From: Robert S. Coren

To: Distribution

Date: April 17, 1973

Subject: CHANGES TO I/O DAEMON

The daemon-print/punch mechanism is in the process of being converted to use message segments to hold requests, in a similar manner to absentee processing. This has the following advantages:

1) It will be possible for a user to cancel his own daemon requests;

2) It will allow for the addition of new options to the dprint command, without freezing the format of a request; i. e., the message containing the request can always be extended to add new features.

In addition, the dprint command will no longer call hcs_$wakeup directly, but will interface through a subroutine called dprint_, passing it an optional structure of arguments. This will allow other programs, particularly subsystems (such as gcos), to issue daemon requests with comparative ease.

New features to be implemented immediately

1) A new control argument to the dprint command, "-destination" or "-ds", will be introduced to specify the location to which printed/punched output is to be delivered. This function will now be separated from the "-he" option. The new format of the header sheet will be as follows: the top line of big letters (where the heading now appears) will contain the destination (if none was specified, this will default to the requestor's project); below it will appear, also in big letters, the specified heading (which defaults to the requestor's name).

2) The list_daemon_requests command, which may be used to print a list of all a user's outstanding dprint/dpunch requests, will be installed at the same time as the new I/O daemon. It is implemented as an additional entry point to list_abs_requests.

New features for the near future

Soon after the new io_daemon is completed, a cancel_daemon_request command will be introduced, to perform a

function analogous to that of cancel_abs_request. It will be implemented as an additional entry point to cancel_abs_request.

Implementation

dprint

The dprint command will function essentially as at present, except for two details: 1) instead of adding a link to the directory qn, where n is the specified request queue, it will call dprint_, which adds a message to the message segment io_daemon_n.ms; 2) before sending a request, the command will check to see if IO.SysDaemon has at least "r" access on the segment to be printed; if not, it informs the user and asks if he still wants the request to be attempted (as at present when the segment is not found). This way, the user can correct the problem immediately, rather than finding out about it the next morning when he gets his output.

dprint_

This subroutine will do the actual adding of the request to the message segment, and call hcs_$wakeup on the io_daemon. The only arguments it absolutely requires are the directory name and entry name of the segment to be printed or punched; all the remaining parameters are passed in a structure to which the caller optionally supplies a pointer. If the pointer is null, dprint_ supplies reasonable defaults (e.g., print rather than punch, one copy, don't delete, no special heading, no destination).

io_daemon

The current set of changes continues to assume the existence of one or more independent IO.SysDaemon processes, each of which receives all the wakeups sent by dprint_ (we are, however, making it easier to change the maximum of such processes, which for the moment will continue to be 2. See the discussion of data bases below). The changes in the daemon's mode of operation are mostly concerned with ensuring that each request is performed exactly once. Since it is impossible to change the identifier of a message within a message segment, this implies deleting each request from the queue when it is started, rather than after it is completed, and saving it elsewhere. For this purpose, a daemon "save" segment is used, which includes an area into which messages are read. The space occupied by such a message is freed when the request has been completed. If the system crashes during the processing of a request, the message describing it is saved in this segment, where it can be found by the daemon_init procedure the next time the daemon is logged in and brought up.

The io_daemon thus has two entry points for processing requests. Its main procedure entry point is used for requests which are still in the message segment; in these cases it ignores the event call message identification passed to it by wakeup, and simply uses the first request in the specified queue. The entry point io_daemon$get_save is used to process a request which has previously been saved in the "save" segment, in which case the event call message id is treated as a pointer to the entry in the save segment describing the request. (This is clarified somewhat in the discussion of data bases below.)

daemon_init

This procedure operates much as it always has, except that when the first daemon is initialized, daemon_init, before scanning the queues for pending requests, checks the save segment for unfinished requests. If it finds any, it signals io_daemon$get_save as described above.

Two other entries to daemon_init, daemon_init$test_io and daemon_init$init_info, are used to initialize data bases, as described below.

In addition, all references to the translator daemon have, at long last, been removed from daemon_init.

Data bases

All of the segments described in this section are expected to reside in a single directory. For normal operation, this directory will presumably be >daemon_dir_dir>io_daemon_dir. However, for testing purposes, the entry point daemon_init$test_io is called with a single argument, which is the full pathname of the directory in which these segments are to be found. (The daemon_init entry passes this information on to io_daemon and dprint_.)

The segments in question are created by daemon_init$init_info, which asks to be told the name of the directory to create them in.

1. Message segments

The request queues are message segments named "io_daemon_n.ms", where n is the queue number (1 to 3). The format of an individual mesage is described by the dprint_message include file, a listing of which is attached. When each message segment is created, the "extended acl" will be set to "adros" for IO.SysDaemon.* and "aos" for *.*.* (thus only the daemon will be able to read and delete messages other than its own). It might be

desirable to give some specific users or projects "aros" for "-admin"-type queue listing, but this can always be done by hand later. (I notice that the absentee queues, at the moment, give "aros" to everybody.)

2. "Info" segment

This segment, with the entry name "daemon_info", is an expanded version of the I/O daemon's old "info0" segment, which has been used primarily to store the names of the event channels over which the daemon is signalled. It is described by the daemon_info_format include file. the "destination" substructure will be used later when a wider choice of devices and locations is available for use as targets of daemon requests.

