1.

(1)

Work = $(1, 0, 1, 2)$

Finish$[3]$ = false, Need$_3 \leq$ Work

∴ Work += Alloc$_3$ → $(1, 0, 2, 2)$
   Finish$[3]$ = true

Finish$[2]$ = false, Need$_2 \leq$ Work

∴ Work += Alloc$_2$ → $(1, 1, 2, 3)$
   Finish$[2]$ = true

Finish$[1]$ = false, Need$_1 \leq$ Work

∴ Work += Alloc$_1$ → $(1, 3, 3, 3)$
   Finish$[1]$ = true

Finish$[4]$ = false, Need$_4 \leq$ Work

∴ Work += Alloc$_4$ → $(2, 4, 3, 3)$
   Finish$[4]$ = true

Need = Max − Alloc

	A	B	C	D	
P$_1$	2	1	0	0	T
P$_2$	0	0	2	1	T
P$_3$	1	0	0	1	T
P$_4$	0	1	1	1	T

∴ sequence $\langle P_3, P_2, P_1, P_4 \rangle$
Satisfies safety
   requirement

(2) Request$_4$ = $(0, 0, 1, 1) \leq$ Need$_4$ $(0, 1, 1, 1)$
                              ↓
doesn't
satisfy Request$_4$ = $(0, 0, 1, 1) \leq$ Available $(1, 0, 1, 2)$
                              ↓
      P4 waits, insufficient resource

(3) Alloc

	A	B	C	D
P$_1$	0	2	1	0
P$_2$	0	1	0	1
P$_3$	0	0	1	0
P$_4$	1	1	1	1

+ Request =

Alloc

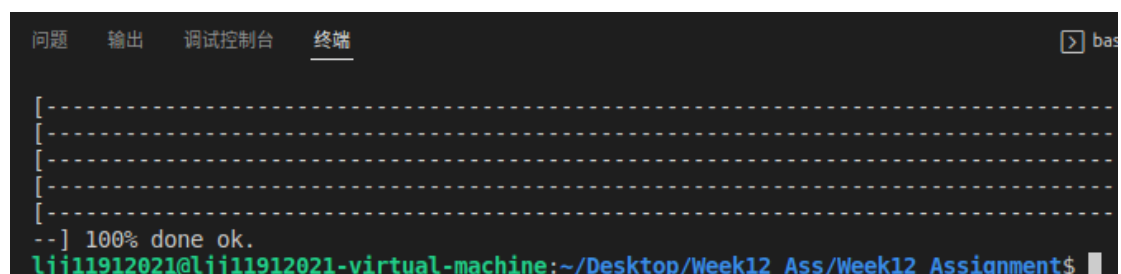|	A	B	C	D
P$_1$	2	3	1	0
P$_2$	0	1	1	1
P$_3$	1	0	1	0
P$_4$	1	2	1	1

Use deadlock detection algorithm

Work = $(-2, -2, -1, 1)$
For all i, Request$_i \leq$ Work is not satisfied
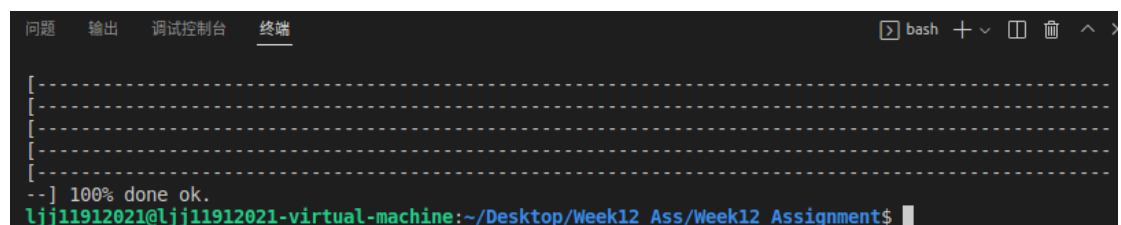∴ Finish$[i]$ = false exists, system deadlock

