# DECavail

*Overview*

Bernard OURGHANLIAN

Symposium DECUS

5 Juin 1991

**VMS 09**

# Sommaire

1. Introduction
2. L'architecture
3. Les composants
4. Le fonctionnement :
   a. Introduction
   b. Le protocole
   c. La collecte des données
   d. L'investigation et la correction des problèmes
5. Exemples
6. Performances

# *Introduction*

**Les informations contenues dans cette présentation ne sauraient constituer un quelconque engagement de la part de DIGITAL sur des fonctionnalités ou des dates de disponibilité**

# Introduction

- DECavail est un outil de surveillance et de contrôle à distance de la disponibilité des systèmes
- DECavail permet :
  - -- D'accroitre la disponibilité d'un système ou d'un *cluster*
  - -- De réduire les nécessités de *reboot* d'un système ou d'un *cluster*
  - -- D'améliorer les performances d'un système ou d'un *cluster*
- DECavail :
  - -- Analyse, signale et corrige les problèmes de disponibilité des systèmes en temps réel
  - -- Collecte les données *remote* à un niveau d'IPL élevé (IPL 6)

-- Corrige les problèmes de disponibilité

-- Présente les informations importantes sous la forme d'un certain nombre de fenêtres dynamiques DECwindows

# L’architecture

DECavail obéit à une architecture de type "maître-esclaves" dans lequel une console peut surveiller et contrôler à distance et en temps réel plusieurs systèmes en même temps

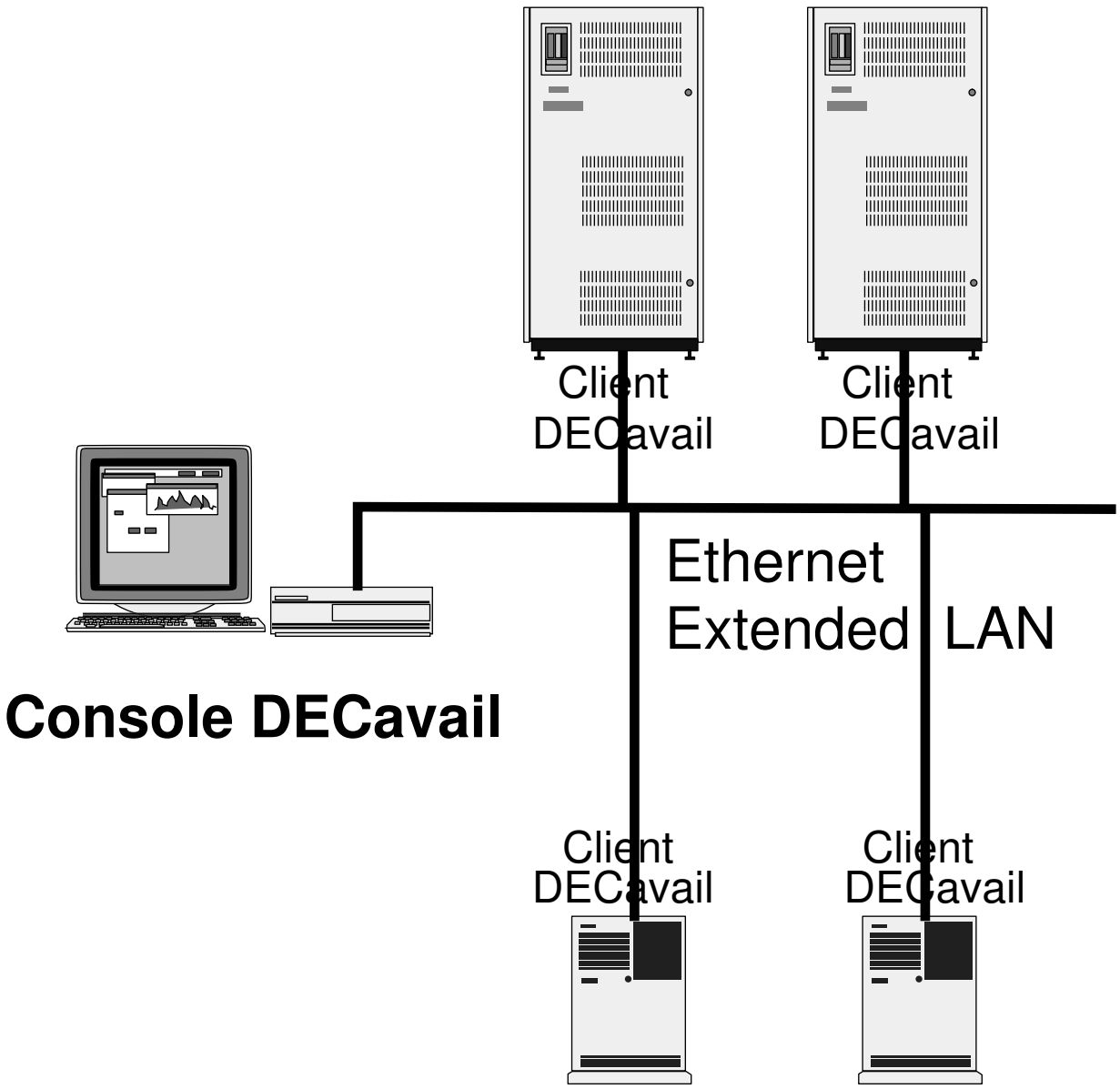
La méthodologie utilisée est la suivante:

