CONCURRENT PROGRAMMING

ASSIGNMENT 2 - SEMAPHORES

Dimitris Voitsidis 03480 Iordana Gaisidou 03570 Stavros Stathoudakis 03512





2.1. Binary Semaphores Library

a) binarySemaphores.c

- a) mysem_init (mysem_t, int n)
- b) mysem_down (mysem_t *s)
- c) mysem_up (mysem_t *s)
- d) mysem_destroy (mysem_t *s)

b) mysem_t

```
struct mysem_t {
   int semid;}
```

c) global variables

static mysem_t **sem_arr -> Array of pointers to semaphores that have been created.

static int sem_arr_size -> size of the array of the previous array.

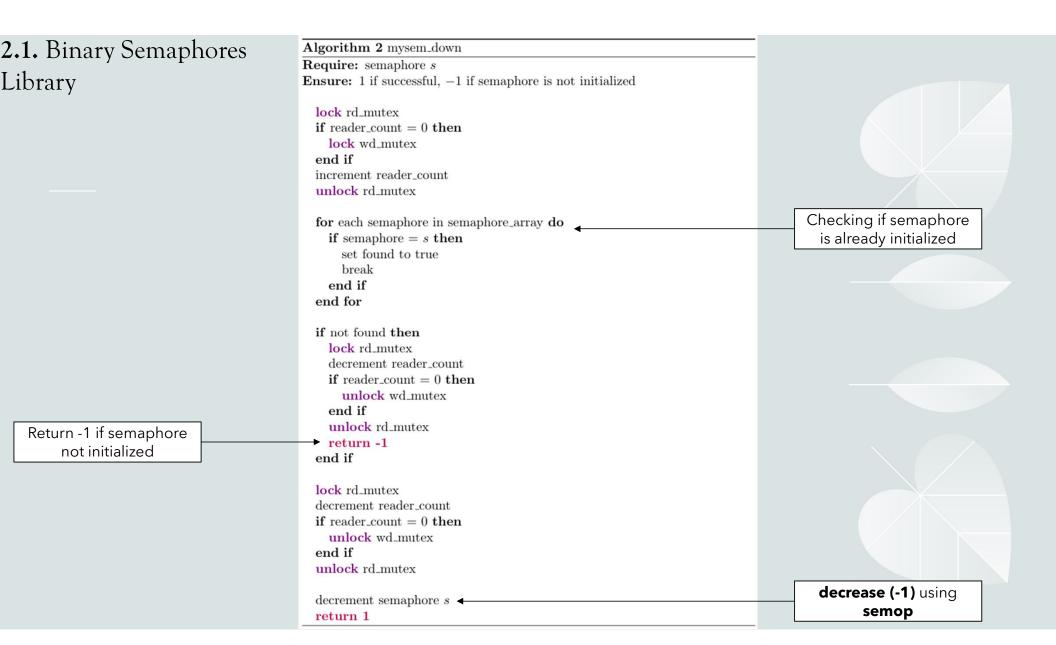
return 1

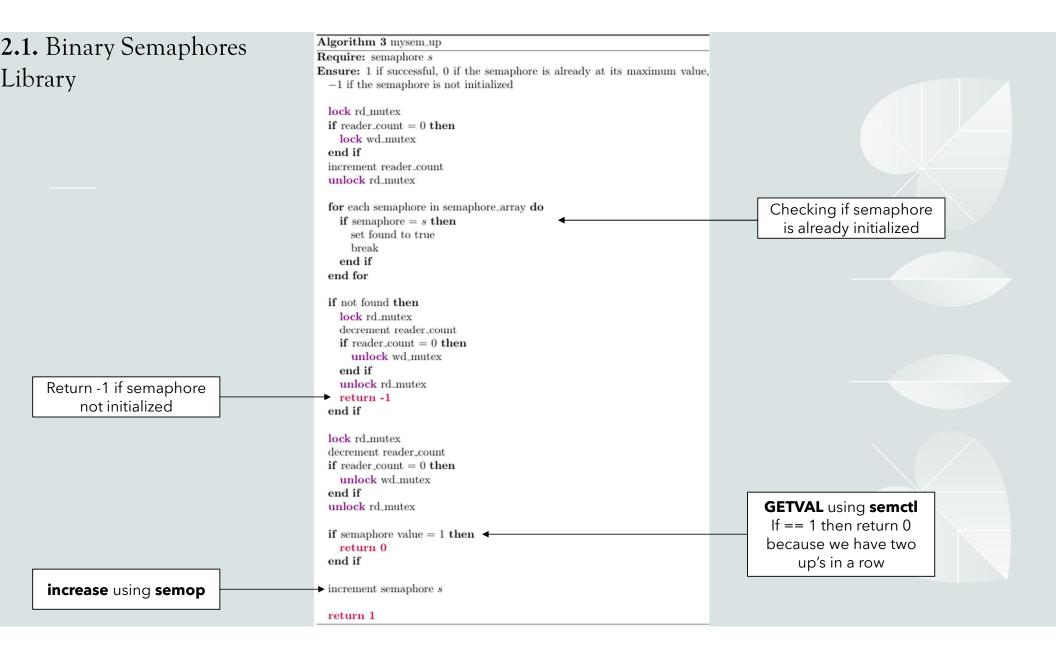
static int rds -> keeping track of readers