2. Design idea one: Use mutex lock. Only one person eats at a time.

```
82    pthread_mutex_t mutex;
83    void init() {
84        // write code if you desire.
85
86    }
87
88    void wants_to_eat(int p_no) {
89        // fixme
90        pthread_mutex_lock(&mutex);
91        pick_right_fork(p_no);
92        pick_left_fork(p_no);
93
94        eat(p_no);
95
96
97        put_left_fork(p_no);
98        put_right_fork(p_no);
99        pthread_mutex_unlock(&mutex);
100
101   }
```
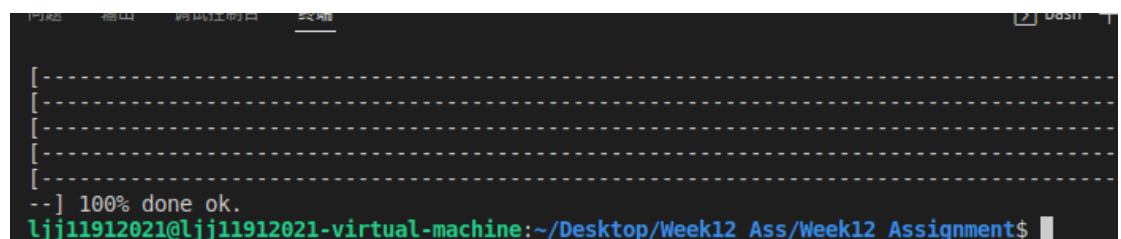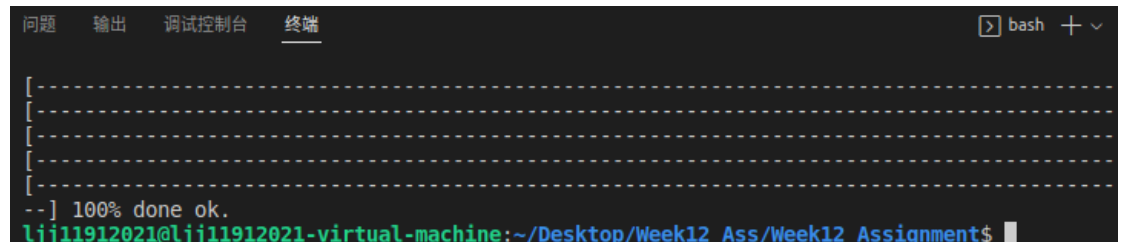
Result (5 times):

```
--] 100% done ok.
ljj11912021@ljj11912021-virtual-machine:~/Desktop/Week12 Ass/Week12 Assignment$
```

Design idea two: Use each person's number. The ones with even number pick their left forks first and then pick the right forks. The ones with odd number pick their right forks first and then pick the left forks.

```c
void init() {
    // write code if you desire.

}

void wants_to_eat(int p_no) {
    // fixme

    if (p_no % 2 == 0) {
        pick_left_fork(p_no);
        pick_right_fork(p_no);
    }
    else {
        pick_right_fork(p_no);
        pick_left_fork(p_no);
    }

    eat(p_no);

    put_left_fork(p_no);
    put_right_fork(p_no);
}
```

Result (5 times):


```
--------] 100% done ok.
ljj11912021@ljj11912021-virtual-machine:~/Desktop/Week12 Ass/Week12 Assignment$
```



```
--------] 100% done ok.
ljj11912021@ljj11912021-virtual-machine:~/Desktop/Week12 Ass/Week12 Assignment$
```

