Dining Philosophers Program

# Problem Statement

The assigned task was to create a solution to the dining philosopher problem based off figure 6.12 which has five philosophers sitting around a table with forks between them all and having to take turns eating based on if other philosophers are eating. Incorporation of the use of semaphores to control access to resources in this case to avoid starvation and deadlocks. Used pthreads from the previous programs to create the five philosophers. The program should run and print out philosophers who are eating and when which is indicated by when they are hungry, in addition the forks they use are numbered to show when they are finished and transition to thinking when it prints out forks have been put down. Think and eat functions have random sleep durations and the program ends after 5 minutes when all threads are killed.

# Approach

The decision was made to run C code in the cslinux1.utdallas.edu remote server using an SSH connection as this was the standard used for previous assignments. Tools used for connection is PuTTY, and the FTP used to transfer the code was WinSCP. Editor used for the code was Notepad++. Threads were created for each of the five philosophers. Sem\_wait is used instead of suggested waits to avoid starvation and deadlock since it is more efficient in this case when we are trying to lock, if other methods were used for locking it won’t wait until the lock becomes available which causes philosophers in program to start thinking after they are hungry when they haven’t eaten yet. Sem\_post unlocks the semaphore after if is locked when it is not waiting. Philosophers default to thinking and then hungry then eating, random sleep time after eating or thinking before next transition occurs. Pthreads cancel after exactly five minutes.

# Solution

The code has defined global variables that can be changed at any time and the code should still run. The global variables are: For the number of philosophers, the number defining the state of philosophers whether they are hungry, waiting, or eating, and left and right values for forks and philosophers next to another. Data structure used to store states is an array. Semaphore use is a way to allow access to the spaghetti or eating state when the forks are available or when other philosophers are not in their critical sections. Sem\_wait was used instead of sem\_trywait because trywait does the same thing basically except it block others from releasing the semaphore and the operation will fail unlike in sem\_wait, so using sem\_trywait causes philosophers to sate hunger when they haven’t even eaten. Sem\_timedwait also does not work as well as sem\_wait since sleep times are randomized it is not as efficient, hence why they submitted code will have sem\_wait but it can be replaced by either of the two semaphore lock methods, they are just not as efficient or reliable as sem\_wait.

Figure 1 below shows the code running with all the philosophers starting off thinking and then based off random certain philosophers or possibly all in the case of figure 1 become hungry and attempts to eat. Figure 2 shows the possibility of not all five becoming hungry based off a random sleep time within the range from zero to five which makes the philosophers more unpredictable. Figure 1 shows that based off the random sleep time philosophers who are hungry are randomly picking up the forks that are beside them at random time when they pick up the forks this signifies the beginning of them eating which shows when the philosopher begins to eat and putting down the forks shows when the philosophers finish eating. The unpredictability of each philosopher eating is also shown in figure 1 since they do not finish eating all the time before another philosopher eats, this is works since the philosophers simultaneously eating are not using the same forks which shows that deadlock and starvation is being avoided. Not shown in the figures is the program ending after exactly five minutes, but it has been tested and it ends exactly after running for five minutes using sleep function for 300 seconds using pthread\_cancel.

Figure 1

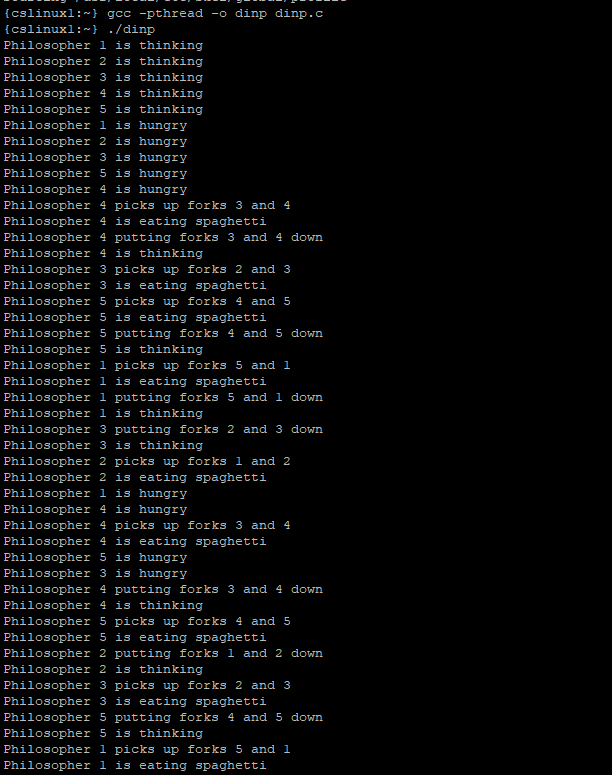


Figure 2

