11/15/67
H. Sturgis

1.0

What follows is a description of a basic time sharing systems, as seen by a knowledgeable user. There are three basic kinds of objects, and some actions that can be performed on them.

1.1

<u>Data Block</u>

A data block is a finite sequence of 60-bit words. They are numbered 0 through N-1 for a data block of N words.

1.2

<u>User Process</u>

A user process consists of two logical flags and a sequence of quadruples. The two flags are called the 'ready' and 'wakeup' flags. The sequence of quadruples is called the memory map. Each quadruple in the map consists of a write bit, a CPU address, the name of a data block and a position in the data block.

From time to time, if the ready flag is on, a series of events takes place. For each quadruple in the map of the process, the indicated word in the data block is copied to a cell in the CPU memory whose address is the address in the quadruple. When all words have been copied, the first 16 words of the CPU memory are used as an exchange jump package.

After some time has elapsed an exchange jump is again performed on the first 16 words. Then for each quadruple in the map of the process, whose write bit is still on, the word in the CPU memory at the CPU address of the quadruple is copied to the indicated word in the data block of the quadruple.

Notice that the words do not actually have to be copied to the CPU started at address 0, but the same results will be obtained if they are copied ~~sharting~~ starting at any relative address, if the first 16 words are suitably modified before and after the exchange jumps.

1.3

System Process

A system process consists of a ready flag and a wake up flag, ~~as for a user process,~~ a type and a state. The state can change only when the ready flag is on.

1.4

Monitor Requests

Either kind of process whose ready flag is on is said to be a running process. A running process may req~~s~~uest certain actions to be done by the system. These are called monitor requests and each is associated with some basic object. Details as to how a user process makes a monitor request will be given later. Here we list the possible monitor requests.

A. Data Blocks

1. Create data block of N cells
2. Write a given K words from the process to the data block starting at cell L. $(L + K \leq N)$
3. Read K words to the process from the data block starting at cell L. $(L + K \leq N)$

B. User Processes

1. Create a user process with a given map

2. wake up the user process

```
if ready then wakeup := true
     else ready := true;
```

3. Block the user process

```
if wake up then wake up := false
     else ready := false;
```

4. Clear the wake up flag

```
wake up := false
```

Note: 3 and 4 are usually done by a process to itself, while 2 is usually done on some other process.

C. System Process

1. Create a system process of given type

2,3,4 as for a user process

2.0 Examples of use of such a system ~~A possible implementation~~ as seen by the user

We assume that the basic objects in the system are named by giving positive integers. We also assume that a FORTRAN-callable subroutine is given for each of the monitor requests.

A1. CREATE D (NAME,CELLS)

creates a data block of CELLS ~~cells~~ cells and places the resulting name in NAME ~~name~~

A2. WRITE D (NAME, START, COUNT ~~WENT,~~ FROM)

writes COUNT ~~count~~ words to data block of name NAME ~~name~~ starting with cell START ~~start~~ and taking words from array FROM ~~from~~.

A3. READ D (NAME, START, ~~WENT~~ COUNT, TO)

reads ~~count~~ COUNT words from data block of name ~~name~~ NAME

U (NAME, MAP, ~~WENT~~)

creates a user process and places name in NAME. MAP is an array of ~~5+ count~~ 5 * COUNT words used to define the memory map as follows: each consecutive group of 5 words defines a sequence of quadruples. If the 5 words are W,A,D,P,C then C quadruples are formed whose write bits are all <u>true</u> if ~~w~~ W ≠ 0 and <u>false</u> if ~~w~~ W = 0; whose data blocks are all the block of name D; whose CPU addresses are A, A+1, ..., A+C-1; and whose indicated data ~~block~~ block cells are P, P+1, ..., P+C-1.

The process always starts with the read and wake up flags off.

B2. WAKE U (NAME)

wakes up user process of name NAME

B3. BLOCK U

blocks its own process

B4. CLEAR U

clears its own wake up flag

C1. CREATE S (NAME, TYPE, D)

creates a system process of type indicated by the integer ~~type.~~ TYPE. (It is assumed that some list of types is given.) Places the name of the process in NAME. System processes are always started with ready flag on and wake up flag off. ~~D is~~ D is assumed to name a data block and the resulting system process will use that data block to communicate with the user process.

