Zach Shim

CSS 430

Program 4

Due February 28th, 2021

Discussions

**Step 5: Run your program with**

./sleepingBarbers 1 chair 200 1000

where chairs should be 1 ~ 60.

Approximately how many waiting chairs would be necessary for all 200 customers to be served by 1 barber?

**Answer:**

In my program, approximately 98 chairs were necessary for all 200 customers to be served by 1 barber.

**Step 6: Run your program with**

./sleepingBarbers barbers 0 200 1000

where barbers should be 1 ~ 3.

Approximately how many barbers would be necessary for all 200 customers to be served without waiting?

**Answer:**

In my program, approximately 5 barbers were necessary for all 200 customers to be served when there were 0 waiting chairs.

**Possible Limitations**

**Clock Skews**

* The number of clock skews affected the number of chairs or barbers that were necessary to serve all customers. A clock skew is a synchronous digital circuit system in which the same sourced clock signal arrives at different components at different times. The instantaneous difference between the readings of any two clocks is called their skew.
* Since the program runs with usleep( ) (which may have some clock skews), results vary in terms of customers served during execution. Depending on when you run the program and how long usleep() takes (it is random), some customer threads may be affected and ultimately not end up getting to run because other threads have not yet been serviced.

**Scalability**

* It is important that multithreaded applications scale well, and as such, the performance of an application should increase as more threads and more processing cores are employed.
* As the number of threads increase in scale, the contention over mutex lock acquisitions which bottlenecks performance.
* If a lock is in sufficiently high demand, threads will block and wait for it at a for it at a high rate, causing many threads to be inactive when more work could be done.
* In addition, there are many wait() operations present. Each wait() causes a context switch to occur, causing the current thread to yield the CPU to another thread. Context switches require a lot of time to save execution state of registers, and then load execution state of other process. If there is high contention over a lock and a thread cannot grab hold of a mutex even after being signaled, it will only lead to more wait() operations and context switches.
* To only add onto this, atomic instructions are also expensive. These operations add overhead each time a lock is reacquired after a signal(), or when a thread enters a function.

**Fairness**

* Example: 3 barbers, 50 waiting chairs, 200 customers 1000 waiting time
* If we have a large amount of waiting chairs (i.e., 50), and all waiting chairs are full with no new incoming customers, then only one barber will be servicing all fifty waiting customers.
* The other two barbers are sleeping (inside the sleeping barbers’ queue) and can only be woken up by a new customer.
* Therefore, whatever barber has just serviced the most recent customer, that barber will be doing all the work for the next 50 customers in the waiting chairs, unless a new customer arrives in the barbershop.

**Possible Extensions**

**Client-Server**

* Ideally, this program could abstractly handle any type of data (can name barber and customer to something else, i.e., tutors and students).
* Therefore, it may be useful to implement this program in a client server architecture, where a server has a pool of barbers to serve customers. Clients will connect provide the customers and connect to the server through a socket.

**Improving Fairness**

* To solve the fairness issue previously described in the limitations section, a customer could provide some method of identifying which barber has been sleeping the longest.
* Therefore, if multiple barbers are sleeping, a new customer will know who has been sleeping the longest and will that barber wake up accordingly.
* From there, the flow of the program from there will continue as normal, where the barber will start the customer’s haircut.