Δεύτερη εργασία Ταυτόχρονου Προγραμματισμού

Ομάδα: 11η

Μέλη:

Παναγιώτης Παυλίδης, 2608

Σταμάτιος Κωφόπουλος - Λυμπέρης, 2548

1η Άσκηση: βιβλιοθήκη

- **Key_t Mysem_create(int val):**

  semid = semget(IPC_PRIVATE) -> creates a set of 2 semaphores,

  semctl(0, SETVAL) -> initializes the first(semid) one to the value we give as argument (the one we use in the program),

  semctl(1, SETVAL) -> initializes the second(sem_mutex) to 1 (we use that one as mutex inside the function),

  return(semid).
- **Mysem_down(key_t semid):**

  semop downs the sem_mutex,

  if(semid value (semval = semctl(GETVAL)) == 0) { semop ups the sem_mutex}

  semop downs the semid, semval--,

  if(semval == 0){ semop ups the sem_mutex}.
- **Mysem_up(key_t semid):**

  semop downs the sem_mutex,

  if(semid value (semval = semctl(GETVAL)) >= 1) { semop ups the sem_mutex, return(-1)}

  semop ups the semid,

  semop ups the sem_mutex,

  return(1).
- **Mysem_destroy(key_t semid):**

  semctl(IPC_RMID) the set of semaphores gets destroyed.

2η Άσκηση : Πρώτοι Αριθμοί

Main:

```
 creates threads and workers
While(i < numbers) {
   down(mutex);
  If(wait_workers != num_threads)
     {up(mutex); wait_main++;
     down(main); down(mutex);}
  for(j < num_threads) {
    if(workers[j].flag == 0 && j < num && I < num) {
      assigns job; flag =-1;
      if(wait_workers != 0) {wait_workers--; up(worker)}
    }
  }
  up(mutex);
}
down(main);
for(all workers) → workers[].flag = 666; up(workers);
down(main);
destroy all sems;
```

Primesearch:

```
down(mutex); num_workers++;
If(last worker && wait_main != 0) → wait_main--; up(main);
wait_workers++; up(mutex); down(workers);
While(1) {
  if(worker.flag != 0) {
    if(terminate) {down(mutex); wait_workers--; up(mutex);
      if(wait_works == 0) → up(main); return(NULL);
    }
    checks if number prime;
  }
  else { down(mutex); up(workers); up(mutex);
          down(workers); continue;}
   down(mutex);
   if(wait_main != 0) → wait_main--; up(main);
   wait_worker++; worker.flag = 0; finished_num++;
   if(all workers blocked && all nums finished) → up(main);
   up(mutex); down(worker);
}
```

