

// Make this Linked List implementation to be thread-safe!

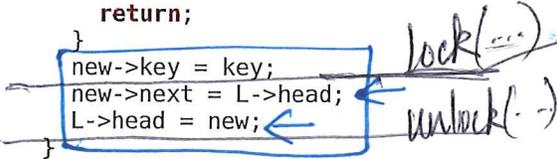
```
#include <stdio.h>
#include <stdlib.h>
```

```
typedef struct __node_t {
    int key;
    struct __node_t *next;
} node_t;
```

```
typedef struct __list_t {
    node_t *head;
} list_t;
lock_t mlock;
```

```
void List_Init(list_t *L) { L->head = NULL; }
```

```
void List_Insert(list_t *L, int key) {
    node_t *new = malloc(sizeof(node_t));
    if (new == NULL) {
        perror("malloc");
        return;
    }
```

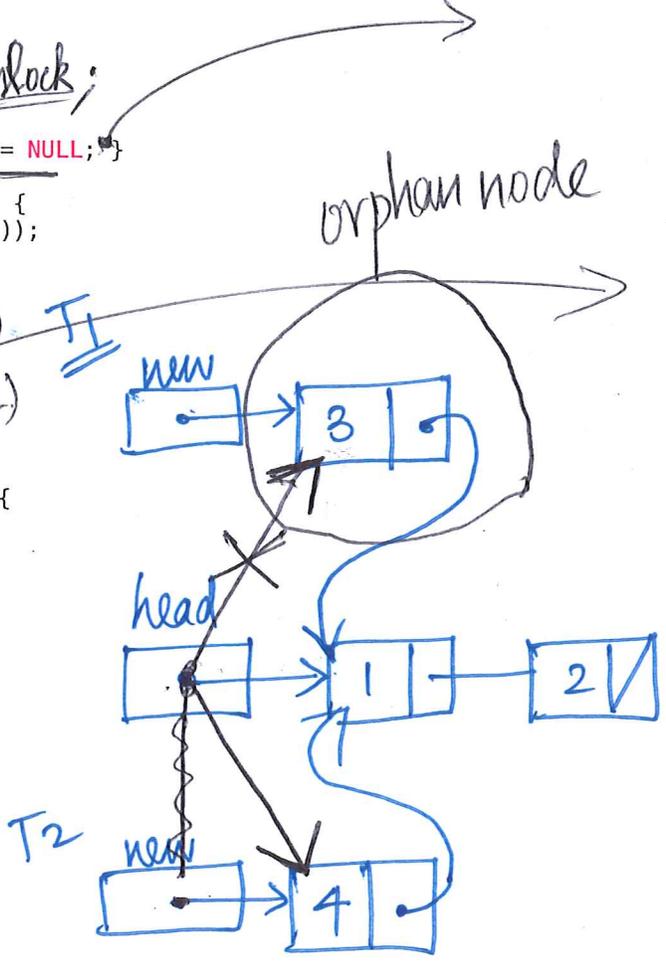


```
int List_Lookup(list_t *L, int key) {
    node_t *tmp = L->head;
    while (tmp) {
        if (tmp->key == key) return 1;
        tmp = tmp->next;
    }
    return 0;
}
```

```
void List_Print(list_t *L) {
    node_t *tmp = L->head;
    while (tmp) {
        printf("%d ", tmp->key);
        tmp = tmp->next;
    }
    printf("\n");
}
```

```
int main(int argc, char *argv[]) {
    list_t mylist;
    List_Init(&mylist);
    List_Insert(&mylist, 10);
    List_Insert(&mylist, 30);
    List_Insert(&mylist, 5);
    List_Print(&mylist);
    printf("In List: 10? %d 20? %d\n", List_Lookup(&mylist, 10),
        List_Lookup(&mylist, 20));
    return 0;
}
```

T₁ T₂



0. Disabling interrupts Locks! (Hardware Locks).

1. TEST AND SET. (ATOMIC XCHG).

x86 - xchgl → locked version.

2. Compare And Swap
x86: cmpxchgl

3. Load Linked & Store Conditional

tricky!

TEMPLATE: FILL THIS IN TO MAKE YOUR OWN LOCK

```
typedef struct __lock_t {  
    // whatever data structs you need goes here  
} lock_t;  
  
void init(lock_t *lock) {  
    // init code goes here  
}  
  
void acquire(lock_t *lock) {  
    // lock acquire code goes here  
}  
  
void release(lock_t *lock) {  
    // lock release code goes here  
}
```

FIRST PRIMITIVE: TEST-AND-SET (or ATOMIC EXCHANGE)

```
// given ptr, sets *ptr to new value; returns the old value at *ptr  
int Exchange(int *lock, int new) {  
    int old = *lock;  
    *lock = new;  
    return old;  
}
```

SECOND PRIMITIVE: COMPARE-AND-SWAP

```
int CompareAndSwap(int *ptr, int expected, int new) {  
    int actual = *ptr;  
    if (actual == expected)  
        *ptr = new;  
    return actual;  
}
```

x86: cmpxchgl

```
void lock (lock_t *m) {  
    while (CompareAndSwap(m, 0, 1) == 1)  
        ;  
}
```

THIRD PRIMITIVE(S): LOAD-LINKED, STORE-CONDITIONAL

```
int LoadLinked(int *ptr) {  
    return *ptr;  
}  
  
int StoreConditional(int *ptr, int value) {  
    if (no one has updated *ptr since LoadLinked to this address) {  
        *ptr = value;  
        return 1; // success  
    } else {  
        return 0; // fail (does not do the store)  
    }  
}
```

```
void lock (lock_t *m) {  
    while (1) {  
        while (LoadLinked(m) == 1)  
            ;  
        if (storeConditional(m, 1) == 1)  
            return;  
    }  
}
```

}}}

Evaluate

| | (mutual exclusion?) | T&S | Q&S | LL&SC |
|-------------------|---------------------|-----|-----|-------|
| 1. correctness | ✓ | ✓ | ✓ | ✓ |
| 2. fairness X | | X | X | X |
| 3. performance. X | | X | X | X |

