Project 3 – Lizards

Elijah Reyna and Ryan Waddington

Systems and Networks 1 - COP4634

10/14/2024

Overview

In this project, we were given a half-built project and asked to fill in the blanks. The core problem was to handle race conditions and slowing the flow of threads at a certain point. In this case, each thread was represented metaphorically by a lizard, and they could eat, sleep, and cross the driveway. However, we didn’t want too many lizards crossing at once (In our experiment the maximum was 4). This metaphor works for many similar situations, and the solution we came up with could be applied to those situations. Our changes involve adding mutex locks for changing values and semaphores that indicate if a lizard is safe to cross the road.

The Changes

|  |  |
| --- | --- |
| **Line(s)** | **Change** |
| 115-120 | Created a printMessage():void function to help organize the code |
| 172 | \_catThread = new thread (catThread, this );   * Creates the cat thread and runs it [Project Requirement] |
| 182-184 | if (\_catThread != NULL) {        \_catThread->join();    }   * Joins the thread to the main thread |
| 206-208 | if(running){      sleep( sleepSeconds );    }   * Quick check if the world is still running before it has the cat thread sleep |
| 344-351 | sem\_wait(&lizSemaphore);    if (debug){      printMessage("[" + to\_string(\_id) + "] checking  sago -> monkey grass");      }    if (debug){      printMessage("[" + to\_string(\_id) + "] thinks  sago -> monkey grass  is safe");      }   * Uses the semaphore to hold the thread in place until there is an opening. |
| 368 | mtx.lock();   * Locks other threads from accessing the variables that are about to be changed |
| 371-373 | if(numCrossingMonkeyGrass2Sago + numCrossingSago2MonkeyGrass > maxNumCrossing){      maxNumCrossing = numCrossingMonkeyGrass2Sago + numCrossingSago2MonkeyGrass;    } |
| 382 AND 386 | mtx.unlock();   * Unlocks the mutex that was locked in line 386. The unlock statement in line 382 unlocks the mutex before the program exits (if it happens) |
| 390-392 | if(running){    sleep( CROSS\_SECONDS );    }   * Quick if-statement to prevent sleeping when the world has ended |
| 397-399 | mtx.lock();    numCrossingSago2MonkeyGrass--;    mtx.unlock();   * Locks and unlocks around the variable change to prevent other threads from accessing it while it is being accessed |
| 417 | sem\_post(&lizSemaphore);   * Uses the semaphore to decrease the value of those “in the driveway” |
| 481 | mtx.lock();   * Locks other threads from accessing the variables that are about to be changed |
| 496 AND 500 | mtx.unlock();   * Unlocks the mutex that was locked in line 481. The unlock statement in line 496 unlocks the mutex before the program exits (if it happens) |
| 505-507 | if(running){    sleep( CROSS\_SECONDS );    }   * Quick if-statement to prevent sleeping when the world has ended |
| 511-513 | mtx.lock();  numCrossingMonkeyGrass2Sago--;    mtx.unlock();   * Locks and unlocks around the variable change to prevent other threads from accessing that variable while it is being accessed |
| 555-562 | aLizard->sleepNow();      aLizard->sago2MonkeyGrassIsSafe();      aLizard->crossSago2MonkeyGrass();      aLizard->madeIt2MonkeyGrass();      aLizard->eat();      aLizard->monkeyGrass2SagoIsSafe();      aLizard->crossMonkeyGrass2Sago();      aLizard->madeIt2Sago();   * Performs the lizard functions [Project Requirement] |
| 606 | sem\_init(&lizSemaphore, 0, MAX\_LIZARD\_CROSSING);   * Initializes the semaphore in the main function |
| 621-623 | for(int j = 0; j < NUM\_CATS; j++)        Cats.push\_back(new Cat(j));       }   * Adds the cat threads to the vector |
| 631-633 | for (int j = 0; j < NUM\_CATS; j++) {      Cats[j]->runCat();      }   * Runs each of the cat threads |
| 649-654 | for(int k = 0; k < NUM\_LIZARDS; k++){      allLizards[k]->wait();    }    for(int l = 0; l < NUM\_CATS; l++){      Cats[l]->wait();    }   * Waits for the cat threads and the lizard threads to finish before moving on |
| 659 | sem\_destroy(&lizSemaphore);   * Destroys the semaphore |
| 664-670 | for(int m = 0; m < NUM\_LIZARDS; m++){      delete[] allLizards[m];    }    for(int n = 0; n < NUM\_CATS; n++){      delete[] Cats[n];    }   * Deletes all the cat and lizard objects |

Results

|  |  |  |
| --- | --- | --- |
| **WORLDEND (s)** | **Maximum Number of Lizards Crossing** | **Lizards Safe?** |
| 30 | 4 | Yes |
| 60 | 4 | Yes |
| 90 | 4 | Yes |
| 180 | 4 | Yes |

Issues Encountered

No issues were encountered.