Fork Protection Memo

Chuck Wall
June 14, 1974

Technical Memo

TM.74-25

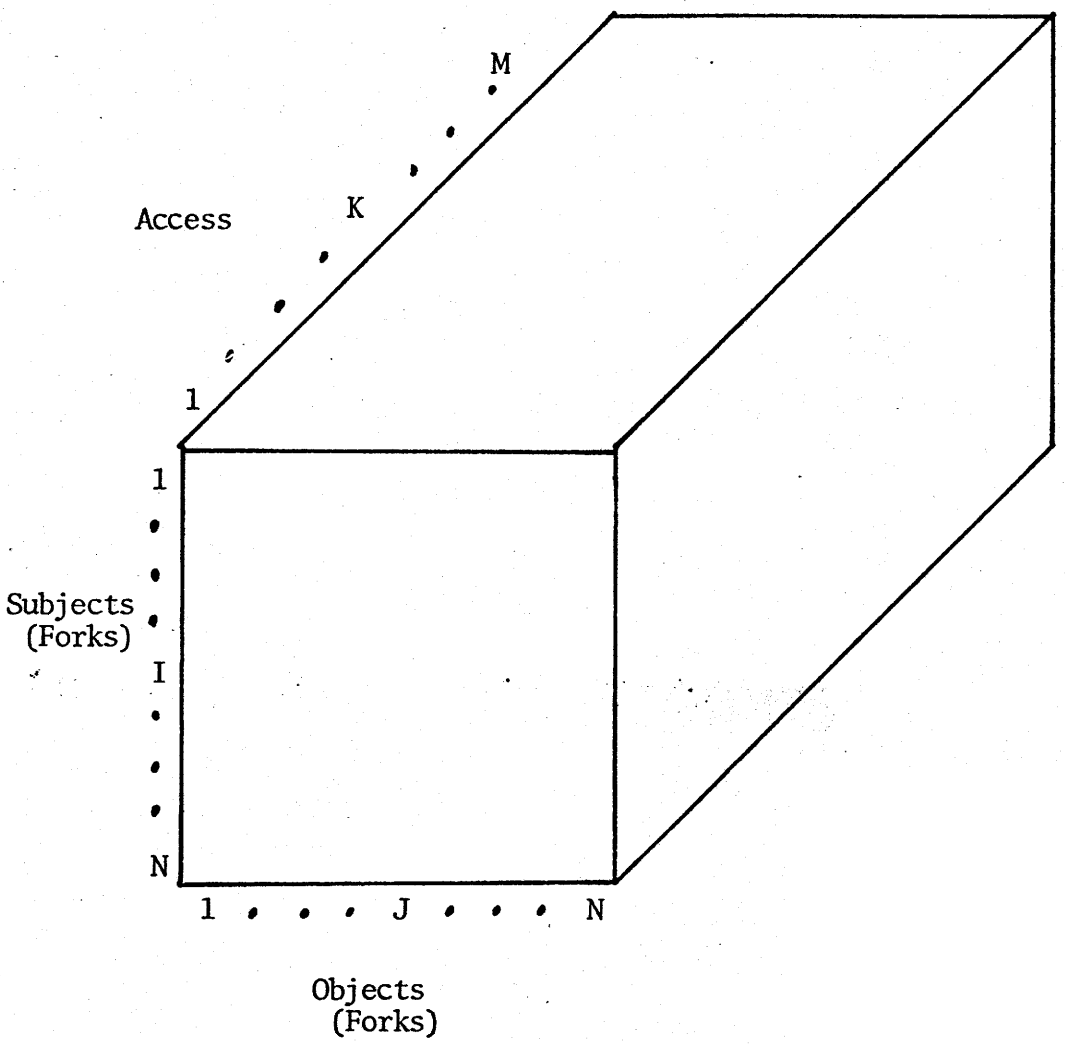
1. Introduction

In this memo we will outline our plans for additional fork protection in TENEX. We will specify the protection required, outline the basic implementation and indicate the changes and additions required. While our central concern is solving the I4 problems, we will be attempting to specify a mechanism that will support a class of fork protection policies.

2. Fork Access Control

The basic problem is to implement access control on forks that will give more control than is currently available in TENEX. In particular, a subsystem might demand to be completely isolated from all other forks in the job.