1. Chaque système esclave notifie le système maître (la console DECavail) de sa présence
2. Chaque système esclave collecte les données à la requête de la console DECavail à un niveau d’IPL élevé afin de pouvoir disgnostiquer les noeuds "bloqués"
3. Les données sont retournées à la console DECavail

4. Les données sont analysées en utilisant des algorithmes heuristiques simples visant à diagnostiquer des problèmes de dépassement de ressources
5. Quand il y a dépassement de ressources, il y a génération d'évènements par le programme de surveillance et de contrôle de la console DECavail
6. Le programme de surveillance et de contrôle de la console DECavail peut répondre en fonction de règles ajustables. Ces réponses peuvent être invoquées automatiquement ou manuellement. Ces réponses sont de type :
   a. Enregistrer les évènements dans une fenêtre DECwindows et/ou dans un fichier
   b. Alerter l'utilisateur par MAIL
   c. Augmenter/Diminuer la fréquence d'analyse des données

d. Enregistrer les données collectées dans un fichier

e. Envoyer des programmes *remote* pour fixer le problème



# Les composants

DECavail est composé de 2 parties :

- Coté *driver* (esclave) :
  - -- Un *driver* RMDRIVER.EXE
  - -- Une programme de contrôle et de chargement du driver RMCP.EXE :
    - > Validation de la licence
    - > Contrôle de la sécurité
    - > Démarrage du protocole
  - -- Un fichier permettant le contrôle de la sécurité : DAVL$ACCESS_NODES.DAT (lu par RMCP)
- Coté console :
  - -- Le programme principal de la console DAVL$CONSOLE.EXE

-- Un programme de communication DAVL$COMM.EXE (lancé dans un sous *process* par DAVL$CONSOLE.EXE)

-- Une série de *program libraries* (librairie de texte binaire utilisé par la *performance machine*) spécifiques à la version du système d'exploitation qui est analysé

-- Le fichier UID du programme de la console (DAVL$CONSOLE.UID)

-- Un fichier permettant le contrôle de la sécurité : DAVL$ACCESS_NODES.DAT

-- Un fichier permettant de paramètrer les ressources utilisées par le *display* DECwindows de la console : DAVL$CUSTOMIZE.DAT

-- Un fichier permettant de paramètrer et de contrôler la sévérité des problèmes rencontrés : DAVL$SEVERITY_DEFS.DAT

-- Un fichier permettant de paramètrer et de contrôler les niveaux limites au-delà desquels un problème est déclaré : DAVL$THRESHOLD_DEFS.DAT

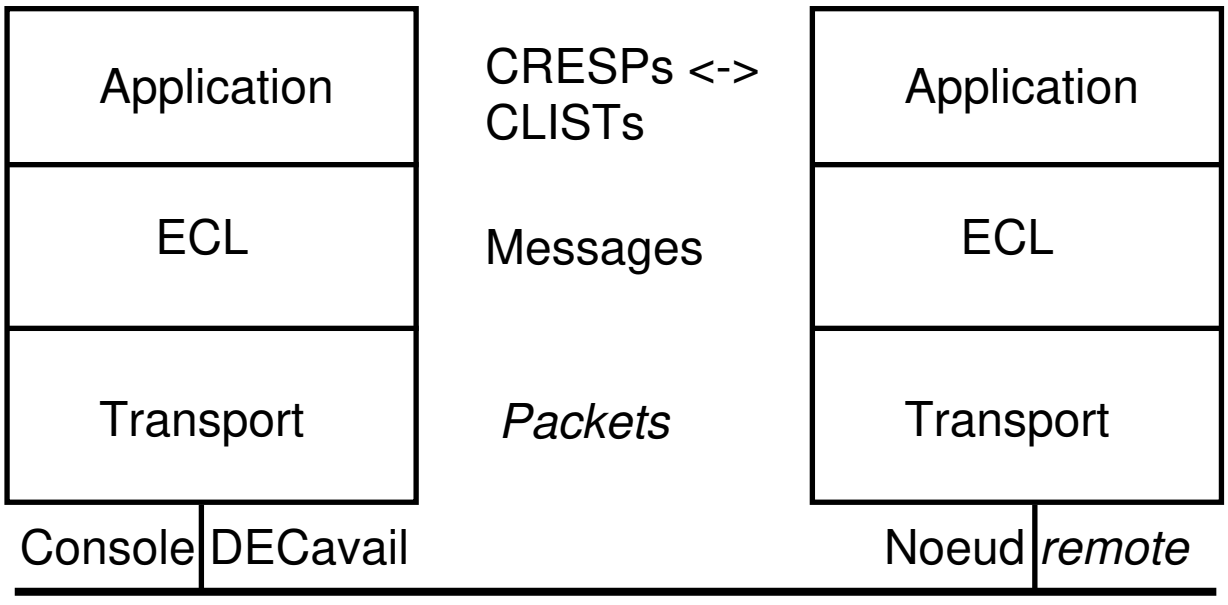
-- Un fichier de déclaration de noms logiques DECavail DAVL$LOGICALS.COM

- Tous ces fichiers sont localisés dans la *directory* pointée par le nom logique DAVL$SYSTEM (SYS$COMMON:[DECAVAIL])
- Il est possible de grouper des systèmes à surveiller au moyen du nom logique DAVL$GROUP_NAME (défini sur le noeud surveillé)
- Les fichiers DAVL$EVENT_LOG et DAVL$LOCK_LOG contiennent respectivement les enregistrements d'*events* DECavail et les enregistrements de problèmes de type *lock contention*

# *Le fonctionnement*

## *Introduction*

- Il est possible de visualiser l'architecture de l'implantation de DECavail de la façon suivante :



