FILE SYSTEMS

file chains (8604)

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 to eP 0
CONTROL → set read only bit +/or perm file bits

FNT / FST, password, 4th wd FNT, RBT chain saved on disc

GENERAL FILE SYSTEM

directory

objects in system 1) files!
2) directory pages
3) accounting blocks (allocation) →

allocation blocks
1) 1/2 tracks (large blocks)
2) sectors (small blocks)
3) entries (1 per object)

capabilities ← object
allowed operations
eg write
change page
etc

directory page

name → capability

eg system page

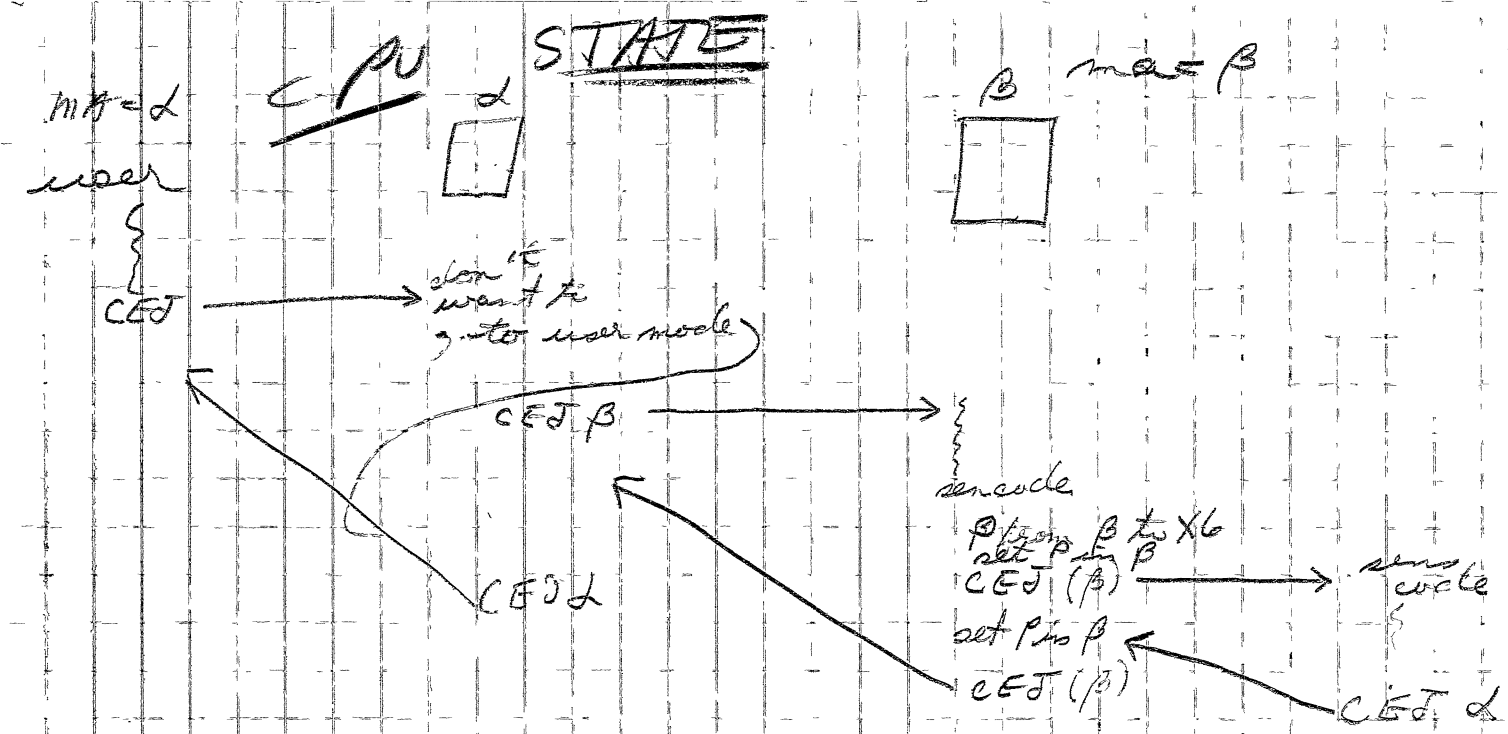
| names → capability
| Job # dir page

24/60/640

Objects in ECS system

files
process
subprocess
map
capability list
event channel
allocation block
operations

STATE



user type

PPU interrupt - not go to user mode

LDC	Xpack
CRD	CM
LDN	1
STD	CM+4
LDD	XPACK
CWD	CM
MEJ	
CRD	CM
LDD	CM+4
EJN	
LDC	XPACK
JDN	

to go to user

~~state 1 in B0~~

CPU

~~test~~ if monitor > 50 ms
set B0 = 0
< 50 ms
if monitor test B0 ≠ 0 then

PPU

read B0
B0 ≠ 0
set B0 = 1
MEJ
read B0
B0 ≠ 0
set B0 = 1

Howard & Burtell **OLP**

user system objects → file — addressable [illegible]
→ processes — scheduled
→ subprocesses
→ operations

capability [illegible] to object 1) object
2) type
3) options

capability of OBJECT

→ capability list
→ [illegible] page → pairs
name ↔ capability
→ [illegible] ([illegible])

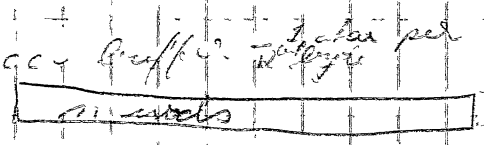
operation 1) [illegible] of pairs
type of [illegible] ↔ option mask
2) procedure ([illegible] abs [illegible] list [illegible])