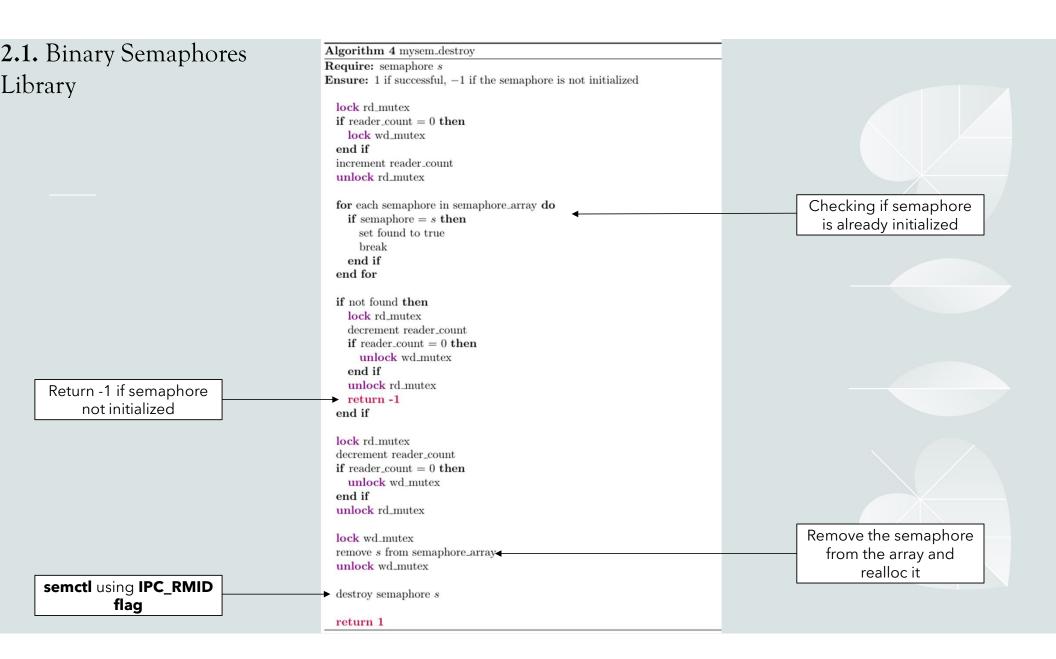
d) global mutexes

```
static pthread_mutex_t rd_mutex static pthread_mutex_t wd_mutex
```

```
Algorithm 1 mysem_init
Require: semaphore s, integer n
Ensure: 1 if successful, 0 if invalid n, -1 if already initialized
 if n \neq 0 and n \neq 1 then
                                                                      Checking if n is not
     return 0
                                                                         equal to 0 or 1
 end if
 lock rd_mutex
 if reader_count = 0 then
   lock wd_mutex
 end if
 increment reader_count
 unlock rd_mutex
                                                                    Checking if semaphore
 for each semaphore in semaphore_array do
                                                                      is already initialized
       if semaphore = s then
     lock rd_mutex
     decrement reader_count
     if reader_count = 0 then
              unlock wd_mutex
     end if
     unlock rd_mutex
     return -1
   end if
 end for
 lock rd mutex
 decrement reader_count
                                                                          semget with
 if reader_count = 0 then
     unlock wd_mutex
                                                                    IPC PRIVATE flag for a
 end if
                                                                            unique id
 unlock rd_mutex
                                                                   semctl with SETVAL flag
                                                                       for initializing to n
 create semaphore s with initial value n
 lock wd_mutex
                                                                     Put the semaphore in
 append s to semaphore_array \leftarrow
 unlock wd_mutex
                                                                            the array
```







