Parallel Programming ECE321 Assignment Set 2

Team Members:
Konsoulas Konstantinos
Avramidis Prodomos
Tsirka Chara
Gerontidis Giorgos

Assignment 1: Binary semaphores

```
struct mysem {
        int id;
        char initialized;
}
```

```
mysem_init(sem, value) {
        if invalid value or sem.initialized = ‘T’:
                return
        do {
                sem.id = get semaphore group id of 2
        } while(failure)
        initialize the first to value
        initialize the second to 1
        sem.initialized = ‘T’
        return
 }
```

```
mysem_up(sem) {
        if not initialized
                return
        down(second group sem)
        if value == 1
                up(second group sem)
                return lost up

        up(first group sem)
        up(second group sem)

 }
```

```
mysem_down(sem) {
        if not initialized
                return
        down(first group sem)
 }
```

```
mysem_destroy(sem) {
        destroy group sem
 }
```

Assignment 2: Prime numbers

Semaphores

init(availableWorkersSem,1)

init(bossSem,0)

init(worker.activeSem,0)

Flags

availableWorkers = 0

maxWorkers

worker.active = 0

finishedWorkers = 0

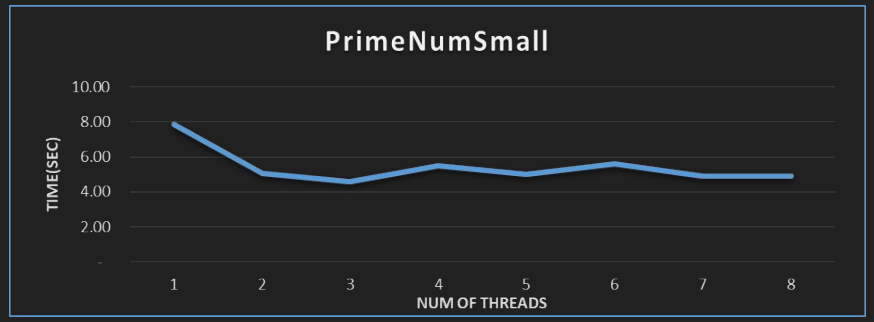
main thread

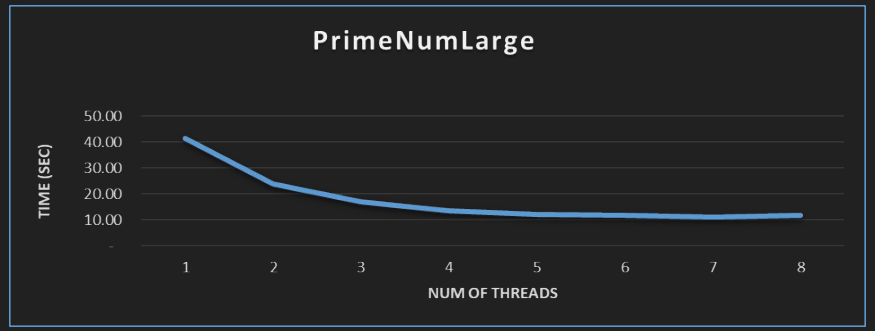
```
create N workers
while (input exists) {
        down(availableWorkersSem)
        availableWorkers - = 1
        if available workers == -1
                up(availableWorkersSem)
                down(bossSem)
        else
                up(availabeWorkersSem)

        find the inactive worker and assign work
        up(worker.activeSem)
 }

up(allWorkers.activeSem)
//wait for all workers to terminate
down(bossSem)
destroy everything
```

