

Εργασία 3η

Ομάδα 10

Αξελού Ολυμπία 2161

Τσιτσοπούλου Ειρήνη 2203

3.1 Παράλληλος υπολογισμός fractals

```
volatile int *res, maxIterations, nofslices;    volatile mandel_Pars * slices;
```

```
volatile int main_draw_w = 0, main_assign_w = 0, threadycast = 0;  
cond c_args, c_assign, c_m_assign, c_draw, c_workers_block;  
mutex mtx;
```

MAIN THREAD

```
...  
//allocate res, slices, draw_array, init mutex  
for(#nofslices){  
    lock(mtx);  
    //create thread;  
    wait(c_args); //wait for it to take its args  
    unlock(mtx);  
}  
...  
while(1){  
    //create #nofslices jobs and assign to workers  
    lock(mtx);  
    if(threadycast < nofslices){ main_assign_w = 1; wait(c_m_assign); }  
    threadycast = 0; //init for next computation  
    unlock(mtx);  
    for(#nofslices) signal(c_assign); //notify workers  
  
    while(1){  
        lock(mtx);  
        if(nofjustfin == 0){ main_draw_w = 1; wait(c_draw); }  
        for(#nofslices){  
            if(draw_array[i] == JUST_FIN){ //found a just-finished worker  
                draw_array[i] = !JUST_FIN; w_done++; break; } }  
        unlock(mtx);  
        // printing slice  
        if(all workers done) for(#nofslices) signal(c_workers_block); break; //-> next job  
    } //take coordinates (next job)  
} ...
```

WORKER THREAD

```
//retrieve arguments  
lock(mtx);  
signal(c_args); //notify main that arguments have been retrieved  
unlock(mtx);  
  
while(1){  
    lock(mtx); //wait for main to assign job  
    threadycast++;  
    if(threadycast == nofslices){ // it's the last worker to arrive  
        if(main_assign_w){main_assign_w = 0; signal(cond_m_assign);} //wake up main  
        wait(c_assign);  
        unlock(mtx);  
  
        perform Mandelbrot computation  
  
        lock(mtx);  
        draw_array[my_no] = JUST_FIN; //notify main to draw my slice  
        nofjustfin++;  
        if(main_draw_w){main_draw_w = 0; signal(c_draw); } //wake up main if sleeping  
        wait(c_workers_block);  
        unlock(mtx);  
    }  
}
```

3.2 Γέφυρα

bridge_enter

```
lock(mtx);
if(carsPassing[my_color] >= 0){ //there is a car from the other color waiting
    carsPassing[my_color]++;
}

//full bridge or other color (waiting too long || on bridge)
if(onbridge[!my_color] || (onbridge[my_color] >= bridgeCapacity) ||
carsPassing[my_color] > MAX_PASSING){

    waiting[my_color]++;

    if(carsPassing[!my_color] == -1){ carsPassing[!my_color] = 0; }
    wait(queue[my_color], mtx);
    if(carsPassing[!my_color] >= 0){ carsPassing[!my_color] = -1;}

    //awakens the one behind him if there is one and if the bridge is not full
    if((waiting[my_color] > 0) && (onbridge[my_color] < bridgeCapacity)){
        waiting[my_color]--; onbridge[my_color]++;

        signal(queue[my_color]);
    }
}

else{
    onbridge[my_color]++;
}

unlock(mtx);
```

```
mutex mtx;
cond wait_to_fill, train_start, wait_to_empty, pas_entering, pas_exiting;
volatile int waiting = onboard = 0;
volatile int trainCapacity;
volatile int train_w_fill = train_w_empty = 0;
```

passenger

```
train_enter();
// on ride
train_exit ( );
```

bridge_exit

```
mtx_lock(&mtx, __LINE__);
onbridge[my_color]--;

//there is none left on bridge while there are of the other color waiting
if(onbridge[my_color] == 0 && waiting[!my_color] > 0 ){
    waiting[!my_color]--;
    onbridge[!my_color]++;
    cond_signal(&queue[!my_color], __LINE__);
}
else if(onbridge[my_color] >= bridgeCapacity - 1 && waiting[my_color] > 0 ){
    //the bridge is not full anymore, allows of same color to pass
    waiting[my_color]--;
    onbridge[my_color]++;
    cond_signal(&queue[my_color], __LINE__);
}
mtx_unlock(&mtx, __LINE__);
```

3.3 Τρενάκι

train_enter()

```
lock(mtx);
waiting++;
// enough passengers to notify the train to start taking passengers
if(waiting == trainCapacity && train_w_fill){
    train_w_fill = 0;
    signal(wait_to_fill);
}
wait(pas_entering, mtx); //wait train to allow them to get in
if(onboard < trainCapacity){ //unblock 1 passenger behind him
    waiting--;
    onboard++;
    signal(pas_entering);
}
else if(onboard == trainCapacity && train_w_start) // train full: start
    train_w_start = 0;
    signal(train_start); //notify the train to start
unlock(mtx);
```

train_exit()

```
lock(mtx);
wait(pas_exiting, mtx);
if(onboard > 0){
    onboard--;
    signal(pas_exiting);
}
else if(onboard == 0 && train_w_empty){
    train_w_empty = 0;
    signal(wait_to_empty);
}
unlock(mtx);
```