[illegible]
[illegible] process
a) no map [illegible] change (RA - FL

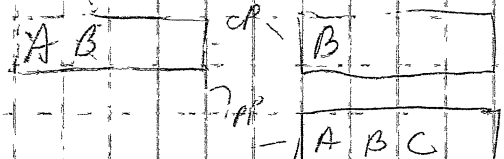
[illegible] of maps
collection of subprocesses

Aug 26

PPU for MUX (TSS)



when should wakeup get sent?
1) output — on almost empty buffer
2) input — on every ctrl char detected by central



output

user process — system privileged

operation: convert a string of text to the output buffer & return # of words converted

operands: ~~1) type of conversion~~
~~a) string stored~~
2) location of string — string packed 6 char per word
~~3)~~ 3) something about setting wake ups
4) [illegible] deferred echo [illegible]

Method

if ecs buffer empty then convert to fill on empty words; convert to fill ECS buffer;
if not whole string out and parameter 3) then
begin set wake up to self;
block and;

NOTE: the PPU will be fussing with the first ecs buffer word — this interface should not fuss with it

on input ^ppu^ fill as usual

if break char then ^ppu^ fill rest of wd w/ empty
and call cpu w/ line number to do wake up

PPU load break table from CM on every char 17 µsec

```
2       LDC   central tables
4       ADD   line #
10      CRD   tables
14      LDM   Break tabs, tables
17µsec  STM   break table search
```

SPECIAL CASE
ALL BREAK

(code modification)

Echo   if not echoed   set bit 12 ⟶ don't echo if 1) ... 2) flag non zero

```
        LPD   char       2
        LMC   4000B      4 µsec plus testing
```

countup counter in data w/word

```
   LDD   BUF        2 ) if not
   AON   1          3 ) echoed
   STD   BUF+1      5
   LDD   CHAR
   STD   BUF
   LDD   BUF+4
   LMN   EMPTY
   ZJN
   LDC   TABLES
   ADC   LINE #
   CWD   BUF+1
   UJN         out
   LDC   TABLEL
   ADC   LINE #
   ADN   1
   CWD   BUF
   SBN   1
   CWD   EMPTYWORD  — wipes out echo count
```

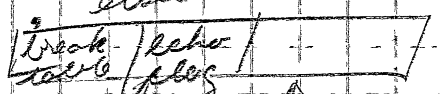
move to new buffer

input buffer

echo count — always count up

maybe the rest of input process

elsewhere



process echo count

ECJ — MEJ

AUG 27

when can't CPU be interrupted

1) when fiddling stuff in ECS OR CM which is fiddled by other components of the system

In general "fiddling" means modifying data

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

~~When can't be interrupted~~

Protection of data sets from 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 gets 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 acquire access(numflags, myflag)
begin for access := true
      for i := 1 step 1 until numflags do
          if flag[i] then begin access := false;
                               goto end end

      flag[myflag] := true
      acquire access := true
      for i := 1 step 1 until numflags do
          if (i ≠ myflag ∨ flag[i]) then
              begin flag[myflag] := false;
                    access := false end;
end : end
```

file system

1) representation of swapped out process
have enough to run - discard

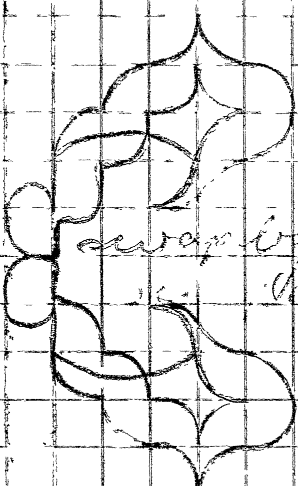
? if disc system [illegible] to either [illegible] remove [illegible] [illegible] ref cnt ≠ 0 [illegible] perm object)

add option [illegible]

[illegible]

1) [illegible]
2) not cause swapout

- force out - force out process call



swap by [illegible]

ref counts of # of capabilities pointing to it

Disc system

There should be a process to coordinate ~~[illegible]~~ the transfer of information to and from the disc. When blocks arrive, it should attach them to the proper files, mark the file element as in ~~use~~ core ~~busy~~ and notify (wake up) the required process when all the blocks it requested are in.

When a subprocess of a user process in the disc system desires a transfer, it should get space (for a read) from the CCS allocator and mark the file ~~block~~ as "busy". then it should notify the process described above its need for a transfer & wait for a wake up — then it ~~should~~ may proceed —

18 [illegible] 24 [illegible], 16 block size, 18 disc file
[illegible] file 17

Proposed
event channels may have
multi data words

Queue theory for tty inputs — TTY terminals — 9/3

time D to deal w/a line

lines active at Poisson distribution

→ last line

let α = number which come in in D

probability that I lines ready at equilibrium

prob of j new line in D

$$P_j = e^{-\alpha} \frac{\alpha^j}{j!}$$

prob of I flags at D time

$$q_I = \sum_{i=1}^{I+1} q_i * P_{I+1-i} + q_0^t * P_I$$

---

n DCT

D_{OUT} service time

T < 3 ms

$T - nD_{out}$ is time left to service tty

$$\frac{T - nD_{DCT}}{D_{tty}} \leq \# \text{ of tty flags allowed in } T \quad (= 9, \ T = 3\text{ms})$$

$$p(I) = e^{-\alpha} \sum_{i=I}^{\infty} \frac{\alpha^i}{i!}$$

(3.7 tty char/msec 3.7/3 TTY 1/msec 9 P_{TTY}

BRKRET

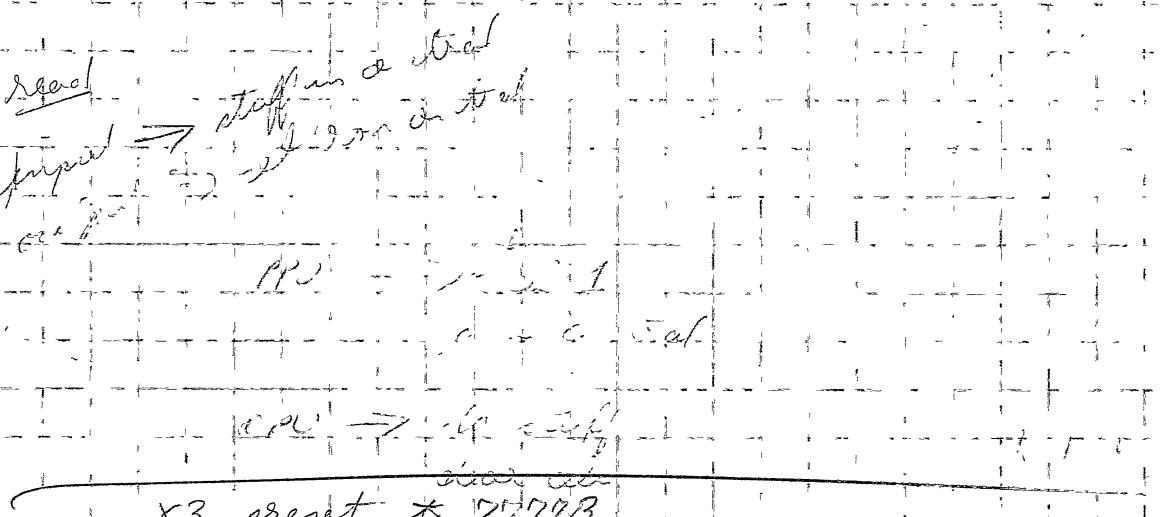
on break - what happens to data in cm?