C2. WAKE S (NAME)

wakes up system process of name NAME.

The following two will be only used in the description of system processes.

C3. BLOCKS

blocks its own process

C4. CLEARS

clears its own wake up flag

2.1 Low speed terminal communication (simple)

We assume the existence of two particular types of system processes, which will be described by FORTRAN programs, and then show how a FORTRAN program can be written to communicate with a terminal.

A. A device input process

We assume that every time a character comes in it is placed in word CHAR, that word count is incremented by 1 and that the process is awakened, and that the data block used for communication is named in D.

```
      COUNT = 0
10    CALL BLOCKS  COUNT
           IF(~~WRITE~~ .EQ. 0) GO TO 10
      CALL READ D(D,0,3,DATA)
12         IF(DATA(2) .NE. 0) GO TO 15
      DATA(2) = 1
      DATA(3) = CHAR
           CALL WRITE D (D,0,3,DATA)
      COUNT = 0
           IF DATA(~~2~~) .NE. 0) CALL WAKE U (DATA(1))
      GO TO 10       1
C
15    CALL BLOCKS
      GO TO 12
```

This is a simple process, and does not detect the loss of a character.

B. A device output process

This is similar to the input process. We assume the existence of a logical function WRITE(CHAR) which will, if possible, send CHAR to the terminal and returns the value .TRUE. , or, if not possible, will return the value .FALSE. . We assume this process awakened each time it becomes possible for a character to be sent.

```
10   CALL BLOCKS
     CALL READ D (D,0,3,DATA)
          IF (DATA(2) .EQ. 0 ) GO TO 10
12        IF (WRITE(DATA(3) ) GO TO 15
     CALL BLOCKS
     GO TO 12
15   CALL WRITE D (D,1,0)
          IF (DATA(1) .NE. 0) CALL WAKE U (DATA(1) )
     GO TO 10
```

Notice in both A) and B), how the process does not block if any wake ups have occurred since the last block.

C. We now give 3 subroutines that can be written by a user for the purpose of communicating with a teletype.

C1. Create the system processes

We assume that variable TIN and TOUT contain the numbers of the types of the appropriate teletype processes. We assume parameter US contains the name of the user process.

```
SUBROUTINE CREATE (IND,INP,OUTD, OUTP, US)
INTEGER IND, INP, OUTD, OUTP, US
INTEGER DATA(3)
DATA DATA /0,0,0/
DATA(1) = US
CALL CREATE D (IND,3)
CALL WRITE D (IND, 0,3,DATA)
CALL CREATES (INP, TIN, IND)
CALL CREATED (OUTD, 3)
CALL WRITED(OUTD, 0,3,DATA)
CALL CREATES(OUTP,TOUT,OUTD)
RETURN
END
```

C2. <u>Get a character</u>

```
          INTEGER FUNCTION GETCHAR(IND,INP)
          INTEGER IND,INP
          INTEGER DATA(3)
     C
     10        CALL READ D(IND,0,3,DATA)
                    IF (DATA(2) .NE. 0) GO TO 15
          CALL BLOCK U
          GO TO 10
     C
     15   CALL WRITE D(IND,1,0)
          CALL AWAKES(INP)
          GETCHR = DATA(3)
          RETURN
          END
```

C3. <u>Write a character</u>

```
          SUBROUTINE WRITE (OUTD, OUTP, CHAR)
          INTEGER OUTD, OUTP
          INTEGER DATA(3)
     C
     10   CALL READ D (OUTD, 0,3,DATA)
               IF (DATA(2) .EQ. 0) GO TO 15
          CALL BLOCK U
          GO TO 10
     C
     15   DATA(2) = 1
          DATA(3) = CHAR
          CALL WRITE D(OUTD, 0,3, DATA)
          CALL AWAKES(OUTP)
          RETURN
          END
```