- L'application se réfère à la partie algorithmique de contrôle et de surveillance de la console de DECavail
- La communication entre la console de DECavail et les noeuds *remote* utilise un protocole asymétrique et opère normalement suivant un modèle client - serveur dans lequel le noeud "console DECavail" envoit des requêtes et le noeud *remote* répond
- Sur les noeuds *remote*, un driver spécial RMDRIVER, est utilisé pour traiter les messages
- Sur la console DECavail, un *process* de communication DAVL$COMM envoit des $QIOs (802.3) à Ethernet qui envoit le message au RMDRIVER situé sur le noeud *remote*

- Un noeud peut se surveiller lui-même; dans ce cas, le module de communication *datalink* n'est pas utilisé

# Le fonctionnement

## Le protocole

- DECavail utilise un protocole spécifique basé 802.3 avec les caractéristiques suivantes :
  - -- Pas de segmentation de messages : taille maximum d'un *packet* : 1480 *bytes*
  - -- Pas de contrôle de flux au niveau transport
  - -- Pas d'acquittement de messages :
    - > protocole de type *request-response*
    - > détection des messages perdus en utilisant des *timers*

- La couche transport accepte un message de la couche ECL, l'intègre dans un enveloppe contenant :
  - -- Un type de message et quelques champs permettant de contrôler la validité
  - -- Un *timer* permettant la détection des messages perdus
  - -- Un *checksum* pour pouvoir détecter les corruptions éventuelles
- Le code détecte la duplication des messages et permet d'initialiser la retransmission de messages perdus
- Le protocole est asymétrique en ce sens que la "console DECavail" peut être vue comme ayant plusieurs "extensions" sur les noeuds *remote*
- Toutes les fonctions de contrôle de *timing* sont exécutées sur la "console DECavail"

- Types de messages :
  - -- "*Hello messages*"
  - -- Messages de contrôle
  - -- "CLIST" : demande de données depuis la "console DECavail"
  - -- "CRESP" : réponse du noeud *remote*
- Initialisation :
  - -- Les noeuds *remote* (RMDRIVER) envoient toutes les 30 secondes un message *multicast* de type "HELLO" sur une adresse déterminée donnant quelques informations vitales :
    - > nom du groupe de noeud
    - > nom du noeud
    - > SCSSYSTEMID
    - > système d'exploitation
    - > version

-- Il n'y a que la "console DECavail" qui consomme un *multicast message slot* car les noeuds *remote* ne recoivent **pas** de messages *multicast*

-- Le *driver* DECAVAIL est démarré au moyen du programme RMCP.EXE.

-- RMCP :

> lit le contenu du fichier DAVL$ACCESS_NODES.DAT

> ce fichier contient des informations de sécurité sous la forme : *adresse\mot-de-passe\accès* ou l'adresse peut être sous la forme d'un format :

+ phyique : AA-00-04-00-xx-yy
+ DECnet : area.number

> l'accès désigne un accès en :

+ écriture : W
+ lecture : R

-- Quand RMDRIVER et la "console DECavail" démarrent :

> il y a construction un *buffer (*dans l'extension de l'UCB de RMDRIVER) contenant chacune des lignes du fichier sécurité

-- Quand la "console DECavail" démarre :

> elle écoute les messages *multicast* : quand elle reçoit un "*hello message*" :

+ elle effectue un certain nombre de vérifications de sécurité en échangeant le mot de passe contenu dans le fichier avec le RMDRIVER *remote*

+ elle n'autorise le démarrage du protocole que si les mots de passe sont en accord

-- Une console DECavail peut contrôler plusieurs groupes ou *clusters* de noeuds, chacun ayant un mot de passe différent

-- A aucun instant le noeud *remote* ne sait que son "*hello message*" a été reçu : il n'y a pas de connexion entre la "console DECavail" et le noeud *remote*

- Opérations normales :

-- Quand l'initialisation est effectuée, sur la "console DECavail", le transport est implanté sous la forme d'un *process* (DAVL$COMM) qui effectue des $QIOs en direction du *driver* Ethernet

-- Appel de la couche ECL au transport:

> création d'un *buffer* d'émission avec un type de message CLIST avec un numéro de séquence et un *checksum*

- envoi du *buffer* à l'aide d'un $QIO avec établissement d'une *timer* AST (via $SETIMR) pour détecter les paquets perdus
- placement du *buffer* de réception sur une liste

-- Réception :

- vérification du *checksum* et du type de message
- utilisation du numéro de séquence pour retrouver le *buffer* de réception
- suppression du *buffer* de la liste de *buffers* de réception

-- *Timeout* :

- si un message est perdu, celui-ci est retransmis avec incrémentation d'un compteur d'erreur
- le cas où un noeud a été arrêté ou a *crashé* est géré (dans ce cas, un

status de type SS$_PATHLOST est retourné)

- La couche "ECL" :
  - -- Chaque commande de l'utilisateur passée dans une fenètre de DECavail provoque la création de *Command Lists* (CLISTs) pour informer le noeud *remote* du type d'information demandée
  - -- Le noeud *remote* retourne ces informations sous la forme de *Command Responses* (CRESPs)
  - -- La couche ECL alloue les *buffers* utilisés par la couche transport et multiplexe les messages en un *packet* de façon analogue au protocole LAT

# Le fonctionnement
## La collecte des données

Pour essayer de comprendre le fonctionnement du mécanisme de collecte de données par DECavail, imaginons que l'on désire collecter les données qui sont montrée dans un "$MONITOR PAGE" comme :

