2nd Project Report

Epaminondas Ioannou
Petros Tsotsi
Panagiotis Katsos

2019-2020

Introduction

This second assignment is a continuation of the first. The core of the code is the same. Some additional changes have been made in the header and source files, that were required for this project's tasks. Both of these files are analyzed below.

1 Header File

The following libraries are included in the header file:

- stdlib
- stdio
- pthread.h, so that threads can be used
- stbool, so that bool types can be used
- unistd, so that the sleep function can be used
- ctype, so that the isdigit function can be used

There are also declarations of constants and variables such as Ncook (available cooks), Noven (available ovens), Ndeliverer (available deliverers), Norderlow, Norderhigh which are the lower and upper limits for each order's pieces, Torderlow, Torderhigh which are the lower and upper limits for the amount of time it takes for a next order to take place, Tprep (time to prepare a pizza), Tbake (bake time), as well as Tlow and Thigh which are the lower and upper limits for the delivery time of the order. As this system works with synchronization, appropriate mutexes are declared such as oven_lock (for ovens), cook_lock (for cooks), deliverer_lock (for deliverers), screen_lock (to lock the screen when the output is printed), time_lock mutex (when each specific order time is being placed in a dynamic table that all threads have access to) and time2_lock mutex (when each specific cold time is being placed in a dynamic table). Three more declarations regarding pthread_cond are made which represent the conditions of the cooks, ovens and deliverers, as well as the declaration of a timespec struct variable (used in the gettime function to calculate the order completion time), the F_times pointer (dynamic table with the times of orders) and the Cold_times pointer (dynamic table with cold times). Finally the declarations of the header file are completed with the routine function declaration that each thread must implement, the declaration of isNumber function (checks if the command line arguments are numbers), the declaration of memory_check function (checks for possible memory leaks), the declaration of rc_check function (checks for pthread actions) and the declaration of a struct with id, number of pizzas and del_time (delivery

time) as parameters. We use this specific struct because we want to have access to 3 variables in each thread.

2 Main function

As for the main program, some appropriate checks have to be made initially. These checks concern the number of arguments (they should be exactly 3) and whether these arguments are positive numbers (via isNumber function). Once we have confirmed that the arguments are in the appropriate form we assign to the Ncust variable the value of the first argument (which is the number of orders), to the seed variable the value of the second argument (which is the random seed) and we dynamically assign through malloc, the appropriate space for the id table(will contain order ids), F_times table (will contain order times) and pthreads table (will contain actual threads). Then all mutexes and conditions (declared in the header file) are initialized through init. With a for loop from 0 to Ncust (number of orders) we create the threads as follows:

- We pass in the id table the id of this specific thread.
- We create a variable of pizzas_ids type which is the struct declared in the header file.
- We calculate the total pizzas this order will have using rand_r which will use the seed that has been given as an argument. Then we assign this integer value to the first parameter (number_of_pizzas) of the variable mentioned above.
- We assign the id to the second parameter.
- We calculate the delivery time that this specific order requires through rand_r. Then we assign this integer value to the third parameter (del_time).
- We create this thread through pthread_create where we pass as arguments the memory address of the thread, the routine it should follow and a pointer to the variable x.

- Once the thread is created, we wait for y amount of time through sleep function, which will be in the range $[Torderlow, Torderhigh]$, until the next order arrives.
- Then another for loop is used in which pthread_join is called to wait for each thread to finish its routine.
- Finally, we destroy all mutexes and conditions through destroy function, print on the screen the maximum and average time of the orders, the maximum and average time of the orders being cold and release the memory where needed through free function.

3 Order function

In the order function is where the multithreading takes place, ensuring that there are no overlaps in the critical areas of our program memory. As mentioned before we pass as an argument a variable of pizzas_ids type and we also use 3 local variables called id, pizzas, deliv_time to store the values of the struct of this thread. We then mark the start time of the thread by specifying it with the gettime function and by storing it in the F_times table. A lock is used for the mutex time_lock as we change memory which is accessible to all threads and thus is a critical area. Then a mutex_unlock happens. Now the thread searches for an available cook, so it locks the mutex cook_lock and checks if there are any available cooks. If not it enters in a while loop where it 'sleeps' through cond_wait until at least one cook is available. If this is not the case then a cook handles this thread (order) so the Ncook variable is reduced by 1, a cook_unlock takes place (since we are leaving the critical area) and the thread waits for pizzas*Tprep amount of time which is equal to the order's preparation time. Then we go back to a critical area, since we have to check for available ovens. Therefore, by following the same pattern, the lock for the ovens takes place (oven_lock) and if there is no oven available (i.e Noven = 0) the thread enters in a while loop and sleeps again through cond_wait until an oven is available. If an oven is available, the Noven variable is reduced by 1 and it unlocks. It is at this point that the first major change since the 1st Assignment, takes place. Cooks now become available right after placing the order in the oven and do not have to wait for it to bake. Therefore, after the ovens unlock, a cook_lock must take place so that a cook becomes available and the Ncook

variable is increased by 1. A signal is also sent to any threads that were waiting, via cond_signal. Then the cooks unlock takes place and the order starts to bake for a Tbake amount of time via sleep function. After the pizzas are baked, they start to get cold until they are delivered to the customer. To calculate the amount of time the pizzas of a specific order were cold we use a mutex_lock(time2_lock), and then save this amount to the Cold_times table with the use of gettime function. This mutex then unlocks. After the baking is over we must also check if there are any available deliverers. By following the same pattern, a deliverer_lock takes place and we check if there are any available deliverers (if not the order waits via cond_wait). If a deliverer is available the Ndeliverer variable is reduced by 1 and this specific deliverer takes the pizzas out of the oven so that this oven is released. Deliverer_lock is then unlocked. Since the oven was released a oven_lock takes place to increase the ovens by 1 and signal the orders that were waiting for an oven, via cond_signal. The deliverer now starts from the pizzeria to deliver the pizzas to the customer and via sleep function he/she requires a deliv_time amount of time which will be in the range [5,15]. Appropriate locks in mutexes time2_lock, time_lock have to be made in order the total delivery time and cold time of the order are saved to F_times, Cold_times tables. Then the output message regarding the specific thread is printed (id, delivery time, cold time). The screen_lock mutex is used. Finally, for the thread's routine to be completed the deliverer must return to the pizzeria in a deliv_time amount of time, via sleep function. Once he/she returns he/she is available to deliver a new order. For that reason, a lock in deliverer_lock mutex must take place. Ndeliverer variable is increased by 1 and a signal is sent to threads that were waiting for a deliverer, via cond_wait. The thread has now finished its routine and exits.

4 Program Restrictions

The only restriction in the program, besides the command line arguments check, is that each command such as mutex_lock, mutex_unlock, cond_wait, cond_signal etc, is checked as to whether it returns a number other than 0. If this is the case the program terminates with an error code.