Control cards involved in permanent file system
perm files stored on disc

common → as usual except for perm read only files
return → drop file or return 2 or 0
control → set read only bit for perm file bits

FMT/FSF, password, YH and RP, RBT chain saved on disc

**General File System**

- directory
  - allocation blocks
    - 1) IN blocks (large blocks)
    - 2) sectors (small blocks)
    - 3) entries (1 per object)
- objects in system: 1) files
  - 2) directory pages
  - 3) accounting blocks
  - directories (object)
  - allowed operations
    - rm, write, close, page
  - eg: system page
    - name → capability
    - job #
    - dir page

eg system page
| name → capability
| job #
| dir page
Object in ECS system

files
process
subprocess
map
capability list
event channel
allocation block
operations
P10 for MUX (TSS)

Output

User process - system privileged

Operation: convert a string of text to the output buffer
and return # of bytes converted

Points:
1) Type of conversion
   a) binary
   b) location of string - string packed 5 octet per word
   c) something about setting wake-up
   d) something else
   e) packed character

2) If acc buffer empty, then fill in empty words;
   if not, fill buffer and parameter 3)
   begin set wake-up to self

NOTE: the process will be freezing with the first empty buffer
      double - this interface should not freeze ever
when can't be interrupted
1) when, fixing stuff in E5 or cm which is
fielded by other components of the system.
In general, fielding "means modifying data"

This suggest that strict conventions be
established to identify the data sets and the
operators on data sets. This is necessary to
prove anything about the reliability of the system

Protection of data sets from the overlapping
updates may be either by 1) monitor mode
during modification of data. 2) flag on the data
which is set while in monitor mode (note
that this grin in trouble if the flag is never
reset). 3) the multiple flag procedure with a flag
cell assigned to each user of the data base.

Boolean procedure: ensure access (num-flags/myflag)
begin
for i := 1 step 1 until numflags do
if flag [i] then begin access := false;
end
end
end

flag[myflags] := true
for i := 1 step 1 until numflags do
if (i = myflags) V flag [i] then
begin flag [myflags] := false
end else access := false end
1) Representation of expected output process.

2) Not cause overlap.

force out of cell until process ends.

Accuracy never equiv to phi line.

This counts as # of capabilities pointing to it.
There should be a process to coordinate the transfer of information to and from the disc. When blocks arrive at a disc they should go to the proper file, mark the file block as "ready" and notify the required process when all the blocks it requires are in.

When a subprocess of a main process in the disc system desires a transfer, it should get space from a read from the disc allocator and mark the file block as "busy". It should notify the process described above and wait for a transfer request for a wake-up. If it does not, a process

Proposed "grandmother" multi-data words
Queue theory

for the inputs

- Time $D$ to clear a line
- Lines active at Poisson distribution
- Let $x$ = number which come in in $D$
- Probability of $I$ flags at $P_{D}$

$P_{I} = \sum_{i=0}^{I} \frac{e^{-x} x^i}{i!}$

$P(1) = 2 - \frac{2}{e}$

$m per T$

$D_{serv. time} < 3$ ms

$T = m D_{serv.} + \text{left to service} + T$

$p(1) = 2 - \frac{2}{e}$

$\frac{q p}{T}$
on break what happens
Do or don't B72?

LOAD CBUF
STORE INPUT
LOAD OUTPUT

INPUT
LPC  BBUF+1
ADM  BBUFFTB, CBUF
CPO  CMBUF
LAD  CMBUF
SNC  EMPTY
NNU  FULL
LAD  CBUF+1
STD  CMBUF+5
LAD  BBUF+1
RPM  BBUF+1, CBUF
CUD  CMBUF+1