```
      FNC
      ACN
      LGN
      LPN
      ZJN     --->
      LDN  2
      LAM  CBUF
      DCN
  14

      LDD  CBUF
      SHN
      ZJN  INPUT
      LJM  OUTBYT

INPUT LDC  BUFBASE  ~~BUFCMBUF~~+1
      ADM  BUFADTB, CBUF
      CRD  CMBUF
      LDD  CMBUF
      LMC  EMPTY
      NJN  FULLI
      LDD  CBUF+1      <- LPC 177B / STD CBUF+1
      STD  CMBUF+5
      LDC  BUFBASE+1
      ADM  BUFATB, CBUF
      CWD  CMBUF+1
  32+4

      LDM  BRKTAB, CBUF
      STD  TEMP
      LJM  0, TEMP

~~BRKRET~~
```

not included
1) word full ÷ 60 × 1/5
2) break 150 × 1/20
3) start buffer 100 × 1/100

```
eg break
      LDM  BTAB, CBUF+1
      LPC  MYBMASK
      ~~ZJN~~  ~~*+3~~ K
      RJM  BREAKIN
      LDM  BTAB, CBUF+1
      LPC  MYEMASK
      NJN  *+3
MYBR  LDM  BRKRET        no echo
      LDM  BRKTAB, CBUF
      SHN  6
      MJN  MYBR
      SHN  1
      MJN  MUXFUL  -->
      FCN
      ACN
```