2.2. Primary Numbers Tester

```
struct workerArg {
    int *job
    bool *terminate
    mysem_t *semWorker
    mysem_t *semMain }
```

```
void *worker(void *Wargs)
while (1):
                                                 Worker x took the job
    mysem_down(semWorker) ←
                                                    offered by Main
    if terminate
                                                 Wake the rest workers
      mysem_up(semWorker) ◆
                                                      to terminate
      print terminate message and terminate
    get the job (number to check)
                                                 Inform main that I took
   mysem_up(semMain) ◆
                                                 the job to let her give
                                                 the rest. (wake Main)
   if num <= 1
      print number is not equal
    else
      use square root algorithm to check if the number is prime
      print result (prime or not prime)
```

Wake workers to let them terminate

Main Function:

Get number of workers from command line as workersNo

Initialize semaphores semMain to 1 and semWorker to 0

Assign the following to worker_args:

job = address of num

semWorker = address of semWorker

semMain = address of semMain

terminate = address of terminate

Allocate memory for worker threads array t

For each worker:

Create a worker thread and pass worker_args

For each job:

Sleep till a worker take the job main offered

Assign job number from command line to num

mysem_up(semWorker) ← Wake v

Wake workers to take the job main offers

mysem_down(semMain) 🔨

mysem down(semMain) ←

Set terminate flag to True mysem_up(semWorker)

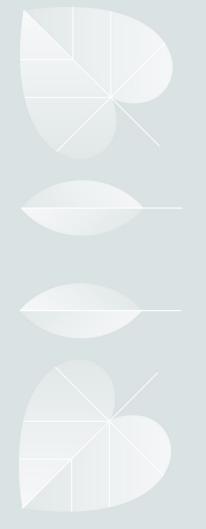
Main should sleep till the last job is taken by a worker

For each worker:

Wait for worker thread to finish using pthread_join

Destroy semaphores semMain and semWorker

2.3. Narrow Bridge



```
struct carArg {
  int maxCarsCrossing; // Maximum number of cars of the same color that can cross the bridge consecutively
  int maxCarsCrossed; // Maximum number of cars allowed on the bridge at the same time
  int *carsCrossing;
                      // Current number of cars crossing the bridge
  int *carsCrossed;
                       // Current number of cars (same colored) that have passed the cross consecutively
  int *waitingUs;
                       // Current number of cars of my color waiting to cross the bridge
  int *waitingOthers;
                       // Current number of cars of the other color waiting to cross the bridge
  carColorE color;
                       // My color
  carColorE *color_bridge; // Current color of cars crossing the bridge
  mysem_t *semUs;
  mysem_t *semOthers;
  mysem_t *mtx;
```

```
void *car(void *Cargs):
mysem_down(mtx)
  increase waiting cars
mysem_up(mtx)
while (1) {
    mysem_down(mtx)
    if (bridge is empty && bridge color != color) {
      change bridge color
      reset cars crossed (0)
    if (bridge color != color || bridge is full ||
      (max cars crossed = cars_crossed && other cars waiting > 0)) {
      mysem up(mtx)
      mysem_down(semUs)←
                                       Sleep
      continue
    decrease waiting cars
    increase crossing cars
    increase crossed cars
    if (same color cars are waiting) {
                                         If a car of our team is
      mysem_up(semUs)
                                         waiting to cross the
                                          bridge, we wake it
    mysem_up(mtx)
    break
  simulate passing bridge time using sleep
  mysem_down(mtx)
  Decrease cars crossing
```

```
if (bridge is empty && other cars waiting > 0) {
    mysem_up(semOthers) 
}

if ((cars_crossed < max_cars_crossed || other cars waiting = 0)  
    && same color cars waiting > 0) {
    mysem_up(semUs)  
}

wake us

mysem_up(mtx) }

Wake us
```

```
Main Function:
initialize_semaphores (semBlue, semRed, mtx) to 1

while (1) {
    read_user_input(carColor)

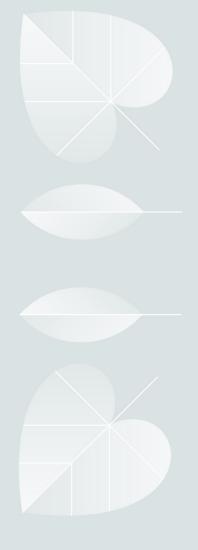
    if (carColor == 'R' || carColor == 'r') {
        create_red_car_thread
    } else if (carColor == 'B' || carColor == 'b') {
        create_blue_car_thread
    } else {
        break
    }
}

wait for threads to join using pthread_join

destroy_semaphores(semBlue, semRed, mtx)
```

