ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΑΥΤΟΧΡΟΝΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

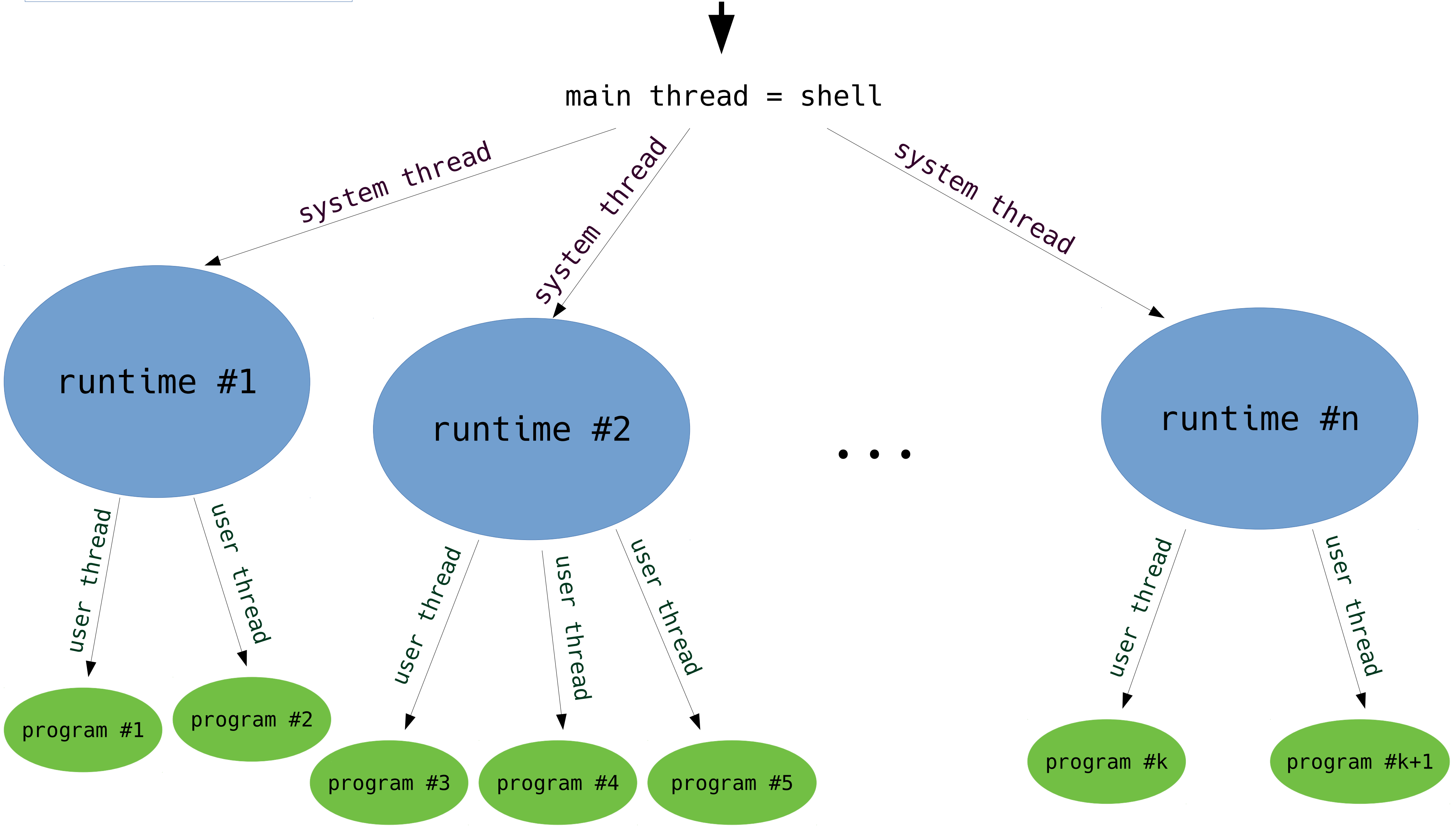
ΑΚΑΔΗΜΑΪΚΟ ΕΤΟΣ 2017 – 2018

HOMEWORK IV

ΕΠΩΝΥΜΟ	ΟΝΟΜΑ	ΑΡ. ΜΗΤΡΩΟΥ
ΚΟΣΚΕΡΙΔΗΣ	ΓΙΩΡΓΟΣ	1660

n = number of CPU cores

Simbly



main thread = shell

system thread

system thread

system thread

runtime #1

runtime #2

. . .

runtime #n

user thread

user thread

user thread

user thread

user thread

user thread

user thread

program #1

program #2

program #3

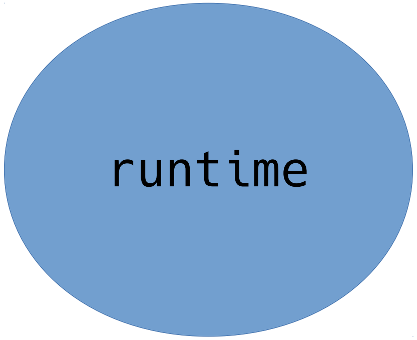
program #4

program #5

program #k

program #k+1

objects:



=

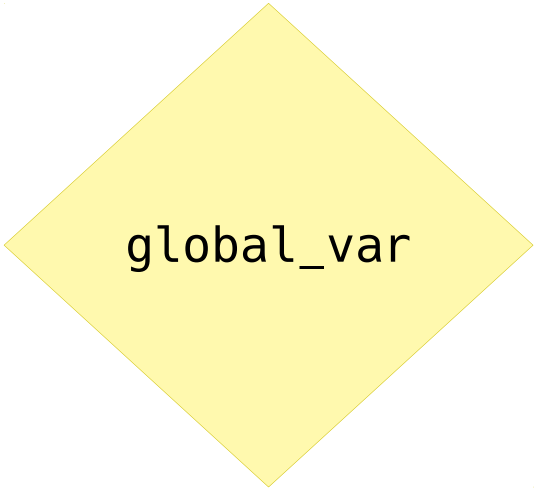
```
list programs;

mutex mtx;

condition list_not_empty;

int program_cnt;

(-)

int thread_id;
```

program

=

```
global_var blocked_sem;

int blocked_idx;

program_state state;

(-)

int argv[], id;

ringbuffer line;

hashtable vars;

long sec_sleep, nsec_sleep;
```

```
hashtable global_vars;

mutex global_table_lock;
```



global_var

=

```
int count[], len;

mutex mtx;

condition cond;
```

functions:

```
up(global_var *var, int idx):

lock(global_table_lock);

if (hashtable_find(global_vars, var)) {
 unlock(global_table_lock);

 lock(var->mtx);
 if (idx < var->len) {
   var->count[idx]++;
   signal(var->cond);
 } else {
   //realloc var->count
   var->count[idx] = 1; //ή 2, αν αρχικοποιούνται σε 1
   var->len = idx + 1;
 }
 unlock(var->mtx);

} else {
  var->count[idx] = 1; //ή 2, αν αρχικοποιούνται σε 1
  hashtable_insert(var);
  unlock(global_table_lock);
}
```

```
hashtable global_vars;

mutex global_table_lock;
```

```
down(program *prog, global_var *var, int idx):

lock(global_table_lock);

if (hashtable_find(global_vars, var)) {
 unlock(global_table_lock);

 lock(var->mtx);
 if (idx >= var->len) {

   //realloc var->count

   var->len = idx + 1;
 }
 unlock(var->mtx);

} else {
  hashtable_insert(var);
  unlock(global_table_lock);
}

prog->blocked_idx = idx;
prog->blocked_sem = var;
prog->state = BLOCKED;
```

```
blocked_handler(program *prog, long sleep_nsec):

global_var *var = prog->blocked_sem;

lock(var->mtx);

if (var->count[prog->blocked_idx] <= 0) {

  cond_timedwait(var->cond, var->mtx, nsec);

  if (var->count[prog->blocked_idx] > 0) {

    var->count[prog->blocked_idx]--;
    prog->state = INSTRUCTION_LINE;

  }

} else {

  var->count[prog->blocked_idx]--;
  prog->state = INSTRUCTION_LINE;

}

unlock(var->mtx);
```

part 1

```
runtime_thread(runtime *rt):

list_node *curr;
while (1) {
  lock(rt->mtx);
  while (!rt->programs) {
    wait(rt->list_not_empty, rt->mtx);
  }
  curr = rt->programs->root;
  unlock(rt->mtx);

  while (curr) {
    program *prog = curr->data;
    long time_slice = generateTimeSlice();

    switch (prog->state) {
       case MAGIC_LINE:
       case INSTRUCTION_LINE:
         //execute instructions until time_slice <= 0
         break;
       case SLEEPING:
         //sleep for time_slice or if time_slice > than the time left
         //sleep for the time left and move state to INSTRUCTION_LINE
         break;
       case BLOCKED:
         blocked_handler(prog, time_slice);
         break;
    }
    (...)
```

