George Costas
CS450 HW 4

The dining philosophers, is a classical synchronization problem. The issue is that there are n philosophers at a table, and there are n forks with which, the philosophers need two forks to eat. The goal is to get all of these philosophers to be full (ie. Each must each a certain number of times). Synchronization issues arise when one philosopher eats too many meals or fails to pass the forks to another, hungrier philosopher. In the problem philosophers can think, eat and/or be hungry, these items compose every state a philosopher can exist in. This problem lends itself to ending in a deadlocked state where, one or more philosophers are hungry and unable to find enough forks to eat. There were four proposed solutions to the synchronization issues that were to be implemented. These were the “no holding solution”, “footman and napkin solution”, “asymmetric handedness solution” and “Tenanbaums solution”. Here are the results testing these solutions on Alpha:

#Philos	#Meals	No-Hold	Footman	Asymmetric	Tenanbaum
2	1	3.1986s	2.8332s	1.5109s	1.4570s
2	2	3.4581s	3.8826s	5.3748s	2.9643s
2	3	5.9927s	6.3226s	4.9990s	3.6297s
2	5	9.9201s	11.110s	9.5074s	3.8438s
2	7	16.420s	13.577s	14.582s	7.8266s
2	10	20.768s	21.731s	18.178s	11.373s
3	1	2.4932s	3.1189s	2.7409s	1.0884s
3	3	6.3240s	9.0567s	7.6201s	2.8911s
3	10	23.706s	27.101s	26.122s	10.591s
4	1	1.6284s	3.3351s	2.7719s	1.6240s
4	5	12.602s	20.539s	10.227s	5.7887s
4	10	24.057s	43.519s	22.485s	12.532s
5	1	3.2630s	5.5594s	1.8831s	1.1565s
5	2	6.4336s	9.0580s	5.8158s	2.5703s
5	3	8.5689s	14.773s	7.0422s	3.9461s
5	5	12.450s	24.792s	12.823s	6.3711s
5	7	16.679s	34.210s	18.482s	10.093s
5	10	20.977s	50.958s	25.284s	10.853s
7	1	2.1076s	5.9672s	3.3940s	1.3418s
7	3	10.318s	20.431s	8.0999s	4.6636s
7	10	24.938s	68.777s	24.489s	11.391s
10	1	2.388s	9.070s	3.680s	1.515s
10	5	12.803s	47.593s	14.053s	6.304s
10	10	25.126s	101.01s	24.858s	12.079s
15	1	3.0288s	13.681s	3.9605s	1.5153s
15	3	8.8056s	44.426s	9.6818s	3.9326s
20	1	3.7691s	20.314s	3.8668s	1.6870s
20	3	8.9772s	59.976s	8.8847s	4.0977s

The no-holding solution runs reasonably quickly for being the simplest solution. In this solutions the philosopher attempts to grab the fork to it's right, and if it succeeds then the philosopher checks to see if the left fork is available. If the left fork is not available after picking up the right fork, then the philosopher drops his fork, if the left fork was available it eats. This algorithm for fork acquisition works reasonably well. The other algorithms, seems to outpace no-holds in situations with only 1 or 2 meals. Overall the no-holding solution almost always is faster than the footman solution and is occasionally faster than the asymmetric hand solution. This solution is immune to deadlock, but prevents starvation.

The footman and napkin solution uses a semaphore to control which philosopher can grab a free fork. For this reason the footman must acquire the forks before the philosophers can. This solution seems to be the worst in terms of run time. It is almost always slower than no holding and the asymmetric hand solutions. This makes sense as the footman slows down the action of all threads by making P operations after each eat is completed. In situations with many philosophers (10+) this algorithm had very significant slow down. This solution also performs signifigantly worse than all others in situations with many meals(10+).

