System Programming Lecture - Report 7

Student Number: 201520740

Name: LIN WEI

Exercise 1

Program

**main.c**

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

#include <pthread.h>

#define ADD\_ENTRY\_NUM 30

#define DELETE\_ENTRY\_NUM 10

#define ENQUEUE\_THREAD\_NUM 2

#define DEQUEUE\_THREAD\_NUM 6

/\* Global mutex for structured code locking. \*/

pthread\_mutex\_t list\_lock = PTHREAD\_MUTEX\_INITIALIZER;

struct entry {

struct entry \*next;

void \*data;

};

struct entry \*entry;

struct list {

struct entry \*head;

struct entry \*\*tail;

};

struct list \*list;

struct list \*

list\_init(void)

{

struct list \*list;

list = malloc(sizeof \*list);

if (list == NULL)

return (NULL);

list->head = NULL;

list->tail = &list->head;

return (list);

}

int

list\_enqueue(struct list \*list, void \*data)

{

struct entry \*e;

char \*d = (char \*) data;

e = malloc(sizeof \*e);

if (e == NULL)

return (1);

e->next = NULL;

e->data = data;

pthread\_mutex\_lock(&list\_lock);

\*list->tail = e;

list->tail = &e->next;

printf("Enqueued [%s]\n", d);

pthread\_mutex\_unlock(&list\_lock);

return (0);

}

struct entry \*

list\_dequeue(struct list \*list)

{

struct entry \*e;

char \* d;

pthread\_mutex\_lock(&list\_lock);

if (list->head == NULL)

{

printf("Dequeue failed: empty list\n");

pthread\_mutex\_unlock(&list\_lock);

return(NULL);

}

e = list->head;

list->head = e->next;

if (list->head == NULL)

list->tail = &list->head;

d = (char \*)e->data;

printf("Dequeued [%s]\n", d);

pthread\_mutex\_unlock(&list\_lock);

return (e);

}

struct entry \*

list\_traverse(struct list \*list, int (\*func)(void \*, void \*), void \*user)

{

struct entry \*\*prev, \*n, \*next;

pthread\_mutex\_lock(&list\_lock);

if (list == NULL)

{

pthread\_mutex\_unlock(&list\_lock);

return (NULL);

}

prev = &list->head;

for (n = list->head; n != NULL; n = next) {

next = n->next;

switch (func(n->data, user)) {

case 0:

/\* continues \*/

prev = &n->next;

break;

case 1:

/\* delete the entry \*/

\*prev = next;

if (next == NULL)

list->tail = prev;

pthread\_mutex\_unlock(&list\_lock);

return (n);

case -1:

default:

/\* traversal stops \*/

pthread\_mutex\_unlock(&list\_lock);

return (NULL);

}

}

pthread\_mutex\_unlock(&list\_lock);

return (NULL);

}

int

print\_entry(void \*e, void \*u)

{

printf("%s\n", (char \*)e);

return (0);

}

int

delete\_entry(void \*e, void \*u)

{

char \*c1 = e, \*c2 = u;

return (!strcmp(c1, c2));

}

/\* Thread of enqueue. \*/

void \* enqueue(void \*arg)

{

int i;

char buffer[100];

for(i = 0; i < ADD\_ENTRY\_NUM; i++)

{

sprintf(buffer, "%d", i);

list\_enqueue(list, strdup(buffer));

}

}

/\* Thread of dequeue. \*/

void \* dequeue(void \*arg)

{

int i;

for(i = 0; i < DELETE\_ENTRY\_NUM; i++)

{

list\_dequeue(list);

}

}

int

main()

{

pthread\_t enq[ENQUEUE\_THREAD\_NUM], deq[DEQUEUE\_THREAD\_NUM];

int i;

list = list\_init();

/\* Create enqueue and dequeue threads. \*/

for (i = 0; i < ENQUEUE\_THREAD\_NUM || i < DEQUEUE\_THREAD\_NUM; i++)

{

if(i < ENQUEUE\_THREAD\_NUM)

pthread\_create(&enq[i], NULL, enqueue, NULL);

if(i < DEQUEUE\_THREAD\_NUM)

pthread\_create(&deq[i], NULL, dequeue, NULL);

}

for(i = 0; i < ENQUEUE\_THREAD\_NUM; i++)

pthread\_join(enq[i], NULL);

for(i = 0; i < DEQUEUE\_THREAD\_NUM; i++)

pthread\_join(deq[i], NULL);

/\* Delete the entry with data "2". \*/

entry = list\_traverse(list, delete\_entry, "2");

if (entry != NULL) {

free(entry->data);

free(entry);

