选择题

11章

1. 以下符合对TCP协议描述的是

①可靠传输 ②在主机间传输 ③可以基于IP协议

④全双工 ⑤可以基于DNS

1. ①②④ B. ③④⑤ C. ①③④ D. ②④

2. 对于建立客户端与服务器的TCP连接，以下关于套接字接口，正确的是：

A. 对于TCP连接而言，其可以由连接双方的套接字对四元组（双方的IP地址和端口号）唯一确定。

B. connect函数用于客户端建立与服务器的连接，其参数中的套接字地址结构需要指明本客户端使用的端口号

C. socket函数返回的套接字描述符，可以立即使用标准 Unix I/O 函数进行读写

D. bind 函数用于将用于客户端的主动套接字转化为用于服务器的监听套接字

3. 下列有关Web服务错误的是

A. Web服务使用客户端-服务器模型，使用了HTTP协议，可以传输文本，HTML页面，二进制文件等多种内容

B. HTTP响应会返回状态码，它指示了对响应的处理状态

C. Web服务使用URL来标明资源，这提供了一层抽象，使得客户端仿佛在访问远端的文件目录，而服务器处理了URL资源和具体文件/动态内容的映射关系

D. 多个域名可以映射到同一个IP地址，但一个域名不可以映射到多个IP地址

备选：

1. 下列关于全球IP因特网的说法中，正确的是
2. TCP/IP协议规定网络字节顺序取决于不同主机的字节顺序，从而方便主机和网络之间收发数据
3. 域名集合和IP地址集合的映射关系目前由http来维护
4. 多个域名可以映射到同一个IP地址，但一个域名不可以映射到多个IP地址
5. 从内核的角度来看，套接字是通信的一个端点；从用户程序的角度来看，套接字是一个有相应描述符的打开文件
6. 下列关于网络的说法中，错误的是
7. 在TCP连接中，客户端和服务器都是指主机
8. 网络按照覆盖范围的大小可以分成局域网和广域网，其中一种流行的局域网技术是以太网
9. 网络可以被主机认为是一种I/O设备，是数据源和数据接收方
10. 路由器可以把多个不兼容的网络连接起来组成一个互联网络

12章

1.下列关于C语言中进程模型和线程模型的说法中，错误的是：

A.每个线程都有它自己独立的线程上下文，包括线程ID、程序计数器、条件码、通用目的寄存器值等

B.每个线程都有自己独立的线程栈，任何线程都不能访问其他对等线程的栈空间

C.不同进程之间的虚拟地址空间是独立的，但同一个进程的不同线程共享同一个虚拟地址空间

D.一个线程的上下文比一个进程的上下文小得多，因此线程上下文切换要比进程上下文切换快得多

大题

并发编程（15分）

“生产者-消费者”问题是并发编程中的经典问题。本题中，考虑如下场景：

1. 所有生产者和所有消费者**共享同一个buffer**
2. 生产者、消费者各有NUM\_WORKERS个（**大于一个**）
3. buffer的容量为BUF\_SIZE，**初始情况下buffer为空**
4. 每个生产者向buffer中添加一个item；若buffer满，则生产者等待buffer中有空槽时才能添加元素
5. 每个消费者从buffer中取走一个item；若buffer空，则消费者等待buffer中有item时才能取走元素
6. 阅读以下代码并回答问题（**代码阅读提示：主要关注producer和consumer两个函数**）
7. */\* Producer–Consumer Problem (Solution 1) \*/*
8. #include "csapp.h"
9. #define BUF\_SIZE 3
10. #define NUM\_WORKERS 50
11. #define MAX\_SLEEP\_SEC 10
12. volatile static int items = 0; */\* How many items are there in the buffer \*/*
13. static sem\_t mutex; */\* Mutual Exclusion \*/*
14. static sem\_t empty; */\* How many empty slots are there in the buffer \*/*
15. static sem\_t full;  */\* How many items are there in the buffer \*/*
16. static void sync\_var\_init() {
17. Sem\_init(&mutex, 0, 1);
18. */\* Initially, there is no item in the buffer \*/*
19. Sem\_init(&empty, 0, BUF\_SIZE);
20. Sem\_init(&full, 0, 0);
21. }
22. static void \*producer(void \*num) {
23. ①;
24. ②;
25. */\* Critical section begins \*/*
26. Sleep(rand() % MAX\_SLEEP\_SEC);
27. items++;
28. */\* Critical section ends \*/*
29. V(&mutex);
30. V(&full);
31. return NULL;
32. }
33. static void \*consumer(void \*num) {
34. ③;
35. ④;
36. */\* Critical section begins \*/*
37. Sleep(rand() % MAX\_SLEEP\_SEC);
38. items--;
39. */\* Critical section ends \*/*
40. V(&mutex);
41. V(&empty);
42. return NULL;
43. }
44. int main() {
45. sync\_var\_init();
46. pthread\_t pid\_producer[NUM\_WORKERS];
47. pthread\_t pid\_consumer[NUM\_WORKERS];
48. for (int i = 0; i < NUM\_WORKERS; i++) {
49. Pthread\_create(&pid\_producer[i], NULL, producer, (void \*)i);
50. Pthread\_create(&pid\_consumer[i], NULL, consumer, (void \*)i);
51. }
52. for (int i = 0; i < NUM\_WORKERS; i++) {
53. Pthread\_join(pid\_producer[i], NULL);
54. Pthread\_join(pid\_consumer[i], NULL);
55. }
56. }
    1. 补全代码（请从以下选项中选择，可重复选择，每个1分，共4分）