LPM  BRTAB, CBUF
STD  TEMP
LDM  O, TEMP

BREAK

LOC  4008
ADD CBUF
LDP CBUF+1
SHH  4000 B
LNM  BRTAB, CBUF; LAM BREAK

BREAK

RPM  BRTAB, CBUF; LAM BREAK

LPM  BRTAB, CBUF
LPC  m&mmask
STD  3
RDM BREAK
LPM  GETHS, CBUF+1
LPC  m&ymask
NNU  3
SNI  1
MNR  BRTAB, CBUF
LMP  BRTAB, CBUF
SRH  1
MNR  m&uxful
PCR  10
read

Jmp = 2

10: c = c + 1

\[ 1^{10} \Rightarrow 1^{c} < \mathcal{F} \]

X3 PRINT \# YHHH

MPU SET BL addr of len \# Yc/hr

\[ = \text{addr of len} \]

6

SA1, B6

6X6, X1 \# X3

\[ 2 \rightarrow X6 \]

LX1

\[ \Rightarrow 5 \times 3 \text{ F} \]

X1 \# 5003

output

set ou

\[ \text{set output} \]

5

SBI, X1

SA4, B1 + buffers-

LX4

12

13X5, X4 \# X1

R \# X4

4B

BX3, X4 - empty

NZ, next

write to 205

next

\[ \text{get control and} \]

SA1

SB3, break line

5X3, B3

BX2, X1 \# X3

SA3, BL + break tab

BX4, X3 \# X2

ZR

\[ \text{jump if no break} \]
only one CPU and one central program may be allowed to execute an interrupt code. To prevent the same CPU from calling interrupt code before the previous call is completed the following algorithm should be used.

I)
1) check if $B_0 = 0$ or exchange package
2) initial interrupt (MST) (EXW)
3) set $B_0 = 0$ or exchange package once
   zero

II) in CPU interrupt code
   a) process interrupt
   b) start for non-zero $B_0$ or exchange package
   c) return to interrupted program (CET)

This algorithm works well on
without hardware changes and a double
exchange jumps to nonmemory+
from the interrupt

\[
\begin{align*}
\lambda &= \text{original } MA, \text{ in } A = \beta \\
\beta &= \text{initial process monitor stack} \\
\end{align*}
\]

1) initial MA, in $A = \beta$
2) CET $\beta$
3) $\beta$ contains MA = $A$
4) CET $A$

note: keep registers
one, missed address
$A$, not $\lambda$, are ok
version $A$, MA and $A$
must be reset
In critical code:

1) set flag
2) do crit stuff & reset 0\[0\]
3) test for existing interrupts
4) save current X'pack' from loc \[
\text{set } \text{new X'pack'} = x\]
\[
\text{set } \text{deferred interrupt flag} \leftarrow \text{0} + \text{loc} \text{-} \text{16}\]
5) save return info for new X'pack'
6) set \text{return info in new X'pack}
7) set \text{BE in deferred interrupt} \rightarrow \text{0}
8) copy deferred in current X'pack'
9) set \text{MM} in \text{5} = \text{current X'pack address}
10) \text{GET current X'pack}

In code initiated by 10:

1) \text{GET 5}

II. Interrupt routine

1) test for crit code entry

   current + routine

2) test to see if this is deferred exception

3) load address of X'pack' and do by
   if deferred cell

4) jump to spec code

Special case

1) \text{set } \text{BE in in current X'pack}
2) \text{reset deferred interrupt flag}
3) \text{return orig } \text{X'pack'} \text{ name } \text{(addr } \text{c1} \text{ = }\text{4)}
4) \text{return three return info at } \text{d}
1) remove waiting interrupt from list (actually one cell for each interrupt) and clear flag for the interrupt; clear chip cycle active flag

2) copy X pack at d to e

3) copy new X pack. IV to d
   set $MA$ at $d = 5$

4) CSE $d$
   in package N

5) set $BO$ at waiting interrupt ≠ 0

6) copy package at $\beta$ (waiting interrupt) to $\delta$; set $MA = \delta$

7) CSE $5$
   in interrupt

8) process interrupt

9) check for flag indicating this is a repeat performance - if flag in special code
   jump to special code

