This is a preliminary document describing the interrupt handling code at the command level of the new system.

Objective

This is designed to handle 2 kinds of interrupts:
- major panic — break the system
- minor panic — currently control shift P.

Implementation

The action created by these panics is to appear as follows:

If they occur while user code is running:

a) minor panic,

The user sub process at top of the stack will receive an interrupt with datum "internal minor panic". If no user sub process currently running which is a proper or improper ancestor of the current top of stack, user sub process has interrupts enabled, the interrupt will be accepted by the boot sector, which will call a debugger.
b) major panic

The head ghost will be called, and will return call a debugger.

If they occur while system code is running, either call explicitly or implicitly (error and panic) by user code then:

Each system routine returns on the stack cleans up. If that system routine considers itself to be a command type routine, it in turn informs the user of the interrupt and requests a new command (this stopping the clean up procedure.)
(The debugger is such a subprocess.) If that system routine does not consider itself to be a command type routine, it removes itself from the stack.
(one way to clean up is to do nothing, e.g. the low level disk system.)

If user code can call a command processor, the user should be able to instruct said code to return with (major) interrupt;

can user code explicitly invoke a CP? (it can implicitly, with an error.)
Fact:

If a leaf system subprocess gets an interrupt or an error, it was on the stack immediately below.

Note:

These algorithms use various portions of my latest proposal on the subprocess call, return and interrupt mechanism. In particular, at least inhibit interrupt in the subprocess descriptor, and the return will inhibit operation.

*unfortunately, this fact is sometimes irrelevant.*
System Leaf

Interrupt ↓

save reg(I)
circle old stack entry (und-save it)

p ≠ Normal entry?

no

yes

restore reg(I)
save reg(n)

p ≠ Entry entry?

no

yes

set ensn
set pc = y
return
d := now
interrupt inhibited
users
get register from N
at abnormal level in stack

non-command case

↓

major panic?

yes

no

allow Int
restore reg(n)
Jump call good foot
special entry

command case

↓

Bekrofense.

allow Int
restore reg(n)
major panic return
System leaf continued

Normal entry ↓

- Save reg(N)

↓

Body of code

- Inhibit interrupt
- Save reg(5)
- Save calls stack entry
- Set pc = ρ
- Return
- Return

Process Give

↑

Error entry ↓

- Inhibit interrupt
- Save reg(5)
- Save calls stack entry
- Set pc = ρ
- Return
- Return

ρ = stacktop

↑

from Interrupt code

- Restore reg(N)
- Allow Int
- Return

- Restore reg(N)
- Allow Int
- Error return
System leaf continued

How to send and get events and be sure

**send**
- inhibit interrupt
- sendev
- set flag for sent
- allow interrupts

**get**
- check got flag?
  - set trying flag
  - gotevent hung
  - set got flag
    - (save x, y, z?)
    - clear trying flag
  - (depends on whether command or otherwise)

Place at ds in the interrupt code:

```
d1
txtry flag on? "n"
  yes
  check pcounter and pcm
  i) about to execute gotevent
  ii) almost finished with gotevent
  iii) high p counter value
      set got flag
  clear trying flag
      (save x, y, z?)
```

Continue
(ends ghost special entry)

↓
\[ \text{check}(\text{entry}) \]

↓
\[ \text{save}(\text{N}) \]

↓
\[ \text{check}(\text{is } \text{N}, \text{ special entry}) \]

↓
\[ \text{system below?} \quad \text{YES} \rightarrow \text{restore}(\text{N}) \]

↓
\[ \text{no} \]

↓
\[ \text{debug} \]

↓
\[ \text{save}(\text{E}) \]

↓
\[ \text{restore}(\text{I}) \]

↓
\[ \text{save}(\text{N}) \]

↓
\[ \text{restore}(\text{E}) \]

↓
\[ \text{internal major panic return} \]

Note: The board ghost has interrupt in 4.5.0 and all times.