M0066

JHS — yr comments noted. you may have this back, for now.

RESOURCE MANAGEMENT AND ACCOUNTING FOR MULTICS

T. Van Vleck

March 23, 1966

Resource Management and Accounting

I. Motivation and design goals

II. Definitions

A. Person

B. Account

C. Process

D. User

E. Metering Unit

F. Cost

III. Plan of attack

A. The accounting procedure and data segment

1. collection and meter-reading

2. conversion

3. thresholds

4. unique interface

B. Metering

1. requirements

2. quantities to be metered

3. calls to CHARGE

4. calls from outside the process

C. Conversion

D. Thresholds

IV. Accounting Structure

- A. Account files
  - 1. location
  - 2. contents
  - 3. control
- B. Creating a process
  - 1. same data segment
  - 2. withdrawal
- C. Destroying a process

V. Other Thoughts

- A. Books must balance
- B. Protection
- C. Daemon processes
- D. Reliability in event of crash
- E. Utility daemons
- F. Extra accounting-procedure entries
- G. Rate control

VII. Pricing Policies

- A. Overview
- B. Policy objectives
- C. Saturation
- D. Side effects
- E. Example - the on-resource system

RESOURCE MANAGEMENT AND ACCOUNTING FOR MULTICS

If Multics is to become a utility system capable of meeting computation needs of business as well as universities, service centers as well as in-house operations, it must posses an extensive resource-management facility which includes capabilities for charging for resources expended, regulating the use of resources, and evaluating the demands made on the resources available. Such a facility must be precise enough to satisfy the strictest auditor; must be clearly and flexibly designed so that different installations needs and policies may be accomodated; must be tamper proof, as far as possible; and must be reliable in the event of trouble elsewhere in the system.

As explained in this paper on System Metering, the ~~system~~ scheme for measuring use of the Multics sytem computes the consumption of system resources in arbitrary units on a per-process basis. This paper concerns itself with the problems encountered in accumulating these units and translating them from multiple usage measures attributed to a single process into a single cost attributed to an account.

Definitions of Terms

A person is just that, a human being, who may sit at a terminal and communicate with the system; he possesses a name, a social security number, and a thumbprint, etc.

An account is an administrative entity which receives bills for system resources expended.

A process is the usual Multics concept, with a descriptor segment and so forth. . Every process is associated with some account, and resources used by a process cause charges to be made to the process's account.

A user consists of one or more processes, all of which have certain data segments (such as the user profile) in common. A process may be under the direction of a person. The identification of a user consists of a project number and a person's or daemon's identification.

A metering unit is a number representing the quantity of a resource which has been expended by a process in some period. The dimensions of a metering unit reflect the kind of resource being metered and the way in which it is measured. For example, secondary storage residence might be measured in "block-seconds".

A cost is a number expressing, in administration-oriented units, the usage of the computer system attributed to an account. Cost is in identical units for all resources; dollars (or "micro-pennies") is an example of a unit of cost.

Plan of Attack

Within Multics, each process has an accounting procedure and an accounting data segment, whose responsibilities include

1. collection of metering units which represent the resources used.by the process.

2. conversion of these metering units into const units, and the updating of the balance maintained in the accounting data segment.

The accounting procedure maintains a balance of cost units which the process may expend. This balance is a single number, not a vector. Each time that the accounting procedure is informed that the process has used some resources, it computes a total cost as a function of the metering units, and decreases the balance by that amount.

The accounting procedure will be the only interface to the accounting data segment. Thus, all questions of format and structure of the data segment are localized, and, indeed, changes in format require no expensive reformatting.

Associating Use With Accounts

The first step in resource-accounting consists in metering the resource usages of every process. The paper on metering covers the plan of attack in detail; from the point of view of this paper, the method used does not matter so long as the metering is one-one: that is,

1. every use of a resource must be associated with some account.

2. all resource use charged to an account must represent resource use by a process "belonging to" that account.

Note that dummy accounts are not ruled out, and may be even required by this scheme to absorb otherwise unchargeable use. Also, cost units may be transferred from one account to another.

The initial design for the resource-usage meters distinguishes 6 quantities to be measured:

a subroutine, and that shift differentials or non-linear cost functions can be implemented easily.

If the accounting procedure sees the balance drop below a preset threshold, it will call an "out-of-funds" procedure. The action to be taken in such a case will depend on circumstances and on policy. Some processes may be allowed to continue working, while others should be stopped. Certain installations may wish, in the case in which a user has no more funds for any processes, to log the user out. Others may wish to allow him to negotiate - or to expunge him completely from the system.