10) copy $\delta$ to $\delta = \delta$

11) set $BO$ at $\beta = 0$

12) CSE $d$
I. In critical code

A) if cell 1 occurs in step 1
 intervening rate that can
 set third cell = 1

B) copy X back at X

C) copy package IV to L

D) SET X

II. in package IV

A) set B0 at existing

interrupt = 1

B) copy package at

to (interrupt)

C) set MA at 5

D) SET 5

III. in interrupt code

A) process interrupt

and exit code

B) check to see if this

was a deferred

entry - if so jump

to spec code

IV. spec code

A) copy X to L

B) set B0 at 0

C) check if 60

D) SET 0

E) take return saved

in reg. at L (after first SET)
Outcode

data area in B1
return in B2

check in 1st and data area

SX1 BX + 1
S\A1 MY F\A\G

LE \circ A\D\I\E

SA1 MY F\A\G

B\X1 X2 - X2

SB3 LEX \L\I\ST

LOOP

S\A2 B3 + B8 \L\U\L\I\ST

NZ: X2, G\O\T\O\E

SB3 B3 + B1

EQ: 00, BQ, LOOP

JP: B2

G\O\T\O\E

SX3 B3

EX3 4

BX0 X3 \M\Y \L\I\S\1

S\A2 B3

SA0 X1

SB4 16

X: X + 0

ERROR inst

SX0 \M\Y \P\K\R

SB0 X1

SB 16

PX B4 + 10

X: X + 0

SX6 B - E

SA10 A0 + 9

SX7 13 7

SA7 10 + 10

S14 11

X: X + 2

25 \O\U\E \R\E\T\U\N

information
Code in interrupt routine

**Flag of critical event cycle**

- **Z**
  - **X**
  - **OK**

- **5X5**
  - **MYPACK**

- **SB2**
  - **myindex**
  - in the datum list

- **S A 5**
  - **B 2 + LIST**

- **EQ**
  - **1 3 0, 1 3 0, exit**

---

**DOX**

- set return & data area
  - **5X5**
  - **MYPACK**
  - as 1st word of data area & go to exit cycle

---

**return from exit cycle**

---

- **exit**
  - in extern proc

- **S A 3**
  - **X 3**
  - **MYPACK**

- **S A 5**
  - **B 2 + LIST**

- **EQ**
  - **8 0, 3 0, S A 5, exit**

---

**return**

---

**S K E X I T**

- **5 X 2**

- **L X 2**
  - **X**
  - **X**

- **S X 0:**
  - **X 2 + MYEXEC**

- **S A 1**
  - **B 2**

- **S H 1**
  - **X 1 = 1**

- **S B 4**
  - **X 1**

- **R E**
  - **B 4**

- **S R E X I T**
  - **exit**

- **S A 1**
  - **A 1 + 1**

- **S B 3**
  - **X 1**

- **J P**
  - **B 3, 0**

---

**B X 5**

- **X 4 + X 4**

- **S H 3**
  - **LIST # 1 3 2**

---

**return (next X pack)**

---

**return colors of data X pack from WEIPER**

---

**get return**

---

**return**

---

**return (waiting flag)**
8/10/68

To refer to sublp

Don't use caps for sublp - get rid of C! subproces
use -CAP for class code
index - process

Starting sublp
by index in c! for process

proc & subprocess 0 means self (calling sublp)

sort of P Pointer or much a call

Create proc C - list - from C - list without class code & index
wrote

talk of proc C - list of other sublp by index to class code

proposed cpl
S/MAP
MRA - map PA - MFL

with proc... - only Obj group C-list
subprocess

(1) (conf. class code)
(2) or within CPP - CFL
MRA - MFL of such sublp
reading within process -- relative to CRAD -- CPC

modify within process -- with new power domain

process - object + subroutines

+ object + subp domain
need class code

process uses: (can have subprocesses)
genie & file blocks (I read only)

