TO: J. SALTZER
C. MARCEAU
K. MARTIN

FROM: P. BELMONT

DATE: 1/5/68

SUBJECT: HOLD, UNHOLD; COMPUTATIONS and COMPUTATION STATES.

The attached major revision of User Control concepts is intended to bring greater clarity to the valuable concept of "computation" (see BQ.2.00, p3, 11/3/67).

I think that acceptance of these ideas and an appreciation of the state-transitional nature of quit, start, hold, etc., will lead to the use of the concepts (if not the terms)

Computational

and Meta-computational

to refer to the two rather different activities of the user as he communicates with the system from his console and quit button. We will, I think say that the work which the Listener does before passing to the Shell* is Meta-computational whereas things done within the Shell are Computational.

* Reset, Hold, Logout, Start, Unhold, etc.

Jerry Saltzer

IDENTIFICATION

Hold, Unhold
P.A. Belmont

PURPOSE

The MULTICS user may wish to quit his work, and later return to the execution of it, having in the meantime used the quit button freely.

Explicitly by use of the "HOLD" command or implicitly by use of (one of) the debug commands, he may cause his quit computation (working processes and I/O status) to be preserved, safe from subsequent quits, resets, or starts. When he is ready to continue execution of his held computation, he may, explicitly through use of the "UNHOLD" command or by use of the debug procedures, cause the held computation to be restarted.

Discussion

A held computation will not be saved with regard to non-console I/O. Thus, if computation A uses a tape and is held; and computation B also uses that tape; then unholding A may not in fact get the user back to where he was.

Concerning Computations and Computation States

The design of hold has led to the following definition of "computation". A "computation" is one or more working processes in the same computation state together with the associated I/O status.

By the I/O status of a non-current computation, I mean information sufficient to reconstruct the I/O situation obtaining at the time the computation was quit. Regarding a user console, this implies being able to type out the whole last line whose output was interrupted by the quit and being able to type out the current content of the input buffer. This is similar to the requirement for "save/resume" except that code conversion problems presumably don't occur.

With regard to tapes, this means being able to reposition the correct reel at the correct record. Of oourse, the contents of the records, like the pile of cards read or punched, is beyond the scope of this definition of I/O status.

Computation States are labeled CURRENT, QUIT, HELD (or: HELD-a, HELD-b, HELD-c, etc.) and are subject to these rules:

a) when a working process is created, it becomes part of the computation in the current state ("the current computation").

b) computations are Indivisible and Uncombinable: two working processes which have ever been in the same (different) computation(s) are always in the same (different) computation(s).

c) there is at most one computation in any computation state.

Caution on Terminology

The Quit Computation is the computation in the QUIT State. A quit computation is any non-current computation.

Computation State Transformations

The simple operations on computations are:

destroy computation in state x

move computation in state x to state y.

Start, quit, reset, hold, and unhold are (<u>conceptually</u>) examples of such transformations or combinations of them. The action taken by the overseer in stopping the current computation in response to an automatic logout event is not conceptually a state transformation since no computation changes state or is destroyed.

This is important to note since the automatic logout is caused by the system whereas all computation state transformations are caused by the user.

Concerning the Stacking of I/O Status

User Control derives no discernable advantage from the stacking of I/O status (see divert, revert, invert) per ioname. The possibility of reinstating the I/O status of any previously ~~periodically~~ quit computation would permit great ~~quit~~ flexibility. Presently, it is impossible to restart a ~~the~~ HELD computation without:

a) sequentially temporarily reestablishing** the I/O status of eabh computation quit since the held computation was quit;

b) lo~~o~~sing all of this I/O status and hence lo~~o~~sing all of these computations.

Implications

With such a "hold" capability, the user may acquire a large number of held computations. In the most general case, he could restart them in any order without prejudicing his ability to restart any other.

Holding will clog up the system and must be a deliberate rather than an accidental act. Quitting automatically destroys the computation, if any, in the quit state. The listener usually gets there first by reseting the quit computation if no effort is made to save, hold, or restart it. There is no automatic cleanup of "held" computation

** This includes the possiblity that the console might spew forth a few random characters, perhaps one per reestablishment.

Less generally, held computations would have to be restarted in the reverse order of their being held. Still less generally, at most one held computation would be permitted at all.

Somewhere between these extremes, the Overseer can be instructed (at its own initialization) which of these situations is to obtain. (an option).

Interaction with Save/Resume and Automatic Logout

We desire that the overseer respond to the automatic logout event by:

a) quickly stopping the user's current computation

and then b) cleaning up after the user

and then c) saving what's left after the cleanup.