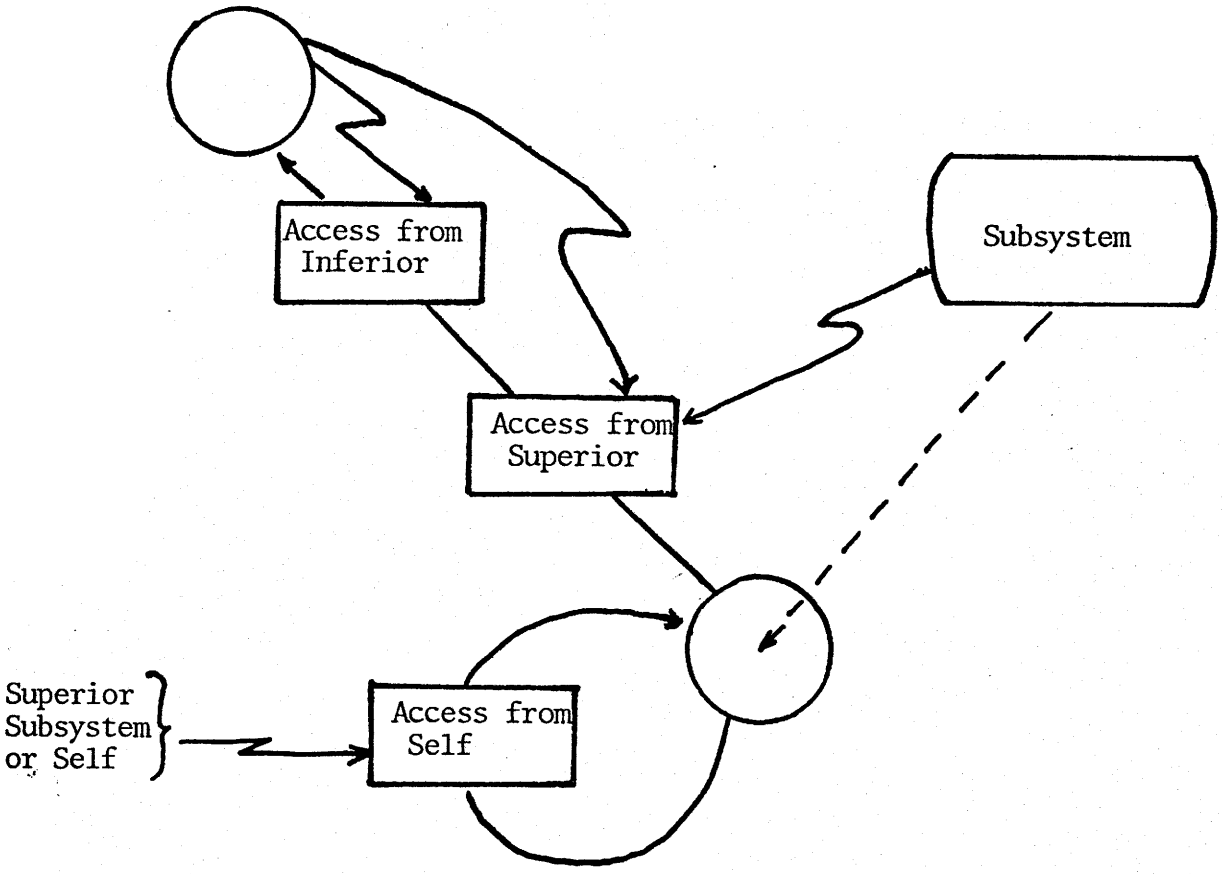
The general solution (at the conceptual level) is of course to specify in a three dimensional access matrix all the fork to fork access types possible. (see fig. 2.1) This access matrix representation contains enough information to provide protection for a graph structure. Since TENEX utilizes a tree structure for control it is not necessary to consider such a general solution. In fact all that is necessary is to control the access that superior and inferior forks have, and the access it has to itself. The specification of who controls the access information will depend on how the fork has been set up. If the superior fork has created the fork, then control will reside with the superior. If a PGET (protected subsystem GET) has set up the fork, then the control will reside with the fork itself. In this way a subsystem can bring with it abilities that the superior fork does not and should not have and protect those abilities by controlling the access that the superior has to it. (see fig. 2.2) With this scheme of access control the fork structure can change and the access from superior specified by a subsystem would still be enough to protect the subsystem. Also subsystems can be protected at any level in the fork structure.



