Middle East Technical University Department of Computer Engineering

CENG 334

Introduction to Operating Systems

Spring 2017-2018

HW2

Due date:25th April 2018, 23:55

1 Objectives

In this assignment you will implement a simple ant simulator. The simulator will be a multi threaded application where threads share the same resources. The simulation takes place on a 2D grid where only ants and foods exist. Ants have a very limited behavior. They move randomly, if they see a food they take it and start carrying. If they came across another food while they are carrying food, they leave the food there and continue moving.

2 Program Specification

In your program, the main thread will read the inputs and print the status of the program while reading the user inputs to exit the simulation prematurely, to change the delay parameter or to update number of sleeper ants which will be explained later. All work will be done on separate threads that each control one ant. An outline of the main function exist in the example code that is given. This code includes the visualization and user interactions, you can use this code in your solution.

2.1 The Grid

The 2D world is a 30×30 grid. Each cell of the grid only exist in one of six states: empty cell represented with `-`(minus sign), cell with food represented with `o`(lowercase letter o), cell with ant represented with `1`(one), cell with food carrying ant represented with `P`(uppercase letter P), cell with sleeping ant represented with `S`(uppercase letter S) and cell with food carrying sleeping ant represented with `$`(Dollar sign). An ant cannot occupy or pass through a cell which has a food or another ant in it.

2.2 Ant Behavior

An ant ca be one of the four states: 'with food', 'without food', 'tired' and 'sleeping'. Tired ant does not have a symbol represented on the grid. While an ant is in 'without food' state:

- Look at the immediate neighborhood for food. There are 8 cells including diagonals.
  - If one exist, move into that cell and become an ant 'with food'.
  - If not, Select a random direction among the available directions and move there. Cannot move into a cell already occupied or out of border.

- Delay specified amount of time that will be explained later.
- One step concluded, start over.

Table 1: While an ant is in 'without food' state, If food exist move into that cell and become an ant 'with food'

-	-	o
-	1	-
-	-	-

(a)

-	-	P
-	-	-
-	-	-

(b)

While an ant is in 'with food' state:

- Look at the immediate neighborhood for food. There are 8 cells including diagonals.
  - If one exist, leave the food that is beeing carried to the current cell. Move into one of the empty neighboring cells as a 'tired' ant.
  - If not, Select a random direction among the available directions and move there. Cannot move into a cell already occupied or out of border.
- Delay specified amount of time that will be explained later.
- One step concluded, start over.

Table 2: While an ant is in 'with food' state, If food exist leave the food that is beeing carried to the current cell. Move into one of the empty neighboring cells as a 'tired' ant.

-	-	o
-	P	-
-	-	-

(a)

-	-	o
1	o	-
-	-	-

(b)

While an ant is in 'tired' state:

- Look at the immediate neighborhood. There are 8 cells including diagonals.
- Move to a random empty cell as a ant 'without food'.
- Delay specified amount of time that will be explained later.
- One step concluded, start over.

If no possible cell exist for the movement of the ant, it should stay in place for that specific step.

Each ant should read the amount of Delay time in milliseconds for every step and wait an amount equal to read Delay time value plus a random amount between 0 and 9. For example, If Delay time is 50 then an ant waits between 50 and 59 milliseconds.

Sleep Behavior If an ant is in 'sleeping' state it does not have any actions and waits for the signal to wake up. There should not be any busy waiting. You should give each ant an index number as an ID. An ant goes into sleep or wakes up depending on its ID and the number of ants that should be sleeping. For example, if 3 ants must be sleeping, ants with the IDs 0, 1 and 2 must be in sleep state. The number of sleeping ants can be changed in run time. If number of sleeping ants is decreased by one then ant with ID 2 must wake up in previous example.

Implementation of run time changes that affect the number of sleeping ants and the delay times are explained in the later sections.

2.3 Grid Access Strategy

You must limit the access of the threads to the data by locking the whole grid, rows/columns or locking individual cells. This effect the total number of actions that can be taken in a given amount of time by your ants. Selected strategy will effect your final grading. Maximum grades will be 80 for whole grid locking, 90 for row or column locking and 100 for cell locking.

