ΕΡΓΑΣΙΑ 4

Στολτίδης Αλέξανδρος 2824
Κουτσούκης Νικόλαος 2907

COROUTINE LIBRARY

```
struck{
    ucontext_t
    char Stack[]
}co_t

co_t   *ntx, *curr
```

```
mycoroutines_init(){

    getcontext(cmain)
    nxt = cmain
}
```

```
mycoroutines_destroy{
    uc_stack.ss_sp = NULL
}
```

```
mycoroutines_create() {
    getcontext(co_t)

    uc_stack.ss_sp= stack
    uc_stack.ss_size = sizeof(stack)
    co.uc_link = nxt

    makecontext()
}
```

```
mycoroutines_switchto(){
    curr = ntx
    nxt = coroutine

    swapcontext(curr, nxt)
}
```

PIPE FIFO

```
writer_function(){
    while(fscanf(letter) != EOF){
        for()
            to push element +1 in fifo

        FIFO[0] = letter

        position++

        if(position == fifo_size)
            mycoroutines_switchto(reader)
    }
    complete = true
    mycoroutines_switchto(reader)
}
```

```
reader_function()
    while(1){
        for(i < size_of_fifo)
            fputs()

        position = 0

        if(complete)
            break

        mycoroutines_switchto(writer)
    }
```

```
main()
    mycoroutines_init(cmain)

    mycoroutines_create(writer)
    mycoroutines_create(reader)

    mycoroutines_switchto(writer)
    mycoroutines_switchto(writer)

    mycoroutines_destroy(writer)
    mycoroutines_destroy(reader)
```

PRIME

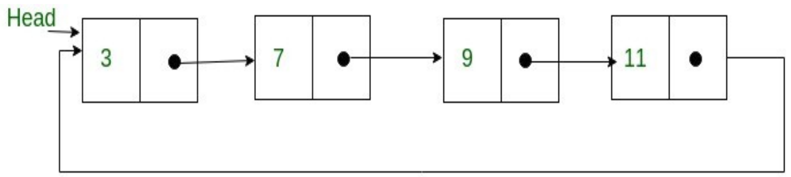
```
Main() {
//while(job exist)
      While() {
            down(cmain)
            CS
            up(cthread)
      }
//Wait the Threads to Finish
      down(cmain)
      up(cmain)
      for(i< total_Threads){
            down(cmain)
            CS
            up(cthread)
      }
      down(cmain)
}
```

```
Fuction_of_Thread(){
      while(1){
            down(cthread)
            if(file finish)
                  break
            up(cmain)
            PRIME
      }
      up(cmain)
}
```

Semaphores
Threads: init(cthread, 0)
Main: init(cmain, 1)

Scheduler

```
mythreads_init(){
    //Create Scheduler Linked List (FIFO)
    …

    //Append main() Context on FIFO
    enqueue(main)

    //Get Context for Thread Termination
    terminate_context()

    //Handle Signals With scheduler()
    sigaction(…)

    //Run Timer
    create_timer();

}
```



```
create_timer(){
    //Set Time Values
    tv_sec = …
    tv_usec = …
    …

    //Set Timer
    setitimer()
}
```

Helper Functions

block_sigalarm(): Blocks Signal Handling with sigprocmask()

unblock_sigalarm(): Unblock Signal Handling sigprocmask()

```
context_make(){
        //Create New Context
        context = malloc(...)

        getcontext()
        //Initialize Stack
        makecontext(...)
}
```

```
scheduler_enqueue(){
        if(empty)
                head = tail = task

        //Append on End
        task→next = head
        tail->next = task
        tail = task

        //Initialize Data
        task→thread = ...
}
```

Helper Functions

signal_scheduler(): Signal Scheduler with kill()

Terminate_context: Runs Every Time a Thread Body has Returned

```
terminate_context(){
        //Set State of Thread to Terminated
        thread→state = terminated
        //Set Context to The Next Available Context
        setcontext(...)
}
```

Scheduler

```
scheduler(){
   block_sigalarm()

     //Find Next Ready Job
     while(!ready) ...

     //Run Ready Context
     swapcontext(current, available)

   unlock_sigalarm()
}
```

API Functions

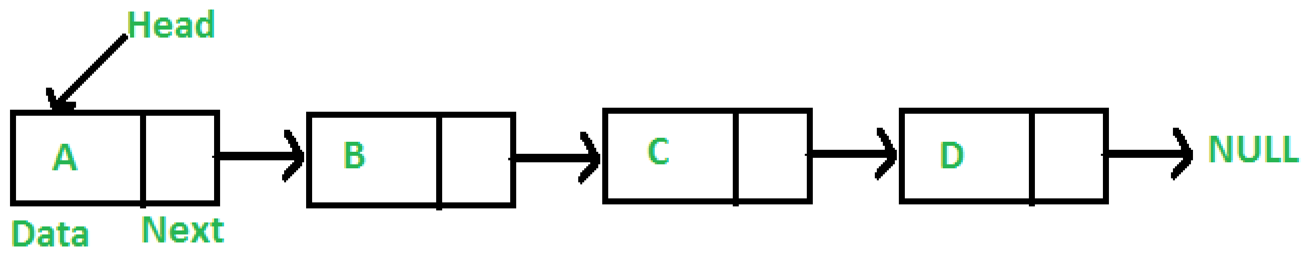
```
mythreads_create(){

    //Create Context
    context_make(...)


      //Append Context on Scheduler
      enqueue(context)

    unlock_sigalarm()
}
```

```
scheduler(){
    block_sigalarm()
      //Allow Scheduler to Switch Ready Job
      signal_scheduler()

    unlock_sigalarm()
}
```

API Functions

```
mythreads_join(){
    while(thread→state not terminated)

}
```

API Semaphores



```
mythreads_sem_init(){
      block_sigalarm()
      //Set Value and Pointers Of Semaphore
      ...
      unblock_sigalarm()
}
```

```
mythreads_sem_down(){
    block_sigalarm()
    semaphore_value – 1
    if(semaphore_value < 0)
        //Set Current Running Thread to Blocked
        current_thread→state = blocked
        //Add Task to the End of the FIFO Queue
        if(empty) head = task
        tail→next = task
        task→next = NULL
        //Set semaphore Data to -1

        //Signal Scheduler to Decide Next Job
        signal_scheduler()

    unblock_sigalarm()
}
```

API Semaphores

```
mythreads_sem_up(){
    block_sigalarm()
    if(blocked)
        thread→state = ready
        //Remove thread From FIFO

    //Increment Semaphore Value
    unblock_sigalarm()
}
```

DESTROY

```
mythreads_sem_destroy(){
    if(head || tail == 0)
        return 0
    For(list)
         free(curr)

    head = tail = NULL
}
```

```
mythreads_destroy() {
    lock_sigalarm()
    for(list)
        free(curr)

unblock_sigalarm()
}
```