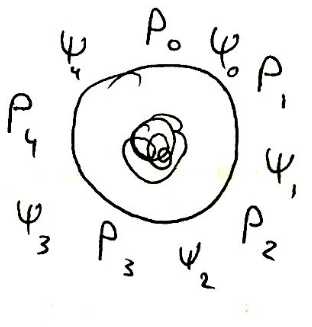
Dining Philosophers

```
Semaphore forks[] = {1, 1, 1, 1, 1}      <- shorthand init.
                                          ^ each 1 represents availability of fork
Semaphore numAllowedToEat (4)
```



```
void Philosopher (int id)
{
    for (int i = 0; i < 3; i++) {        <- # think, eat cycles
        Think();
        SemaphoreWait (numAllowedToEat);
        SemaphoreWait (forks[id]);                 <- if all 5 threads are
        SemaphoreWait (forks[(id+1) % 5]);            removed from process
        Eat()                                          at this point
        SemaphoreSignal (forks[id]);                   ↳ deadlock
        SemaphoreSignal (forks[(id+1) % 5]);
        SemaphoreSignal (numAllowedToEat);
    }
    Think();
}
```

if only 4 threads are allowed, at least one thread is guaranteed to run at any time
$\Rightarrow$ no deadlock

Also try use minimal amount of work to prevent deadlock to grant thread manager maximal flexibility.

File download Example

```
int DownloadSingleFile (const char * server,
                        const char * path);
# bytes
   ↓

int DownloadAllFiles (const char * server,
                      const char * files[],
                      int n)
{ Semaphore ChildrenDone = 0;
  int totalBytes = 0;
  Semaphore lock = 1;
  for (int i=0; i<n; i++) {
     ThreadNew("…", DownloadHelper, 5, server, files[i], & [redacted] totalBytes,
                                                          lock, [redacted] ChildrenDone);
  }
  for (int i=0; i<n; i++){SemaphoreWait (ChildrenDone);}
  return totalBytes;       ← wait untill all threads
}                            exit before returning

                                                    1-to-N Rendezvous
                                                    a thread with
                                                    n children threads
                                                    [redacted]
void DownloadHelper (const char * server,
                     const char* path,
                     int *numBytesp,
                     Semaphore lock, [redacted] Semaphore parentToSignal){
   int BytesDownloaded = DownloadSingleFile (server, path);
   SemaphoreWait (lock);
   (*numBytesp) += BytesDownloaded;   ) critical region
   SemaphoreSignal (lock);
   SemaphoreSignal (parentToSignal);
}
```

when
[redacted] SemaphoreWait (ChildrenDone) has completed
n 1→0 decrements, all n threads have been
completed until and including
SemaphoreSignal (parentToSignal).

The DownloadAllFiles thread may return [redacted] before
} in all [redacted] DownloadHelper threads is completed.

CS 107 Lecture 17

Ice Cream Store Simulation

$$\begin{tikzcd}
\text{Manager} \arrow[d, leftrightarrow] \arrow[d, leftrightarrow, bend left] \\
\text{10-40 Clerks} \arrow[d, leftrightarrow] \arrow[d, leftrightarrow, bend left] \\
\text{10 Customers [1-4 ice cream cone order]} \arrow[d, leftrightarrow] \\
\text{Cashier}
\end{tikzcd}$$