```
LDC  400B
ADD  CBUF
OAN
LDD  CBUF+1
SHN  2
ADC  400B B
OAN
DCN
LDC  2000B
RAM  BRKTAB, CBUF ; LJM BRKRET
```



X3 preset * 7777B

PPU SET B6 addr of lesi # + char

6	SA1	B6	
5	BX6	X1 * X3	char → X6
6	RX7	12	lesi #2 → X1
6	LX1	5X3	X1 − 400B
5	SB1	X1	output
	SA4	B1 + buffers-	get our word
	4X4	12	
	13X5	X4 + X1	
	RX4	48	
	BX3	X4 - empty	
	NZ	next	

write to 205

next	SA1	B4 + link	get control word
	SB3	break disc	
	SX3	B3	
	BX2	X1 * X3	
	SA3	A6 + break to 6	
	BX4	X3 * X2	
	ZR		jump if no break

Interrupt routines

9/4

only one PPU & no central program may be allowed to initiate interrupt code. To prevent the same PPU from calling interrupt code before the previous call is completed the following algorithm should be used

I) in the PPU
a) check for BO = 0 in exchange package
b) initial interrupt (MEJ)(EXN)
c) set BO in exchange package non zero

II) in CPU interrupt code

a) process interrupt
b) wait for non zero BO in exchange package
c) return to interrupted program (CEJ)

This algorithm works with or without hardware modification — without hardware changes a double exchange jump is necessary to return from the interrupt

α = addr Xchange package for interrupt
β = " " " for dummy CEJ to insure proper monitor mode

1) original MA in $\alpha = \beta$
2) CEJ β
3) β contains MA $= \alpha$
4) CEJ α

note: leave registers etc unused up in α; RA & FL are ok however MA in α must be reset

SENSITIVE CODE PROTECTION

private

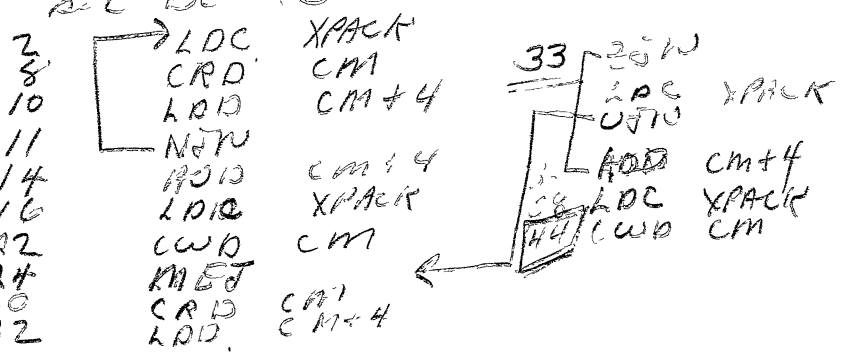
I

II

IV

V

VI



I In critical code

1) set flag
2) do crit stuff & reset flag
3) test for waiting interrupts
4) save current XPack from loc 4
5) put new XPack in 4
   set MA = 5 ≠ deferred interrupt code - 16
6) save return info in new XPack
7) set BO in deferred interrupt ≠ 0
8) copy deferred in current XPack
   to loc 5 = deferred interrupt - 16
9) set MA in 5 = current XPack addr
10) CEJ current XPack

II In code initiated by 10)

1) CEJ 5

III Interrupt routine

1) test for crit code copy
   interrupt routine
2) test to see if there is deferred interrupt
3) load addr of XPack and index
   of deferred call
4) jump to spec code

IV Special code

1) reset BO in current XPack
2) reset interrupt deferred flag
3) activate orig in XPack (can do 4 ≠ 4)
4) return thru return info at 5

original ~~copy~~ X pack at α
~~wait~~ interrupt at β

1) remove waiting interrupt from list (actually one cell for each interrupt) and ~~place~~ ~~replace by~~ set α for flag for the interrupt; clear ~~entr~~ exit code active flag

2) copy X pack at α to Σ

3) copy new X pack N to α
set MA at $\alpha = \delta$

* 4) CCT α

5) in package N
set BO at wait; interrupt $\neq 0$

6) copy package at β (wait; interrupt) to δ; set MA = δ

* 7) CCS δ
is interrupt

8) process interrupt

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

in special code ← clear flag — after getting α from flag

10) copy Σ_1 to $\alpha_0 = \lambda$

11) set BO at $\beta = 0$

~~12) CCT α_N~~

Action	M MODE	U MODE	PIH
I in critical code	M	U	α
A) if self indicate interrupt waiting reset that cell & set this next cell = α	M	U	α
B) copy X) pack at α to Σ			
C) copy package N to α w/ MA = δ	M	U	α
D) CET α	U	M	δ
II in package N			
A) set BO at waiting interrupt $\neq 0$			
B) copy package at ω (waiting interrupt) to δ			
C) set MA at $\delta = \alpha$			
D) CET δ	M	U	α
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 Σ to α			
B) set BO at $\omega = 0$			
C) reset all flags			
D) CET α	U	MA	2
D) take return saved in reg at α (before first CET)			

Critcode

data area in B1

return in B2

Hook in 1st wd data area

```
        SX1   B0+1
        SA1   MYFLAG
        [ do critical
          stuff ]

        BX1   X2-X2        clear flag
        SA1   MYFLAG
        SB3   LENLIST
LOOP    SA2   B3+BEGLIST
        NZ    X2,GOTONE    check for waiting entrys
        SB3   B3-1
        EQ    B0,B0,LOOP

        JP    B2           normal return

GOTONE  SX3   B3           X2 contains addr of interrupt routine
        LX3   4            X pack
        BX0   X3+MYECS1
        SA2   B7
        SA0   X1
        SB4   16           write out current X pack
  +     WE    B4+0
        error instr
        SX0   NEWPACK
        SA0   X1           get new package
        SB4   16           MA in new package = DELTA
  +     RE    B4+0
        error instr
        SX6   B7           SET X1, X2 for X pack to B1 and X2
        SA6   A0+4
        SX7   B2           eg save return
        SA7   B0+10        information

        SB4   11
              13
```

~~Instruction code~~

```
SA3   X2
BXB   X3+1
SA6   A3

SX0   X3+MYEGS2
SA0   X2
SB4   16
CE    B4
error exit
SX0   X3+MYEGS2
BA0   ALTPACK
SB4   16
RE    B4
error exit
SA4   ALTPACK+6
BMX3  36
LX3   36
BX4   X4*X3
SX3   A1
LX1   36
SB5   X4+X1
SA5   A4

SB4   X1
CEJ   B4
```

next set B0 ≠ 0 ↑
new ~~pointer~~ interrupt package

next copy away interrupt X pack

NOTE: ALT PACK is just before the package of the waits routine

set MA in alternate package to current package addr of current X pack still in X1

package control from current X pack loc

```
SB4   ALTPACK
CEJ   B4
```

code in interrupt

Code in interrupt routine

```
SA1   flag   of critical exit code
ZR    X1, OK
SX5   MYXPACK
SB2   myindex   in the waiting list
SA5   B2 + LIST
EQ    B0, B0, exit