2.4 Implementation, Compilation & Execution Details

Your program should get three input parameters. First is for the number of ants, second for the number of foods and the last one is the maximum amount of time the simulation will run in seconds. Given number of foods and ants should be placed randomly on the grid as the starting point of the simulation. Your simulation should end and your program should exit without any error code or outputs to the standard output after given amount of time.

There are functions given with the homework document that you have to use. These functions governs the access to the data structures of the program, mainly the grid that ants live on. You **have to** use these functions to access the data. None of the functions have any access control. You should ensure data integrity while using this functions. The correctness of the state will be checked in the `drawWindow()` function, you must lock all resources accessed by your ant threads before calling this function. The correctness is existence of correct number of ants and foods on the grid in any call of the `drawWindow()` function.

`void setDelay(int d)`: Sets the delay to the given number.

`int getDelay()`: Returns the amount of delay.

`void setSleeperN(int d)`: Sets the number of sleepers to the given number.

`int getSleeperN()`: Returns the number of sleepers.

`void putCharTo(int i, int j, char c)`: Replaces the character in coordinates `i` and `j` of the grid with the character `c`.

`char lookCharAt(int i, int j)`: Returns the character in coordinates `i` and `j` of the grid.

`void startCurses()`: Initializes the `Ncurses`[1] for the visualizations. Example usage exist in given code.

`void endCurses()`: Releases the `Ncurses` structures.

`void drawWindow()`: Draws the visualization of the data structures.

[1] Ncurses is a terminal control library. You can get more information here if you are interested.

These functions are implemented in the `do_not_submit.h` file that is given with the homework. You have to include this file in your solution but do not modify or submit this file. There are also a definition to represent the grid size which is `GRIDSIZE` and a definition to represent amount of delay between screen draws `DRAWDELAY`. You must wait `DRAWDELAY` amount of time between `drawWindow()` calls. Do not use anything else from this file in your solution or it will not compile during grading.

There is also `main.c` file that includes the usages of these functions and you can use it as a starting point for your solution. It also have the code to read from the keyboard 'q','-', '+', '*', '/' characters to end the simulation, increase/decrease the delay time and increase/decrease number of sleeping ants respectively.

You can use any function included in `stdlib.h` but nothing else. Pay attention to the fact that only the `stdlib.h` not the whole standard library.

You **have to** provide a makefile with your implementation which creates an executable named `hw2`. You should have parameters `-lncurses -lpthread` for the libraries your are using. You must **not** use any compiler optimizations (O2, O3, etc.). It will be checked during grading. You can use the example makefile given with the homework.

Your submission should be a tar file, only containing **your code** and the `Makefile` to compile your code in its root. You can do this with a command such as this: `tar -cvzf hw2.tar.gz *.c makefile`.

Your code will be executed with `./hw2 5 5 5`. You can assume valid parameters.

2.5 Hints & Tips

- You can start by implementing one ant in a single threaded manner. Then add access control with mutexes and/or semaphores. Then turn your program into a multi-threaded one with multiple ants. Then add condition variables for sleeper ants.
- You can use `valgrind` and `helgrind` to check memory leaks and deadlocks.

```
$ valgrind --leak-check=yes ./a.out
$ valgrind --tool=helgrind ./a.out
```

3 Regulations

1. Your code have to be in **C**.
2. Submission will be done via COW. Create a tar.gz file named *hw2.tar.gz* that contains all your source code files and a makefile. The executable should be named *hw2*.
3. Following sequence of commands should compile and run your code, otherwise you **lose 10 points**.

```
$ tar -xf hw2.tar.gz
$ make all
$ ./hw2 5 5 50
```

4. Your codes will be evaluated with a black-box approach and **have to compile and run on lab machines**.
5. Please ask your questions related to the homework on piazza instead of email. Your friends, who may face with the same problem, can see your questions and answers.
6. Do not cheat.