- *Remote system time* (EXE$GQ_SYSTIME)
- *# of Page faults* (PMS$GL_FAULTS)
- *# of Page read* (PMS$GL_PREADS)
- *# of paging I/O read* (PMS$GL_READIO)

La méthode utilisé par DECavail est celle d'une "*Performance Machine*" :

- DECavail utilise un méta-langage permettant la collecte des informations de performance

- Chaque *request buffer* contient un certain nombre d'"instructions" dans un langage permettant de construire rapidement des listes de commandes
- Par exemple, la collecte des données du $MONITOR PAGE précédent peut être effectuée de la façon suivante :

*Request packet*

| | |
|---|---|
| 8 | GET_MEM |
| EXE$GQ_SYSTIME | |
| 4 | GET_MEM |
| PMS$GL_PAGEFLTS | |
| 4 | GET_MEM |
| PMS$GL_PREADS | |
| 4 | GET_MEM |
| PMS$GL_READIO | |

*Response packet* (HEX)

| | |
|---|---|
| EXE$GQ_SYSTIME | E2850000 |
| | 009096F3 |
| PMS$GL_PAGEFLTS | 00008522 |
| PMS$GL_PREADS | 00002E33 |
| PMS$GL_READIO | 00000747 |

- Dans cet exemple, toutes les requêtes sont envoyées sous la forme "GET_MEM" qui n'est autre que l'"OPCODE" de la *performance machine*

- La *performance machine* a 256 "registres" de 32 bits qui sont utilisés pour manipuler et charger les données
- Cette *performance machine* est écrite en Macro-32
- On a alors la possibilité d'ecrire des routines ECL sous la forme :

```
L_PCB     1,213000E5              ; Get PCB address in register 1
LOAD      2,PCB$L_PHD(1)          ; Get PHD address in register 1
GET_MEM 4,PHD$L_CPUTIM(2) ; Output 4 bytes
```

pour récupérer le temps CPU pour le *process* 213000E5

- Cette approche permet la flexibilité et la simplicité avec l'inconvénient d'avoir un code interprété : l'exemple ci-dessus représente la génération de 24 instructions VAX
- Pour une machine de 1 VUP, cela représente 1 micro-seconde par *input packet* de 1 *byte* : ceci est acceptable

# Le fonctionnement

## L'investigation et la correction des problèmes

- La "console DECavail" analyse les noeuds *remote* en :
  - -- Envoyant des demandes d'informations en direction de chacun de ces noeuds
  - -- Stockant les réponses dans des structures de données contenant les précédentes réponses
- Ceci permet d'appliquer des algorithmes heuristiques simples :
  - -- Indépendants de la taille du *hardware* concerné
  - -- Utilisant des valeurs de seuils modifiables par l'utilisateur (DAVL$THRESHOLD_DEFS.DAT)

-- Tenant compte du système dans son intégralité et non d'un composant particulier

- Si un problème existe, un *event* est généré : cet évènements sont alors re-dirigés vers l'*event window*
- Ces *events* ont des niveaux de sévérité de 0 (information) à 100 (crash) modifiables par l'utilisateur (DAVL$SEVERITY_DEFS.DAT)
- Ces *events* peuvent être filtrés de façon à ne pas apparaitre dans l'*event window* ou bien triés par le temps ou la sévérité
- La plupart des *events* non informationnels sont associés à un *display* de type *investigate* ou *fix*

- DECavail sait reconnaitre les *events* :
  - -- *Clusters* :
    - > *State transition, lock contention*
  - -- *Node :*
    - > *Page/Swap file* plein
    - > Pas assez de mémoire (*free list*)
    - > Trop de *DIO, BIO, Fault IOs, Modified Page Writer IOs*
    - > Taille de *COM queue* trop grande, trop de temps CPU passé sur l'*Interrupt Stack*
    - > Pas assez de temps passé en mode *user*
    - > Trop de *processes* en état inhabituel (PFW, COLPG, FPG, MUTEX)

-- *Process* :
  > Pas assez de temps CPU consommé (*process computable* depuis longtemps sans avoir accumulé de *CPU time*)
  > Trop de *DIO,BIO, Fault IOs*
  > Quotas trop faibles
-- Disques :
  > Taux d'erreur trop important
  > Taille de la queue d'attente trop importante
  > *Mount verify status*
  > Pas assez de place libre