3η Άσκηση: Στενή Γέφυρα

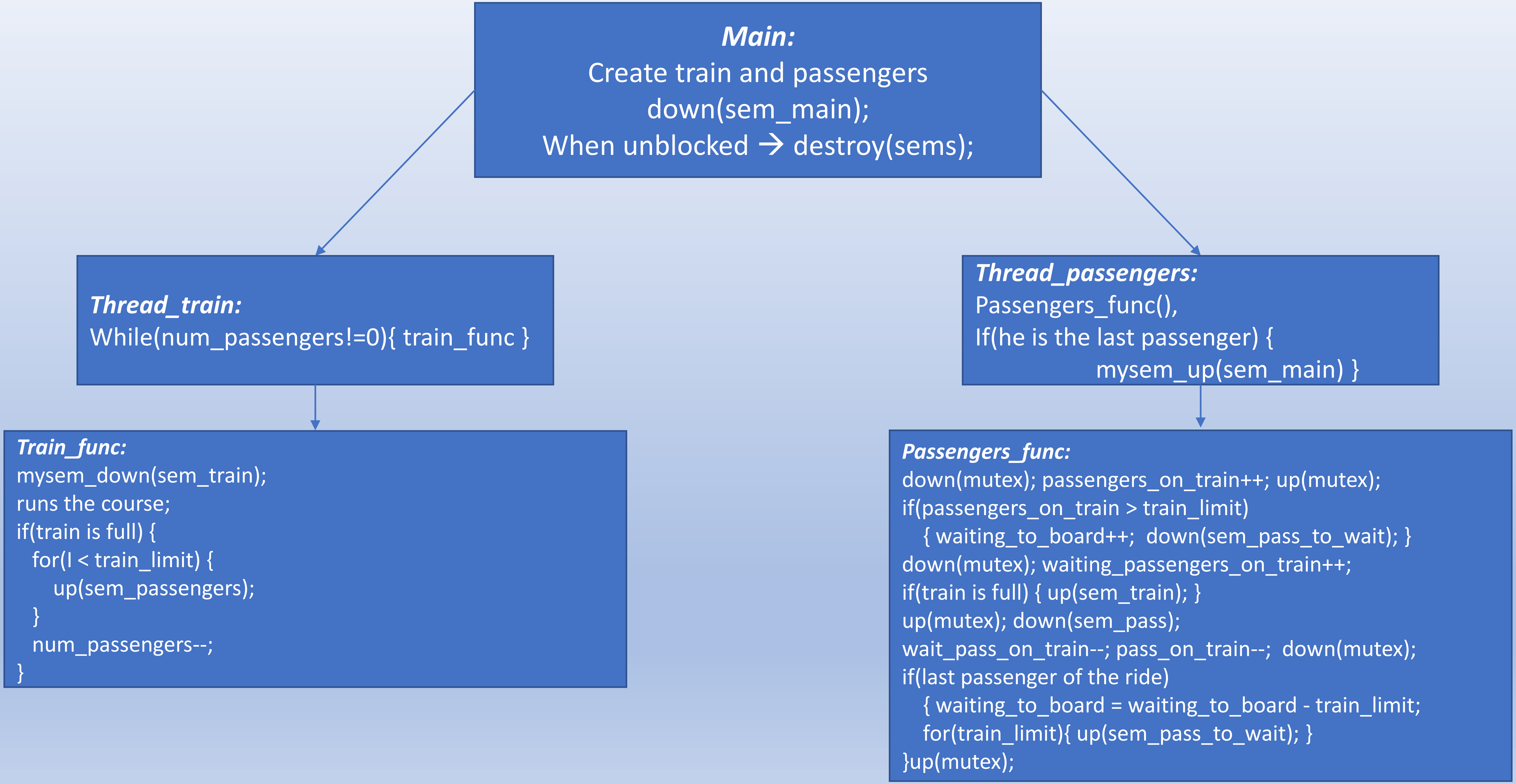
```
Global variables
colour_same_move = -1;
colour_opp_move = -1;
```

```
Main:
Creates sems;
Randomly assigns the color while creates car-thread;
down(main);
destroys sems;
```

```
Threads_func:
down(mutex); checks the color of the first car that arrives;
If(colour==1) {car_in_move = blue}
else{car_in_move = red}, up(mutex);
//Depending on color :
Before bridge Cs
If(colour_opp_move != -1)
  { wait_same_colour++; down(same_colour);}
Down(mutex);
if(bridge_counter >= specific amount of cars)
  {bridge_counter++; colour_opp_move =0/1;
  wait_same_colour++; up(mutex); down(same_colour)
  colour_opp_move=-1; down(mutex);}
Bridge_counter++; up(mutex);
Down(mutex); same_in_bridge++; up(mutex);
if(same_in_bridge > bridge_space){ down(sem_bridge_limit); }
```

```
After bridge Cs
down(mutex);
If(same_in_bridge > bridge_space) { up(sem_bridge_limit);}
same_in_bridge--;
If(same_in_bridge == 0) {
  colour_same_move =-1; colour_opp_move =0/1; bridge_counter = 0;
  If(wait_same_colour != 0 && wait_opp_colour == 0) {
    colour_opp_move = -1; same_colour_move = 0/1;
    for(i < wait_same_colour && i<CARS_POP) {up(same_colour);}
    wait_same_colour -= i;
  }
  for(i < wait_opp_colour && i<CARS_POP)
    {up(opp_colour);}
  wait_opp_colour -= i; up(mutex);}
//after car exits bridge
down(mutex); num_of_cars--; up(mutex);
If(num_of cars == 0) {up(main);} //signal to end main
return(NULL);
```

4η Άσκηση: Τρενάκι



Main:

```
Create train and passengers
down(sem_main);
When unblocked → destroy(sems);
```

Thread_train:

```
While(num_passengers!=0){ train_func }
```

Thread_passengers:

```
Passengers_func(),
If(he is the last passenger) {
               mysem_up(sem_main) }
```

Train_func:

```
mysem_down(sem_train);
runs the course;
if(train is full) {
  for(I < train_limit) {
      up(sem_passengers);
  }
  num_passengers--;
}
```

Passengers_func:

```
down(mutex); passengers_on_train++; up(mutex);
if(passengers_on_train > train_limit)
   { waiting_to_board++;  down(sem_pass_to_wait); }
down(mutex); waiting_passengers_on_train++;
if(train is full) { up(sem_train); }
up(mutex); down(sem_pass);
wait_pass_on_train--; pass_on_train--;  down(mutex);
if(last passenger of the ride)
   { waiting_to_board = waiting_to_board - train_limit;
   for(train_limit){ up(sem_pass_to_wait); }
}up(mutex);
```