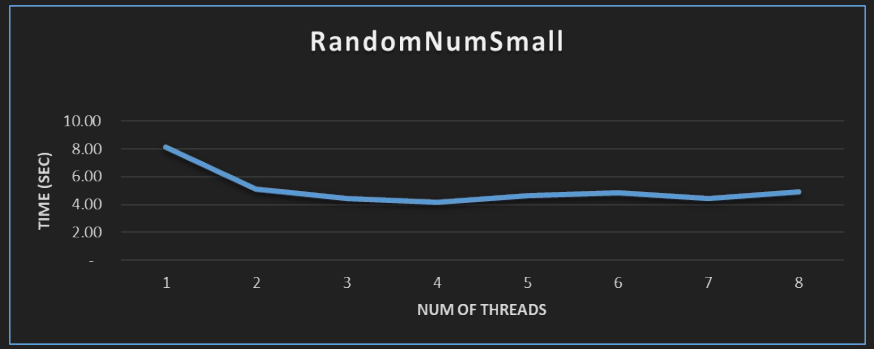
worker thread

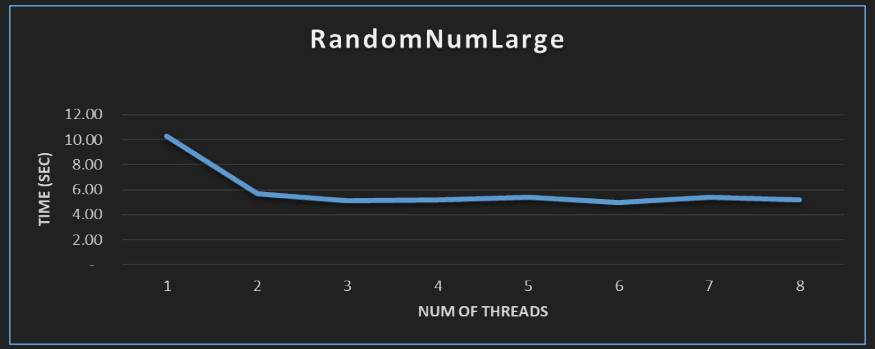
```
while (1) {
        down(availableWorkersSem)
        availableWorkers++
        if(availableWorkers == 0)
                up(bossSem)
        up(availableWorkersSem)
        down(worker.activeSem)
        if(worker.active == 0)
                break
        run algo
        worker.active=0
}
down(availableWorkersSem)
finishedWorkers++
if(finishedWorkers == maxWorkers)
        up(availableWorkersSem)
         up(bossSem)
else
        up(availableWorkersSem)
```

Performance on a 4 core 8 thread system









Assignment 3: Narrow Bridge

Semaphores

init(bridgeAccess, 1)

init(redCarsAccess, 1)

init(blueCarsAccess, 1)

init(myQ, 0)

init(oppositeQ, 0)

Common Variables

bridgeColor

bridgeCapacity

myColorCars

oppositeColorCars

carsPassed

Parameters

maxBridgeCapacity

fairnessFactor

crossTime

```
crossBridge {
        down(myCarsAccess)
        myColorCars++
        up(myCarsAccess)
        down(bridgeAccess)
        if (bridge empty)
                bridge color = my color
        if (bridge full || bridge different color)
                up(bridgeAccess)
                down(myQ)
                down(bridgeAccess)

        bridgeCapacity -=1
        if (cars on the other side)
                carsPassed++
        else
                carsPassed = 0
```

```
down(myCarsAccess)
myColorCars -= 1
up(myCarsAccess)
if(bridge has space && cars on my side
   && bridge same color && fair pass)
        up(myQ)
up(bridgeAccess)
//Crossing bridge safely….
```

```
    down(bridgeAccess)
    bridgeCapacity ++

    if (bridge same color && cars on my side
        && first to get off bridge
        && (fair pass || no cars on the other side))
            up(myQ)

    if (last to get off bridge && cars on the other side
        && (no cars on my side || exceeded fair pass))
            change bridge color
            carsPassed = 0
            up(oppositeQ)

    up(bridgeAccess)
}
```

Assignment 4: Roller Coaster

Semaphores

init(trainSem, 0)

init(onBoardSem, 1)

init(boardingSem, 1)

init(gettingOffSem, 0)

Common Variables

onBoard

boarding

maxTrainCapacity

passengers

```
while(1) {
        down(boardingSem)
        down(onBoardSem)
        if(train has space && boarding == 1)
                onBoard++
                if(train full)
                        boarding = 0
                        up(trainSem)
                        up(onBoardSem)
                        break;
                up(boardingSem)
                up(onBoardSem)
                break;
        up(onBoardSem)
}
down(gettingOffSem)
down(onBoardSem)
onBoard -= 1

//passengers continue…
```

```
//passengers continue here
if(train empty)
        boarding = 1
        up(boardingSem)
        up(onBoardSem)
        return
up(onBoardSem)
up(gettingOffSem)
```

train

```
while(1) {
    down(trainSem)
    sleep(T)
    up(gettingOffSem)
}
```