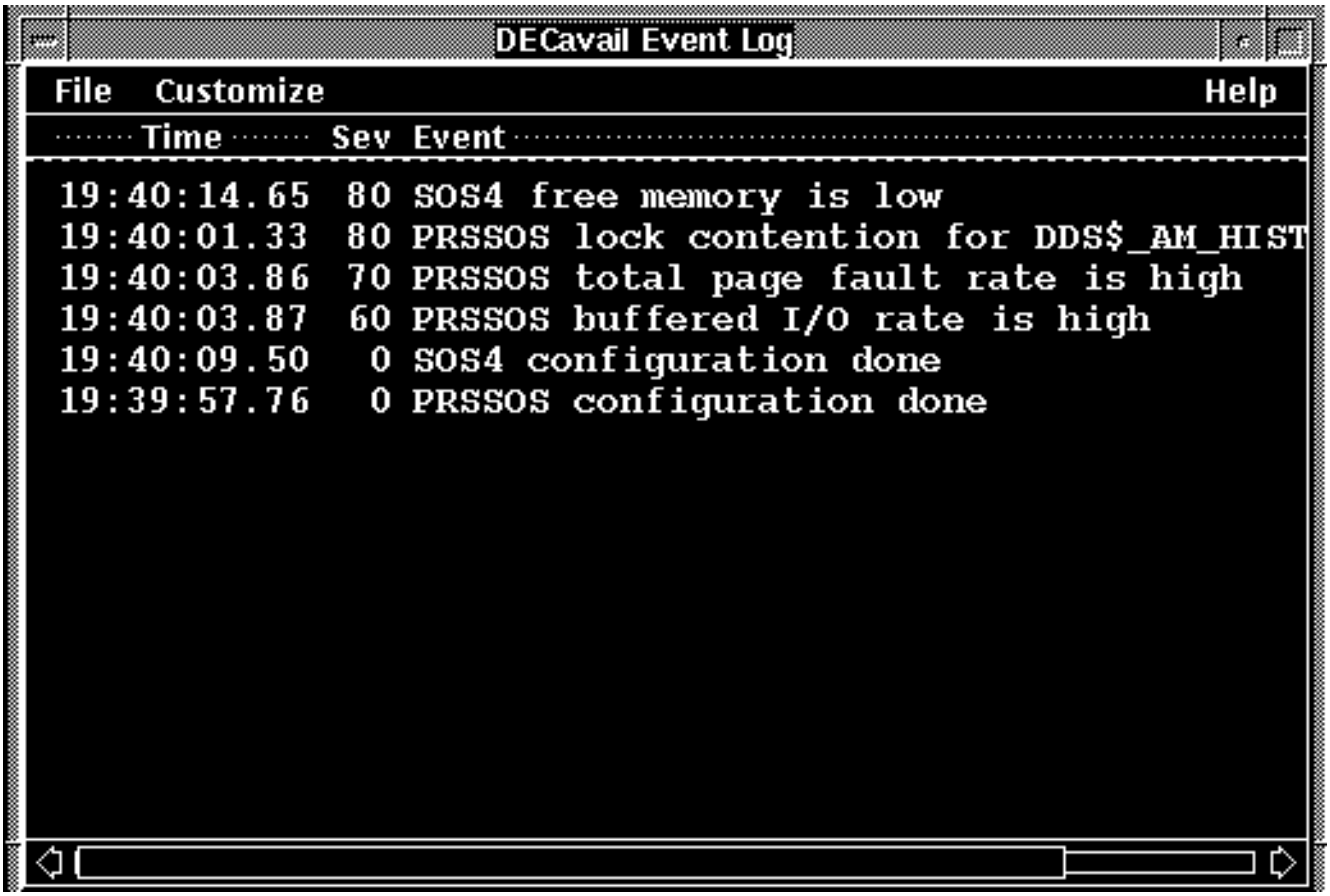
- DECavail peut "fixer" les problèmes suivants :
  - -- *Clusters* :
    - > Réduction du *quorum* (équivalent du CTRL-P, >>> D/I 14 C, IPC>Q)
  - -- *Node* :
    - > *Crash*
  - -- *Processes* :
    - > *Processes* "mangeant la machine":
      - + *Stop, Suspend, Force Exit, Resume*
      - + Abaissement de la priorité, purge du *working set*
    - > *Processes* "en *hang*" :
      - + Augmentation de la priorité pour les *processes* en COM
      - + Augmentation des quotas pour les *processes* en RWAST

+ Débloquage des *processes* en attente de *locks*
+ *Cancel I/O* sur les *channels* ouverts

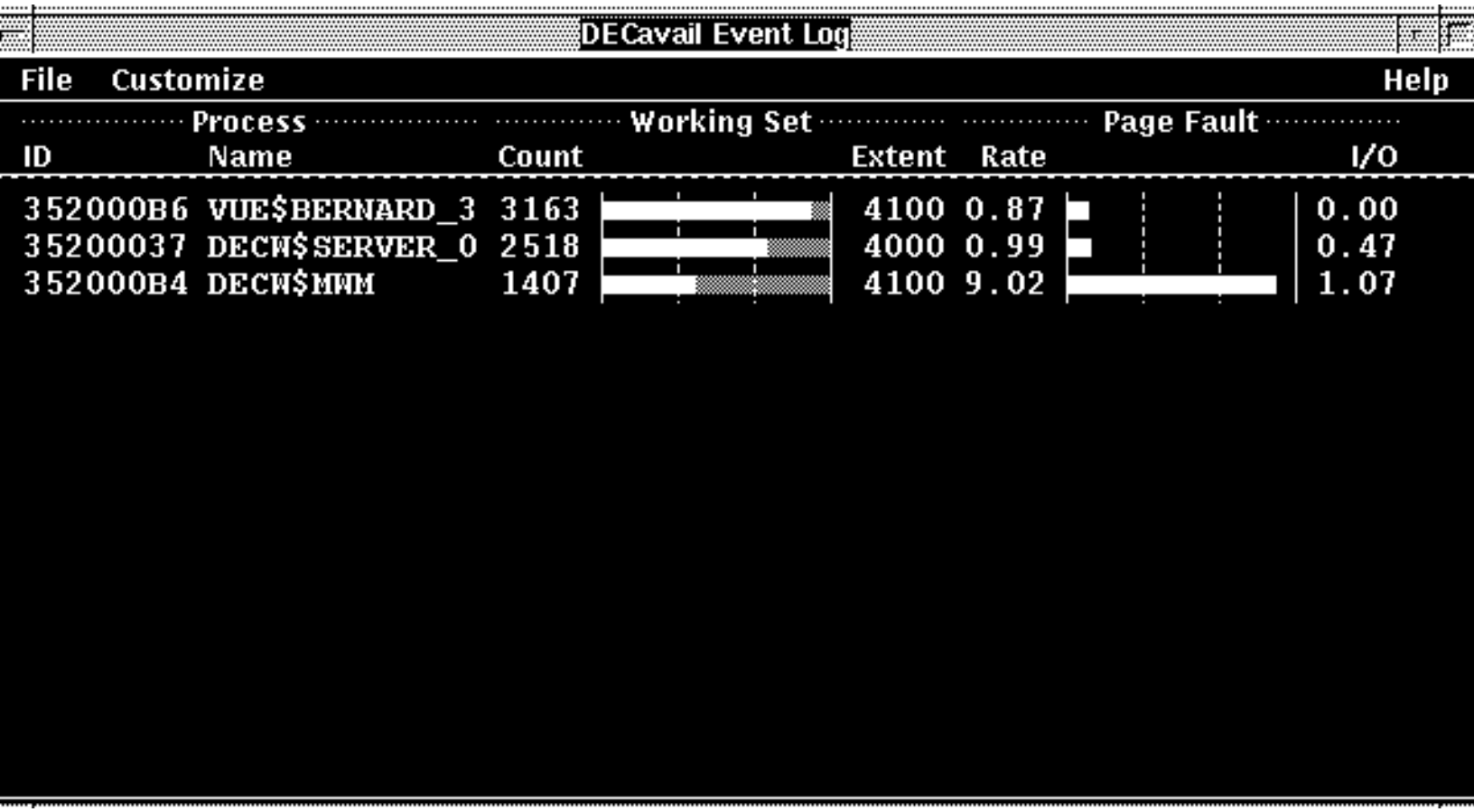
-- Disques :