3. Design idea: Use two semaphores: sem_drink and sem_buy. Sem_drink is initialized as 2 and sem_buy is initialized as 0. At the beginning of every producer (mom, dad, grandfather), execute wait(sem_drink), and at the end of every producer, execute post(sem_buy). At the beginning of consumer (son), execute wait(sem_buy), and at the end of consumer, execute post(sem_drink). Therefore, before the fridge is empty, there would always be someone to buy milk, and before the fridge is full, there would always be the consumer to drink milk. Therefore the fridge will not be empty or full.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <pthread.h>
#include <semaphore.h>
#include <fcntl.h>


pthread_mutex_t fri_lock;
sem_t * sem_buy;
sem_t * sem_drink;

void *mom(int *num){
    for(int i=0;i<10;i++){
        sem_wait(sem_drink);
        pthread_mutex_lock(&fri_lock);
        printf("Mom comes home.\n");
        sleep(rand()%2+1);
        printf("Mom goes to buy milk.\n");
        *num += 1;
        if (*num > 2){
            printf("What a waste of food! The fridge can not hold
            while(1)printf("TAT~");
        }
        printf("Mom puts milk in fridge and leaves.\n");
        pthread_mutex_unlock(&fri_lock);
        sem_post(sem_buy);
    }
}
```

```c
31    void *dad(int *num){
32        for(int i=0;i<10;i++){
33            sem_wait(sem_drink);
34            pthread_mutex_lock(&fri_lock);
35            printf("Dad comes home.\n");
36            sleep(rand()%2+1);
37            printf("Dad goes to buy milk.\n");
38            *num += 1;
39            if (*num > 2){
40                printf("What a waste of food! The fridge can not hold
41                while(1)printf("TAT~");
42            }
43            printf("Dad puts milk in fridge and leaves.\n");
44            pthread_mutex_unlock(&fri_lock);
45            sem_post(sem_buy);
46        }
47    }
```

```c
49    void *grandfather(int *num){
50        for(int i=0;i<10;i++){
51            sem_wait(sem_drink);
52            pthread_mutex_lock(&fri_lock);
53            printf("Grandfather comes home.\n");
54            sleep(rand()%2+1);
55            printf("Grandfather goes to buy milk.\n");
56            *num += 1;
57            if (*num > 2){
58                printf("What a waste of food! The fridge can not hold
59                while(1){
60                    printf("TAT~");
61                }
62            }
63            printf("Grandfather puts milk in fridge and leaves.\n");
64            pthread_mutex_unlock(&fri_lock);
65            sem_post(sem_buy);
66        }
67    }
```

```c
69    void *son(int *num){
70        for(int i = 0; i < 30 ; i++){
71            sem_wait(sem_buy);
72            pthread_mutex_lock(&fri_lock);
73            printf("Son comes home.\n");
74            if(*num == 0){
75                printf("The fridge is empty!\n");
76                while(1){
77                    printf("TAT~");
78                }
79            }
80            printf("Son fetches a milk\n");
81            *num -= 1;
82            printf("Son leaves\n");
83            pthread_mutex_unlock(&fri_lock);
84            sem_post(sem_drink);
85        }
86    }
```

```c
88      int main(int argc, char * argv[]) {
89          srand(time(0));
90
91          int num_milk = 0;
92          pthread_t p1, p2, p3, p4;
93          pthread_mutex_init(&fri_lock,NULL);
94
95          sem_drink = sem_open("empty", O_CREAT, 0666, 2);
96          sem_buy = sem_open("full", O_CREAT, 0666, 0);
97
98          // Create two threads (both run func)
99          pthread_create(&p1, NULL, mom, &num_milk);
100         pthread_create(&p2, NULL, dad, &num_milk);
101         pthread_create(&p3, NULL, grandfather, &num_milk);
102         pthread_create(&p4, NULL, son, &num_milk);
103
104         // Wait for the threads to end.
105         pthread_join(p1, NULL);
106         pthread_join(p2, NULL);
107         pthread_join(p3, NULL);
108         pthread_join(p4, NULL);
109
110         sem_close(sem_buy);
111         sem_close(sem_drink);
112
113         printf("success!\n");
114     }
```

Result:

```
问题    输出    调试控制台    终端                          >_ bash  + ∨  ⊓ 🗑 ∧

Son leaves
Grandfather comes home.
Grandfather goes to buy milk.
Grandfather puts milk in fridge and leaves.
Son comes home.
Son fetches a milk
Son leaves
success!
ljj11912021@ljj11912021-virtual-machine:~/Desktop/Week12 Ass/Week12 Assignment$
```