Several thresholds may actually be maintained at once, some set by "higher authority" and others established by the user himself. These may be in the form of additional actions to be taken when either the balance or the usage of a single resource passes a certain point. Thus an administrator may say, "don't let user x spend more than half his balance on secondary storage," or the user may say, "tell me if I spend more than 18 units an hour on processor 2 for my process Z."

<u>The Accounting Structure</u>

Each accounting data segment contains a pointer which specifies the account associated with the process. Accounts themselves are represented in the machine as files within a special directory, the <u>accounting directory</u>. <u>Account files</u> contain the following items:

1. Information as to the "owner" of the account, where to send the bill, etc.

2. The total amount of cost units which the account may spend in a period.

3. The amount unspent.

4. A list of users authorized to charge this account, and, for each user,

   a) history information, showing how much this user has spent.

   b) control information, describing how the user may spend.

The administrator for an account will be able to delegate his authority to control ~~who charges to~~ the account to several sub-administrators, each of whom may control the resources of some group of users. In such a case, there is an entry or entries which specifies a subsidiary accounts, instead of a users, in (4) above.

When a process is created, there are two possibilities: either the new process can share ~~contain~~ the same accounting data segment with ~~as that of~~ the creating process, and thus charge its use to the same account automatically; or a new accounting data segment can be created. If the second option is taken, we have a situation like that found when a user ~~first~~ logs in. An account number must be specified, in this case, so that an initial balance can be obtained. The given account identifier is used to find the account file, and the file is searched for the user's entry. Then subject to the limitations contained in the account file, a <u>withdrawal</u> of a certain number of cost units is made: the amount unspent in the account file is decreased, and the data segment for the process being created has that amount established as its initial balance.

Whenever the user destroys one of his processes, or at any other time he chooses, the accounting data segment for a process may have its balance set to zero, and the unspent cost units will be returned to the account file from which they were withdrawn. Such a procedure may be used to switch the charges being incurred by a process from one account to another.

Other Thoughts

All of the system bookkeeping will be multiple-entry, so that at any time, all of the use of the system can be accounted for. It is intended that all resources will be completely charged, whether to a user or to some idle or shared-overhead account, in order that the books will always balance. The cost incurred by shared-overhead accounts may be transferred to other accounts, if the installation chooses.

Careful checks must be designed so that no user may meddle with the data in the accounting data segments without special privilege.

A daemon process, which is doing work for all users will have a move extensive accounting procedure, perhaps able to reflect charges set to it to the account for which it is working. These procedures will keep a complete internal set of records, which must balance. In this way, users can be charged for the secondary storage backup necessary to keep their files safe, and so forth.

Reliability of the accounting system is helped by the "multiple-entry" philosophy on charging, and by a design which attempts to limit the amount of information kept in core to an amount we are prepared to lose if core is

A User, From the Point of View of the Accounting System

User Profile

Shared Data Segment

Shared Data Segment

Process A

Process B

Process C

Process D

Accounting data segment 1

Account data segment 2

Accounting data segment 3

Account File

User Entry

Account File

User Entry

Terminal

Person

cleared or scrambled. We must have some way of forcing important information onto secondary storage at intervals.

In the event of a system crash, there will be tools available for the administrator to "pay users back" for lost work.

A number of utility processes will be necessary to produce a coherent resource-management system. Some of these will be daemon processes which remain blocked for long periods, and then perform various updating operations for every user. Automatic billing and other end-of-period operations are examples of this, as are statistics - gathering functions of various types. Administrative utility functions must also be provided, such as creation of accounts, repairs to and searches of the "accounting tree," and so forth.

An auditor process will be one such daemon. It will have the responsibility of checking the system books at intervals to make sure that they balance, and to detect mistakes in or tampering with the accounting system. This process will remain blocked for random lengths of time with mean, say, 2 days, and then scan the whole system's status.

An entry to the accounting procedure must be available so that the process may request status information, such as how much time has been used, what the accounting data segment's balance is, and so on.

Control of the rate at which a process expends resources can be accomplished by setting a very low balance in the data segment, and requesting that a special procedure be called when the process is out of funds. The

special procedure would examine the clock, and withdraw more funds if the rate was low enough - otherwise, it might print an on-line comments.

Pricing Policies

As we have noticed in passing, administrative policy touches many phases of the accounting system on the programming level. It is hoped that the design of the resource-management sytem so far set forth is sufficiently flexible to enable the system administrators to make whatever policy decisions they choose without being restricted by the implementation.