> "Désallocation" d'un disque
> *Cancel mount verification*

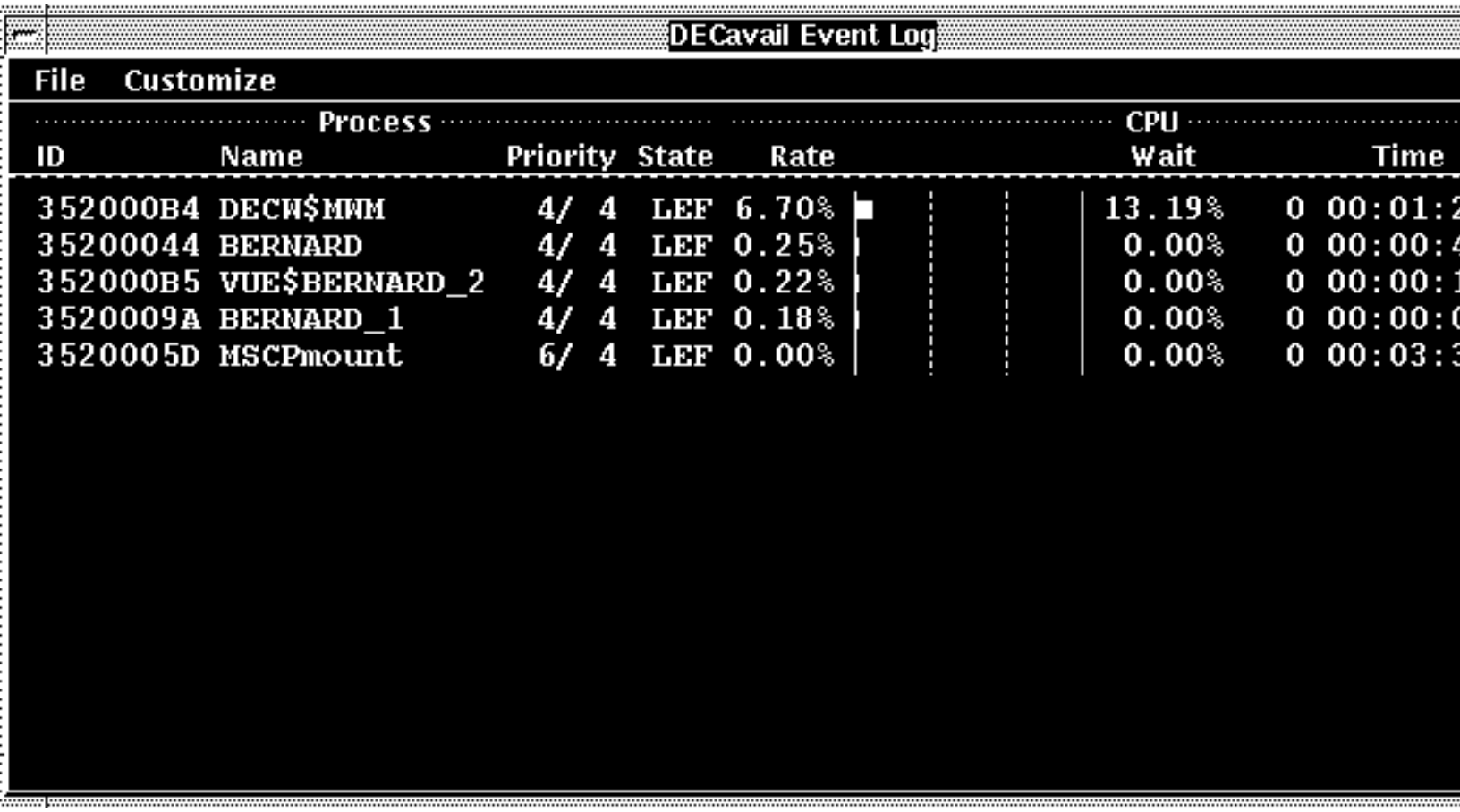
# Exemples


DECavail Event Log
File
Customize
Help
Time
Sev
Event
19:40:14.65 80 SOS4 free memory is low
19:40:01.33 80 PRSSOS lock contention for DDS$_AM_HIST
19:40:03.86 70 PRSSOS total page fault rate is high
19:40:03.87 60 PRSSOS buffered I/O rate is high
19:40:09.50 0 SOS4 configuration done
19:39:57.76 0 PRSSOS configuration done

