Middle East Technical University Department of Computer Engineering

CENG 334

Operating Systems

Spring 2021-2022

Homework 2 - All About (Cigarette) Butts

Due date: 25 May 2022, Wednesday, 23:59

1 Objective

The objective of this homework is to gain hands-on multi-threaded programming experience by simulating an environment involving multiple agents and actions. To this end, you will create and run threads and use synchronization primitives such as mutexes, semaphores and condition variables using the POSIX threads library `pthreads` on Linux.

The homework materials are at `https://github.com/denizmsayin/metu-os-22-hw2`.

2 Scenario

One of the core peace-time military activities performed by privates is the daily cleaning of assigned areas, including the terrain outside buildings (*mıntıka temizliği* in Turkish). Ironically, 95% of the litter outside consists of cigarette butts generated by the privates themselves. Every single morning, the privates gather all the cigarette butts from the ground everywhere and leave *absolutely nothing*. Somehow, by the time the next cleaning activity comes around (a day later), hundreds of cigarette butts have reappeared all over the terrain! Guess who smoked all those cigarettes? It's a pointless endless cycle of work, but everyone needs something to do...

Sounds like something fun to simulate, right? No?.. Well, we're going to do it anyway!

Note: Cigarette butts will be affectionately referred to as *cigbutts* from now on for brevity since we will be mentioning them *a lot*.

3 Simulation

You will simulate the daily cleaning scenario in multiple development steps, where each step will add extra complexity to the simulation.

The core of the simulation will be a rectangular region (the *grid*) split into G_i rows and G_j columns, having $G_i \times G_j$ dimensions. Each equally-sized cell will initially be filled with a certain amount of cigbutts.

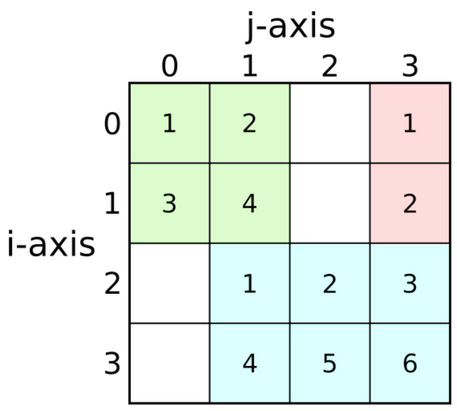

Figure 1: Emptying order of cells for different areas, shown in different colors

3.1 Part I - Gathering the Cigbutts (25 points)

3.1.1 Definition

In the first part of the simulation, we will simulate some poor *proper private*s gathering cigarette butts from rectangular areas in the grid. Proper privates have a bunch of properties:

- Each proper private has a unique ID `gid` and should be simulated by a single thread.
- Proper privates lock down rectangular areas consisting of multiple cells to gather cigbutts. Each private has their own given constant dimension for the areas they lock down: $s_i \times s_j$ with s standing for span. For example: one private may always lock down 1×2 areas, while another could always lock down 3×3 areas.
- Two proper privates' gathering areas should not intersect. If a proper private wants to lock down an area that intersects with another proper private's currently locked down area, they will have to wait for the other to finish.
- Once a proper private locks down an area, he should start gathering one cigbutt every t_g milliseconds. He should also wait t_g milliseconds before gathering his *first* cigbutt after locking down an area, i.e. he should not take a cigbutt immediately as soon as he locks down the area. The value of t_g is given as a parameter for each proper private and remains fixed.
- Proper privates should first start with the top-left cell of the area they want to gather from and remove one cigbutt at a time. When the top-left cell becomes **empty**, they will move to the one on the right and gather from that cell. When the whole row becomes clear of cigbutts, they will continue starting at the leftmost cell of the next row. In short: start from the top left, move right first and then down, ending at the bottom right as cells become empty. See Figure 1 showing the gathering order of cells in 3 different areas across a 4×4 grid.
- Every proper private has a given number of areas they want to empty (remove all cigbutts from), n_g, which is at least 1. The coordinates of the top-left corners of these areas will be given in the form (i_0, j_0), (i_1, j_1), ..., (i_{n_g-1}, j_{n_g-1}). They should only stop gathering cigbutts from one area and move on to the next when no cigbutts are left. These areas will not go over the bounds of the grid, i.e. it is guaranteed that $i_k + s_i < G_i$ and $j_k + s_j < G_j$ for all k.
- Once a proper private finishes gathering cigbutts from all n_g of its assigned areas, it exits and its thread stops.

