread\_mutex\_t \*mutex);
4. 成功返回0，失败返回其他值

注意：临界区锁住后，忘记解锁，则其他尝试进入临界区的线程将会“死锁”。

4. 信号量的创建、销毁

1. **int** sem\_init(sem\_t \*sem,**int** pshared,unsigned **int** value);
2. **int** sem\_destroy(sem\_t \* sem);
4. 成功返回0，失败返回其他值
6. sem：创建信号量时传递保存信号量的变量地址值
7. pshared：传递其他值时，可创建多个进程共享的信号量。0时，只允许一个进程内部使用该信号量
8. value：指定信号量初始值。

5. post/wait

1. **int** sem\_post(sem\_t \*sem);
2. **int** sem\_wait(sem\_t \*sem);
4. 成功返回0，失败时返回其他值
6. sem：传递保存信号量读取值得变量地址值

调用sem\_init时，操作系统创建信号量对象，并赋初始值。

调用sem\_post函数值，对象+1，sem\_wait函数时-1.

信号量的值不能小于0，在信号量为0的情况，sem\_wait函数会进入阻塞直到值大于0。

6. 关于控制访问顺序的同步

1. #include <stdio.h>
2. #include <pthread.h>
3. #include <semaphore.h>
5. **void** \* read(**void** \* arg);
6. **void** \* accu(**void** \* arg);
8. **static** sem\_t sem\_one;
9. **static** sem\_t sem\_two;
10. **static** **int** num;
12. **int** main(**int** argc, **char** \*argv[]){
13. pthread\_t id\_t1,id\_t2;
15. sem\_init(&sem\_one,0,0);
16. sem\_init(&sem\_two,0,1);
18. pthread\_create(&id\_t1,NULL,read,NULL);
19. pthread\_create(&id\_t2,NULL,accu,NULL);
21. pthread\_join(id\_t1,NULL);
22. pthread\_join(id\_t2,NULL);
24. sem\_destroy(&sem\_one);
25. sem\_destroy(&sem\_two);
27. **return** 0;
28. }

31. **void** \* read(**void** \* arg){
32. **int** i;
34. **for**(i=0;i<5;i++){
35. printf("Input num %d: ",i+1);
36. sem\_wait(&sem\_two);
37. scanf("%d",&num);
38. sem\_post(&sem\_one);
39. }
41. **return** NULL;
42. }
44. **void** \*accu(**void** \*arg){
45. **int** sum = 0,i;
46. **for**(i=0;i<5;i++){
47. sem\_wait(&sem\_one);
48. sum += num;
49. printf("sum = %d \n",sum);
50. sem\_post(&sem\_two);
51. }
53. printf("result : %d \n",sum);
55. **return** NULL;
56. }

7. 线程的销毁

线程不是在线程main返回时自动销毁，而是使用下面的方法销毁，不然会一直存在内存空间：

a. pthread\_join函数//线程一直等待，阻塞

b. pthread\_detach函数//通常使用这种方法销毁线程，不会引起线程终止或进入阻塞。

调用个了pthread\_detach后，不能再调用pthread\_join

1. **int** pthread\_detach(pthread\_t **thread**);
3. 成功：返回0，失败返回其他值

8. 多线程并发服务器端的实现

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
4. #include <unistd.h>
5. #include <signal.h>
6. #include <sys/wait.h>
7. #include <arpa/inet.h>
8. #include <sys/socket.h>
9. #include <pthread.h>
10. #include <semaphore.h>
12. #define BUF\_SIZE    100
13. #define MAX\_CLNT    256
15. **void** \* handle\_clnt(**void** \* arg);
16. **void** send\_msg(**char** \*msg,**int** len);
17. **void** error\_handling(**char** \*message);
19. **int** clnt\_cnt = 0;
20. **int** clnt\_socks[MAX\_CLNT];
21. pthread\_mutex\_t mutx;
23. **int** main(**int** argc, **char** \*argv[]){
24. **int** serv\_sock;
25. **int** clnt\_sock;
26. **struct** sockaddr\_in serv\_adr;
27. **struct** sockaddr\_in clnt\_adr;
28. **int** clnt\_adr\_sz;
29. pthread\_t t\_id;
31. **if**(argc != 2){
32. printf("Usage : %s <port>\n",argv[0]);
33. exit(1);
34. }
35. pthread\_mutex\_init(&mutx,NULL);
37. serv\_sock = socket(PF\_INET,SOCK\_STREAM,0);
38. **if**(serv\_sock == -1){
39. error\_handling("socket() error");
40. }
42. memset(&serv\_adr,0,**sizeof**(serv\_adr));
43. serv\_adr.sin\_family=AF\_INET;
44. serv\_adr.sin\_addr.s\_addr=htonl(INADDR\_ANY);
45. serv\_adr.sin\_port=htons(atoi(argv[1]));
47. **if**(bind(serv\_sock,(**struct** sockaddr\*)&serv\_adr,**sizeof**(serv\_adr)) == -1){
48. error\_handling("bind() error");
49. }
51. **if**(listen(serv\_sock,5) == -1){
52. error\_handling("listen() error");
53. }
55. **while**(1){
56. clnt\_adr\_sz = **sizeof**(clnt\_adr);
57. clnt\_sock = accept(serv\_sock,(**struct** sockaddr\*)&clnt\_adr,&clnt\_adr\_sz);
58. pthread\_mutex\_lock(&mutx);
59. clnt\_socks[clnt\_cnt++] = clnt\_sock;
60. pthread\_mutex\_unlock(&mutx);
62. pthread\_create(&t\_id,NULL,handle\_clnt,(**void** \*)&clnt\_sock);
63. pthread\_detach(t\_id);
64. printf("Connetcted client IP :%s \n",inet\_ntoa(clnt\_adr.sin\_addr));
65. }
67. close(serv\_sock);
69. **return** 0;
70. }
72. **void** error\_handling(**char** \*message){
74. fputs(message,stderr);
75. fputs("\n",stderr);
76. exit(1);
77. }
79. **void** \* handle\_clnt(**void** \* arg){
80. **int** clnt\_sock = \*((**int** \*)arg);
81. **int** str\_len =0;
82. **int** i;
83. **char** msg[BUF\_SIZE];
85. **while**((str\_len = read(clnt\_sock,msg,**sizeof**(msg))) != 0)
86. send\_msg(msg,str\_len);
88. pthread\_mutex\_lock(&mutx);
89. **for**(i=0;i<clnt\_cnt;i++){
90. **if**(clnt\_sock == clnt\_socks[i]){
91. **while**(i++ < clnt\_cnt - 1)
92. clnt\_socks[i] = clnt\_socks[i+1];
93. **break**;
94. }
95. }
97. clnt\_cnt--;
98. pthread\_mutex\_unlock(&mutx);
99. close(clnt\_sock);
101. **return** NULL;
102. }
104. //send to all
105. **void** send\_msg(**char** \*msg,**int** len)
106. {
107. **int** i;
109. pthread\_mutex\_lock(&mutx);
110. **for**(i=0;i<clnt\_cnt;i++){
111. write(clnt\_socks[i],msg,len);
112. }
114. pthread\_mutex\_unlock(&mutx);
115. }