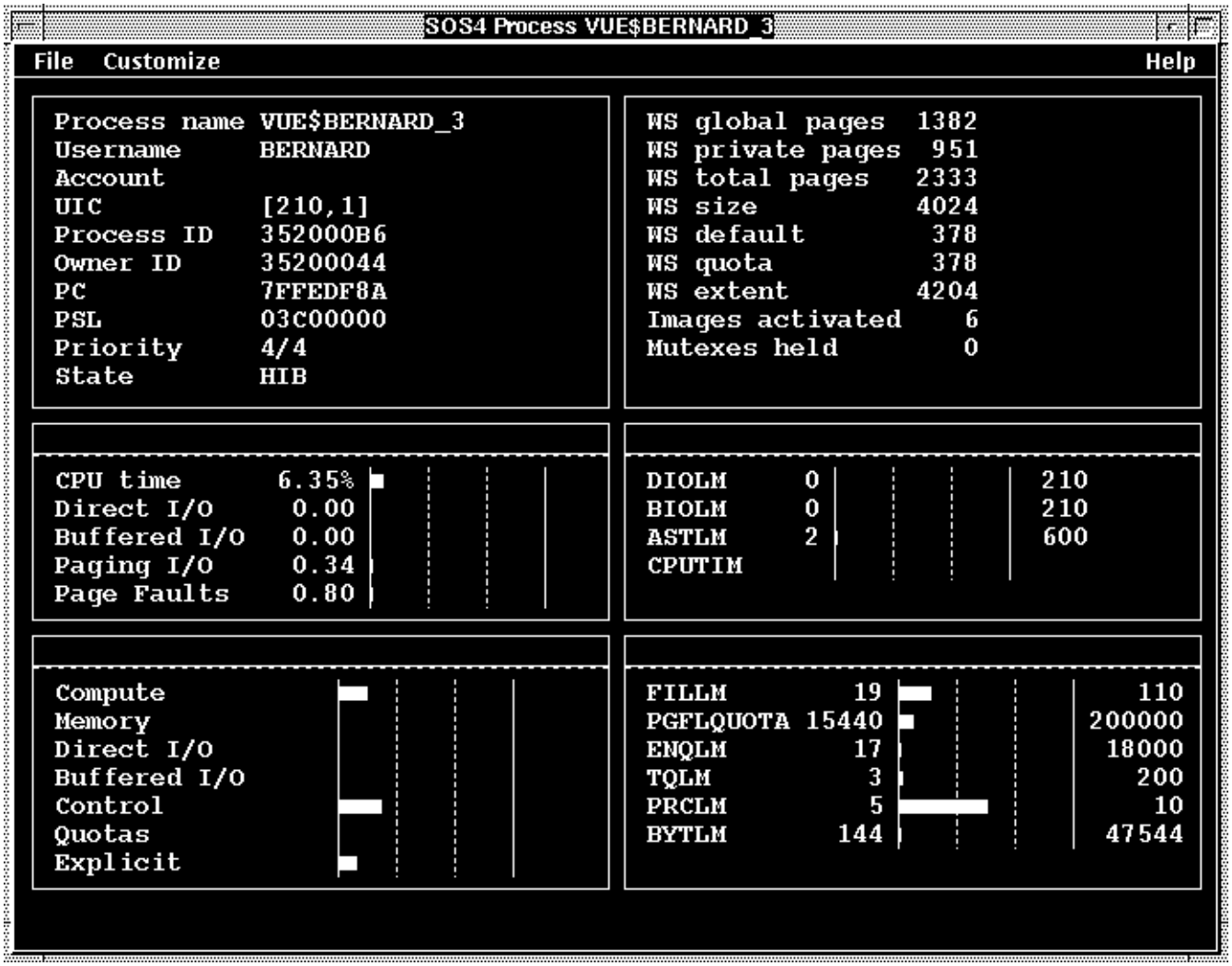
# Exemples


DECavail Event Log
File
Customize
Help
Process
Working Set
Page Fault
ID
Name
Count
Extent
Rate
I/O
352000B6 VUE$BERNARD_3 3163 4100 0.87 0.00
35200037 DECW$SERVER_0 2518 4000 0.99 0.47
352000B4 DECW$MWM 1407 4100 9.02 1.07