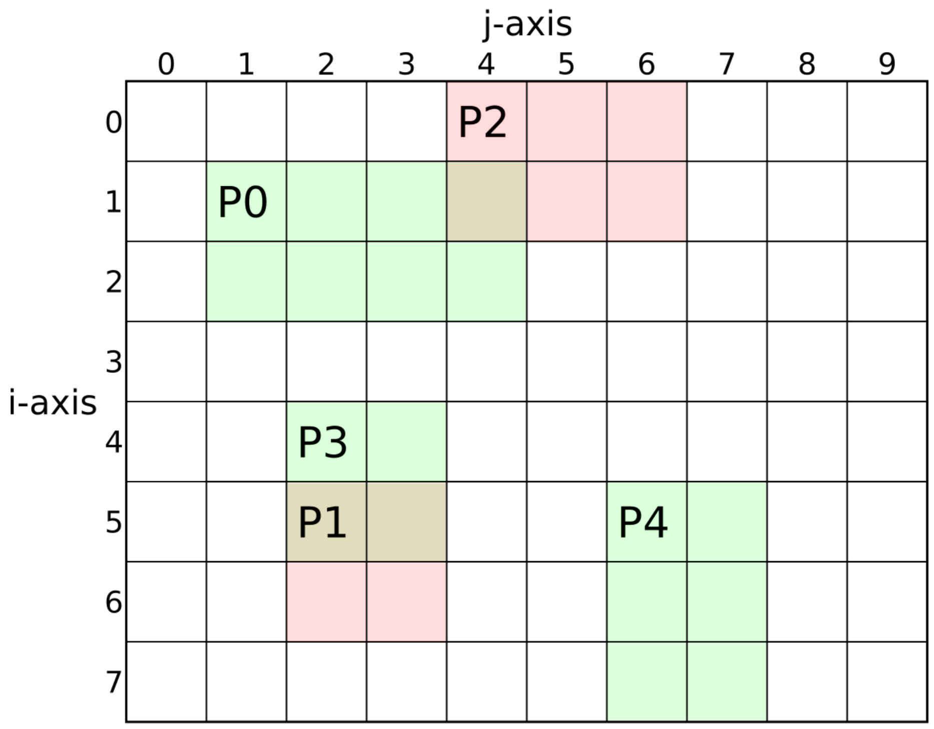

Figure 2: Proper privates gathering on a grid, see 3.1.2 for an explanation

3.1.2 Example on Figure

An example 8×10 grid is shown in Figure 2 having 5 proper privates (cigbutt amounts are not shown). Private P0 has a 2×4 gathering area and its gathering area is given as (1, 1), which is the top-left corner of the area. Similarly, P3 has a 2×2 gathering area and is gathering from (4, 2); while P4 has a 3×2 area and is gathering from (5, 6). These areas are marked in green.

P1 has a 2×2 area and wants to gather from (5, 2), but cannot do so because the gathering area intersects with P3's area and P3 is currently gathering. P1 has to wait for P3 to finish. In the same way, P2 has a 2×3 gathering area and wants to gather from (0, 4), but cannot because one cell of that area intersects with P0's. So, P2 will have to wait for P0 to finish. The intersections are marked in brown, while the rest of the areas are marked in red, since the privates cannot start gathering before locking down their whole area.

3.1.3 Output Format

The output of your program will consist of notifications about the simulation, all printed through the function `hw2_notify` declared in `hw2_output.h`. Proper privates will call this function right after or before certain events:

1. Immediately after creation (thread starting), each proper private should call:
   `hw2_notify(PROPER_PRIVATE_CREATED, gid, 0, 0)`
2. After locking-down an area for gathering (not while waiting!):
   `hw2_notify(PROPER_PRIVATE_ARRIVED, gid, area_top_left_i, area_top_left_j)`
3. Right after picking up a cigbutt:
   `hw2_notify(PROPER_PRIVATE_GATHERED, gid, gathered_cell_i, gathered_cell_j)`.

