Homework 4 – Ταυτόχρονος Προγραμματισμός

Γαρυφαλλιά Αναστασία Παπαδούλη | 3533

Δημήτριος Τσαλαπάτας | 3246

Νικόλαος Μπέτσος | 3267

Ιωάννης Ρείνος | 3390

31/01/2024

Ομάδα 4

Assignment 1

Coroutine library using <ucontext.h>

```
struct coroutine
{
    ucontext_t *cot;
    ucontext_t *next_cot;
    char stack[16384];
};
typedef struct coroutine co_t;
```

Struct coroutine:
cot: pointer to current context
next_cot: pointer to next context
Stack: holds space to keep data before switching

int mycoroutines_destroy(cot *coroutines)

```
mycoroutines_destroy():
   thread.cot->uc_link = 0
   thread.cot->uc_stack.sp = NULL
   thread.cot->uc_stack.size = 0
   thread.cot->uc_stack.flags = 0
    return SUCCESS
```

int mycoroutines_init(cot *coroutines)

```
mycoroutines_init()
    getcontext(main->cot)
    main.cot->uc_link = 0
    main.cot->uc_stack.sp = main.stack
    main.cot->uc_stack.size = STACK_SIZE
    main.cot->uc_stack.flags = 0
    main.next_cot = NULL
    return SUCCESS
```

int mycoroutines_switchto(cot *coroutines)

```
mycoroutines_switchto()
    swapcontext(thread1->cot, thread1->next_cot)
    if swapcontext fail:
        return ERROR
    else
        return SUCCESS
```

int mycoroutines_create(cot *coroutines, void (*func)(void *), void *arg)

```
mycoroutines_create()
    getcontext(thread->cot)
    thread.cot->uc_link = 0
    thread.cot->uc_stack.sp = thread.stack
    thread.cot->uc_stack.size = STACK_SIZE
    thread.cot->uc_stack.flags = 0
    makecontext(thread->cot, (void (*)(void))routine, 1, arg)
    return SUCCESS
```

Coroutines testing:

Reader: reads the file and writes the data to the pipe until it's full.

Writer: reads data from the pipe until it's empty and copies it to another file.

```
main:
    open pipe
    read input
    create_coroutine(main)
    create_coroutine(reader)
    create_coroutine(writer)
    start_write = 1 //notify coroutine to startwriting in the pipe
    switch to main coroutine
    return
```

```
writer:
    while(pipe is open for reading)
        pipe_read()
        if pipe is empty
            switch to reader
    switch to main
```

```
reader:
    wait notification from main to start writing in the pipe (start_writing)
    while (input exists)
        if pipe is full
            switch to writer
        else
            pipe_write()

    write_done
```

Assignment 2

Our library implementation consists of 3 elements:
Semaphores, threads (using coroutines) and the scheduler.

Semaphores

Threads

Scheduler

Semaphore implementation

```
struct semaphore
{
    unsigned short value;
    int *queue;
    int size;
};
```

Value: semaphore value
Queue: list of blocked threads' ids.
Size: size of queue

sem_t *sem_create(sem_t *sem, int value);

```
sem_t* sem_create(sem_t *sem, int value):
    if sem = NULL:
        allocate memory for sem
    sem->value = value
    sem->queue = NULL
    sem->size = 0
    return sem
```

void sem_destroy(sem_t *sem);

```
int sem_destroy(sem_t *sem):
    if sem = NULL:
        return FAILED

    free(sem)
    return SUCCESS
```

Semaphore implementation

int sem_down(sem_t *sem)

```
int sem_down(sem_t *sem):
    block signals for scheduler

    if sem->value = 0:
        add current thread to sem->queue
        sem->size = sem->size + 1
        make current thread BLOCKED
        find next thread to run and switch to it
        mycoroutines_switchto(scheduler)
        unblock signals for scheduler
        return SUCCESS (does not change value)

    sem->value = sem->value - 1
    unblock signals for scheduler

    return SUCCESS
```

int sem_up(sem_t *sem)

```
int sem_up(sem_t *sem):
    block signals for scheduler

    if sem->value = 1:
        return ERROR lost up
    if sem->size = 0:
        sem->value = 1
        return SUCCESS

    make first thread in queue READY
    shift_left sem->queue
    re-allocate memory for sem->queue
    sem->size = sem->size - 1

    unblock signals for scheduler

    return SUCCESS
```