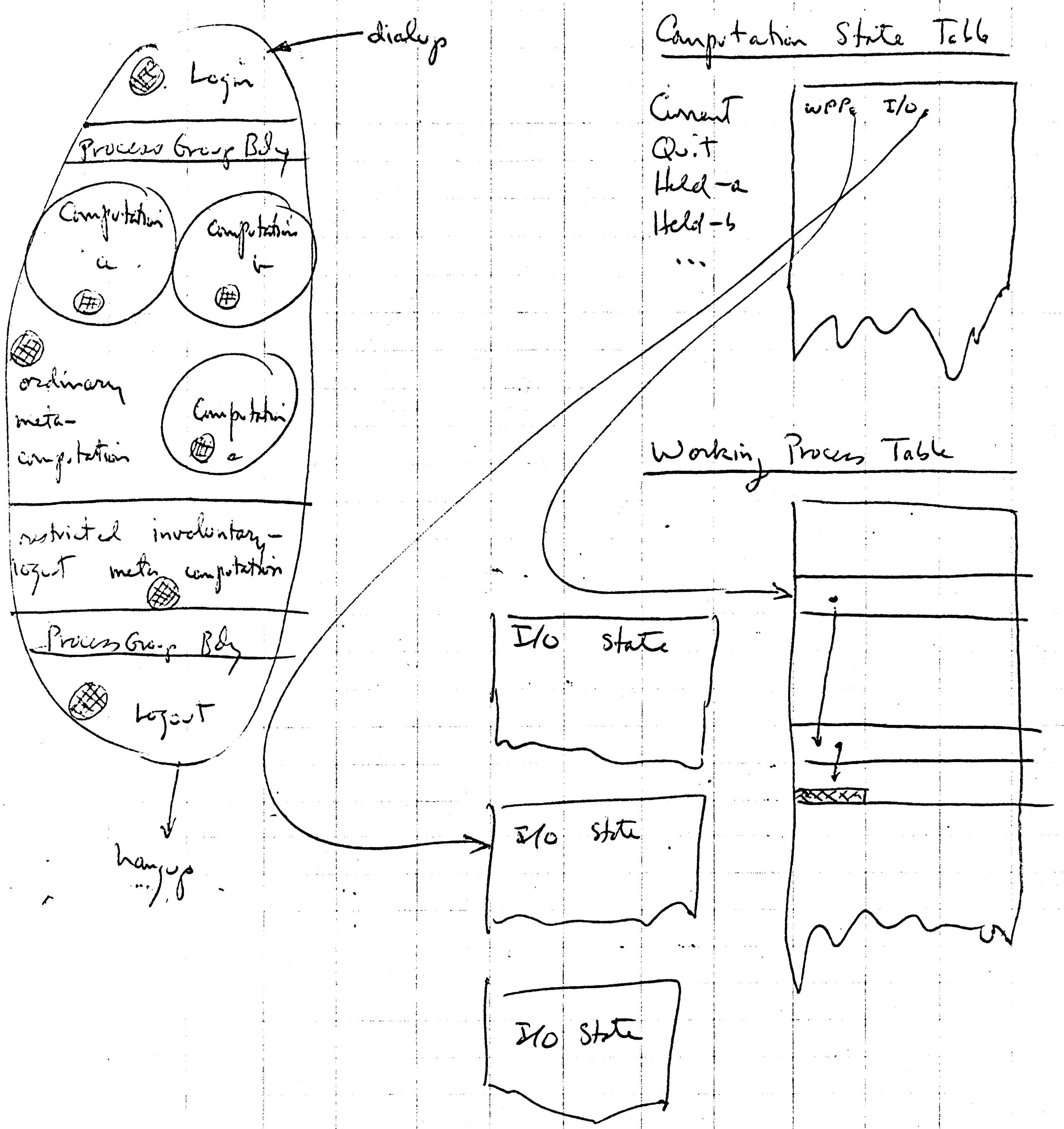
If (c) is not to be done, then (b) too can be skipped.

If the user has "held" a lot of stuff which is only marginally important to him, and he is to be involuntarily logged out, he may wish to issue some meta-computational commands (i.e., commands about his computations such as "destroy the computations in states A, B, C" and "save what's left") to reduce the work of getting him off the machine.

Otherwise, and except for the cost of doing it, the user should not find that save/logout followed by login/resume leaves him in a different situation than he was in. The transparency of it should be like the transparency of quit/start.

Functional Description of Computation / meta Computation View of User Control

▨ present user of console (1 at a time)



dialup
Login
Process Group Bdy
Computation a
Computation b
ordinary meta-computation
Computation a
restricted involuntary-logout meta computation
Process Group Bdy
Logout
hangup
Computation State Table
Current
Quit
Held-a
Held-b
...
WPP
I/O
Working Process Table
I/O state
I/O state
I/O state