If (I, J, K) = 1 allow fork I, access K to fork J

else protection error

Fig. 2.1: Fork Access Matrix



Access from superior or self would be set and controlled by either the superior or the subsystem depending on the situation.

Access from the inferior would always be specified by the superior.

Access from self may be specified by either the superior or the subsystem or control could be given to the fork itself.

Fig. 2.2: Fork Access Control

We have assumed in this solution that parallel forks are independent entities in the sense that there will not be any operations that are directly performed on them. Should this change for any reason it would simply mean adding another access control list.

3. Implementation

Basically, the implementation of access control on forks will be based on the current TENEX method of determining the relationship of an object fork to a subject fork. This involves searching the FKPTRS table in the JSB. All we need to do is insert the protection in parallel with this search to ensure that at each fork in the structure the access being attempted is to be allowed. Exactly what information needs to be encoded depends on the access types we identify.

Our first task then is to identify all the access types for each of our access lists. The format for this list will be <JSYS> <ACCESS> (<from>) where <JSYS> is the name of the JSYS in which the access occurs, <ACCESS> is the access involved and <from> is a list of where in the structure the access is from (S=Superior, I=Inferior, C=Current or Self). The code S=Superior means the access or operation the immediate superior has on the fork, "I=Inferior" means the access an inferior has on a fork and "C=Current or Self" mean the access or operations it may perform on itself. Since a fork can have multiple inferiors, the "from Inferior" access will have to be associated with the individual inferiors involved. An alternative way to look at this is "to Superior" access. A " * " will indicate no access.

The relevent access types are as follows:

<JSYS>	<ACCESS>	(<from>)
PMAP	Map page from fork	(S, I, C)
	Map page to fork	(S, I, C)
RPACS	Read the accessibility of a page	(S, I, C)
SPACS	Set the accessibility of a page	(S, *, C)
RMAP	Acquire a handle on a page	(S, I, C)
GPJFN	Get primary JFN	(S, *, C)
SPJFN	Set primary JFN	(S, *, C)
RUNTM	Runtime of one process or whole job	(-------)
GETER	Get most recent error conditions	(-------)
GTRPI	Get trap information	(S, *, C)
SIR	Setup PSI table address	(S, *, C)
RIR	Read PSI table address	(S, *, C)
EIR	Enable PSI for process	(S, *, C)
SKPIR	Test PSI it see if it is enabled	(S, *, C)
DIR	Disable PSI for a process	(S, *, C)
AIC	Activate specified channels	(S, *, C)
IIC	Initiate PSI on specified channel	(S, I, C)
DIC	Deactivate specified channels	(S, *, C)
RCM	Read channel-activated word-mask	(S, *, C)
RWM	Read waiting channel interrupts word-mask	(S, *, C)
SIRCM	Set inferior reserved channel mask	(S, *, *)

<JSYS>	<ACCESS>	(<from>)
RIRCM	Read inferior reserved channel mask	(S, *, C)
DEBRK	Dismiss current PSI in progress	(*, *, C)
STIW	Set terminal interrupt word	(S, *, C)
RTIW	Read terminal interrupt word	(S, *, C)
CIS	Clear interrupt system	(-------)
RWSET	Release working set	(*, *, C)
GTRPW	Get trap words	(S, I, C)
RPCAP	Read process capabilities words	(S, I, C)
EPCAP	Set process capabilities words	(S, *, C)
KFORK	Kill one or more forks	(S, *, *)
SPLFK	Splice fork structure	
	fork to become new superior	(S, *, C)
	fork to become new inferior	(S, *, *)
FFORK	Freeze one or more forks	(S, *, *)
RFORK	Resume frozen fork	(S, *, *)
RFSTS	Read fork status	(S, I, C)
SFORK	Starts a fork	(S, *, *)
SFACS	Set fork AC's	(S, *, *)
RFACS	Set fork AC's	(S, *, *)
HFORK	Halts one or more forks	(S, *, C)
WFORK	Wait for one or more forks to terminate	(S, *, *)
GFRKH	Get a fork handle not currently known	(S, *, *)

