Program A: a list in which is a mutex is used so that it is **THREAD SAFE**

However in the case of the function contains where we only read data from the list the normal mutex lock is too restrictive for what we need. That is why a read/write lock would be a good idea -> Program B

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

struct lnode {

int value;

struct lnode\* next;

};

struct list {

struct lnode\* head;

pthread\_mutex\_t lock;

};

void init(struct list\* l) {

l->head = NULL;

pthread\_mutex\_init(&l->lock, NULL);

}

void append(struct list\* l, int v) {

struct lnode\* p;

struct lnode\* n = (struct lnode\*)malloc(sizeof(struct lnode));

n->value = v;

n->next = NULL;

pthread\_mutex\_lock(&l->lock);

if (l->head == NULL)

l->head = n;

else {

p = l->head;

while(p->next != NULL)

p = p->next;

p->next = n;

}

pthread\_mutex\_unlock(&l->lock);

}

void delete(struct list\* l, int v) {

struct lnode\* p;

struct lnode\* n;

pthread\_mutex\_lock(&l->lock);

if (l->head != NULL) {

p = l->head;

if (p->value == v) {

n = p->next;

free(p);

l->head = n;

}

else {

while(p->next != NULL && p->value != v)

n = p;

p = p->next;

if ((p->next != NULL) || (p->next == NULL && p->value == v)) {

n = p->next;

free(p);

}

}

}

pthread\_mutex\_unlock(&l->lock);

}

int contains(struct list\* l, int v) {

/\*although contains only iterates the list and only "reads" its interaction with the other functions may lead to problems\*/

pthread\_mutex\_lock(&l->lock); /\*the lock should be made here so that it is safe to access the head\*/

struct lnode\* p = l->head;

while(p != NULL) {

if(p->value == v)

return 1;

p = p->next;

}

pthread\_mutex\_unlock(&l->lock);

return 0;

}

int destroy(struct list\* l) {

struct lnode\* x;

/\*from a logocal point of view destroy should be protected as it iterates throuh the list but from a user's point of view it shouldn't so that any possible errors that may arise are shown\*/

struct lnode\* p = l->head;

while(p != NULL) {

x = p;

p = p->next;

free(x);

}

pthread\_mutex\_destroy(&l->lock);

return 0;

}

int main(int arg, char\*\* argv) {

struct list l;

init(&l);

append(&l, 1);

append(&l, 2);

if(contains(&l, 3))

printf("Oops!\n");

if(contains(&l, 1))

printf("OK!\n");

return 0;

}

Program B: using read/write lock so that it is not overly restrictive

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

struct lnode {

int value;

struct lnode\* next;

};

struct list {

struct lnode\* head;

pthread\_rwlock\_t lock;

};

void init(struct list\* l) {

l->head = NULL;

pthread\_rwlock\_init(&l->lock, NULL);

}

void append(struct list\* l, int v) {

struct lnode\* p;

struct lnode\* n = (struct lnode\*)malloc(sizeof(struct lnode));

n->value = v;

n->next = NULL;

pthread\_rwlock\_wrlock(&l->lock);

if (l->head == NULL)

l->head = n;

else {

p = l->head;

while(p->next != NULL)

p = p->next;

p->next = n;

}

pthread\_rwlock\_unlock(&l->lock);

}

void delete(struct list\* l, int v) {

struct lnode\* p;

struct lnode\* n;

pthread\_rwlock\_wrlock(&l->lock);

if (l->head != NULL) {

p = l->head;

if (p->value == v) {

n = p->next;

free(p);

l->head = n;

}

else {

while(p->next != NULL && p->value != v)

n = p;

p = p->next;

if ((p->next != NULL) || (p->next == NULL && p->value == v)) {

n = p->next;

free(p);

}

}

}

pthread\_rwlock\_unlock(&l->lock);