OK    set return & data area
      in MYXPACK as 1st word of data
      area & go to exit code

      return from crit code

      rest of interrupt proc
      SX5   MYXPACK
      SB2   MYINDEX
      SA3   B2 + LIST      check to see if this
      ZR    X3, exit       was a deferred req
      EQ    B0, B0, SPEC EXIT

      SA3   X3             reset B0
      MX4   42
      SA5   X3 ≠ X4
      SA5   A3
SPEC EXIT  SX2   B2        retrieve orig X pack
      LX2   4
      SX6   X3 + MYECS1
      SA1   B2             retrieve addr of orig
      SA1   X1 = 6         X pack from MULTPACK
      SB4   X1             in X1
      RE    B4
      error exit
      SA1   A1 + 1         get return
      SB3   X1             return
      JP    B3, 0

      BX5   X4 - X4
      SA5   LIST + B2      reset waiting flag
```

9/19/68

to refer to subp

don't use cap for subp - get rid of c: subprocess
use - cap. for class code
- index
- process

to find subp
by index w c: bar process

note: subprocess '0' means self (calling subp)
what of P-counter or such a call

create pre-C-lists - from c-list w/ class code & index
route

talk of pre-C-lists of other subp by index & class code

prepare CRA
SRA
MRA - map RA MFL

with process cap - to subp any obj { map / pre-c-list / subprocess

(1) { map in its own class code

(2) or with CRA-CFL
MRA-MFL of own subp

8/19/68

__reading__ within process – relative to CRA – CPC
MRA – MPC

any process – process
obj index + subin...

__modify__ with process — with subprocess domain
[illegible] what [illegible] want

+ outside sub p domain
need class code

process creation (w/one subprocess)
__give__ 2 file blocks (1 read only) ← proc map
P-counter
class code
C-list

async error

two ~~fields~~ pattern ① async error priv (more bits ⇒ less important)

② error class, number

don't process error in tcb (subp w/ all bits set which are w/ error and others proc normally

[illegible]

---

async error — ① subp to field error (index)
② error class
③ [illegible] number

~~Clear result~~

CCF k field — if k > 0 then RA + k for parameter

-9 < k < 0 then RA + B(-k)

-18 < k ≤ 9 then RA + X(-(k+9))

return: if single data word return ⇒ X6

if 2 word return ⇒ X7

✓ process waits queue. Occurs process queue

parameter:

P1 ⇒ Pointer parameter

C: & P: repeat

AP = actual parameter in

IP = what the user supplies (input param)

parameter specification

PS

SPRA } subprocess ref address

SPLF

EPE — enterprise environment

sub proc

addr's of 'os
error proc mask
error subp number

action — set bits in error proc mask
para
1: subp
2: D: ... wd
3: D: ... wd (fixed)
2 and 3 ~~AND~~ 'or' in mask

action — turn off bits

action — ~~multiple~~ multiple ... turns off action flags

---

capabilities proc-c-list and map

---

action → hang on event channel
— decrement P-counter

→ increment P-counter when event comes in

---

subproc error

call subp # specified when mask admits

...

---

cap ~~for proc-C ... a given P ... is whose ... a subp ... be ... have ... more bits on a fixed or default parameter~~

9/24/68

c-list req

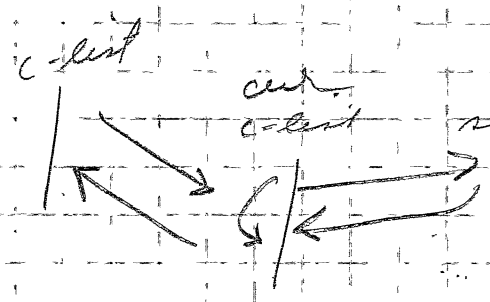
① read req

② set req
mask

③ current-proc c-list ⟷ req

④ copy p-c-list

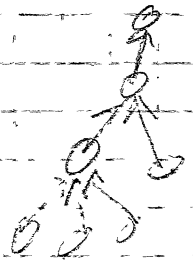
subp has cap for a c-list



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

subprocess

organize subprocess in a chain of subp which are likelely to reference adata space of each other



each subp has its own map - all maps back up the chain are loaded - also current mapsubp

Procs has 0 current active procs
~~current map of proc~~

RA
FL

9/25/68

in subp

ptr to another subp (back ptr)

~~map~~ map index (C-list) (like FL of map)

class code