2.3. Narrow Bridge

2.4. Train Ride



```
struct Targs {
  int train_capacity; // Train capacity

  bool* destroy; // Flag to terminate train

  int* boarded_passengers; // Number of passengers boarded
  int* waiting_passengers; // Number of passengers waiting

  mysem_t* mtx; // Semaphore to signal waiting passengers
  mysem_t* semBoarding; // Semaphore to signal boarding passengers
  mysem_t* semTrain; // Semaphore to signal train
  mysem_t* semDisembark; // Semaphore to signal disembarking passengers
}
```

2.4. Train Ride

```
void *train(void *Targs)
print_train_start_message()
  while (destroy_flag == false) {
                                    If train is not full then
    mysem_down(semtrain) ←
                                           sleeps
    if (destroy_flag = true && passengers_boarded = 0) {
      break
    sleep()
                                   Disembark the boarded
    mysem up(semdisembark)←
                                         passengers
  mysem_up(mtx)
  if (waiting_passengers != 0) {
                                       Tell the waiting
    mysem_up(semBoarding) ◆
                                     passengers to leave
  mysem_up(mtx)
  while (boarded_passengers != 0) {
    mysem_down(semTrain) 🔻
                                       Tell the waiting
                                     passengers to leave
  return
```

```
void *passenger(void *Pargs)
passenger(args) {
  mysem up(mtx)
  increase waiting passengers
  mysem_down(mtx)
                                      Waits till someone tells
  mysem_down(semBoarding) 
                                            it to board
  mysem up(mtx)
  decrease waiting passengers
  mysem down(mtx)
  if (destroy flag = true) {
                                         Wakes up the next
    mysem_up(semboarding) -
                                       waiting passenger to
    return
                                           destroy itself
 increase boarded_passengers
                                      If the train is full, wake it
  if (train is full) {
                                       up to start its journey
    mysem_up(semTrain) 
 } else {
    mysem_up(semBoarding) ←
                                        else wakes the next
                                        passenger to board
  mysem_down(semDisembark)
                                         Waits the train to
  decrease boarded passengers
                                        complete its journey
```

2.4. Train Ride

```
...rest of passenger
if (destroy_flag = true && boarded_passengers = 0) {
                                                          Wakes up train to
    mysem_up(semTrain) ←
                                                             terminate
  if (train is empty && destroy_flag = false) {
    mysem_up(semBoarding)
                                                                 Main Function:
  } else {
                                                                 main() {
    mysem_down(semDisembark) // Signal other passengers
                                                                   mysem_init(semTrain (0), semBoarding(1), semDisembark(0), mtx(1))
to disembark
                                                                   while (1) {
                       Tell waiting passengers
                                                                     read_user_input()
                       to board the train if its
                               empty
                                                                     if (user input = enter) {
                                Else
                                                                       destroy_flag = true
                                                                                                  Wake up the train if the
                         Tell the rest of the
                                                                       mysem_up(semTrain) ←
                                                                                                 user wants to terminate it
                       boarded passengers to
                                                                       break
                             disembark
                                                                     create_passenger_thread(passengers, i, passenger_args)
                                                                  join threads
                                                                   mysem_destroy(semTrain, semBoarding, semDisembark, mtx)
```



Thank you