}

int contains(struct list\* l, int v) {

/\*although contains only iterates the list and only "reads" its interaction with the other functions may lead to problems\*/

pthread\_rwlock\_rdlock(&l->lock); /\*the lock should be made here so that it is safe to access the head\*/

struct lnode\* p = l->head;

while(p != NULL) {

if(p->value == v) {

pthread\_rwlock\_unlock(&l->lock);

return 1;

}

p = p->next;

}

pthread\_rwlock\_unlock(&l->lock);

return 0;

}

int destroy(struct list\* l) {

struct lnode\* x;

/\*from a logocal point of view destroy should be protected as it iterates throuh the list but from a user's point of view it shouldn't so that any possible errors that may arise are shown\*/

struct lnode\* p = l->head;

while(p != NULL) {

x = p;

p = p->next;

free(x);

}

pthread\_rwlock\_destroy(&l->lock);

return 0;

}

int main(int arg, char\*\* argv) {

struct list l;

init(&l);

append(&l, 1);

append(&l, 2);

if(contains(&l, 3))

printf("Oops!\n");

if(contains(&l, 1))

printf("OK!\n");

return 0;

}

Program C: using a semaphore so that the access to the list simultaneously is restricted to a number of threads (5 in this case)

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

struct lnode {

int value;

struct lnode\* next;

};

struct list {

struct lnode\* head;

pthread\_rwlock\_t lock;

sem\_t sem;

};

void init(struct list\* l) {

l->head = NULL;

pthread\_rwlock\_init(&l->lock, NULL);

sem\_init(&l->sem, 0, 5);

/\* 0: only the threads in this process can fight for the semaphore

\* 1: the threads from ouside of the process can also fight for it

\*/

}

void append(struct list\* l, int v) {

struct lnode\* p;

struct lnode\* n = (struct lnode\*)malloc(sizeof(struct lnode));

n->value = v;

n->next = NULL;

sem\_wait(&l->sem);

pthread\_rwlock\_wrlock(&l->lock);

if (l->head == NULL)

l->head = n;

else {

p = l->head;

while(p->next != NULL)

p = p->next;

p->next = n;

}

pthread\_rwlock\_unlock(&l->lock);

sem\_post(&l->sem);

}

void delete(struct list\* l, int v) {

struct lnode\* p;

struct lnode\* n;

sem\_wait(&l->sem);

pthread\_rwlock\_wrlock(&l->lock);

if (l->head != NULL) {

p = l->head;

if (p->value == v) {

n = p->next;

free(p);

l->head = n;

}

else {

while(p->next != NULL && p->value != v)

n = p;

p = p->next;

if ((p->next != NULL) || (p->next == NULL && p->value == v)) {

n = p->next;

free(p);

}

}

}

pthread\_rwlock\_unlock(&l->lock);

sem\_post(&l->sem);

}

int contains(struct list\* l, int v) {

/\*although contains only iterates the list and only "reads" its interaction with the other functions may lead to problems\*/

sem\_wait(&l->sem);

pthread\_rwlock\_rdlock(&l->lock); /\*the lock should be made here so that it is safe to access the head\*/

struct lnode\* p = l->head;

while(p != NULL) {

if(p->value == v) {

pthread\_rwlock\_unlock(&l->lock);

sem\_post(&l->sem);

return 1;

}

p = p->next;

}

pthread\_rwlock\_unlock(&l->lock);

sem\_post(&l->sem);

return 0;

}

int destroy(struct list\* l) {

struct lnode\* x;

/\*from a logocal point of view destroy should be protected as it iterates throuh the list but from a user's point of view it shouldn't so that any possible errors that may arise are shown\*/

struct lnode\* p = l->head;

while(p != NULL) {

x = p;

p = p->next;

free(x);

}

pthread\_rwlock\_destroy(&l->lock);

sem\_destroy(&l->sem);

return 0;

}

int main(int arg, char\*\* argv) {

struct list l;

init(&l);

append(&l, 1);

append(&l, 2);

if(contains(&l, 3))

printf("Oops!\n");

if(contains(&l, 1))

printf("OK!\n");

return 0;

}