EX1:

TA:

sem_init (&sem1, 0, 1)

Others:

sem_init (&sem, 0, 0)

There must be a separate semaphore for everyone because of the thread cyclic.

EX2:

1.Vector generate:

```
for (j=0; i<n; j++) {
        dnum = rand()/(double)(RAND_MAX); //生成向量ab
        a[j] = dnum;
        dnum = rand()/(double)(RAND_MAX);
        b[j] = dnum;
 }
```

2.Shared variables to making child threads and main thread inform to each other

```
double *a,*b,*ans;
int homeworks[10]={0},homeworks_done[10]={0};
int vec_index=0,hm_index=0;
```

3.Shared variables used to hold vectors and answers

```
double *a,*b,*ans;
```

4.thread function

```
while (1) {
    while (!homeworks[hm_index]); //wait for main thread's available information
     sem_wait (&sem);
     ai = a[vec_index];
```

```
        bi = b[vec_index];
        abi = ai*bi;
        printf("%lf ",abi);
        vec_index++;   //vector index move
        ans[hm_index]+=abi;    //add the product of a[i] and b[i] to current
answer
        if (vec_index == 10){
           homeworks_done[hm_index]=1;   //informs the main thread that it has
completed its job
          hm_index++;
        }
        sem_post (&sem);
        sleep (1);
}
```

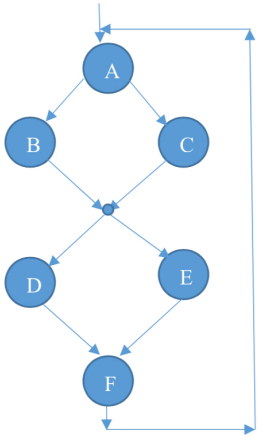
6.main thread lock:

```
for (i=0; i<10; i++) {
        sem_wait (&sem);
        vec_index=0;
        for (j=0; i<n; j++) {
           dnum = rand()/(double)(RAND_MAX);
           a[j] = dnum;
           dnum = rand()/(double)(RAND_MAX);
           b[j] = dnum;
        }
        homeworks[i]=1;
        sem_post (&sem);
        printf("homework[%d]:\n",hm_index);
        while (!homeworks_done[i]);
        printf("homework[%d] dot product:%lf\n",hm_index-1,ans[hm_index-1]);
  }
```

7.kills child threads

```
for (i=0; i<t; i++) {
        pthread_kill(tids[i],SIGQUIT);
}
```

EX3:



transfer to: