Άσκηση 1(buffer)

Consumer

```
while(1){
    if(pipe is full){
        for(#buffer_size){
            pipe_read(1 byte);
            printf(1 byte);
        }
        switchto(producer);
    }
    else if(EOF){
        for(#bytes left to read){
            pipe_read(1 byte);
            printf(1 byte);
        }
    }
    else { //term; }
}
```

Producer

```
cons.from = prod;
while(number > 1){
    if(buffer is full){
        switchto(consumer);
    }
    scanf(1 byte);
    if(EOF){
        if(no bytes read) { flag = -2; }
        else { flag = #bytes left to read;}
        switchto(consumer);
    }
pipe_write(1 byte);
pos++
}
```

Main

```
init(main_cor);
prod.link = cons;
create(prod);

cons.link = main;
create(cons);

switchto(prod);
```

```
mycoroutines_init{
    getcontext for main
}

mycoroutines_create{
    getcontext for prod/cons
    allocate stack size
    makecontext();
}
```

```
mycoroutines_switchto{
    swap coroutine
}

mycoroutines_destroy{
    free allocated space
}
```

Άσκηση 2(primes)

scheduler

```
sigaction(ignore sigalarm);
//check if a node does not exist
anymore, wake up thread that
is blocked because of him(join);
if(anyone terminated){
        flag = 1;
}
if(flag == 1){ //finish
        delete(curr);
        make next RUNNING;
}
else if(anyone blocked){//sem down
                                        or join
        delete from running list
        add to blocking list
        make next RUNNING;
}
else{
        find_running();
        make next RUNNING;
}
switchto(RUNNING);
```

master_thread

```
//allocate space for workers
sem_init(job, 0);
sem_init(available, nthreads);

mythreads_init();

for(#nthreads){
        mythreads_create();
}

while(1){
        scanf(); //user gives number
        if(exit){
                exited = 1; break;
        }
        else{
                mysem_down(available);
                //assign job
                mysem_up(job);
                mythreads_yield();
        }
}
//wake up workers
mythreads_join(); //wait for them to
                                    terminate
//free allocated space
```

```
thr_t{
        int thread_id, status, finish
             wait_id;
        co_t context;
        sem_t sem;
        thr_t* next, prev, block;
}
```

worker_thread

```
while(1){
        mysem_down(job);
        if(term){
                //check for any number
                left to test
                //primetest;
                //break;
        }
        //take job assigned by main
        //primetest;
        mysem_up(available);
}

//finished
```