<JSYS>	<ACCESS>	(<from>)
RFRKH	Release a fork handle	(*, *, C)
GFRKS	Get fork structure	(-------)
DISMS	Dismiss this process for Δ t	(*, *, C)
HALTF	Halt this process	(*, *, C)
BPT	Currently HALTF	(*, *, C)
WAIT	Dismiss this process indefinitely	(*, *, C)
GET	Get a SAVE file	(S, *, C)
SFRKV	Start fork using entry vector	(S, *, *)
SAVE	Non-sharable SAVE	(S, *, C)
SSAVE	Sharable SAVE	(S, *, C)
SEVEC	Set entry vector	(S, *, C)
GEVEC	Get entry vector	(S, *, C)
SCVEC	Set compatibility entry vector	(S, *, C)
GCVEC	Get compatibility entry vector	(S, *, C)

In the above list RUNTM, GETER, CIS and GFRKS do not fit into the tree structure, but it is possible under some circumstances that these access types as they are associated with particular forks would need to be protected. We have decided not to protect forks from these access types. We have decided also not to protect forks from themselves and to protect superior forks from inferiors by allowing the superior to set bits associated with these operations in bits 9-17 of the capabilities word. This leaves us with the "from superior" access control. Remember that the control on the "from superior" access will be specified by either the superior or the "protected subsystem" when a PGET is issued.

In order to implement access control on the superior then all we need do is set up a table that is parallel to FKPTRS and put into SETLFK, SETJFK and MAPFKH in SWPMON code that will check the protection information in the parallel table. If we let a bit stand for access allowed and initialize this table to -1 (e.g. all bits on) then this will be equivalent to the current TENEX except for the small amount of code we need for the protection test. If the bit is off then the access by the superior is not allowed and current error returns can be utilized. It would be convenient if we could specify less than 36 access categories, since that would mean an increase of only NUFKS words in the JSB. We will need one bit to indicate control by superior. We will attempt a tentative specification of access groups and bit assignments as follows:

Bit #	Access Group	Access types included	
BØ	Superior Access Control	SFACL: RFACL:	Set fork access control Read fork access control
B1	Map from	PMAP: RPACS: RMAP: SAVE: SSAVE:	Map page from fork Read accessibility of page Acquire a handle on page Non-sharable save Sharable save
B2	Map to	PMAP: SPACS: GET:	Map page to fork Set accessibility of page Get a save file
B3	Get primary JFN	GPJFN:	Get primary JFN
B4	Set primary JFN	SPJFN:	Set primary JFN
B5	Get trap information	GTRPI:	Get trap information
B6	Read PSI information	RIR: SKPIR: RCM: RWM: RIRCM: RIIM: GTRPW:	Read PSI table address Test PSI system Read channel-activated word-mask Read waiting channel word-mask Read inferior reserved channel Read terminal interrupt word Get trap words
B7	Set PSI information	SIR: SIRCM: STIW:	Set PSI table address Set inferior reserved channel mask Set terminal interrupt word
B8	Control PSI system	EIR: DIR:	Enable PSI system Disable PSI system
B9	Control channels	AIC: IIC: DIC:	Activate channel Initiate PSI on a channel Disable a channel
B1Ø	Read process capabilities	RPCAP:	Read process capabilities
B11	Enable process capabilities	EPCAP:	Enable process capabilities
B12	Read state information	RFSTS: RFACS:	Read fork status Read fork AC's

Bit #	Access Group	Access types included
B13	Control state	SPLFK: Splice fork FFORK: Freeze fork RFORK: Resume fork SFORK: Start fork SFACS: Set fork AC's HFORK: Halt fork WFORK: Cause wait GFRKH: Get a fork SFRKV: Start fork using entry vector
B14	Read entry vector	GEVEC: Get entry vector GCVEC: Get compatibility entry vector
B15	Set entry vector	SEVEC: Set entry vector SCVEC: Set compatibility entry vector

We note that, with a grouping like this that does not exceed 18 bits, the left half of the SYSFK table in the JSB could be used to contain the protection information.

3. Additions Required

The following additions are needed to support fork protection:

- Additional assignment of bits (in the B9-B17 range) in the capabilities word to allow the superior to protect itself from the inferior for the following access types:

  IIC - Initiate PSI on superiors channel

  GTRPW - Get superiors trap words

  RFSTS - Read status of superior fork

- Provide a PGET and PSAVE (detailed in separate memo)
- Provide a SFACL (Set Fork Access Control List) and a RFACL (Read Fork Access Control List) JSYS.

- Initialize the protection bits to ones
- Allow forks to commit suicide and send an interrupt to the superior to indicate that they have