Ταυτόχρονος Προγραμματισμός

ΣΕΙΡΆ ΕΡΓΑΣΙΏΝ 4

Άσκηση 1 – Κορουτίνες

Στην βιβλιοθήκη υπάρχουν οι συναρτήσεις:

```
int mycoroutines_init(co_t *main_co)
int mycoroutines_create(co_t *co, void (*body)(void*), void *arg)
int mycoroutines_switchto(co_t *co)
int mycoroutines_destroy(co_t *co)
```

Δομή για την χρήση κορουτίνας:

```
struct {
    ucontext_t context;
    ucontext_t *co_swap;
} co_t
```

Δομές που βοηθάνε στην εκτέλεση του παραδείγματος

Ενδιάμεση κοινή αποθήκη:

```
struct {
    char data[BUFFER_SIZE];
    int size;
    bool end;
} buffer_t;
```

Απαραίτητα arguments:

```
struct  {
    buffer_t *arg_buffer;
    co_t *arg_context;
}args_t;
```

Συναρτήσεις

```
int mycoroutines_init(co_t *main_co)
...
getcontext(&main_co->context)
return (0);
...
```

```
int mycoroutines_switchto(co_t *co)
...
swapcontext(&co->context,  co->co_swap)

return 0;
...
```

```
int mycoroutines_create(co_t *co, void (*body)(void*), void *arg)
...
getcontext(&co->context)
co->context.uc_link = &arguments->arg_context->context;
co->context.uc_stack.ss_sp = malloc(STACK_SIZE);
co->context.uc_stack.ss_size = STACK_SIZE;
arguments->arg_context = co;
makecontext(&co->context, (void (*)(void))body, 1, arguments);

return (0);
...
```

```
int mycoroutines_destroy(co_t *co)
...
free(co->context.uc_stack.ss_sp);

return 0;
...
```

Παράδειγμα

```
void producer(void *args) {
    buffer_t *buffer = arguments->arg_buffer;
    co_t *context = arguments->arg_context;
    Open File
    Ψα filesize
    while (bytesRead != filesize) {
        while (buffer->size == BUFFER_SIZE) {
            mycoroutines_switchto(context);
        }
        input_size = fread(buffer->data, 1 , BUFFER_SIZE,
input_file);
        bytesRead += input_size;
        buffer->size += input_size;
        if (input_size != BUFFER_SIZE|| input_size == 0) {
            if (bytesRead == filesize) break;
        }
    }
    buffer->end = true;
    mycoroutines_switchto(context);
    close file
    return;
}
```

```
void consumer(void *args) {
    buffer_t *buffer = arguments->arg_buffer;
    co_t *context = arguments->arg_context;
    Open file
    while (buffer->size > 0 || buffer->end != true) {
        while (buffer->size == 0 && buffer->end != true) {
            mycoroutines_switchto(context);
        }
        if(buffer->size < BUFFER_SIZE) {
            for ( i = 0; i < buffer->size; i++) {
                fwrite(&buffer->data[i], 1, 1, output_file);
            }
        }
        else {
            fwrite(buffer->data, 1, BUFFER_SIZE, output_file);
        }
        buffer->size = 0;
    }
    Close file
    return ;
}
```

Άσκηση 2 – Αυτοσχέδια νήματα

Στην βιβλιοθήκη υπάρχουν οι συναρτήσεις:

```
int mythreads_init();
int mythreads_create(struct thr_t *thr, void(body)(void*), void *arg);
int mythreads_yield();
int mythreads_join(struct thr_t *thr);
int mythreads_destroy(struct thr_t *thr);
int mythreads_sem_create(sem_t *s, int val);
int mythreads_sem_down(sem_t *s);
int mythreads_sem_up(sem_t *s);
int mythreads_sem_destroy(sem_t *s);
```

Δομή σηματοφόρου:

```
typedef struct {
    int value;
}sem_t
```

Δομή κορουτίνας:

```
typedef struct {
    ucontext_t context;
}co_t;
```

Δομή νήματος:

```
struct thr_t{
    co_t *co;
    int thread_id;
    struct thr_t* next_thread;
    struct thr_t* previous_thread;
    int critical_segment;
}
```

Global μεταβλητές:

```
struct thr_t *current_thread, *main_thread;
```

Συναρτήσεις κορουτίνων που τροποποιήθηκαν

```
int mycoroutines_create(co_t *co, void (*body)(void*), void *arg)
...
getcontext(&co->context);
co->context.uc_link = &my_return.context;
co->context.uc_stack.ss_sp = malloc(STACK_SIZE);
co->context.uc_stack.ss_size = STACK_SIZE;
makecontext(&co->context, (void (*)(void))body, 1, arg);
return (0);
...
```

```
int mycoroutines_switchto(co_t *co_sig, co_t *co_switch){
...
current_thread = current_thread->next_thread;
swapcontext(&co_sig->context, &co_switch->context);
return (0);
...
```