4. Once an area has been completely emptied of cigbutts and the proper private is leaving: `hw2_notify(PROPER_PRIVATE_CLEARED, gid, 0, 0)`.

5. Finally, once the proper private has finished emptying *all* its assigned areas and is about to exit: `hw2_notify(PROPER_PRIVATE_EXITED, gid, 0, 0)`

3.1.4 Input Format

- The first line will contain G_i and G_j, the size of the cigbutt grid.
- Each of the following G_i lines will contain G_j natural numbers, the initial number of cigbutts in the grid.
- The next line will contain N_p, the number of proper privates.
- Each proper private is defined on multiple lines. The first line contains the `gid`, s_i, s_j, t_g and n_g. These are the core parameters of the proper private, as explained in 3.1.1. Remember that t_g is in **milliseconds**.
- Each of the following n_g lines will contain a pair i_k and j_k, the top-left corner of the next area the proper private wants to gather from.

Input Example:

```
3 5             // Grid size
10 0 8 9 16     // Cigbutt counts
3 18 5 0 0
4 7 3 3 3
2               // Number of proper privates
0 2 2 1000 3    // Proper private 0 parameters
0 0             // 3 areas to gather from
1 0
1 2
7 2 4 700 1     // Proper private 7 parameters
1 1             // 1 area to gather from
```

Please do note that the actual input will contain no comments, those are just for clarification:

```
3 5
10 0 8 9 16
3 18 5 0 0
4 7 3 3 3
2
0 2 2 1000 3
0 0
1 0
1 2
7 2 4 700 1
1 1
```

3.2 Part II - Obeying Orders (35 points)

3.2.1 Definition

So far so good. Time to add a layer of complexity!

The proper privates are of course subject to their commander's orders. Luckily, the commander has a big heart and realises that having the proper privates gather non-stop is kind of cruel. Therefore, the commander will issue orders at given times to ease the proper privates' burden a bit:

1. **BREAK!**: This order will cause the proper privates to take a break, which means they will stop gathering cigbutts and disperse until the order to continue is given.

   Proper privates are supposed to take the break immediately. They should not wait to wake up and gather another cigbutt if they are in the process of gathering, and they should not wait to acquire an area in case they were waiting for an area to be unlocked. They need to stop whatever they are doing: proper privates should generate no other notifications (cigbutt gathered, area locked etc.) other than taking a break.

   They should also unlock the areas they've locked down when leaving for the break: the whole grid should be unlocked during the break.

   In case the proper privates are already on a break, this order should be ignored.

2. **CONTINUE!**: This order will cause the proper privates that are on break to continue gathering cigbutts, once again immediately.

   Note that the proper privates may not be able to continue from where they left off instantly: someone else may lock the area they were at before the break since all the areas were unlocked, in which case the proper private will have to wait.

   In case the proper privates are already working, this order should be ignored.

3. **STOP!**: This last order will cause all the proper privates to unlock their areas and exit the program immediately, ending the simulation. Again, no cigbutts should be gathered after this order. This order could also come while on a break.

3.2.2 Output Format

We will be adding some extra notifications for part 2:

1. When the BREAK! order is issued, the simulation should notify us with `hw2_notify(ORDER_BREAK, 0, 0, 0)`. Then, each proper private should notify of their break with `hw2_notify(PROPER_PRIVATE_TOOK_BREAK, gid, 0, 0)`.

   Once again, proper privates should generate no other notifications than `PROPER_PRIVATE_TOOK_BREAK` after the `ORDER_BREAK` notification is sent. No gathering, arriving or clearing.

   In case the proper privates were already on break, the `ORDER_BREAK` notification should still happen, but no proper privates will react.

2. Similarly, as soon as the CONTINUE! order is issued, `hw2_notify(ORDER_CONTINUE, 0, 0, 0)` should go out. Afterwards, proper privates that are waking up should notify with `hw2_notify(PROPER_PRIVATE_CONTINUED, gid, 0, 0)`. Proper privates will of course also send `PROPER_PRIVATE_ARRIVED` notifications as they re-lock areas for gathering after continuing.

   Similar to the BREAK! case: If the proper privates were already gathering, `ORDER_CONTINUE` will go through, but the proper privates will not react.

