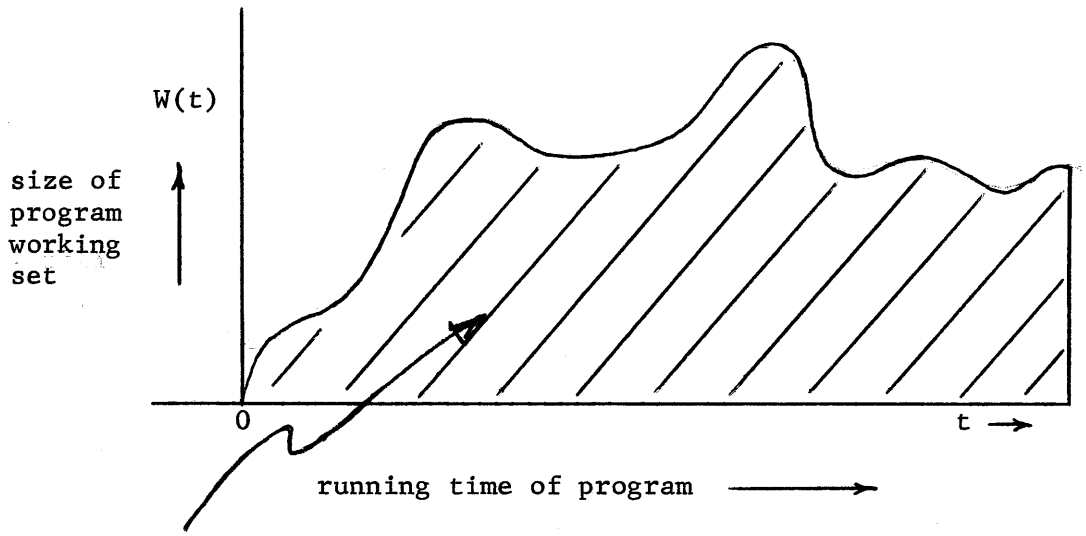
DRAFT
2/14/73

A Brief Description of the Multics memory-charging algorithm

by J. H. Saltzer

Problem: to estimate accurately, the size of the user's working set as his program runs, and charge him for the area under the curve.



Area = a suitable memory charge, in page-seconds.

$$U = \int_{\substack{\text{running}\\ \text{time of}\\ \text{program}}} W(t)dt$$ measured in page-seconds

---

We arbitrarily define the working set size W, to be the amount of real memory which results in a paging rate of f_o page faults per second, where f_o is the paging rate at which the system operates most efficiently. For example, f_o might be 20 p/s.

The linear model predicts a relation between the paging rate, f_m, and the actual memory size used, M.

$f_m \times M = \text{constant}$ call the constant C (it will drop out anyway)

In other words, the bigger the memory, the smaller the paging rate. Then the working set size is related to f_o by

$$f_o \times W = C$$

and the paging rate versus memory size relation may be written

$$f_o \times W = f_m \times M$$

and we discover a method of estimating W, the working set size:

$$W = \frac{f_m}{f_o} \times M$$

All we need do is estimate the actual memory size, M, and measure the actual paging rate, f_m. If the observed paging rate is 40 page faults/second and M is 50 pages, W is

$$W = \frac{40}{20} .50 = 100 \text{ pages}$$

The method of estimating U is then: at every page fault estimate W, and add $\Delta U = W \times$ (time since last page fault) to a location collecting U. that is,

$$U \leftarrow U + \Delta U$$

where

$$\Delta U = \frac{f_m}{f_o} \times M \times \text{(time since last page fault)}$$

We can take one more step to simplify the computation which will be required. We notice that the page fault rate, f_m, is exactly

$$f_m = \frac{1}{\text{time since last page fault}}$$

so

$$\Delta U = \frac{M}{f_o}$$

Finally, since f_o is constant, we may simplify the estimating procedure further by calculating, instead of U, a charge of $U' = U \times f_o$ which is exactly proportional to U. We then have

$$\Delta U' = M$$

So our entire strategy of measuring memory usage reduces to estimating the amount of real memory in use whenever a page fault occurs, and adding that number to the memory usage accounting meter. (And remembering to divide the value in that meter by f_o if we wish to normalize it to a page-second measure.)

It remains to estimate M, the actual size of the memory being used to contain the program. Consider a program which has just started running in an N page memory all by itself, but has only one page in core at the moment. We can say that its size, M, is one page, at the time it takes its first page fault. That page fault causes a second page to be assigned, so its size is now 2 pages, and this new size holds up until the time it takes its 2nd page fault. Continuing this line of reasoning, we have a size estimate

$$M = \# \text{ of page faults taken so far.}$$

This line of reasoning holds up to the point at which the real memory is filled up, after which point the size N is a better estimate of M. Thus

$$M = \min (\# \text{ of page faults taken so far, } N)$$

The final step is an arbitrary one. In Multics, the real memory is shared by some number of processes being multiprogrammed. We assume arbitrarily that the real memory is equally divided among the multiprogrammed processes. Thus, if there are p processes being multiprogrammed in a memory with N unwired pages, we estimate

$$M = \min (\# \text{ of page faults since loading, } N/p)$$

The resulting memory usage value, U', may be interpreted as the memory, in page seconds, that the process would have used in a system maintaining a paging rate of 1 page/second.

For convenience in calculation and discussion, the actual implementation reports $U'/1000$, since the numbers then resulting are typically comparable in value to the number of CPU seconds used. Thus usage is being measured in kilo-page seconds.