Συναρτήσεις νημάτων

```
int mythreads_init()
...
init main thread and its context
threads_created = 1;
mycoroutines_init(main_thread->co)

main_thread->next_thread = main_thread;
main_thread->previous_thread = main_thread;
main_thread->thread_id = 0;
current_thread = main_thread;
main_thread->critical_segment = 0;

mycoroutines_create(&my_return, mythreads_return,
NULL);
initialize timer for changing threads
return 0;
...
```

```
int mythreads_create(struct thr_t *thread, void (*body)(void
*), void *arg)
...
thread->critical_segment = 1;
thread->co = malloc(sizeof(co_t));
thread->next_thread = main_thread;
thread->previous_thread = main_thread->previous_thread;
main_thread->previous_thread->next_thread = thread;
main_thread->previous_thread = thread;
thread->thread_id = threads_created++;
mycoroutines_create(thread->co, body, arg);

thread->critical_segment = 0;
return 0;
...
```

```
int mythreads_yield()
...
current_thread->critical_segment = 1;
mycoroutines_switchto(current_thread->co, current_thread->next_thread->co);
current_thread->critical_segment = 0;
return 0;
...
```

Συναρτήσεις νημάτων

```
int mythreads_join(struct thr_t *thread)
...
while (thread->thread_id > 0 ) {
    mythreads_yield()
}
return 0;
...
```

```
int mythreads_destroy(struct thr_t *thread)
...
if (thread->thread_id == 0 ) {
    turn off timer
    free(main_thread->co);
    free(main_thread);
    current_thread=NULL;
}
else {
    mycoroutines_destroy(thread->co);
    free(thread);
}
return 0;
...
```

Συναρτήσεις σηματοφόρων

```
int mythreads_sem_create(sem_t *s, int val)
...
if (val != 1 && val != 0) {
    return (-1);
}
s->value = val;
return 0;
...
```

```
int mythreads_sem_up(sem_t *s) {
...
current_thread->critical_segment = 1;
if (s->value == 1) {
    return 0;
}else {
    s->value = 1;
}
current_thread->critical_segment = 0;
return 0;
...
```

```
int mythreads_sem_down(sem_t *s) {
...
current_thread->critical_segment = 1;
while (s->value <= 0) {
    current_thread->critical_segment = 0;
    mythreads_yield()
    current_thread->critical_segment = 1;
}
s->value--;
current_thread->critical_segment = 0;
return 0;
...
```

```
int mythreads_sem_destroy(sem_t *s) {
// Nothing specific to destroy for this simple semaphore
return 0;
```

Βοηθητική συνάρτηση

```
void mythreads_return()
...
current_thread->critical_segment = 1;

current_thread->next_thread->previous_thread = current_thread->previous_thread;
current_thread->previous_thread->next_thread = current_thread->next_thread;
current_thread->thread_id = -1;

current_thread->critical_segment = 0;
mythreads_yield();
...
```

Άσκηση 3 – reader/writer πρόβλημα

```
void reader(void *arg) {
    while(1) {
        mythreads_sem_down(&mtx);
        n_reader++;
        if(n_reader == 1) {
            mythreads_sem_down(&wrt);
        }
        mythreads_sem_up(&mtx);
        if(count == limit){
            break;
        }
        mythreads_sem_down(&mtx);
        n_reader--;
        if(n_reader == 0) {
            mythreads_sem_up(&wrt);
        }
        mythreads_sem_up(&mtx);
    }
}
```

```
void writer(void *arg) {
    while(1) {
        mythreads_sem_down(&wrt);
        count++;
        if(count == limit) {
            break;
        }
        mythreads_sem_up(&wrt);
    }
    mythreads_sem_up(&wrt);
}
```

mtx : 0
wtr : 0

Άσκηση 3 – reader/writer πρόβλημα

```
void reader(void *arg) {
    while(1) {
        mythreads_sem_down(&mtx);
        n_reader++;
        if(n_reader == 1) {
            mythreads_sem_down(&wrt);
        }
        mythreads_sem_up(&mtx);
        if(count == limit){
            break;
        }
        mythreads_sem_down(&mtx);
        n_reader--;
        if(n_reader == 0) {
            mythreads_sem_up(&wrt);
        }
        mythreads_sem_up(&mtx);
    }
}
```

```
void writer(void *arg) {
    while(1) {
        mythreads_sem_down(&wrt);
        count++;
        if(count == limit) {
            break;
        }
        mythreads_sem_up(&wrt);
    }
    mythreads_sem_up(&wrt);
}
```