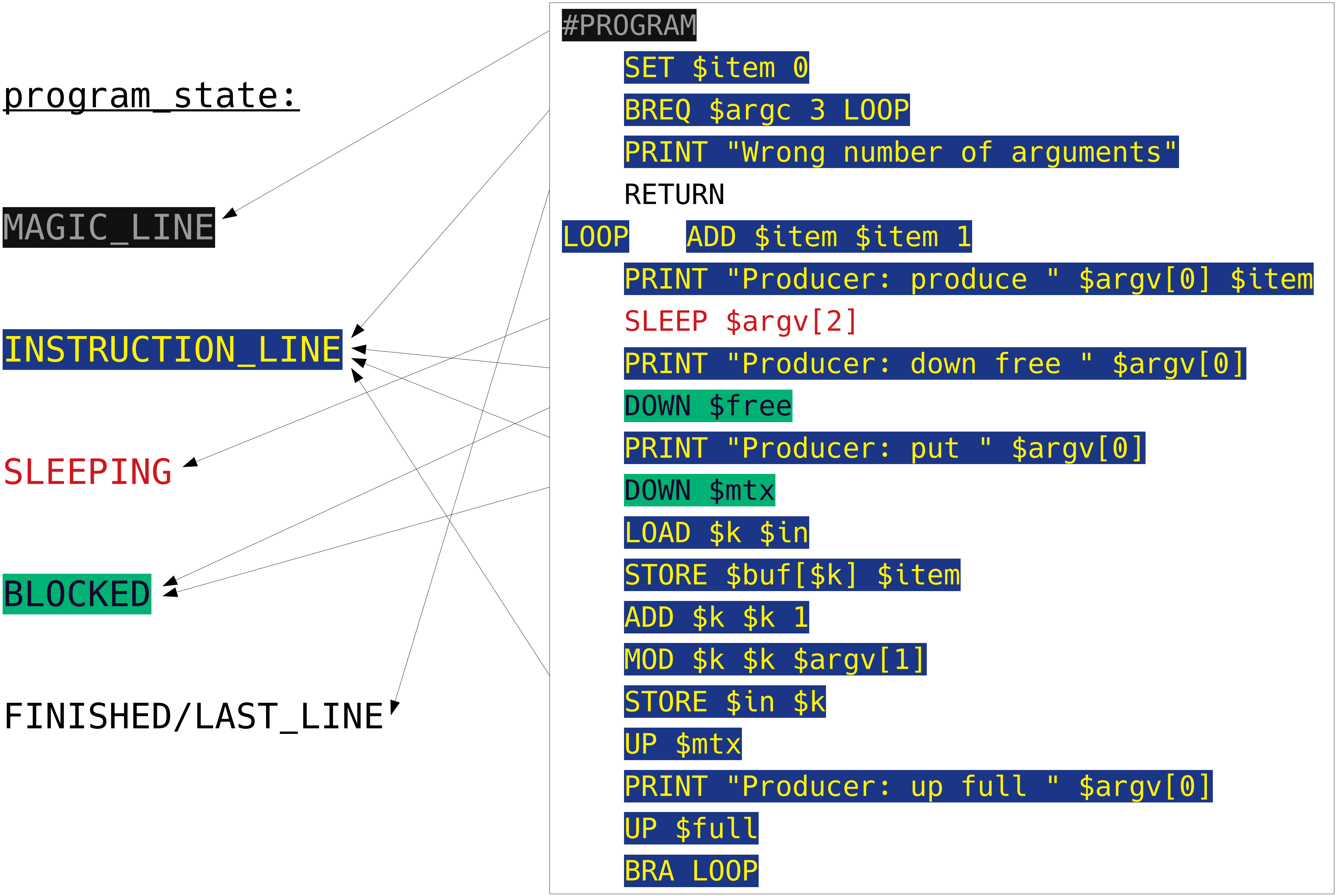
part 2

```
runtime_thread(runtime *rt):

    (...)
    if (prog->state == FINISHED) {
       lock(rt->mtx);
       curr = curr->next;
       rt->program_cnt--;

       delete_node(rt->programs, curr->prev);
       unlock(rt->mtx);
    } else {
       curr = curr->next;
    }
  }
}
```

```
runtime_attach_program(runtime *rt, program *prog):

lock(rt->mtx);

rt->program_cnt++;
append_node(rt->programs, prog);

if (rt->program_cnt == 1) {
  signal(rt->list_not_empty);
}
unlock(rt->mtx);
```

αλγόριθμος ομοιόμορφης κατανομής προγραμμάτων:

```
main:

int rt_cnt, rt_min_idx;
runtime *rt_arr[rt_cnt];

(...)

int i, min_prog_cnt;

lock(rt_arr[0]->lock);
min_prog_cnt = rt_arr[0]->program_cnt;
unlock(rt_arr[0]->lock);

rt_min_idx = 0;

for (i = 0; i < rt_cnt; i++) {
  lock(rt_arr[i]->lock);

  if (rt_arr[i]->program_cnt < min_prog_cnt) {
    min_prog_cnt = rt_arr[i]->program_cnt;
    rt_min_idx = i;
  }
  unlock(rt_arr[i]->lock);
}

runtime_attach_program(rt_arr[rt_min_idx], prog);
```



program_state:

MAGIC_LINE

INSTRUCTION_LINE

SLEEPING

BLOCKED

FINISHED/LAST_LINE

```
#PROGRAM
    SET $item 0
    BREQ $argc 3 LOOP
    PRINT "Wrong number of arguments"
    RETURN
LOOP   ADD $item $item 1
    PRINT "Producer: produce " $argv[0] $item
    SLEEP $argv[2]
    PRINT "Producer: down free " $argv[0]
    DOWN $free
    PRINT "Producer: put " $argv[0]
    DOWN $mtx
    LOAD $k $in
    STORE $buf[$k] $item
    ADD $k $k 1
    MOD $k $k $argv[1]
    STORE $in $k
    UP $mtx
    PRINT "Producer: up full " $argv[0]
    UP $full
    BRA LOOP
```