Simon Yoon
Operating Systems
Problem Set 6

Problem 1 -- Short Answer
Below is a function which inserts list element `what` after existing element `where` in a singly-linked list.

```
struct ll {
          struct ll *fwd;
          /* and other stuff */
};
void ll_insert(struct ll *where,struct ll *what)
{
          what->fwd = where->fwd;
          where->fwd = what;
}
```

A) Discuss why this function is not safe in a concurrent situation, i.e. where two or more threads try to execute `ll_insert` at the same time. Give specific examples of what can go wrong.

If multiple threads execute `ll_insert` "simultaneously", the insertion of two '`what`'s into the node given what all we know is not atomic and therefore, it is prone to a race condition. Race conditions lead to erroneous program behavior and an intentional update of '`what`' could be lost.

B) Modify the code above to make it safe for the STU case (as defined in the lecture notes) where our concern is only with a signal handler.

```
struct ll {
          struct ll *fwd;
          /* and other stuff */
};
void ll_insert(struct ll *where,struct ll *what)
{
          sigset_t oldmask, newmask;
          sigset_fill(&newmask);
          sigprocmask(SIG_BLOCK, &newmask, &oldmask);
          what->fwd = where->fwd;
          where->fwd = what;
          sigprocmask(SIG_SETMASK, &oldmask, NULL);
}
```

C) Now modify the original code in a different way so it can be safe for the MTU case, e.g. a multithreaded process, or multiple single-threaded processes sharing a data structure in shared memory

```
struct ll {
            int spinlock; /* 0 is unlocked, NZ is locked */
            struct ll *fwd;
            /* and other stuff */
};
void ll_insert(struct ll *where,struct ll *what)
{
            while (TAS(&where->spinlock)!=0);
            /* BEGIN CRITICAL REGION */
            what->fwd = where->fwd;
            where->fwd = what;
            /* END CRITICAL REGION */
            where->spinlock =0; /*unlock spin lock*/

            /*
            A mutex such as a spinlock is sufficient. The critical
            region is small, so the number of retries is minimal and
            won't invoke an endless locking mechanism and the atomic
            test and set instruction (TAS) prevents data races.
            */
}
```