382C: Multicore Computing : Assignment 1: Fall 2020

Trevor Anderson - tba428
Roberto Longo Pazos - rclp1973

Sept 20, 2020

0. Create a TACC UserID and add it to googledoc on Canvas.
   Link: https://portal.tacc.utexas.edu/

1. (a)
   - When the light is initially off, the prisoners select 1 prisoner to be the "counter"
   - When other prisoners go in, if the light is off and that prisoner has never turned it on, the prisoner turns the switch on. If the switch is already on, the prisoners do nothing
   - When the counter goes in, if the switch is on, they turn it off, and add 1 to their internal count. If the switch is off, the counter does nothing
   - When the counter finally reaches 99, it means that all 99 of the other prisoners have entered the room to turn on the switch

   (b)
   - When the initial state of the switch is unknown, the prisoners again select 1 individual to be the "counter"
   - The counter again will be the only person to turn off the switch, and will add 1 to their counter each time they do so, except for the first time, when they will not add 1
   - When the count reaches 197, the counter can declare that all prisoners have visited the room

2. a) A process in Peterson's algorithm sets the *turn* variable to itself instead of setting it to the other process.
   - Say P0 sets wantCS[0] = true, then P1 sets wantCS[1] = true
   - P0 sets turn = 0
   - P0 now evaluates (wantCS[1] turn == 1), which evaluates to false, so P0 enters the critical section
   - P1 now sets turn = 1

  - P1 now evaluates (wantCS[0] turn == 0), which evaluates to false, so P1 enters the critical section
  - Both processes are now in critical section, algorithm no longer guarantees safety

b) A process sets the *turn* variable before setting the *wantCS* variable.

  - P0 sets turn = 1
  - P1 sets turn = 0
  - P1 now sets wantCS[1] = true
  - P1 evaluates (wantCS[0] turn == 0) which evaluates to false, so it enters critical section
  - P0 now sets wantCS[0] = true
  - P0 evaluates (wantCS[1] turn == 1) which evaluates to false, so it enters critical section as well
  - Both processes are now in critical section, algorithm no longer guarantees safety

3. Modified filter algorithm:

```
int N;
int l; // l<N
int [] gate;
int [] last;
int [] number;

entry protocol for  thread i, loop over the threads

for (int i = 1; i < N; i++)
number[i]=0;
for (int k = 1; k < N; k++)
gate[i]= k;
last[k]= i;
for (int j = 0; j<N; j++)
while ((j != i && (gate[j] >= k) && (last[k] = i))
{};
number[i]++;
if (number[i] <= l) return;

exit protocol  gate[i] = 0;
               number[i] = 0;
```

4. See Attached Files

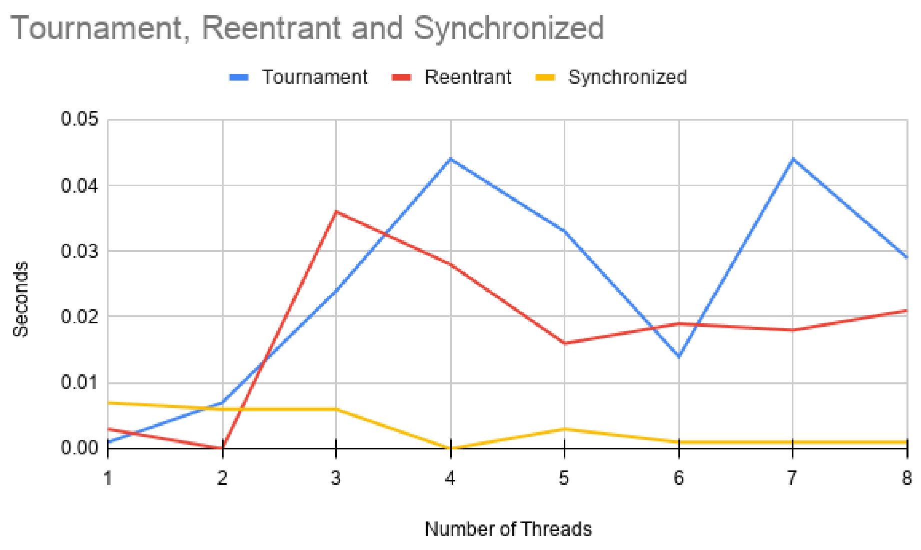

Figure 1: On my hardware, the results showed that the built in methods were the most consistent

5. Below is the plot for the tournament, reentrant lock, and synchronization methods for the incrementation.