}

/\* Traverse the list. \*/

list\_traverse(list, print\_entry, NULL);

free(list);

return (0);

}

Result

lw@lw-VirtualBox:~/Documents/Report7/Exercise1$ ./a.out

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Enqueued [0]

Enqueued [1]

Enqueued [2]

Enqueued [3]

Enqueued [4]

Enqueued [5]

Enqueued [6]

Enqueued [7]

Enqueued [8]

Enqueued [9]

Enqueued [10]

Enqueued [11]

Enqueued [12]

Enqueued [13]

Enqueued [14]

Enqueued [15]

Enqueued [16]

Enqueued [17]

Enqueued [18]

Enqueued [19]

Enqueued [20]

Enqueued [21]

Enqueued [22]

Enqueued [23]

Enqueued [24]

Enqueued [25]

Enqueued [26]

Enqueued [27]

Enqueued [28]

Enqueued [29]

Dequeued [0]

Dequeued [1]

Dequeued [2]

Dequeued [3]

Dequeued [4]

Dequeued [5]

Dequeued [6]

Dequeued [7]

Dequeued [8]

Dequeued [9]

Enqueued [0]

Enqueued [1]

Enqueued [2]

Enqueued [3]

Enqueued [4]

Enqueued [5]

Enqueued [6]

Enqueued [7]

Enqueued [8]

Enqueued [9]

Enqueued [10]

Enqueued [11]

Enqueued [12]

Enqueued [13]

Enqueued [14]

Enqueued [15]

Enqueued [16]

Enqueued [17]

Enqueued [18]

Enqueued [19]

Enqueued [20]

Enqueued [21]

Enqueued [22]

Enqueued [23]

Enqueued [24]

Enqueued [25]

Enqueued [26]

Enqueued [27]

Enqueued [28]

Enqueued [29]

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

0

1

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

Consideration on Exercise 1

As we can see, all these threads ran correctly. First, the dequeue threads tried to delete entries but failed because the list was empty. After the enqueue threads have added some entries into the list, the remaining dequeue threads began to delete entries from the list. Finally, other enqueue threads added more entries into the list. And after the program deleted the entry with data”2”, *list\_traverse* function showed there was no data “2” among the remaining entries.

Exercise 2

Program

**main.c**

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

#include <pthread.h>

#define ADD\_ENTRY\_NUM 30

#define DELETE\_ENTRY\_NUM 10

#define ENQUEUE\_THREAD\_NUM 2

#define DEQUEUE\_THREAD\_NUM 6

struct entry {

struct entry \*next;

void \*data;

};

struct entry \*entry;

struct list {

struct entry \*head;

struct entry \*\*tail;

/\* Mutex for structured data locking. \*/

pthread\_mutex\_t list\_lock;

};

struct list \*list;

struct list \*

list\_init(void)

{

struct list \*list;

list = malloc(sizeof \*list);

if (list == NULL)

return (NULL);

list->head = NULL;

list->tail = &list->head;

/\* Init of the mutex. \*/

pthread\_mutex\_init(&list->list\_lock, NULL);

return (list);

}

int

list\_enqueue(struct list \*list, void \*data)

{

struct entry \*e;

char \*d = (char \*) data;

e = malloc(sizeof \*e);

if (e == NULL)

return (1);

e->next = NULL;

e->data = data;

pthread\_mutex\_lock(&list->list\_lock);

\*list->tail = e;

list->tail = &e->next;

printf("Enqueued [%s]\n", d);

pthread\_mutex\_unlock(&list->list\_lock);

return (0);

}

struct entry \*

list\_dequeue(struct list \*list)

{

struct entry \*e;

char \* d;

pthread\_mutex\_lock(&list->list\_lock);

if (list->head == NULL)

{

printf("Dequeue failed: empty list\n");

pthread\_mutex\_unlock(&list->list\_lock);

return(NULL);

}

e = list->head;

list->head = e->next;

if (list->head == NULL)

list->tail = &list->head;

d = (char \*)e->data;

printf("Dequeued [%s]\n", d);

pthread\_mutex\_unlock(&list->list\_lock);

return (e);

}

struct entry \*

list\_traverse(struct list \*list, int (\*func)(void \*, void \*), void \*user)

{

struct entry \*\*prev, \*n, \*next;

pthread\_mutex\_lock(&list->list\_lock);

if (list == NULL)

{

pthread\_mutex\_unlock(&list->list\_lock);

return (NULL);

}

prev = &list->head;

for (n = list->head; n != NULL; n = next) {

next = n->next;

switch (func(n->data, user)) {

case 0:

