Part 1

1. D
2. B D D C B A
3. D
4. A
5. D
6. D
7. A

1.(1) 2^N

(2) 不相同，子进程退出的顺序不相同

(3) 将第 12 行改为 `pid_t wpid = waitpid(pid[i], &child_status);`

2.

```c
int main(int argc, char *argv[]) {
    pid_t pid1, pid2;
    int status;

    pid1 = fork();
    if (pid1 == 0) {
        sleep(5);
        exit(5);
    }

    pid2 = fork();
    if (pid2 == 0) {
        sleep(1);
        exit(1);
    }

    waitpid(pid1, &status);
    waitpid(pid2, &status);
    return 0;
}
```

3.(1) 6 种

(2) 13-21 行修改为:

```
printf("main: begin\n");
rc = pthread_create(&p1, NULL, mythread, "I ");
assert(rc == 0);
rc = pthread_join(p1, NULL);
assert(rc == 0);
rc = pthread_create(&p2, NULL, mythread, "LIKE ");
assert(rc == 0);
rc = pthread_join(p2, NULL);
assert(rc == 0);
rc = pthread_create(&p3, NULL, mythread, "OS ");
assert(rc == 0);
rc = pthread_join(p3, NULL);
assert(rc == 0);
printf("\nmain: end\n");
```

Part 2

1. D
2. C
3. C
4. B
5. A
6. A
7. C

1.若P不是原子操作，那么当一个进程检查信号量发现满足进入临界区条件、并且还未减1时，另一个进程可能也检查信号量发现满足进入临界区条件。这样会导致多个进程同时进入临界区，从而破坏互斥机制。
2.初始化信号量s为1，当有进程需要进入临界区时，检查s，若s为1，则将s减1并进入临界区，否则阻塞等待；从临界区离开时，将s加1，并唤醒等待中的进程。这样确保了n个进程在同一时刻仅有一个能访问临界区。
3.设 n 为计数值，s1, s2 为两个二进制信号量。
s1 初始值为 1，s2 初始值为 0.

```
P_new() {
    P(s1);
    n = n - 1;
    V(s1);
    if (n <= 0) {
        P(s2);
    }
}

V_new() {
    P(s1);
    n = n + 1;
    if (n > 0) {
        V(s2);
    }
    V(s1);
}
```