3. For the STOP! order, the `hw2_notify(ORDER_STOP, 0, 0, 0)` notification should go through as soon as the order is given. Just like taking a break, proper privates should stop what they are doing and exit with `hw2_notify(PROPER_PRIVATE_STOPPED, gid, 0, 0)`. No other notifications including `PROPER_PRIVATE_EXITED` should be sent by the proper privates.

3.2.3 Input Format

- The initial part of the input will be as in Part I: containing information about the grid and proper privates.
- The next line will contain N_o, the number of orders.
- The following N_o lines will each contain a number t_o and a string s_o. t_o is *when* in milliseconds after the start of the program the order should be given. s_o will be one of `break`, `continue` or `stop`; always in lowercase. It is guaranteed that the times will be in ascending order (i.e. the orders will be given in sequence). You do not have to sort them yourself!

Example:

```
3 3             // Grid and proper private information, as before
1 2 3
4 5 6
7 8 9
1
0 1 1 2500 5
0 0
1 1
2 2
2 1
1 2
6               // Number of orders
1000 break      // Break order at 1000th msec
3000 continue   // Continue at second 3 (NOT 3s AFTER THE BREAK!)
5000 continue   // Pointless continue order at second 5
10000 break     // Break at second 10
13500 stop      // Stop at second 13.5
20000 break     // Break at second 20, pointless since everyone left
```

As you can see, it is possible to have 'pointless' orders. In this case, the `ORDER_*` notifications should still go through, but there will be no one to react to the orders. *This is fine.*
All orders should be delivered before shutting down the program, even if all proper privates have already stopped/finished.

3.3 Part III - Sneaking in a few Ciggies (40 points)

3.3.1 Definition

That second part is much tougher to be sure. But we're not done yet!

In this following part we introduce some *sneaky smokers* who will be smoking and generating cigbutts around where they smoke while the proper privates are working. [1]

[1] The more law-abiding smokers are proper privates and put the cigarettes they smoke during breaks in trash bins. Since they do not litter, they have no effect on the simulation!

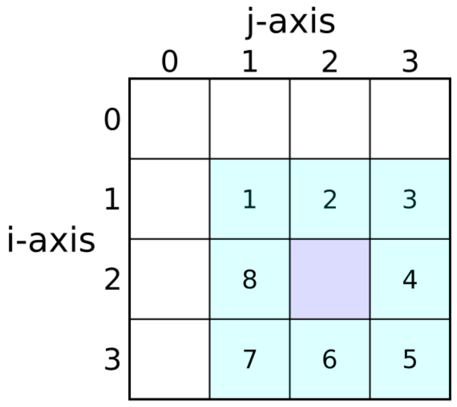

Figure 3: Littering order for a sneaky smoker at (2, 2)

- Sneaky smokers also have their own unique ID `sid` separate from proper private IDs and should be simulated by a single thread.
- Sneaky smokers will smoke in cells. While smoking, they will always litter around themselves in a 3×3 area centered on the cell in which they smoke.
- Multiple sneaky smokers cannot be in the same cell. But their 3×3 littering areas may intersect; they can be next to each other in which case they will litter under each other's feet.
- Sneaky smokers' littering area should not intersect with an area currently locked down by a proper private. That would be disrespectful! In case a sneaky smoker wants to litter in an area currently locked down by a proper private, they have to wait. Similarly, proper privates will have to wait in case a sneaky smoker is currently littering in the area where they want to gather at. In short, there should be no intersection between active sneaky smoker and proper private areas!
- After they lock down a cell for smoking, sneaky smokers will start generating one cigbutt every t_s milliseconds. Again, they should wait for t_s milliseconds upon arrival and not litter instantly for their first cigarette. They will litter rotating through the small 8-cell perimeter around them in clockwise order starting from the top left, getting to the top left again at their 9th, 17th, 25th cigarette etc. See Figure 3 for an example sneaky smoker in a cell.
- Every sneaky smoker will have n_s cells they want to smoke at (at least 1). The coordinates of these cells (center of the 3×3 littering area) as well as how how many cigarettes they intend to smoke there will given as 3-tuples (i_0, j_0, c_0), (i_1, j_1, c_1), ..., $(i_{n_s-1}, j_{n_s-1}, c_{n_s-1})$. It is guaranteed that the 3×3 area will fit inside the grid, i.e. $1 \leq i_k \leq G_i - 2$ and $1 \leq j_k \leq G_j - 2$ for all k.
- Once a sneaky smoker finishes smoking all the cigarettes it intended to smoke, it exits and its thread stops.
- Sneaky smokers do not care for BREAK! and CONTINUE! orders. They will keep smoking through those, which may cause proper privates to have to wait for many sneaky smokers to leave their cells when returning from their break. This is normal.
- Sneaky smokers will also stop and leave the simulation when the STOP! order arrives.