/\* continues \*/

prev = &n->next;

break;

case 1:

/\* delete the entry \*/

\*prev = next;

if (next == NULL)

list->tail = prev;

pthread\_mutex\_unlock(&list->list\_lock);

return (n);

case -1:

default:

/\* traversal stops \*/

pthread\_mutex\_unlock(&list->list\_lock);

return (NULL);

}

}

pthread\_mutex\_unlock(&list->list\_lock);

return (NULL);

}

int

print\_entry(void \*e, void \*u)

{

printf("%s\n", (char \*)e);

return (0);

}

int

delete\_entry(void \*e, void \*u)

{

char \*c1 = e, \*c2 = u;

return (!strcmp(c1, c2));

}

/\* Thread of enqueue. \*/

void \* enqueue(void \*arg)

{

int i;

char buffer[100];

for(i = 0; i < ADD\_ENTRY\_NUM; i++)

{

sprintf(buffer, "%d", i);

list\_enqueue(list, strdup(buffer));

}

}

/\* Thread of dequeue. \*/

void \* dequeue(void \*arg)

{

int i;

for(i = 0; i < DELETE\_ENTRY\_NUM; i++)

{

list\_dequeue(list);

}

}

int

main()

{

pthread\_t enq[ENQUEUE\_THREAD\_NUM], deq[DEQUEUE\_THREAD\_NUM];

int i;

list = list\_init();

/\* Create enqueue and dequeue threads. \*/

for (i = 0; i < ENQUEUE\_THREAD\_NUM || i < DEQUEUE\_THREAD\_NUM; i++)

{

if(i < ENQUEUE\_THREAD\_NUM)

pthread\_create(&enq[i], NULL, enqueue, NULL);

if(i < DEQUEUE\_THREAD\_NUM)

pthread\_create(&deq[i], NULL, dequeue, NULL);

}

for(i = 0; i < ENQUEUE\_THREAD\_NUM; i++)

pthread\_join(enq[i], NULL);

for(i = 0; i < DEQUEUE\_THREAD\_NUM; i++)

pthread\_join(deq[i], NULL);

/\* Delete the entry with data "2". \*/

entry = list\_traverse(list, delete\_entry, "2");

if (entry != NULL) {

free(entry->data);

free(entry);

}

/\* Traverse the list. \*/

list\_traverse(list, print\_entry, NULL);

free(list);

return (0);

}

Result

lw@lw-VirtualBox:~/Documents/Report7/Exercise2$ ./a.out

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Dequeue failed: empty list

Enqueued [0]

Enqueued [1]

Enqueued [2]

Enqueued [3]

Enqueued [4]

Enqueued [5]

Enqueued [6]

Enqueued [7]

Enqueued [8]

Enqueued [9]

Enqueued [10]

Enqueued [11]

Enqueued [12]

Enqueued [13]

Enqueued [14]

Enqueued [15]

Enqueued [16]

Enqueued [17]

Enqueued [18]

Enqueued [19]

Enqueued [20]

Enqueued [21]

Enqueued [22]

Enqueued [23]

Enqueued [24]

Enqueued [25]

Enqueued [26]

Enqueued [27]

Enqueued [28]

Enqueued [29]

Dequeued [0]

Dequeued [1]

Dequeued [2]

Dequeued [3]

Dequeued [4]

Dequeued [5]

Dequeued [6]

Dequeued [7]

Dequeued [8]

Dequeued [9]

Enqueued [0]

Enqueued [1]

Enqueued [2]

Enqueued [3]

Enqueued [4]

Enqueued [5]

Enqueued [6]

Enqueued [7]

Enqueued [8]

Enqueued [9]

Enqueued [10]

Enqueued [11]

Enqueued [12]

Enqueued [13]

Enqueued [14]

Enqueued [15]

Enqueued [16]

Enqueued [17]

Enqueued [18]

Enqueued [19]

Enqueued [20]

Enqueued [21]

Enqueued [22]

Enqueued [23]

Enqueued [24]

Enqueued [25]

Enqueued [26]

Enqueued [27]

Enqueued [28]

Enqueued [29]

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

0

1

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

Consideration on Exercise 2

All these threads ran correctly and the result was the same as exercise 1.By these two exercises, I have gotten the difference between code locking and data locking in programming.

Review of this lecture

In this lecture, I have learned the basic of code locking, data locking and the deadlock problem. I find I always tend to forget unlocking the mutex after the *return* statement within some *if* blocks. We really should take care of this kind of miss because it may cause the deadlock problem when all the threads are sharing one Mutex lock.