Homework #9

Problem 1

A. The program in the figure below has a bug. The thread is supposed to sleep for 1 second and then print a string. However, when we run it on our system, nothing prints. Why?

B. You can fix this bug by replacing the exit function in line 9 with a Pthread function calls. What's it?

```
1   #include “csapp.h”
2   void *thread(void *vargp);
3
4   int main()
5   {
6           pthread_t tid;
7
8           Pthread_create(&tid, NULL, thread, NULL);
9           exit(0);
10  }
11
12  /* Thread routine */
13  void *thread(void *vargp)
14  {
15          Sleep(1);
16          printf(“Hello, world!\n”);
17          return NULL;
18  }
```

Solution

A. The problem is that the main thread calls exit without waiting for the peer thread to terminate. The exit call terminates the entire process, including any threads that happen to be running. So the peer thread is being killed before it has a chance to print its output string.

B. We can fix the bug by replacing the exit function with Pthread_exit, which waits for outstanding threads to terminate before it terminates the process.

Problem 2

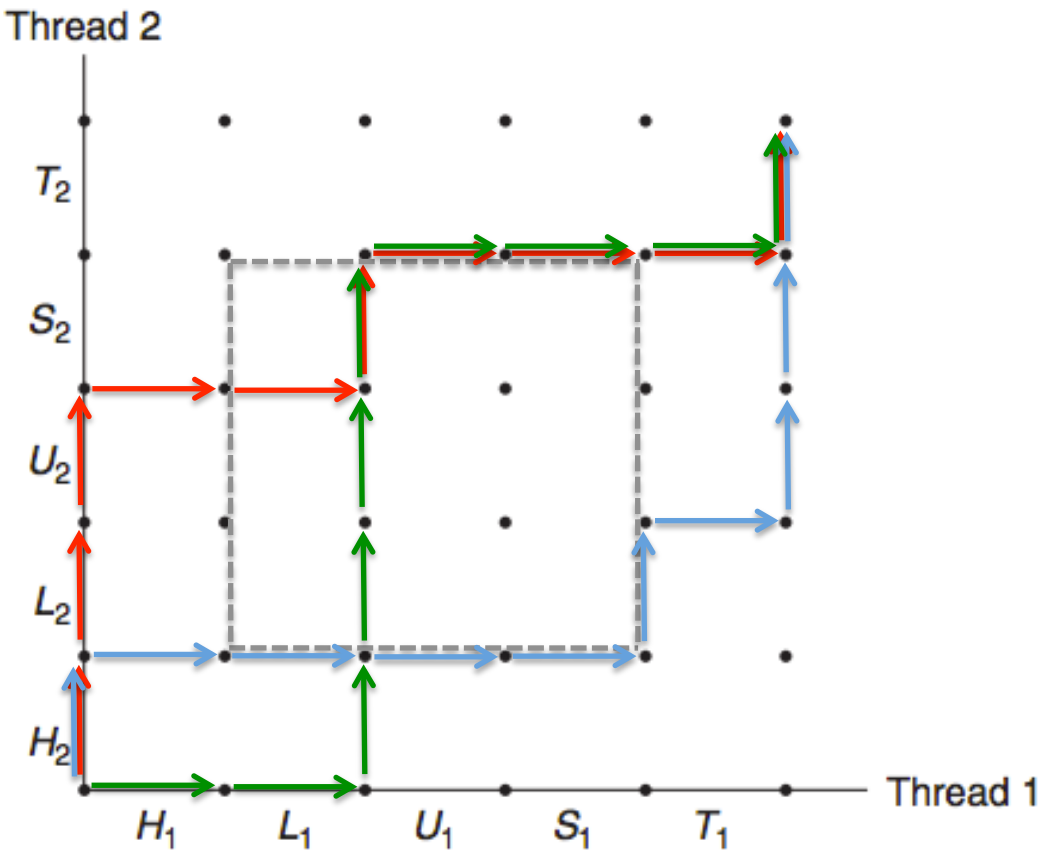
Using the progress graph in Figure 12.21, classify the following trajectories as either safe or unsafe. And draw them out.

A. H2, L2, U2, H1, L1, S2, U1, S1, T1, T2

B. H2, H1, L1, U1, S1, L2, T1, U2, S2, T2

C. H1, L1, H2, L2, U2, S2, U1, S1, T1, T2

Solution



A: unsafe, red line
B: safe, blue line
C: unsafe, green line

Problem 3

Refer to the I/O multiplexing based concurrent echo server in Figure 12.8, indicate the pros and cons of this event-driven programming approach based on I/O multiplexing.

Solution(主观题，答到大致的点即给分)

Pros:

1. Event-driven designs give programmers more control over the behavior of their programs than process-based designs
2. Easy to share data between flows.
3. Event-driven designs are often significantly more efficient than process-based designs because they do not require a process context switch to schedule a new flow

Cons:

1. Significantly more complex to code than process- or thread-based designs.
2. Hard to provide fine-grained concurrency
3. Cannot take advantage of multi-core

Problem 4

Suppose this scenario, there's a lift with a max capacity of 18 persons at the same time (each person implemented as a thread). The only thing that a person will do in the lift is going up and down, which is indicated by the function "onLift()". Please fill in the code segment where is marked by *"Your code here"*. (Use semaphore to guarantee the thread safety.)

Solution

```
#include "csapp.h"
void *thread(void *vargp);
unsigned int cnt;       /* count of persons */
sem_t sem;              /* semaphore */

int main()
{
        pthread_t tid1, ...;

        /* Initialize the semaphore. Your code here. */
        Sem_init(&sem, 0, 18);

        /* Create threads and wait */
        ...
        exit(0);
}

/* Thread routine */
void *thread(void *vargp)
{
        /* Your code here. */
        P(&sem);
        onLift();
        V(&sem);
        return NULL;
}
```