To: Multics Administrative Distribution

From: T. H. Van Vleck

Date: September 13, 1971

Subject: Absentee prices and priorities

Several of us discussed this proposal at a system-administration meeting some time ago, and I promised I'd write it up. The proposal aims at making "what the user buys" for his absentee charges more straightforward and at cleaning up some ungeneralizable code. A specific strategy is also proposed.

We have recognized that judicious use of a "timax" on long-running programs like the backup daemon can improve system operation and allow these programs to keep going even during the day. It has been suggested that absentee be given a larger timax; the most general solution is to provide a table which specifies the timax for each queue level.

A recent change moved the per-shift prices from constants in the billing programs out into an external segment (installation-parms). The absentee prices are now half and half, since the current programs assume that there are exactly three queues and that the prices table looks like this:

	CPU	plus
queue 1	1.5 * current shift	$1/hr
queue 2	1.0 * current shift	$1/hr
queue 3	shift 3 price	$1/hr

Notice that the numbers 1.5 and 1.0 are program constants, as is the

subscript 3 for queue 3. A most general solution would be to make the rate be

$$a(i) * \text{current shift} + b(i)$$

per CPU hour, and similarly for connect ime, where the a's and b's are tabular constants, indexed by queue. Sticking in zeroes in appropriate slots can get us our current rates. (The b(i) term could be eliminated -- currently, there is no difference in price between queues 2 and 3 when the shift is 3, but there is a scheduling difference. However, it is simple to leave in and implement, and some installation may want it, so I think we should have the code.)

How many queues should we have? We have three price queues now, with the "queue 4" compilations crocked to be treated as queue 3. I'm sure everybody can concoct a class of service which isn't covered and needs another. But every queue we add requires more storage in the per-user usage-recording data bases, more columns on the bill, and so on. My vote is for four. Eight isn't unreasonable. More than eight is.

Some of the absentee scheduling parameters should also be made data instead of program constants. The following occur to me:

1. Maximum CPU time, indexed by queue.

2. Minimum underload (i.e. number of "free slots" on the system) if a job is to be scheduled, indexed by queue and by shift. This allows one to shut off queue 3, say, during prime shift.

3. Maximum number of absentee users, indexed by shift. This should be expressed as a percentage of max units to allow for

different configurations.

4. Timax, indexed by queue and shift.

All of the above sound extremely elaborate, although clearly not difficult to implement. Suggestions as to which areas of generality are superfluous would be welcome.

A specific proposal for MIT's prices follows.

queue	CPU	Connect	Timax	Max CPU
1	1.2 * current shift	1.0 * current	16	5 min
2	1.0 * current shift	1.0 *current	8	15 min
3	.8 * current shift	0	8	30 min

The maximum number of absentees table would look like

shift	max
1	1/40 = .025 = 1.5/60
2	2/40 = .05 = 3/60
3	4/40 = .10 = 6/60
4	3/40 = .075 = 4.5/60

The minimum-free-slot table would be

shift	queue 1	2	3
1	.10	.10	1.0
2	.10	.10	.10
3	.10	.10	.10
4	.10	.10	.10

Comments

1. This provides a price break to queue 3 users on shift 3.

2. The free-slot table shuts off queue 3 on prime shift.

3. Super-long jobs must run on queue 3. They are less likely to crash there, and they get a break in that they don't pay a real-time charge.

4. Queue 1 can be used for compilations during the day. It's more expensive than just leaving your console, but it runs better and can restart after a crash.