p-counter

error stuff

active flag → return link

head of map chain index (a subp)

error selection mask

error subp #

(concat of maps in free map)

action (modify subp)

only change P-counter

~~replace map or map~~

all other changes to subp involve destroy & re create

action on map

may operate on maps of descendents

to get at entry say ① subp that backpoint to self
② entry in map (C-list)

on call
orig ~~pass length~~
map, clist
& memory space

SOBP obtained
class code + subp index #

9/27/68

sync error process

| ~~type 1 → spec subprocess~~ P-counter
| ~~type 2 → spec sub…~~

scan up tree to find someone to direct the error

call stack ① return in
② P-counter
③ top of Path

(to call - must leave room in stack at # of ancestors ← space in stack

subprocess - has entry point (P-counter)

async becomes "call that subprocess"
> splice identified subp in the subprocess callstack
> immed above 1st entry of Path (splice)

note: to graft in ...
is ok for succeeding entry if ok for graftee

Event channels

long event channel
(or event channel)

if EC A is hung on EC B &
process P on A

event on A → P
or event on B → A → P

timer ⟷ event channel ; put event on event channel in n ms

act. path is self & root

full path - new B - you
newt = if you ∈ Path(oldt)
then oldt
else you

9/27

Process — look at : i) c: process
create
(destroy : -)
interrupt — c: process

c: class code
D: index
D: err type
D: error num

---

subop — create operation & call — c: class code
D: subindex

call by op

create — c: class code
D: operation

c: code } back ptr
D: subop }
c: "class"
D: address space obj
D: length of msg

---

c-list — gen — c: class code
D: subindex — get C: c-list

---

mem — create — D:

9/30

subproc activation

	Pcounter	~~param~~ ?
call	fixed	actual param
jump	fixed	source of jump
interrupt		
error	fixed	e'or data
return	call stack	none

← interrupt entry point

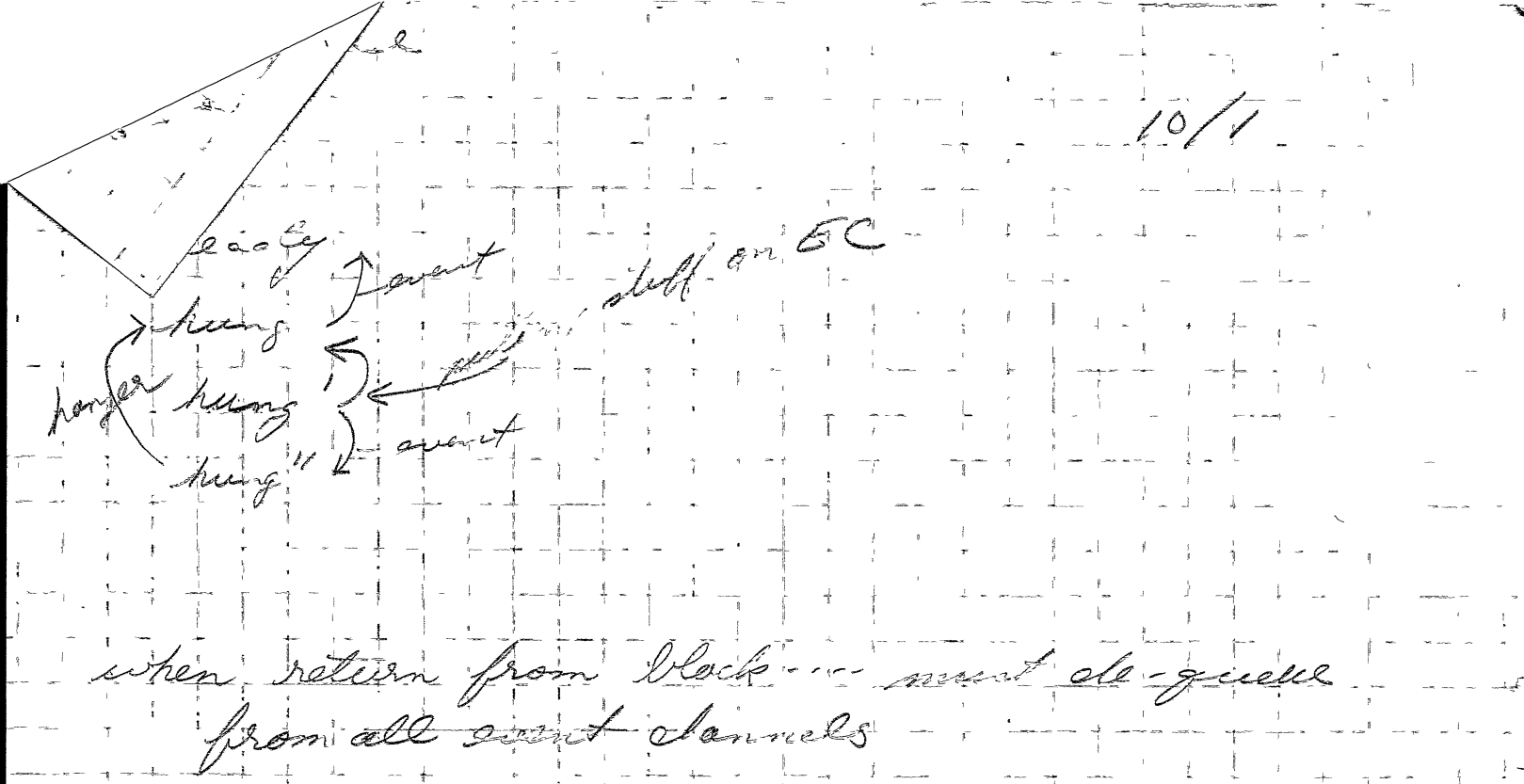
insert in call chain ~~before~~ after 1st ... on path to root (≠ self)

---

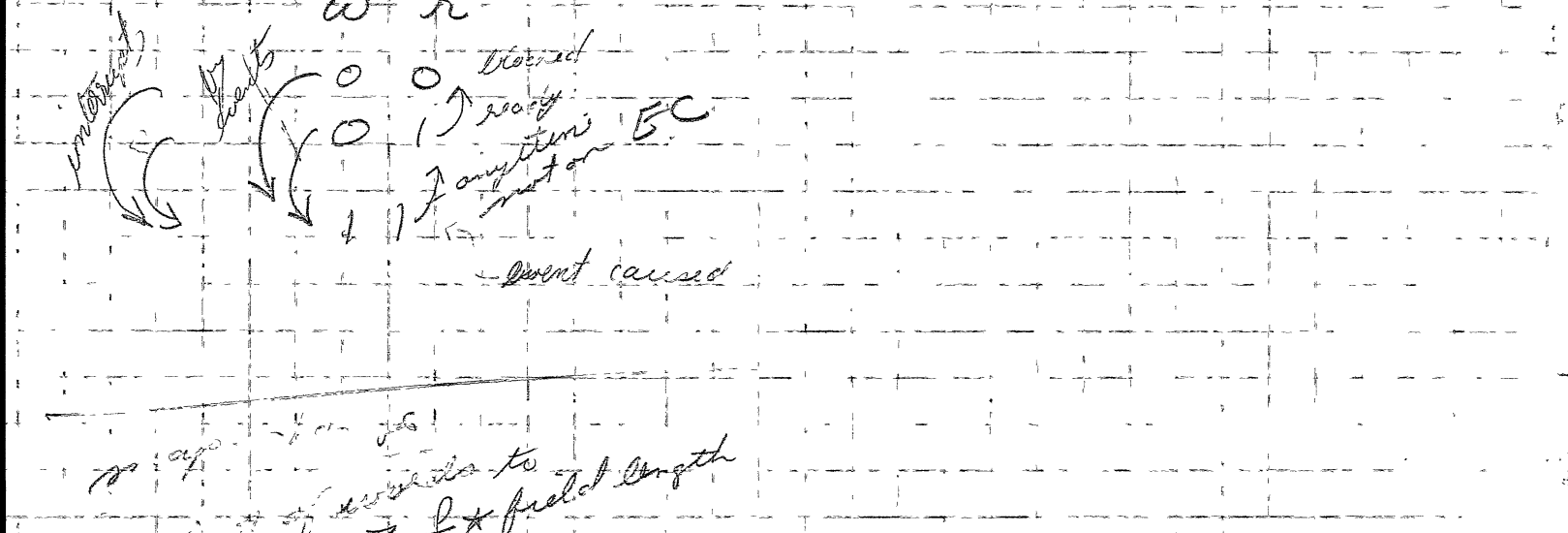
event channel op

① hang on e.c.
② change on in event channels
  [(1) n
  (2) ptr to C-for e.c.

10/1



when return from block ... must de-queue from all event channels



~~Dealing Process~~ 10/1

get block of words → don't get state running in CPU

Interrupt fails if interruptee is running on other CPU (notify)

→ or has wake-up waiters)

① interrupt right away
② " waiting
③ failed

Allocation
- event chan
- Proc
- C-list
- file
- alloc blk
- oper

— must be able to enquire status of a file
— get block to file
— diddle alloc block when moving file block

authority to construct obj of various types w/ memory amt avail per type

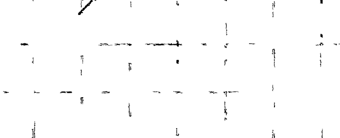
ECS in 2 cat - A - B

↑ available & disk   ↑ - non swap

- create w/o worry $
- add mem
- acct $

---

says you have X words of ECS
capability lets type of obj to construct
allocation block owned by an allocator



in alloc block + current total keep max!

each obj:
- pt to Master Obj table
- 2 ptr for alloc chain
- current size
- last access time

10/3

Disc System

objects → process
file
event ch
c-list
class codes
directories
alloc blk ⇄ disk
ECS
operation ⇄ ECS / DISC

same as additional
fixed
swapped ⟨ in / out attached block

swapped ⟨ attach count
(1) in count
(2) in → out

dependency
disc obj
files

disc space
(1) perm
(2) temp
Tape files
ECS space — perm / temp
$

dire
alloc blk (disc, ECS)

directory → fixed length seq directory entries
dir Entry (DE) : pair
name
disc capability for one of 1) swap file
2) class codes
3) directories
4) alloc blk (disc)

swap algor

swapped process attach 1 block of disc file