3.3.2 Example on Figure

An example grid including both proper privates and sneaky smokers is shown in Figure 4. The grid has the same dimensions as before, 8×10. S0 is smoking at (1, 2) and littering in the 8 cells around (1, 2).

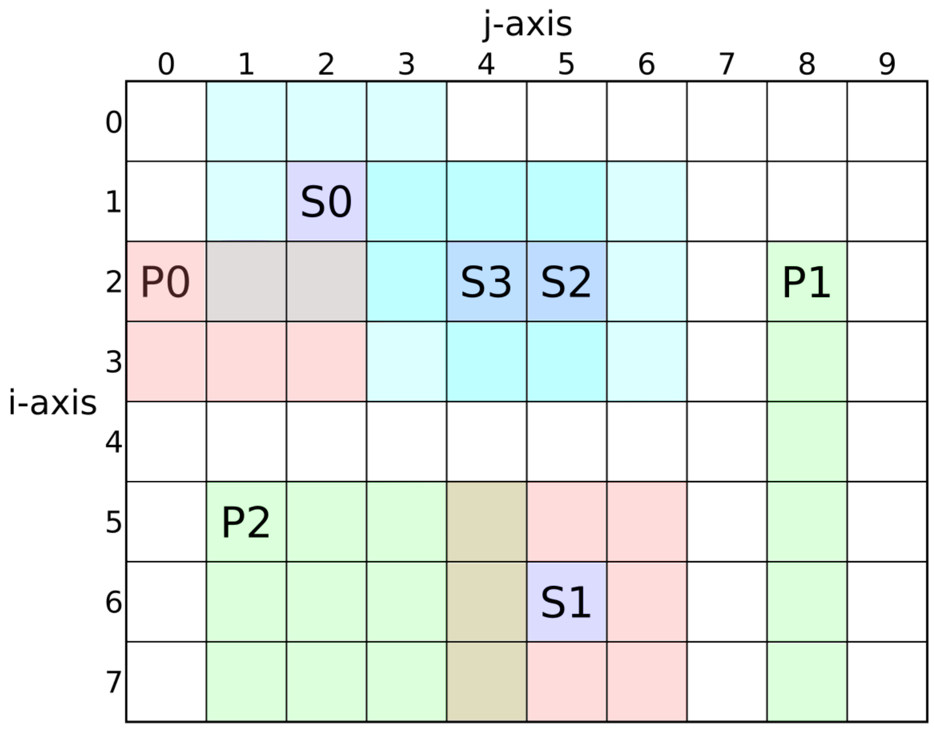

Figure 4: Proper privates and sneaky smokers on a grid, see 3.3.2 for an explanation

Similarly, S2 is smoking at (2, 5) and S3 is smoking at (2, 4). Note that littering areas and smokers can safely intersect; the only constraint is that there should be no more than one sneaky smoker in a cell at a time.

P1 is gathering in a 6×1 area at (2, 8) and P2 is gathering in a 3×4 area at (5, 1).

P0 wants to gather from a 2×3 area at (2, 0), but cannot do that yet since some cells are actively being littered by S0. Similarly, S1 wants to smoke at (6, 5) but has to wait for P2 to finish gathering, since its littering cells would intersect with P2's gathering area.