① （24行）

② （25行）

③ （39行）

④ （40行）

选项：

1. P(&mutex)
2. P(&empty)
3. P(&full)
   1. 如果交换24行与25行（两个P操作）， （单选，2分）
      * 1. 有可能死锁
        2. 有可能饥饿
        3. 既不会死锁，也不会饥饿
   2. 交换32行与33行（两个V操作）是否可能造成同步错误？ （2分）
      * 1. 可能
        2. 不可能
   3. rand函数是不是线程安全的？ （1分）
      * 1. 是
        2. 不是

28行与43行对rand函数的使用是否会导致竞争？ （1分）

1. 会
2. 不会

已知rand函数的实现如下（来源：<https://github.com/begriffs/libc/blob/master/stdlib.h> 和<https://github.com/begriffs/libc/blob/master/stdlib.c>）：

1. #define RAND\_MAX 32767
2. unsigned long \_Randomseed = 1;
3. int rand() {
4. \_Randomseed = \_Randomseed \* 1103515425 + 12345;
5. return (unsigned int)(\_Randomseed >> 16) & RAND\_MAX;
6. }
7. void srand(unsigned int seed) {
8. \_Randomseed = seed;
9. }
10. 考虑“生产者-消费者”问题的另一种解法（**代码阅读提示：12-69行之外均与上一种解法相同**）
11. */\* Producer–Consumer Problem (Solution 2) \*/*
12. #include "csapp.h"
13. #define BUF\_SIZE 3
14. #define NUM\_WORKERS 50
15. #define MAX\_SLEEP\_SEC 10
16. volatile static int items = 0; */\* How many items are there in the buffer \*/*
17. static sem\_t mutex;                */\* Mutual Exclusion \*/*
18. static sem\_t sem\_waiting\_producer; */\* Wait for empty slots \*/*
19. static sem\_t sem\_waiting\_consumer; */\* Wait for available items \*/*
20. volatile static int num\_waiting\_producer = 0;
21. volatile static int num\_waiting\_consumer = 0;
22. static void sync\_var\_init() {
23. Sem\_init(&mutex, 0, 1);
24. Sem\_init(&sem\_waiting\_producer, 0, ①);
25. Sem\_init(&sem\_waiting\_consumer, 0, ①);
26. }
27. static void \*producer(void \*num) {
28. P(&mutex);
29. while (items == BUF\_SIZE) {
30. num\_waiting\_producer++;
31. ②;
32. ③;
33. P(&mutex);
34. }
35. */\* Critical section begins \*/*
36. Sleep(rand() % MAX\_SLEEP\_SEC);
37. items++;
38. */\* Critical section ends \*/*
39. if (num\_waiting\_consumer > 0) {
40. num\_waiting\_consumer--;
41. V(&sem\_waiting\_consumer);
42. }
43. V(&mutex);
44. return NULL;
45. }
46. static void \*consumer(void \*num) {
47. P(&mutex);
48. while (items == 0) {
49. num\_waiting\_consumer++;
50. ④;
51. ⑤;
52. P(&mutex);
53. }
54. */\* Critical section begins \*/*
55. Sleep(rand() % MAX\_SLEEP\_SEC);
56. items--;
57. */\* Critical section ends \*/*
58. if (num\_waiting\_producer > 0) {
59. num\_waiting\_producer--;
60. V(&sem\_waiting\_producer);
61. }
62. V(&mutex);
63. return NULL;
64. }
65. int main() {
66. sync\_var\_init();
67. pthread\_t pid\_producer[NUM\_WORKERS];
68. pthread\_t pid\_consumer[NUM\_WORKERS];
69. for (int i = 0; i < NUM\_WORKERS; i++) {
70. Pthread\_create(&pid\_producer[i], NULL, producer, (void \*)i);
71. Pthread\_create(&pid\_consumer[i], NULL, consumer, (void \*)i);
72. }
73. for (int i = 0; i < NUM\_WORKERS; i++) {
74. Pthread\_join(pid\_producer[i], NULL);
75. Pthread\_join(pid\_consumer[i], NULL);
76. }
77. }
    1. 补全补全代码（请从以下选项中选择，④⑤无需填写，每个1分，共3分）

① （21、22行）

② （29行）

③ （30行）

选项：

* + - 1. 0
      2. 1
      3. P(&sem\_waiting\_producer)
      4. V(&mutex)
  1. 如果27行和50行的while换成if，是否可能造成同步错误？ （2分）
     + 1. 可能
       2. 不可能