A Login Queue.

A major factor affecting user's reactions to the time-sharing system is the disorderly approach the system takes in accepting users when it is overloaded. If the maximum number of users is logged in, all others are given a busy signal and turned away. When space becomes available, the lucky user who happens to try to log in at that instant is able to obtain service; others who may have been waiting longer cannot.

This 'random' approach to scheduling of service tends to have several undesirable results: When a user does finally obtain service he has acquired a very disagreeable frame of mind from having been forced to type repeated login commands to get in. Many users are discouraged from using the system completely; especially those whose time is at a premium. Finally, a practice of "redundant" logins has developed, in which a user logging out first looks around to see if a "friend" would like to use the system; the friend then logs in "on top of" the earlier user, at the same console thereby guaranteeing that he obtains the line of the earlier user. During heavy usage on weekday afternoons, as many as 60 percent of logins have been observed to be redundant.

It is suggested that to help avoid these evils, a limited queue be maintained of users wishing to log in. The appearance of, and implementation of such a queue is described below.

The Login Queue, as seen by the user.

If the system is not "full" the LOGIN command will operate as usual. However, if the system is loaded to capacity, and no party line is available for the user trying to log in, he will receive the message:

PARTY LINE IS BUSY, NO STANDBY LINES ARE AVAILABLE.
YOU ARE NUMBER 6 IN THE QUEUE.

The queue position number will, of course, depend on the number of other users who have arrived ahead of him. The user is not logged in.

As other users log out, this user will receive successive messages:

YOU ARE NUMBER 5 IN THE QUEUE.
YOU ARE NUMBER 4 IN THE QUEUE.

etc. Finally, he will receive the message:

STANDBY LINE IS NOW AVAILABLE. PLEASE LOG IN.

He will have 2½ minutes to log in. If he fails to do so, his turn will be lost. his console will be hung up, and

Two other features will be noted by a user. First, a redundant login will not guarantee a line. If there is a queue, the first user in the queue will receive the released line, and the user trying to login will be placed at the end of the queue. Second, if a user is automatically logged out because of "bumping" by a primary user, he goes to the head of the queue, and will be first to get a new line when one becomes available.

Implementation of a Login Queue.

The time-sharing system is "assembled" for a maximum number of users, N, say 51. Normally, the largest number of users who are allowed to log in is some smaller number, since limited computation capacity is available. Call this smaller number M, and assume that it is 30. A queue will then be maintained of up to N-M-1 users, which in this case would be 20. The remaining "slot" is left open for a user to obtain the message that no queuing space is available.

When the system is full, The LOGIN command will place in the array NOTIME a marker for each user who tries to log in ~~(if the system is loaded),~~ indicating his position in the queue and his party group affiliation. It will also send the user the initial message that he has been queued.

The LOGOUT command will, after logging a user out, chdck to see if a login queue exists. If one does, LOGOUT will update the queue, send updating messages to each user in the queue, and invite the head user to log in. It will place a marker in the NOTIME array indicating ~~that~~ to the LOGIN command that this user may log in. If a primary line is released, ~~LOGOUT~~ LOGUP will determine which user to admit by taking the longest waiting user with the same party group as the user who just logged out. ~~If no user if the same party group is in the queue,~~ Otherwise, the longest waiting user in the queue is then permitted to log in.

On Automatic logouts, the logout command will not ~~perform any operations with the kqueue.~~ If the Automatic logout was called for by LOGIN, in the process of "bumping" for a higher

priority user, the LOGIN command will update the queue with the logged out user at the fromt.

The Scheduling algorithm presently hangs up inactive lines. ~~It will instead only hang up lines which have been~~ It will now check to see if these lines are queued before hanging them up. Hangup would only occur for a line which failed to accept a LOGIN invitation, or never tried to LOGIN at all.

~~The hangup coding in the clock section will need to be modified.~~

The hangup coding in the clock section will initiate a ~~dummy~~ LOGUP command for the user hungup, to update the ~~queues~~ login queue, if necessary. Main Control should accept a LOGUP command from a user who is not logged in.

These last ~~two~~ three changes (to SCHEDL, clock section, & main control) represent the only coding which needs to be done in care A, and should be accomplishable with two or three lines of code each.

Needless to say, the modifications described above will have to wait until the major changes to the I/O file control system are finished and debugged, since ~~the I/O file control~~ this system profoundly affects the LOGIN and LOGOUT commands.