803 操作系统

单选部分

23. D 最先执行的是 BIOS 上的程序
24. A 进程创建之后进入就绪状态
25. B 用户进程可能都被阻塞
26. D 需要互斥访问的那段代码是临界区
27. B X * 10 <= 100ms => X <= 10
28. A 尽量使处理器忙起来是提高吞吐率的首要思想
29. A 使用文件首先使用 open 系统调用，获取指向文件的指针
30. A 树形目录的首要目的就是解决文件重名
31. D 数据块占两块，索引块占一块
32. A 只能顺序，不能跳跃

综合题部分

45.PV 原语题。

此题涉及到两个同步信号量，分别是车门、停车。司机负责车的行驶和停止，售票员负责车门的开关。同时司机需要判断车门开关，售票员需要判断车是否停。

```
Int door = 0; //0 表示车门开着，1 时表示车门已关
Int stop = 1;//0 表示车开着，1 时表示车到站停车
Semaphore mutex = 1;   //互斥控制变量，保护两个 int 变量被互斥访问
```

司机：

```
While(true) {
  Wait(mutex);
  While(door==0);  //开车前检查门是否关了
  Signal(mutex);
  Start vehicle;
  Wait(mutex);
  Stop = stop - 1; //通知售票员车开了
  Signal(mutex);
  Drive;
  Bus stop;
  Wait(mutex);
  Stop = stop + 1;  //告诉售票员车停了
  Signal(mutex);
}
```

售票员：

```
While(true) {
  Passengers on board;
  Close the door;
  Wait(mutex);
```

```
  Door = door + 1;
  Signal(mutex);
  Sell ticket;
  Wait(mutex);
  While(stop==0); //检查车是否停了
  Signal(mutex);
  Open the door;
  Wait(mutex);
  Door = door - 1;  //通知司机车门开了
  Signal(mutex);
  Passengers get off;
}
```

46.

	1	2	3	4	1	2	5	1	2	3	4	5	7
	1	1	1	1	1	1	1	1	1	1	1	5	5
		2	2	2	2	2	2	2	2	2	2	2	7
			3	3	3	3	5	5	5	5	4	4	4
				4	4	4	4	4	4	3	3	3	3
缺页	√	√	√	√			√			√	√	√	√

缺页次数：9

缺页率：9/13

803 数据结构部分：

单选部分

1-5　A　B　A　D　C

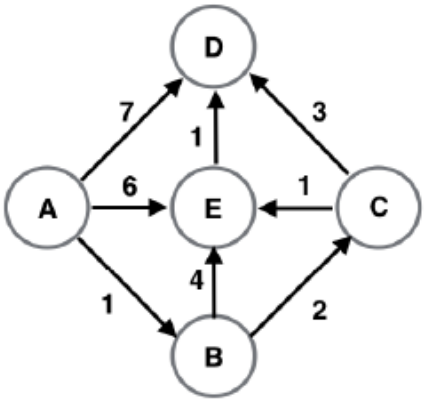
6-10　C　D　A　D　B

11　C

综合题部分

41.

(1)



(2)

Dijkstra 算法求 A 到其他各节点的最短路径：

顶点	第 1 趟	第 2 趟	第 3 趟	第 4 趟

B	1（最短） A->B			
C	∞ 	3（更新）（最短） A->B->C		
D	7 A->D	7 A->D	6（更新） A->B->C->D	5（更新）（最短） A->B->C->E->D
E	6 A->E	5（更新） A->B->E	4（更新）（最短） A->B->C->E	
集合 S	{A, B}	{A, B, C}	{A, B, C, E}	{A, B, C, E, D}

42.

```
// Created by Teacher Dong-Youxue kao yan on 2017/10/18.
// All rights reserved.
#include <stdlib.h>
#include <ctype.h>


typedef struct {
```

```
    int key;
    char info[10];
} elemtype;

typedef struct node {
    elemtype data;
    struct node *lchild, *rchild;
} node, *bitptr;

node *Search_Insert(bitptr root, int e) {
    node *p, *f, *new;
     p = root, f = root;
     while (p != NULL) {
          f = p;
          if ((p -> data).key == e) {
               break;
          }
          else if ((p -> data).key > e) {
               p = p -> lchild;
```

```
    }
    else {
        p = p -> rchild;
    }
}
if (p == NULL) {
    new = (node *)malloc(sizeof(node));
    new -> lchild = NULL, new -> rchild = NULL, (new -> data).key = e;
    if ((f -> data).key > e) {
        f -> lchild = new;
            }
            else {
                f -> rchild = new;
            }
            p = new;
        }
        return p;
    }
```