3. "Save" segment

This segment, called "daemon_save_seg", has two sections. The first is an array of structures (daemon_save_req) each member of which describes one saved message, and the second part is an area into which the messages are read. The daemon_save_req structure includes the argument structure returned by various message_segment_ entries, which in turn includes a pointer to a particular message in the area. Note that the word offset portion of this pointer remains valid between crashes, and makes it possible to locate a still-allocated message in the area.

The "lock" word is set by an io daemon which is preparing to save a request, and thus prevents two or more daemons from trying to use the same entry. The "active" word is not set until a mesage has actually been saved. Both are cleared when a request is finished. (A more sophisticated mechanism can be worked out later to save all requests for a fixed time interval after completion, in case it should subsequently be discovered, say, that a printr has been chugging away without any paper.) The daemon_init program uses the "active" switch to determine whether to wake up the io_daemon$get_save entry; if it finds a slot locked but not active, it clears the lock on the assumption that the system crashed before the daemon got around to moving the request out of the message segment. Failure to do this would result in the daemon being unable to find a free slot in the save segment. With proper management, this should be impossible as long as there are as many slots as daemon processes.

```
/* BEGIN INCLUDE FILE ... dprint_message.incl.pl1 */

dcl  dprint_msg_buffer (64) fixed bin (71);      /* automatic buffer for message */
dcl  MESLTH fixed bin int static init (4032);
                                                 /* length of message in bits */
dcl  VERSION fixed bin int static init (0);      /* version number of this dcl */
dcl  dmp ptr;                                    /* ptr to message */

dcl 1 dprint_msg based (dmp) aligned,            /* structure of a daemon request */
    2 msg_time fixed bin (71),                   /* date and time of request */
    2 dirname char (168),                        /* directory name */
    2 ename char (32),                           /* entry name of file requested */
    2 version fixed bin,                         /* version of this declaration */
    2 print_punch fixed bin,                     /* 1=print, 2=punch */
    2 copies fixed bin,                          /* number of copies */
    2 delete_sw fixed bin,                       /* delete file after print? */
    2 notify fixed bin,                          /* notify user when done? */
    2 heading char (64),                         /* heading or page 1 */
    2 output_module fixed bin,                   /* 1=print, 2=punch, 3= mcc, 4=raw */
    2 carriage_control fixed bin,                /* 1=normal, 2=no pgoflo, 3=ignore, 4=ignore, no pgofl
    2 destination char (12),                     /* routing for output */
    2 forms char (8),                            /* forms required */
    2 lmargin fixed bin,                         /* left margin */
    2 line_lth fixed bin,                        /* line length */
    2 page_header char (120);                    /* heading or every page */

/* END INCLUDE FILE ... dprint_message.incl.pl1 */
```

```
/*        BEGIN INCLUDE FILE ... daemon_info_format.incl.pl1  */

/*        Format of the io_daemon's "info" segment */

dcl  nd fixed bin int static init(2);              /* number of daemon's that may */
                                                   /* be logged in at once */
                                                   /* (put in here for easy changing) */
dcl  dip ptr;

dcl  n_dest fixed bin int static init(4);

dcl  1 iod based(dip) aligned,
       2 version fixed bin,
       2 channels(2),                              /* wakeup message channels */
           3 procid bit(36),
           3 chn_name(0:3) fixed bin(71),
       2 lock fixed bin(35),                       /* used to prevent simultaneous
                                                          accounting calls */
       2 lock_count fixed bin,
       2 save_seg_ptr ptr,                         /* pointer to segment in which daemon
                                                   * saves current request  */
       /* limit of version 1 structure */

       2 destinations(n_dest),
           3 id char(12),
           3 attached fixed bin,
           3 busy fixed bin,
           3 stream_name char(20),
           3 dim_name char(32),
           3 dev_type char(8);

/*        END INCLUDE FILE ... daemon_info_format.incl.pl1  */
```

```
/*        BEGIN INCLUDE FILE . . . daemon_save_seg.incl.pl1 */

/* Format of entries in segment used for saving io daemon's current
*    request. Daemon can search for a free slot. Requests are saved
*    here in case of system crashes.
*/

dcl       daemon_save_ptr ptr;

dcl       daemon_save_ent_len fixed bin int static init(24):          /* length in words */
                                                                      /* of one entry    */

dcl       1 daemon_save_req based(daemon_save_ptr) aligned,
             2 lock fixed bin,                    /* 0 = free, non-0 = locked */
             2 active fixed bin,                  /* 0 = no, 1 = yes */
             2 ms_arg_copy(14) fixed bin,         /* info returned by message_segment_ */
                                                  /* programs          */
             2 queue fixed bin,
             2 pad(7) fixed bin;                  /* force to 8-word boundary */


dcl       1 daemon_save_seg based aligned,        /* template for save segment */
             2 save_req_copy(nd, 24) fixed bin,   /* as many save slots as daemons */
                                                  /* nd is supplied in daemon_info_format */
                                                  /* include file */
             2 msg_area area(536);                /* area for messages */


/*        END INCLUDE FILE . . . daemon_save_seg.incl.pl1 */
```