(1) by putting in map
(2) ... (prefetch)

to swap in (1) ~~compute~~ compute ECS space needed
by counting attached block less stuff in
(charge for stuff attached ⇒ swapped files charged
in disc system)

divide swaped processes in 2 categories

(1) doesn't block → [illegible]

(2) blocks on [illegible] EC → run right away

tell by history + [illegible] [illegible]

to throw out → calculate cost of swap to/from ECS / disc

— good guys (1) — keep swapping down

bad guys (2)

10/3

10/7

If forward error fails - kill

~~put error fails~~ - send to forward error

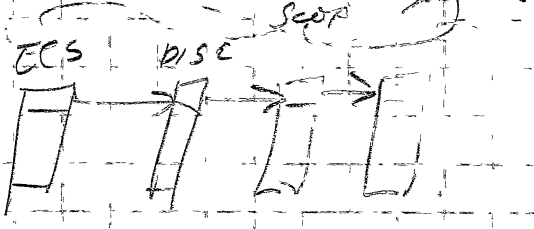
put error address

w/ each call stack entry is room for pointer (name) to operation + counter

operation = op + op' + params

FRETURN causes next level of operation in ~~the~~ preceding stack entry

RETOP is used



get parameters for number of levels specified - each time new AP list is built

within ~~subprocess~~ process read a subprocess by
class code & index

change error masks HES
Examine operation
punt
~~backward errors~~

scans ~~up~~ full path to ~~top of call stack~~ current "top of path" checking in backward ESM

this error removes the ~~the~~ generator (subprocess) from call stack

if not found it fails & is reported

dominate punts → par

interrupt — only interrupt someone below you in tree

---

in ordinary call (+error) → pass origin of callers address space relative to the new address space

caller → origin of caller in full c-list + map
~~names of caller~~

if already at top of path give up

10/4

operations of full map
full c-list
full memory space

explicit op map
by subprocess

explicit op c-list
by cop

subprocess changes

params

call	op + param
jump	op + param
punt error	param → class, → num + address (bottom of CET) orig (~~do op to check params~~)
error	class & ~~num~~ num
interrupt	none

actual param — 1 ⇒ call; origins (3); aps of op

error — 4; e class; e num; origin

jump — 2; origin (3); APs of op

punt error — 3; class; number; 3 origins ←; address of params for orig op causing punters to start (not relocated)

To create obj

(1) c: alloc blk for fixed ECS space

(2) specify right stuff

EC: event queue size

c-list: length

Operation: size of / local stuff

subprocs; size of [illegible] [illegible] [illegible] back pointer entry pts

process: stack size (may be changed?) one map ← two entries; IC-[illegible] (specify one subp); start [illegible] and [illegible] subprocess

files: shape, block size

destroy files

reference count on blocks of maps ~~refering~~ entries referencing

Protection w/ respect to scheduling

On entry to ECS system - the proper monitor bit is set in C.R. and will set ECS sys flag in system core

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

The scheduler will check the ECS sys flag - if it is set, it will queue its request, change the CEJ at S.rete to a jump to the scheduler at an entry point which will ① first restore the instruction at S.rete to a CEJ B1+P. xpack, ② schedule all processes in its queue, ③ jump to S.rete or initiate swap as needed.

Interrupts must be locked out while scheduling since an interrupt may cause scheduling

The swapper protection

Before swapping is a process, it must be marked as "in-core" → to do this, interrupts must be locked out

Event channel

Must lockout PPV interrupts and other event channel accesses

12/12

ECS access protection

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 + sched of E.C.)
3) object destruction

Other areas of concern in ECS →

NO I 1) Storage allocation

~~NO I~~ 2) scheduling? - may be handled by having one scheduler handle all processors and have queues + mailboxes in ECS for communication