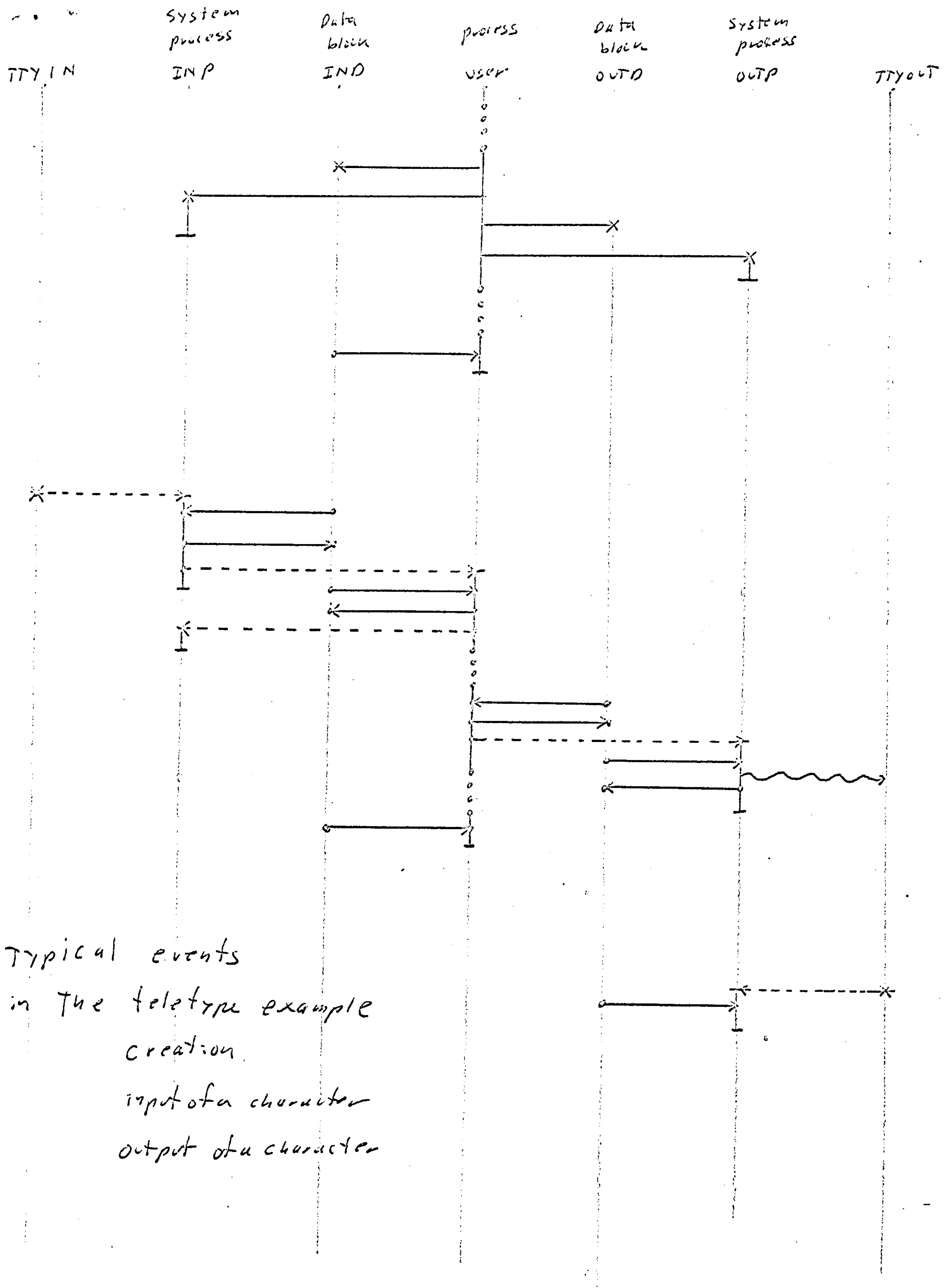
11/7/67

NOTATION

TTYIN	represents the line comming in from a teletype
$*$	represents the arivel of a character
TTYOUT	represents the line going to a teletype
$\rightsquigarrow$	represents the starting out of a character
$*$	represents the completion of output of a charact

Sur processes

$*$	creation
$\vert$	ready flag on
$\vdots$	ready flag on, but in other parts of code not of interes
$\dashrightarrow\top$	wake up
$\perp$	block
$+$	attempted block, but wake up bit on
$\vdash\dashrightarrow$	wake up another process
$\vdash\!\longrightarrow\circ$	write a data block
$\vdash\!\longleftarrow$	read a data block



Typical events
in the teletype example
creation
input of a character
output of a character