* even try class: cycle
  c-list
Design Error

The solution

1. Design error / error in ride (in %)

2. Error class / number

Don't process errors in test as good for all but set which are w/ error at all times of prove normally

Error error # rules to fields error above

Error class

Error number
Action: Set bit or clear bit in mask

2. D. [continued]
   - D. place word (fixed)

2. End of list

Action: Multiple

2. End of action page

4. Process list and map

Action: D. copy on event channel
- Insert if P event

- Increment P when event comes in

Subprime error
- Call NPD if specified when mask activates
- Respond

1/24/68

1. read reg
2. set reg
3. current proc c-list <-> reg
4. copy proc c-list

sublp has cap for a c-list

actions - all changes are to ECS copy - if change to local copy, copy changed ECS copy to local

subprocess

organize subprocesses in a chain of sublp which are likely to reference address space of each other

each sublp has it's own map - all maps back up the chain are loaded - also current map process

RA
FL
In update

Pt to another subp (back ptr.)
if sub-map under
C-list (like FC of map)
else call

proc
set map under
C-list (like FC of map)
else call

error stuff
active flag

set map under
C-list (like FC of map)
else call

error stuff
active flag

(set of maps in felds/stack)

action (modify subp)
  only change P-count
  C-sexp map or map

All other changes to subp in subp

action on map

may update on map of descendants

to pt at entry say
1) subp that backpoint itself
2) entry in map (C-list)
Type 1 error process:
Type 2 error process:
P - counter

Scan up tree to find someone to absorb the error

Call Stack:
1. return
2. B count
3. top of path

Subprocess has entry point (P counter)

Wayne becomes "call that subprocess"
splice identifier out of the subprocess call stack

Event channels

If event on B:
   event on A --> P
   event on B --> A --> P

Path modules:

Final path:
   new B - you
   new = if you ask Path (oldt)
   oldt = oldt
   oldt
   you
Process
  look at i)
  process
    (destroy )
    process
      interrupt
        process
          Class code
          D'index
          D'sec type
          D'offset
          mun

pulp
  create
    operation
    call
      Class code
      D'subindex
      call by op
      create
        Class code
        D'exp
        
        Code 3 check pt
        p.pack
        c'cliof
        D'address space sig.
        D'LENGTH ofubar

C'list
  get
    Class code
    D'sub op
    get C'c'list

Miss
  address
  D:'
Call
Jump
Interrupt
Loop

Premise
Condition
Result
Source of jump
Entry point

Insert in call chain after 1st call met
on path to root (call cell)

Event channel of

1. Hang on call
2. Delay on call channels

1. Set to C far E C
when return from block, mount clearance from all mount channels

101

[Diagrams and notes scattered across the page, difficult to transcribe accurately.]
Disc System

- objects: process, file, etc.
- data: codes, characters, allocated disk blocks, ECS
- operation: release, ECS

Swap Algorithm

- Proposed: panic or attack, block of ECS file
- Dry running on map
- Etc., directly (prefetch)

To swap in: compute ECS space needed
- By counting attached block less stuff in
- Charge for stuff attached
- Swapped files targeted
- in disc system
dwell escaped pressure is 2 categories

1) doesn't block
2) blocks on dwell block

tell by history and load configuration

to know cost to calculate cost of swap to from 231

good guys keep swapping down
If forward error fails, fail

put error code into forward error
put error address

end each stack entry to room for
pointer (name) to operation + count

operation = op + op + param

RETURN causes next level of operation in the
preceeding stack entry

RETURN is invalid

get parameters for number of levels specified
each item new AF list is built
within fast procs read a subprocess by name or by index

change error message
print status

backward errors

scan full path to top of call stack; current "top of path" checking in backward ESF

this error removes the the generator from call stack

if not found if fails it is reported

determine points -> par

interrupt -> only interrupt someone below you in tree

in ordinary call -> pass origin of caller's address space relative to the rest address space

caller

if already at top of call stack
Protection with respect to scheduling.