mutex mtx;

cond wait_to_fill, train_start, wait_to_empty, pas_entering, pas_exiting;

volatile int waiting = onboard = 0;

volatile int trainCapacity;

volatile int train_w_fill = train_w_empty = 0;

```
train_enter( );
// on ride
train_exit ( );
```

passenger

```
while(1){
    lock(mtx);
    if(waiting < trainCapacity){
        train_w_fill = 1;
        wait(wait_to_fill, mtx); //waits until enough passengers have arrived
    }
    onboard++; waiting--;
    signal(pas_entering); // notify passengers to enter
    if (onboard < trainCapacity){
        train_w_start = 1;
        wait(train_start, mtx); // waits until last passenger has got in
    }

    sleep(RIDE_DURATION);

    onboard--;
    signal(pas_exiting); // notifies one passenger to exit
    if (onboard > 0){
        train_w_empty = 1; //train about to wait
        wait(wait_to_empty, mtx); //Waits for the last passenger to exit
    }
    unlock(mtx);
}
```

train

3.4 Condition Critical Region (CCR)

CCR_DECLARE(label)

```
//declare mutexes: mtx_label, mtx_q_label  
  
//declare conditions: queue_label  
  
//declare ints: no_q_label, loop_label
```

CCR_INIT(label)

```
init(mtx_label); init(mtx_q_label);  
  
init(queue_label);  
  
no_q_label = 0; loop_label = -1;
```

CCR_EXEC(label, cond, body)

```
lock(mtx_label); lock(mtx_q_label);  
while(!cond){  
    no_q_label++;  
    if(loop_label >= 0){ //sb woke him up. Not the first time in while  
        loop_label++;  
        if(loop_label == no_q_label){  
            loop_label = -1; // completed a circle. Give mtx to newcomers  
            unlock(mtx_label);  
        }else{ no_q_label--; signal(queue_label); } //wake up next  
    }else{ unlock(mtx_label); } //in queue  
    wait(queue_label);  
}  
  
body  
  
if(no_q_label > 0){  
    loop_label = 0; //init loop counter  
    no_q_label--; signal(queue_label); //wake up 1st  
}else{ // no one waiting in queue  
    loop_label = -1;  
    unlock(mtx_label);  
}  
unlock(mtx_q_label);
```

3.4.1 Παράλληλος υπολογισμός fractals w/ CCR

train_enter()

```
...
//allocate res, slices, draw_array, init mutex
for(#nofslices){
    //create thread;
    region X when (args_taken){
        args_taken = 0;}
    }
...
while(1){
    //create #nofslices jobs and assign to workers
    region X when (1){
        jobs_assigned = nofslices;
        unprinted_slices = nofslices;}

    while(1){
        region X when (nofjustfin > 0){
            //for: breaks when it finds a worker which has just finished
            for(i = 0; i < nofslices; i++){
                if(draw_array[i] == JUST_FINISHED){
                    draw_array[i] = !JUST_FINISHED;
                    nofjustfin--; workersDone++;
                    break;
                }
            }
        }
    }
    // printing slice
    if(all workers done){ for(#nofslices) signal(c_workers_block);
        break; //-> next job}
    }//take coordinates (next job)
}...
```

```
volatile int *res, maxIterations, nofslices;    volatile mandel_Pars * slices;
volatile int main_draw_w = 0, main_assign_w = 0, threadycast = 0;
cond c_args, c_assign, c_m_assign, c_draw, c_workers_block;
mutex mtx;
```

WORKER THREAD

```
//retrieve arguments
region X when (1){
    args_taken = 1;
}
while(1){
    // wait for main to assign job
    region X when (jobs_assigned > 0){
        jobs_assigned--;

        perform Mandelbrot computation

        region X when(1){
            draw_array[my_no] = JUST_FINISHED; //notify main to draw my slice
            nofjustfin++;
        }
        region X when (unprinted_slices > 0){
            unprinted_slices--;
        }
    }
}
```

3.4.3 Τρενάκι w/ CCR

train_enter()

```
region X when (1){
    waiting++;
    if(waiting == trainCapacity){
        wait_to_fill = 1;
    }
}

region X when (pas_entering){
    onboard++; waiting--;
    if(onboard == trainCapacity){
        pas_entering = 0; train_start = 1;
    }
}
```

train_exit()

```
region X when (pas_exiting){
    onboard--;
    if(!onboard){
        pas_exiting = 0;
        wait_to_empty = 1;
    }
}
```

```
volatile int wait_to_fill, wait_to_empty, train_start;
volatile int pas_entering, pas_exiting, waiting, onboard;
volatile int trainCapacity;
```

```
train_enter( );
// on ride
train_exit ( );
```

passenger

```
while(1){
    region X when (wait_to_fill){
        wait_to_fill = 0; pas_entering = 1;
    }

    region X when (train_start){
        train_start = 0;
    }

    sleep(RIDE_DURATION);

    region X when (1){
        pas_exiting = 1;
    }

    region X when (wait_to_empty){
        wait_to_empty = 0;
    }
}
```

train