In the asymmetric hand solution, at least 1 philosopher will be right handed and at least on will be left-handed. This implies that they pick up the fork to their handed side prior to the other fork. This organizes the order in which forks are placed and grabbed so that they continue inline until the opposite handed philosopher is reached. This algorithm performs fairly well. It is only consistently out paced by Tenanbaums solution. It is resistant to the influence of a large number of philosophers and instead is affected by number of meals. This solution is immune to deadlock

Tenanbaums solution uses a test to see what state the philosopher is in. philosophers can only be hungry, eating or thinking. All philosophers begin thinking and then one is tested to see if he can begin to eat, so this philosopher becomes hungry. The test dictates that if he is hungry (which is now true) , and his neighbors are not eating, then he can eat. When a philosopher finishes it tests each of its neighbors to see if they are hungry and able to eat. This is by far the fastest solution in the group tested. It is obviously the most regimented and although introducing the test introduces more operations into run time, it ensures that time is not wasted by philosophers waiting to eat, when both forks are available. This solution is immune to deadlock, but not starvation.

On average Tenanbaums solution is by far the fastest, usually finishing twice as quickly as all other algorithms. The no hold solution and the asymmetric hand solution are almost always within 1 or 2 seconds of each other. The asymmetric solution performs better than the no holding solution in situations with fewer philosophers and fewer meals, while no holding is about equal in other situations. The footman and napkin solution is almost always the slowest due to its single footman controlling the pace. All other solutions are resistant to drastic changes in speed times based on large number of meals or philsophers except no holding.

I did not implement random.seed so, my results are more difficult to interpret with a low number of meals. This made me do more testing with a large number of meals to see if better patterns could be found between each group. I also did a large number of tests using low meal numbers (1-3) , but these tests simply proved my suspicion that the lack using random.seed was confusing my results with low meals. Rather than using seed.random (which I had trouble implementing on time) I used random.uniform(0,1) throughout the program. This introduced +/_ intervals of time that made the asymmetric and no holding solutions difficult to distinguish even with testing. Overall when looking at the variance in range we can see that the range can vary greatly in between algorithims, since little timing error is introduced by uniform.random (0-1) we can assume this is due to threading and hardware

Here are the results of my retesting with greater meals:

#Philos	#Meals	NoHold	Footman	Asymmetric	Tenanbaum
2	5	12.0490s	12.1450s	10.5234s	5.2314s
2	5	12.3364s	9.7798s	12.7462s	4.9600s
2	5	10.7897s	10.0345s	10.3695s	6.9038s
2	5	9.2833s	11.6580s	11.1066s	6.3700s
2	10	21.3139s	21.3576s	19.2376s	11.0489s
2	10	24.1298s	21.7299s	22.6999s	11.2957s
2	10	21.0300s	21.1774s	23.6067s	9.3637s
2	10	21.7470s	21.3782s	19.4453s	10.5415s
2	20	42.4391s	44.1190s	49.1630s	21.2284s
5	10	23.2711s	51.8196s	24.1812s	11.1773s
5	10	23.1826s	49.7133s	24.2011s	10.6876s
5	10	25.0643s	47.4123s	26.2830s	11.7856s
5	10	24.3887s	50.5698s	26.3472s	11.8953s
5	15	36.0205s	77.2040s	37.6889s	17.1460s
5	15	33.1278s	74.8683s	37.2609s	18.3169s
5	15	38.5909s	77.8172s	42.2078s	18.5909s
5	20	49.0149s	106.5287s	47.1307s	20.0103s

Nohold: 2,5 Rng:3.06s 2,10 Rng: 3.1s 5,10 Rng:1.88s 5,15 Rng:5.46s
Footman: 2,5 Rng:2.36s 2,10 Rng:3.1s 5,10 Rng:4.4s 5,15 Rng:2.95s
Asymmetric: 2,5 Rng:2.38s 2,10 Rng:4.36s 5,10 Rng:2.16s 5,15 Rng:4.94s
Teneanbaum: 2,5 Rng:1.94s 2,10 Rng:1.93s 5,10 Rng:1.20s 5,15 Rng:1.44s