mtx : 0
wtr : block

Άσκηση 3 – reader/writer πρόβλημα

```
void reader(void *arg) {
    while(1) {
        mythreads_sem_down(&mtx);
        n_reader++;
        if(n_reader == 1) {
            mythreads_sem_down(&wrt);
        }
        mythreads_sem_up(&mtx);
        if(count == limit){
            break;
        }
        mythreads_sem_down(&mtx);
        n_reader--;
        if(n_reader == 0) {
            mythreads_sem_up(&wrt);
        }
        mythreads_sem_up(&mtx);
    }
}
```

```
void writer(void *arg) {
    while(1) {
        mythreads_sem_down(&wrt);
        count++;
        if(count == limit) {
            break;
        }
        mythreads_sem_up(&wrt);
    }
    mythreads_sem_up(&wrt);
}
```

mtx : 0
wtr : 0

Άσκηση 3 – reader/writer πρόβλημα

```
void reader(void *arg) {
    while(1) {
        mythreads_sem_down(&mtx);
        n_reader++;
        if(n_reader == 1) {
            mythreads_sem_down(&wrt);
        }
        mythreads_sem_up(&mtx);
        if(count == limit){
            break;
        }
        mythreads_sem_down(&mtx);
        n_reader--;
        if(n_reader == 0) {
            mythreads_sem_up(&wrt);
        }
        mythreads_sem_up(&mtx);
    }
}
```

```
void writer(void *arg) {
    while(1) {
        mythreads_sem_down(&wrt);
        count++;
        if(count == limit) {
            break;
        }
        mythreads_sem_up(&wrt);
    }
    mythreads_sem_up(&wrt);
}
```

mtx : 1
wtr : block

Άσκηση 3 – reader/writer πρόβλημα

```
void reader(void *arg) {
    while(1) {
        mythreads_sem_down(&mtx);
        n_reader++;
        if(n_reader == 1) {
            mythreads_sem_down(&wrt);
        }
        mythreads_sem_up(&mtx);
        if(count == limit){
            break;
        }
        mythreads_sem_down(&mtx);
        n_reader--;
        if(n_reader == 0) {
            mythreads_sem_up(&wrt);
        }
        mythreads_sem_up(&mtx);
    }
}
```

```
void writer(void *arg) {
    while(1) {
        mythreads_sem_down(&wrt);
        count++;
        if(count == limit) {
            break;
        }
        mythreads_sem_up(&wrt);
    }
    mythreads_sem_up(&wrt);
}
```

mtx : 0
wtr : block

Άσκηση 3 – reader/writer πρόβλημα

```
void reader(void *arg) {
    while(1) {
        mythreads_sem_down(&mtx);
        n_reader++;
        if(n_reader == 1) {
            mythreads_sem_down(&wrt);
        }
        mythreads_sem_up(&mtx);
        if(count == limit){
            break;
        }
        mythreads_sem_down(&mtx);
        n_reader--;
        if(n_reader == 0) {
            mythreads_sem_up(&wrt);
        }
        mythreads_sem_up(&mtx);
    }
}
```

```
void writer(void *arg) {
    while(1) {
        mythreads_sem_down(&wrt);
        count++;
        if(count == limit) {
            break;
        }
        mythreads_sem_up(&wrt);
    }
    mythreads_sem_up(&wrt);
}
```

mtx : 0
wtr : 0

Άσκηση 3 – reader/writer πρόβλημα

```
void reader(void *arg) {
    while(1) {
        mythreads_sem_down(&mtx);
        n_reader++;
        if(n_reader == 1) {
            mythreads_sem_down(&wrt);
        }
        mythreads_sem_up(&mtx);
        if(count == limit){
            break;
        }
        mythreads_sem_down(&mtx);
        n_reader--;
        if(n_reader == 0) {
            mythreads_sem_up(&wrt);
        }
        mythreads_sem_up(&mtx);
    }
}
```

```
void writer(void *arg) {
    while(1) {
        mythreads_sem_down(&wrt);
        count++;
        if(count == limit) {
            break;
        }
        mythreads_sem_up(&wrt);
    }
    mythreads_sem_up(&wrt);
}
```

mtx : 1
wtr : 0

Άσκηση 3 – reader/writer πρόβλημα

```
void reader(void *arg) {
    while(1) {
        mythreads_sem_down(&mtx);
        n_reader++;
        if(n_reader == 1) {
            mythreads_sem_down(&wrt);
        }
        mythreads_sem_up(&mtx);
        if(count == limit){
            break;
        }
        mythreads_sem_down(&mtx);
        n_reader--;
        if(n_reader == 0) {
            mythreads_sem_up(&wrt);
        }
        mythreads_sem_up(&mtx);
    }
}
```

```
void writer(void *arg) {
    while(1) {
        mythreads_sem_down(&wrt);
        count++;
        if(count == limit) {
            break;
        }
        mythreads_sem_up(&wrt);
    }
    mythreads_sem_up(&wrt);
}
```

mtx : 0
wtr : block