客户端实现：

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
4. #include <unistd.h>
5. #include <arpa/inet.h>
6. #include <sys/socket.h>
7. #include <semaphore.h>
8. #include <pthread.h>
10. #define BUF\_SIZE    100
11. #define NAME\_SIZE   20
13. **void** error\_handling(**char** \*message);
14. **void** \* send\_msg(**void** \* arg);
15. **void** \* recv\_msg(**void** \* arg);
17. **char** name[NAME\_SIZE] = "[DEFAULT]";
18. **char** msg[BUF\_SIZE];
20. **int** main(**int** argc,**char** \*argv[]){
21. **int** sock;
22. **struct** sockaddr\_in serv\_adr;
23. pthread\_t snd\_thread,rcv\_thread;
24. **void** \* thread\_return;
26. **if**(argc != 4){
27. printf("Usage : %s <IP> <port> <name>\n",argv[0]);
28. exit(1);
29. }
31. sprintf(name,"[%s]",argv[3]);
32. sock = socket(PF\_INET,SOCK\_STREAM,0);
33. **if**(sock == -1){
34. error\_handling("socket() error");
35. }
37. memset(&serv\_adr,0,**sizeof**(serv\_adr));
38. serv\_adr.sin\_family = AF\_INET;
39. serv\_adr.sin\_addr.s\_addr = inet\_addr(argv[1]);
40. serv\_adr.sin\_port = htons(atoi(argv[2]));
42. **if**(connect(sock,(**struct** sockaddr \*)&serv\_adr,**sizeof**(serv\_adr)) == -1){
43. error\_handling("connect() error\r\n");
44. }**else**{
45. printf("Connected....");
46. }
48. pthread\_create(&snd\_thread,NULL,send\_msg,(**void** \*)&sock);
49. pthread\_create(&rcv\_thread,NULL,recv\_msg,(**void** \*)&sock);
51. pthread\_join(snd\_thread,&thread\_return);
52. pthread\_join(rcv\_thread,&thread\_return);
54. close(sock);
55. **return** 0;
56. }
58. **void** error\_handling(**char** \*message){
59. fputs(message,stderr);
60. fputs("\n",stderr);
61. exit(1);
62. }
64. **void** \* send\_msg(**void** \*arg){
65. **int** sock = \*((**int** \*)arg);
66. **char** name\_msg[NAME\_SIZE + BUF\_SIZE];
68. **while**(1){
69. fgets(msg,BUF\_SIZE,stdin);
70. **if**(!strcmp(msg,"q\n") || ! strcmp(msg,"Q\n")){
71. close(sock);
72. exit(0);
73. }
75. sprintf(name\_msg,"%s %s",name,msg);
76. write(sock,name\_msg,strlen(name\_msg));
77. }
79. **return** NULL;
80. }
82. **void** \* recv\_msg(**void** \* arg){
83. **int** sock = \*((**int** \*)arg);
84. **char** name\_msg[NAME\_SIZE + BUF\_SIZE];
85. **int** str\_len;
87. **while**(1){
88. str\_len = read(sock,name\_msg,NAME\_SIZE+BUF\_SIZE-1);
90. **if**(str\_len == -1){
91. **return** (**void** \*) -1;
92. }
94. name\_msg[str\_len] = 0;
95. fputs(name\_msg,stdout);
96. }
98. **return** NULL;
99. }