3.3.3 Output Format

We once again have a bunch of notifications for different events, similar to proper privates:

1. Call `hw2_notify(SNEAKY_SMOKER_CREATED, sid, 0, 0)` on thread creation.
2. When a sneaky smoker manages to lock down a cell for smoking, notify with `hw2_notify(SNEAKY_SMOKER_ARRIVED, sid, cell_i, cell_j)`.
3. After a sneaky smoker finishes smoking a cigarette and litters a cell use `hw2_notify(SNEAKY_SMOKER_FLICKED, sid, littered_cell_i, littered_cell_j)`.
4. When a sneaky smoker has smoked all the cigarettes they wanted to smoke in the current area, call `hw2_notify(SNEAKY_SMOKER_LEFT, sid, 0, 0)` to notify about them leaving.
5. Notify via `hw2_notify(SNEAKY_SMOKER_EXITED, sid, 0, 0)` after a sneaky smoker finishes smoking all their cigarettes and is about to exit.
6. Finally, use `hw2_notify(SNEAKY_SMOKER_STOPPED, sid, 0, 0)` when a sneaky smoker stops after a STOP! order. No other notifications should be delivered after a STOP! order goes through, as with the proper privates.

3.3.4 Input Format

- The initial lines will contain grid, proper private and order information as in Part II.
- The following line will contain N_s, the number of sneaky smokers.
- Each following sneaky smoker is defined over multiple lines. The first line will contain `sid`, t_s and n_s as defined in 3.3.1.
- The following n_s lines will contain a triplet i_k, j_k and c_k. The center coordinates of the next cell the sneaky smoker wants to smoke at and the number of cigarettes they will smoke there. $c_k \geq 1$ is guaranteed.

Example:

```
5 4             // Grid, proper private and order information
0 7 6 2
11 2 4 12
5 5 5 7
3 1 3 8
5 6 0 15
2
3 2 2 1700 2
1 1
3 1
9 1 1 2200 1
1 0
2
2222 break
5555 continue
3               // Number of sneaky smokers
0 7000 3        // Smoker 0 parameters (id, time to smoke, cell count)
1 1 3           // 3 cigarettes centered at (1, 1) etc.
2 2 7
3 2 1
1 2200 1        // Smoker 1 parameters
1 1 10
2 1000 2        // Smoker 2 parameters
2 1 3
1 2 5
```

4 Specifications

- Your code must be written in C or C++.
- Your implementation will be compiled and evaluated on the `ineks`, so you should make sure that your code works on them.
- How your implementation should behave in various complex scenarios will be compiled in `complex_-scenarios.pdf` in the homework repository. That PDF will be updated with new scenarios as they come up.

- All of part I, II and III inputs are valid. You can check for the existence of orders by seeing if there is any input left after the proper private definitions are over. Similarly for part III and the existence of sneaky smokers: Check for extra input after the orders.
- You are free to use any standard libraries, but C++11 multithreading libraries as defined in headers such as `<thread>`, `<mutex>`, `<future>`, and `<condition_variable>` are prohibited. You must use `<pthread.h>`. Build any more complicated mechanisms you want to use by using `pthread` primitives.
- You must use the provided `hw2_output.h` and `hw2_output.c` for your notification outputs. These files will be replaced during grading, so submitting them is pointless.
- You must call `hw2_init_notifier()` as the first thing at the start of your main function to initialize the notifier with a precise starting time.
- Coordinates axis values are guaranteed to be less than 2^{16}, which means a grid dimension will be 2^{16} at most.
- Do not use `stdout` at all; it should only be used by `hw2_notify` calls. You can use `stderr` for your debugging needs, via `cerr` in C++ or `fprintf(stderr, ...)` in C. You can leave `stderr` output in your final submission with no problems.
- Do not attempt to *synthesize* fake output without creating threads. This will be checked separately. You will be wracking your brain for it and will get zero for your trouble. Focus on synchronization.
- We have a zero tolerance policy against cheating. All the code you submit must be **your own work**. Sharing code with your friends, using code from the internet or previous years' homeworks are all considered plagiarism and strictly forbidden.
- Follow the course page on ODTUClass and the homework's github repository for updates and clarifications.
- Please ask your questions on ODTUClass instead of sending an email for questions that do not contain code or solutions, so that all may benefit. Use a descriptive subject line to help your friends sort through questions.

5 Submission

Submission will be done via ODTUClass. Create a gzipped tarball file named `hw2.tar.gz` that contains all your source code files together with your Makefile. Your archive file should not contain any subfolders. Your code should compile and your executable should run with the following command sequence:

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

If there is a mistake in any of the 3 steps mentioned above, you will lose 10 points.

Late Submission: The standard penalty of $5 \cdot (\text{late days})^2$ will be applied to your final grade in case of late submissions.