In considering pricing policies, we must first recognize that the objectives of a policy may differ from situation to situation. One objective might be to distribute the cost of the system equitably among those who derive benefit from it. Another objective, which may conflict with the first, might be to use the prices to encourage maximum system effectiveness or perhaps volume efficiency. A third, conflicting objective might be to insure that the same "job" run twice will incur the same cost under the same conditions.

Our next observation is that a policy objective will dictate different strategies depending on the degree of demand. If demand exceeds the supply of computation resources, the simplest way to satisfy the objective "pay for the machine" is to charge each account a fraction of the total cost equal to the fraction of total resource use charged to the account. But it only a few people use the system, and they use only a fraction of the available resources, this strategy results in prices so high that users drop out rather than pay - and the policy defeats itself.

A third observation is that any policy will tend to encourage certain styles of programming and discourage others. Every programmer is familiar w ith the trade-offs possible between program size and program speed: many programs were reworked when they were transferred from batch to time-sharing, because of the several advantages of reduced size in the CTSS environment.

A comprehensive charging policy must, therefore, have well-defined objectives, must take into account the expected demand for each resource, and must consider the kind and quantity of resource use which the policy encourages. Several external factors, such as the amount a user is willing to pay for a computation, and the extent to which the user will be directed by prices, must be considered in the design of any policy.

A Policy Example

An example of a policy which seems workable follows. For simplicity, it assumes a system with only one resource~~, but the extension to many resources is straightforward~~.

The objective of this policy is to distribute the cost of the system equitably among all users, but to protect the user from rising costs in the event of ~~user~~ under-use of the resource.

Let us assume that in the period under consideration the system has a total capability of C units. (For example, 60 * 150 minutes of processor time in one week). Let the resource cost K dollars for this period, and suppose that in the period N users use $u_1 \ldots u_N$ units, totally U units of resource expended per period. Note that U is less than or equal to C.

~~or equal to C.~~

The simplest method, obviously, would be to charge user i

$$K * \frac{ui}{\mathcal{U}} \quad \text{dollars per month}$$

so that he pays for the fraction of the total use which he caused. This is the situation we discussed above, which has the drawback that if the total use $\mathcal{U}$ is much less than the capacity C we are paying for, users' costs become astronomically large. A worse drawback is that the scheme we have suggested for accounting so far requires that $\mathcal{U}$ be known in advance, so that we know the price of a resource as it is used. This will be true only if we know that the system is saturated, so that $\mathcal{U}$=C. Then if we set

$$K' = K/C$$

so that K' is the price per unit capability, we have

$$\text{cost to user } \underline{i} = K' \; u_i \text{ dollars.}$$

D ≤ C at all times, no need for min ( )

w.h. "overcapacity" may mean better service.

For an unsaturated system the problem is more difficult. If the installation has bought too much capacity, it seems unreasonable to attempt to make users pay for this mistake. Suppose that the administrators can estimate the demand which will be made on the resources of the system as some figure D. Then it seems reasonable to set

$$\text{charge to user } \underline{i} = K' \left(\frac{\min(D,C)}{C}\right) u_i \text{ dollars.}$$

You have assumed $K' = \frac{K}{D}$?

If our demand estimates are accurate, then we will recover either enough to pay for the machine, when the system is saturated, or else a fraction

Why not $\frac{K}{C} u_i$?

I don't understand this argument.

U/C of this, when we cannot sell all the machine.

In order to show where additional revenue may arise, and to bring the scheme closer to reality, let us introduce a flat-rate <u>Base charge</u>, B, which represents fixed charges such as "cost of keeping an account on the books," and a <u>Surcharge factor</u> S greater than or equal to 1. Both of these will be applied to all accounts. Our formula becomes

$$\text{cost to user } \underline{i} = K' (B + SRu_i) \text{ dollars}$$

where

$$R = \max\left(\frac{\min(C-B, D)}{C}, 0\right)$$

B will in general be small; it represents the amount of our cost which is paid for by subscription.

R represents the surcharge needed to pay for the entire resource if the expected demand, D, is realized.

With this scheme we may still make a profit or lose money, but what to do with this amount becomes an accounting problem. Losses represent capacity we were unable to sell, and profits represent extra capacity sold beyond projection, and should perhaps be distributed at the end of the accounting period, or used to buy additional capacity.

n.b. "R" can be computed at the end of the accounting period to make the profit = 0.

Can also be used to lower K for the next month.

Additional components may be added, in the form of extra charges for future purchase, or punitive charges to enforce a pattern of use (e.g. shifts). Further complication may arise through round-off problems if there is a mismatch between the units we measure and the smallest amount we are willing to bill. These problems, and the estimation of the demand and base charge parameters, are administrative decisions, which must be made [after we have some idea of system capacity.] ?