Thread implementation

```
enum {READY, BLOCKED, FINISH, YIELD} state_T;
struct my_threads
{
    int id;
    state_T state;
    co_t *coroutine;
    sem_t *finish;
};
typedef struct my_threads thr_t;
```

void mythtreads_init()

```
int mythread_init:
    thread_ids = 1;
    if(timer_init() != SUCCESS):
        return FAILED

    return SUCCESS
```

void mythreads_create(thr_t *thread, void (*func)(void *), void *arg)

```
int mythreads_create(thr_t *thread, void (*func)(void *), void *arg):
    sem_down(lib_mtx)

    thread->state = READY;
    thread->finish = NULL;
    thread->finish = sem_create(thread->finish, 0);

    if thread->finish = NULL:
        return FAILED

    give id to thread
    thread_ids = thread_ids + 1

    if mycoroutines_create(thread->coroutine, func, arg) != SUCCESS:
        return FAILED

    sem_up(lib_mtx)

    return SUCCESS
```

void mythreads_join(thr_t *thread)

```
int mythreads_join(thr_t *thread):
    if thread->finish = NULL:
        return FAILED

    sem_down(thread->finish)
    return SUCCESS
```

Thread implementation

void mythtreads_destroy()

```
int mythread_destroy(thr_t *thread):
    sem_down(lib_mtx)
    for i: 0 -> thread_ids:
        if threads_array[i] = thread:
            mycoroutines_destroy(threads_array[i].coroutine);
            sem_destroy(threads_array[i].finish);
            threads_array[i].id = -1;
            break

    sem_up(lib_mtx)
    return SUCCESS
```

void mythreads_yield()

```
int mythread_yield:
    threads_arrray[current_thread].state = YIELD;
    while(threads_array[current_thread].state != READY):
        do nothing

    return SUCCESS
```

void mythreads_exit()

```
int mythread_exit:
    threads_arrray[current_thread].state = FINISH;
    sem_up(threads_array[current_thread].finish);

    while(1):
        do nothing

    return SUCCESS
```

Scheduler Implementation

void scheduler()

```
void scheduler:
    for i: 0 -> thread_ids:
        if threads_array[i].id != -1:
            if threads_array[i].state = FINISH:
                mythread_destroy(threads_array[i])

    for i = current_thread + 1 -> current_thread:
        if threads_array[i].id = -1:
            continue

        if threads_array[i].id = 0:
            i = -1
            continue;

        if threads_array[i].state = YIELD:
            threads_array[i].state = READY;
            continue;

        if threads_array[i].state = READY:
            found_avaialble_thread = 1
            set current_thread.coroutine->next_cot = threads_array[i].coroutine
            temp = current_thread
            current_thread = i
            mycoroutines_switchto(threads_array[temp].coroutine)
            break

    if found_avaialble_thread = 0:
        give processor to the same thread
```

void timer_init()

```
int timer_init:
    set timer to 50000 us (0.05 sec)
    set signal handler to scheduler
    return SUCCESS
```

void custom_sleep()

```
void custom_sleep:
    get start time  // local time //
    init end_time = start_time + seconds

    while(current_time < end_time):
        do nothing
```

Assignment 3

Testing of assignment 2 with writers-readers

```
writer():
    get id
    down(mtx)
    if (reading + writing > 0)
        writers_waiting++
        up(mtx)
        down(write_sem)
    else
        writing++
        up(mtx)
    sleep()
    down(mtx)
    writing--
    if (readers_waiting > 0)
        readers_waiting--
        reading++
        up(read_sem)
    else
        if (writers_waiting > 0)
            writers_waiting--
            writing++
            up(write_sem)
        up(mtx)
    thread_exit()
```

```
reader():
    get id
    down(mtx)
    if (writing + writers_waiting > 0)
        readers_waiting++
        up(mtx)
        down(read_sem)
        if (readers_waiting > 0)
            readers_waiting--
            reading++
            up(read_sem)
        else
            up(mtx)
    else
        up(mtx)
    sleep()
    thread_yield()
    down(mtx)
    reading--
    if ((reading = 0) & writers_waiting > 0)
        writers_waiting--
        writing++
        up(write_sem)
    up(mtx)
    thread_exit()
```

- semaphore mtx: mutual exclusion
- Write_sem: blocks writers
- Read_sem: blocks readers

- Writing: num of writers in CS
- Reading: num of readers in CS
- Writers_waiting: num of blocked writers
- Readers_waiting: num of blocked readers

```
main:
    if wrong arguments:
        return ERROR

    Initialize variables
    allocate memory for threads_array
    sem_create(mtx, 1)
    sem_create(lib_mtx, 1)
    sem_create(read_sem, 0)
    sem_create(write_sem, 0)

    mythreads_create for main
    mythreads_init()
    while(file != EOF)
        if line[0] = 'w':
            custom_sleep(given time)
            mythreads_create for writer
        else if line[0] = 'r':
            custom_sleep(given time)
            mythreads_create for reader

    for i: 0 -> thread_ids:
        if threads_array[i].id != -1:
            mythreads_join(threads_array[i])
```

Thanks for your attention!

"Team 4"