1. GENERAL

Checkio may supervise all actions on a particular document. That is performed in the following way: Assume that checkio is executed in a job process called dev. Then all messages sent to dev are passed on by checkio to the document supervised. The answer from the document is passed back to the original sender process, which seems to be handling the document in the normal way.

Checkio can easily print all messages and answers as they are passed on, and you can later find out about parity errors from the document, rereading etc.

2. CALL

The process dev must be created with a protection register which enables it to modify the contents of all other processes.

Checkio must be called with one parameter specifying the name of the document to be supervised. The messages and answers are printed on the current output of dev. A console communication to start dev may look like this:

```
to s
new dev size 5000 pr 1 2 3 4 5 6 7 pk 1 run
to dev
o lp              ; print on the line printer
checkio t6        ; check the document t6
```

You will then see nothing from dev until some messages are sent to it, for instance from another job started like this:

```
to s
new sl run
to sl
s=set mto dev     ; edit to the 'magnetic tape' dev,
s=edit tre        ; which acts as the magnetic tape t6.
```

Dev will now start listing the communication on the line printer. When you have finished using dev for supervising of t6, you can proceed like this:

```
to s
proc dev break  ; the remaining output from dev appears on
                ; the printer.
*** fp break
checkio reader  ; select a new document etc.
```

3. RESERVING THE DOCUMENT

Checkio cannot simulate those actions on a document which involve reservation of the document, creation of it as an area process etc. So a process which tries to reserve dev (the document) or create it as an area process, and checks the result, cannot be supervised.

To remedy the problem for other processes, checkio proceeds as follows when the document does not exist: First a call of create area process is attempted. If this does not succeed, checkio outputs the console message

```
mount <document>
```

and performs the usual wait action. Then the operation is repeated before the original sender gets his answer.

If the document rejects the message, checkio reserves the document and repeats the operation before the original sender gets his answer.

4. OUTPUT

The output from checkio consists of one line for each message:

message <operation> <mode> <first addr> <last addr> <sender>

and one line for each answer:

answer <bits of logical status> <bytes> <characters> <file> <block>

If the answer is not transmitted back to the original sender, the reason is printed as one of the lines

create
<result> reserved

information	repl.		ident FGS 850228		page 1/1
	RC8000				class EXT

subj. Corrections to RCSL No 55-D70, Checkio

PN: 991 10220

Replace the following pages: 1 and 2.

RCSL No: 42-i 1321

FILE PROCESSOR UTILITY PROGRAM: CHECKIO

General

Checkio may supervise all actions on a particular document. That is performed in the following way: Assume that checkio is executed in a job process called dev. Then all messages sent to dev are passed on by checkio to the document supervised. The answer from the document is passed back to the original sender process, which seems to be handling the document in the normal way.

Checkio can easily print all messages and answers as they are passed on, and you can later find out about parity errors from the document, rereading, etc.

Call

The process dev must be created with a size big enough to store the data blocks passed by.

Checkio must be called with one parameter specifying the name of the document to be supervised. The messages and answers are printed on the current output of dev. A console communication to start dev may look like this:

```
to s
new dev size 5000 run; if monitor older than 8.0 mode 0
                      ; also
to dev
o lp                  ; print on the line printer
checkio t6            ; check the document t6
```

You will then see nothing from dev until some messages are sent to it, for instance from another job doing like this:

```
t = set mto dev 0 6   ; edit to the 'magnetic tape' dev,
t = edit tre          ; which acts as the magnetic tape
                        file t6.
```

Dev will now start listing the communication on the line printer. When you have finished using dev for supervising of t6, you can proceed like this:

```
to s
proc dev break        ; the remaining output from dev
                        appears on the printer.

***fp break
checkio reader        ; select a new document, etc.
```

Reserving the document

Checkio cannot simulate those actions on a document which involve reservation of the document, creation of it as an area process, etc. So a process which tries to reserve dev (the document) or create it as an area process, and checks the result, cannot be supervised.

Processes running utility programs and algol/fortran programs will not act this way: operations are sent and reserve process/create area process is done if necessary after the first operation.

To remedy the problem for these processes, checkio proceeds as follows when the document does not exist: First a call of create area process is attempted. If this does not succeed, checkio outputs the console message

mount <document>

and performs the usual wait action. Then the operation is repeated before the original sender gets his answer.

If the document rejects the message, checkio reserves the document and repeats the operation before the original sender gets his answer.

If the original sender gets stopped before checkio has received/delivered the data in i/o messages, checkio retries until the sender has delivered/received the data.

To supervise actions on an area process, a file descriptor of kind=6 and document name = name of process executing checkio must be created in advance.

Output

The output from checkio consists of one line for each message:

message<operation><mode><first addr><last addr><sender>

and one line for each answer:

answer<bits of logical status><bytes><characters><file><block>

If the answer is not transmitted back to the original sender, the reason is printed as one of the lines

```
reserved by another process
not user/cannot be reserved
does not exist/not user of area process
```