On entry to ECS system - the proper monitor bit in set in CR and will set ECS app flag in system core.

If an interrupt causes the scheduler to run - the interrupt will make sure the monitor bit in CR is set but will not set the ECS app flag.

The scheduler will check the ECS app flag - if it is set, it will queue its request, change the CR to S, retic to a jump to the scheduler at an entry point which will first restore the instruction at S, retic to a CR, P1 and expect - schedule all processes in its queue, jump to S, retic or initiate swap as needed.

Interrupts must be locked out while scheduling since an interrupt may cause scheduling.
The swapper protection.

Before swapping in a process, it must be marked as "in-core." To do this, interrupts must be locked out.
In general, it is assumed that calculated ECS addresses remain valid. The actions which invalidate calculated ECS addresses are:

1. garbage collection
2. re-allocation of a process (for interruption and scheduling)
3. object destruction

Other areas of concern in ECS:

1. storage allocation
2. scheduling: may be handled by having one scheduler handle all processors and have queues and mailboxes in ECS for communication.
3. interruption: an interrupt may not be presented if one is pending in Process RD storage also not allowed if process is executing
4. event channel: only one person may access with a particular channel at a time - included here is setting of "wake-up waiting" in process (and presentation to scheduler)
5. handling reference counts on file blocks of maps compiled to file blocks
6. allocation block modification
in GR — a mon mode list for each processor
and a linear ordering of prios listed above
such that the ordering does not place any prios
which could be demanded by an interrupt
routine before a prior not used by interrupts.

To do anything with this, all processes
in the mon list must be turned on. Thus, to
do the things mentioned (garbage collect,
etc.) the processor doing it must not
all mon lists to look out for access by
other processors.

For all other prios, there is no processor
may try to get any privileges at
the moment, either other in the prior
ordering. However, the priority of the
processor already has. There
are no restrictions on releasing
prios (note: the mon list is not
considered a prio)
I. Storage allocation

A) Interact with other storage alloc.
B) If obj $\neq$ NOT search
C) File index or data block $\rightarrow$ file access

NOT used by interrupt routines

Suggest ignore C$ \rightarrow$ only need to lock out all other allocation - one prior 1st allocation

II. Scheduling

A) Scheduling may result in a process being swapped in $\Rightarrow$ interferes with interrupted

B) Must lock out process - interrupts to work process as "classic"

Scheduling algorithm creates list in ECS $\rightarrow$ head of list

Interrupt process to resume in either machine $\rightarrow$ then

Scheduler must not run while any MM code $\rightarrow$
III. Process Interrupt

A) Only one processor may try to present an interrupt to a process.
B) Interrupt may cause event channel action if process is blocked.
C) A process may not be swapped in during interrupt action → no scheduling.
D) If process is blocked, event channel action cannot be allowed until "wake-up-waiting" has been set.

IV. Event Channel

A) Only one event channel is involved at a time.
   The dizzle of "wake-up-waiting" is prior.
B) Only one processor may work on a particular event channel at a time.
C) CPU interrupt not allowed if EC has been marked "busy".

Suggest two prior:
1) Mark event channel busy.
2) Dizzle "wake-up-waiting".
Delete file if not open and attempt to open or create it.

Reference count: If not previously opened, then ECS incarnation is created; clear decisions.

Ref count: If goes to 0, ECS incarnation is destroyed.

Open socket or capability for the file.

ECS object: class code, 2 parts: temp & perm,

Perm used to identify user or user group & checked
by disk sge to be sure it agrees with current perm access.
Temp part used to build edge to allow access.

Open file by list of temp parts as file.

ECS acting: take old class code & set same field.

(todo option list)

30

Class code: temp

perm class code

4x(1)
ECS open

return capability for 1st object

allocate block

params: 1) cap allocated block
2) loc in c-list to return

fixed place objects in ECS

1) event channels
2) files

new ECS action

change unique name of object

Two kind object can hang on event channel => 2 entry point