1st Project Report

Epaminondas Ioannou
Petros Tsotsi
Panagiotis Katsos

2019-2020

Abstract

This assignment requires POSIX threads, which will be used in a pizza delivery system. This system was implemented in C programming language and consists of a header file which contains all necessary declarations of functions, constants and variables, and a c file. 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), 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, as well as Tprep (time to prepare a pizza), and Tbake (bake time). As this system works with synchronization, appropriate mutexes are declared such as oven_lock (for ovens), cook_lock (for cooks), screen_lock (to lock the screen when the output is printed) and time_lock mutex (when each specific order time is being placed in a dynamic table that all threads have access to). Two more declarations regarding pthread_cond are made which represent the conditions of the cooks and ovens, as well as the declaration of two timespec struct variables (used in the gettime function to calculate the order completion time) and of the F_times pointer (dynamic table with the times of orders). 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) and the declaration of a struct with id and number of pizzas as parameters. We use this specific struct because we want to have access to 2 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 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 functions, print on the screen the maximum and average time of each order and release the memory where needed through free function.

3 Order function

In the order function is where 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 2 local variables called id, pizzas to store the values of the struct of this thread. Then we 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 allo threads and thus is a ctitical 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, it unlocks and sleeps for as long as the baking requires, i.e Tbake time. After the baking is over, the order is completed and is ready for take away. Both the cook and the oven are released. For this reason we need to increase their corresponding variables and we must enter a critical area again. The oven is the first to be released with the use of its mutex lock, so the Noven variable is increased by 1, and threads that sleep in the 'oven while loop', wake up through the cond_signal and the oven_unlock takes place. This exact procedure is followed for the cooks, only that instead of Noven we have Ncook and instead of oven_lock and oven_cond we have cook_lock and cook_cond. Then we extract the completion time of the thread by completing the appropriate subtraction with the previous value of F_times table and by saving this thread time in F_times table again, with a mutex_lock in time_lock as mentioned above. At this stage our order is finally completed and all that remains is to print an output which will contain the order id, the number of pizzas and the total completion time. Mutex_lock is used again to avoid confusing the print lines. After the unlock the thread terminates via pthread_exit.

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. In this case the program terminates with an error code.