STARVATION

Starvation is the name given to the indefinite postponement of a process because it requires some resource before it can run, but the resource, though available for allocation, is never allocated to this process. It is sometimes called *livelock*, though sometimes that name might be reserved for cases where the waiting process is doing something, but nothing useful, as in a spin lock. However it happens, starvation is self-evidently a bad thing; more formally, it's bad because we don't want a non-functional system. If starvation is possible in a system, then a process which enters the system can be held up for an arbitrary length of time.

To avoid starvation, it is often said that we want the system's resources to be shared "fairly". This is an appealing suggestion, but in practice it isn't much use because it doesn't define what we mean by "fairly", so it's really more of a description of the problem than a contribution to its solution. It's more constructive to investigate the paths by which a system can reach a livelocked state, and try to control them.

CAUSES OF STARVATION.

Starvation is caused by failure to allocate some resource to a process, so to find the causes we must inspect the policies which the system uses in handling resources. Here are some possibilities.

- Processes hand on resources to other processes without control. If decisions about resource allocation are taken locally without considering the overall resource requirements of the system, anomalies can occur. If processes queue for a resource, and the resource is always handed on to the next process in the queue, it is essential that *every* process awaiting the resource must be placed in the queue.

- Processes' priorities are strictly enforced. If a process of worse priority requires a resource in competition with a constant stream of processes of better priority, it might wait for ever.

- "Random" selection is used. If processes awaiting service are not queued, but an arbitrary process is selected whenever the resource becomes available, it is possible for some processes to wait for a very long time. In some circumstances, that doesn't matter too much – for example, if it is known that the average demand for resources is far less than the resource pool available, the congestion is almost certain to last for very short periods. The trouble with this strategy isn't usually that processes wait for ever (unless your "random" number generator isn't), but that you just can't tell what will happen. A good example is the Ethernet communications technique, where processors use a common communications medium without overall synchronisation, and resolve attempts to transmit simultaneously by delaying for an arbitrary interval and then trying again. Once the medium becomes moderately heavily used, quite long delays can be experienced.

- Not enough resources. This is commonly the real problem, so far as physical resources are concerned, though as its solution costs money it might be a hard one to solve. Provided that the supply of resource exceeds demand over a reasonable period of time, it should be possible to satisfy the demand, and strategies can be chosen to provide service to all processes which need it. On the other hand, if demand exceeds supply, no amount of ingenious trickery can serve everything, and under these conditions starvation can often occur.

Starvation can happen at any organised scheduling level, though it is more likely in the automatic allocation processes than in the higher-level manual parts. (We assume that there is more intelligence and flexibility at the manual level; bureaucracy can defeat this assumption.)

REMEDIES.

Cures for starvation are in general based on means of ensuring that the conditions for starvation can't happen. Here is a selection.

- There must be an independent manager for each resource, which must manage all allocations of its resource; this will guarantee that processes don't just pass resources around between themselves without making them available for general allocation.

- Strict priorities should not be enforced. A poor priority should be regarded as a weak claim, but not an overridable claim. There are at least two ways to achieve this end :

  - Improve the priority of a waiting process with time; then even a process with poor priority which has waited for a long time will eventually be able to compete successfully with a newly-arrived process of much better basic priority. Of course, the improved priority doesn't last – after allocation, the process's priorities revert to their original levels.
  - Implement priorities by rationing; regard the priorities not as indications of absolute importance, but as measures of the proportion of the resources which can be consumed.

- Avoid random selections, uncontrolled competition, etc. It is very unusual for random resource allocation techniques to have any intrinsic merit. After all, if a random technique will work, it doesn't matter which process receives the resource, so you might as well queue them and give the resource to the process at the head of the queue. If anything, the queue overhead is less, and you win by the closer approximation to a functional system.

- Provide more resources. This is the only satisfactory solution to continued congestion when demand approaches supply. If the supply and demand are closely balanced, even small fluctuations of demand can cause queues to form which can take a very long time to eliminate. Of course, if you have sufficient resource to reduce queueing to an acceptable minimum with any normal demand, you will have periods when the resource is standing idle.

> *That's why the hospitals have queues. If you have enough resource to handle an epidemic, then most of it is idle for most of the time, and you are accused of mismanagement; if you try to manage with the minimum possible resource and keep it all in use, then long queues are inevitable for most of the time, and you are accused of mismanagement. But try explaining that to a politician. The political solution currently practised is to tell some of the processes that they don't need the resource, or that they must wait until they need far more resource, or that they must get it from some other operating system.*

Whether you can enforce the remedies through operating system constructs alone is not very clear. The system must be designed so that the details of scheduling policy can be set to match local priorities, which might not be determinable from considerations of processing performance alone. This flexibility can lead to unsatisfactory scheduling arrangements. For example, if the algorithm for selecting among the dispatcher queues can be arbitrarily changed, starvation is easy to arrange : just serve the queues of better priority first unconditionally.