NO I 3) Interruption → an interrupt may not be presented if one is pending in process RO storage also not allowed if process is executing

4) Event channel → only one person ~~at~~ may wait on with a particular event channel at a time → included here is setting of "wake up waiting" in process (and presentation to scheduler)

NO I 5) handling reference counts on file blocks of maps compiled & this block

6) allocation block modification

in GR — a MON mode bit for each processor and a linear ordering of priv. listed above such that the ordering does not place any priv which could be demanded by an interrupt routine before a priv not used by interrupt.

To do anything with ECS under the MON bit must be turned on → thus to do the things mentioned (garbage collect, etc) the processor doing it must set all MON bits to lock out ECS access by other processors.

For all other priv. the rule is → a processor may try to get any privilege ~~which~~ whose place in the priv ordering ~~is~~ is later than ~~the~~ any priv that the processor already has. ~~Failure~~ There are no restrictions on releasing priv (NOTE - the MON bit is not considered a priv

Analysis of interactions in ECS

I storage allocation.

A) interact with other storage alloc

B) if obj → NOT search

c) file index or data block → file access

* NOT used by interrupt routines

Suggest ignore c) → only need to lock out all other allocation — one priv I/O stor alloc

II scheduling →

A) scheduling may result in a process being swaped in ⇒ interfer w/ interrupt

B) must lock out process interrupts to mark process as "incore"

Scheduling algorithm creates list in ECS → head of list is next process to run in either machine → then

scheduler must not run while in middle M mode code →

III Proces Interrupt —

A) Only one processor may try to prevent an interrupt to a proces

B) Interrupt may cause event channel action (if proces is blocked)

C) A proces may not be swapped in durring interrupt action ⇒ no scheduling

D) If proces if blocked (+ interrupt lesa) ⇒ event channel action cannot be allowed until "wake-up-waiting" has been set

suggest priv
1) place interrupt
*used by interrupt 2) diddle *wake-up-waiting"
*used by interrupt 3) secure event channel
4) no schedule

IV Event channel

A) only one event channel is involved try but the diddle of "wake-up-waiting" is priv

B) only one proesnor may work on a particular event channel at a time

C) PPI interrupts not allowed if EC has been marked "busy"

Suggest two priv
*used by interrupt
1) to* mark event channel busy
2) diddle "wake-up-waiting"

I. ref count on fil blocks

A) interfers only with map compilation and block deletion

suggest one priv 1) ref count

12/4

Desc file — must be opened which bumps reference count — if not previously opened (w/ unique number) then ECS incarnation is created — close decrements ref count — if goes to '0' ECS incarnation is destroyed

↳ open returns a capability for the file

ECS object - class code - 2 parts - temp & perm

[perm | temp] 64

↳ Perm used to identify user or user group & checked by disc sys to be sure it agrees w/ current perm access

(new class code gives new)

Temp part used in disk sys to allow user to open file by list of temp parts w/ file

ECS action: take old class code & set temp field (requires option bit)

class code

30

29

[perm | temp]

[subp class code]

42 (41)

12/1 approx

ECS oper

return capability for 1st object on alloc block

params 1) cap alloc blk
2) loc in C-list to return

---

fixed place objects in ECS

1) event channels
2) files

new ECS entry:

change unique name of object

Two kind object can have on event channel ⇒ 2 entry point