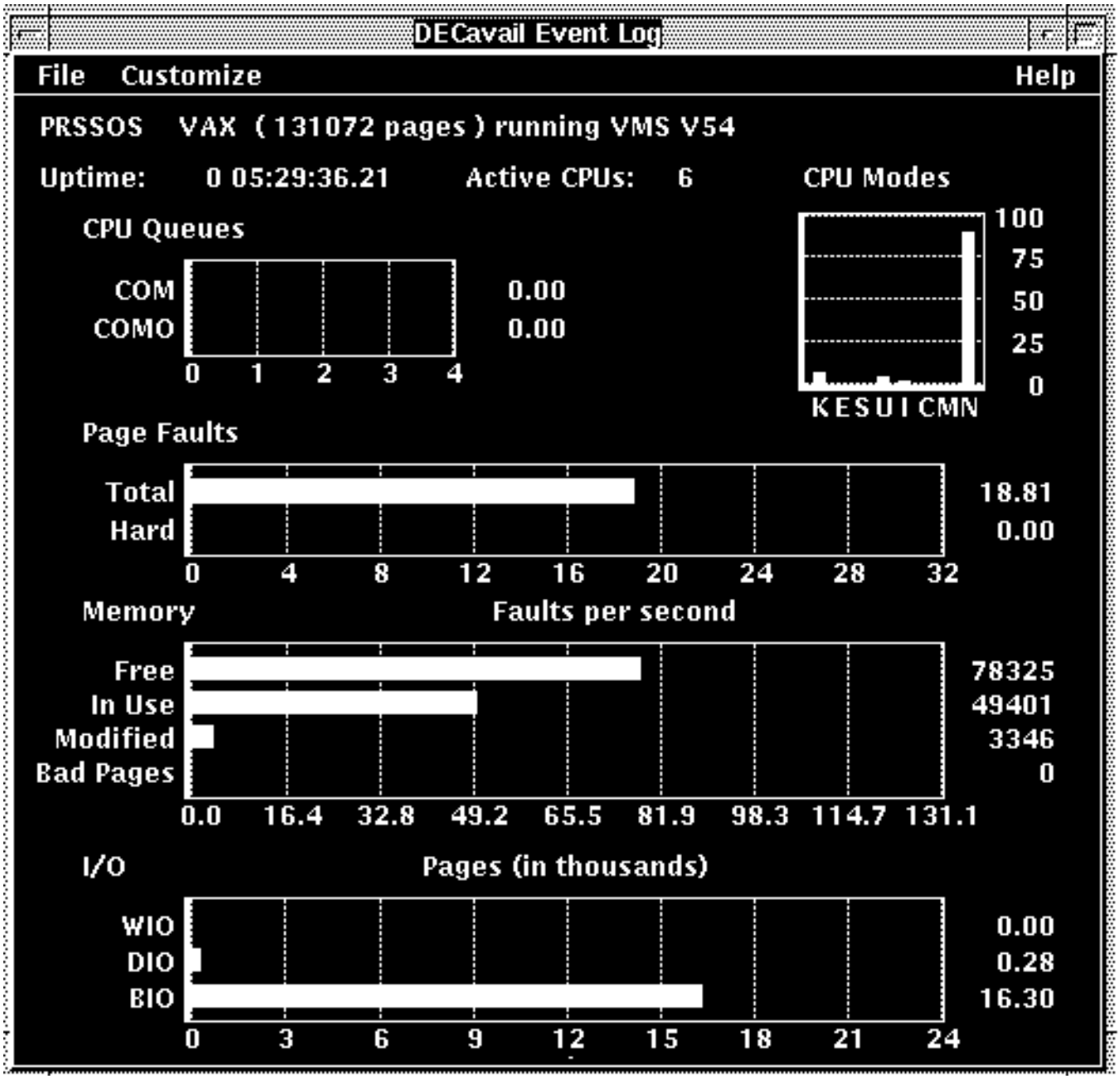
# Exemple


DECavail Event Log
File Customize
Process
CPU
ID Name Priority State Rate Wait Time
352000B4 DECW$MWM 4/ 4 LEF 6.70% 13.19% 0 00:01:2
35200044 BERNARD 4/ 4 LEF 0.25% 0.00% 0 00:00:4
352000B5 VUE$BERNARD_2 4/ 4 LEF 0.22% 0.00% 0 00:00:1
3520009A BERNARD_1 4/ 4 LEF 0.18% 0.00% 0 00:00:0
3520005D MSCPmount 6/ 4 LEF 0.00% 0.00% 0 00:03:3

# Exemple


SOS4 Process VUE$BERNARD_3
File
Customize
Help
Process name VUE$BERNARD_3
Username BERNARD
Account
UIC [210,1]
Process ID 352000B6
Owner ID 35200044
PC 7FFEDF8A
PSL 03C00000
Priority 4/4
State HIB
WS global pages 1382
WS private pages 951
WS total pages 2333
WS size 4024
WS default 378
WS quota 378
WS extent 4204
Images activated 6
Mutexes held 0
CPU time 6.35%
Direct I/O 0.00
Buffered I/O 0.00
Paging I/O 0.34
Page Faults 0.80
DIOLM 0 210
BIOLM 0 210
ASTLM 2 600
CPUTIM
Compute
Memory
Direct I/O
Buffered I/O
Control
Quotas
Explicit
FILLM 19 110
PGFLQUOTA 15440 200000
ENQLM 17 18000
TQLM 3 200
PRCLM 5 10
BYTLM 144 47544

# Exemple


DECavail Event Log
File
Customize
Help
PRSSOS VAX ( 131072 pages ) running VMS V54
Uptime: 0 05:29:36.21
Active CPUs: 6
CPU Modes
CPU Queues
COM
COMO
0.00
0.00
0 1 2 3 4
100
75
50
25
0
KESUI CMN
Page Faults
Total
Hard
18.81
0.00
0 4 8 12 16 20 24 28 32
Memory
Faults per second
Free
In Use
Modified
Bad Pages
78325
49401
3346
0
0.0 16.4 32.8 49.2 65.5 81.9 98.3 114.7 131.1
I/O
Pages (in thousands)
WIO
DIO
BIO
0.00
0.28
16.30
0 3 6 9 12 15 18 21 24

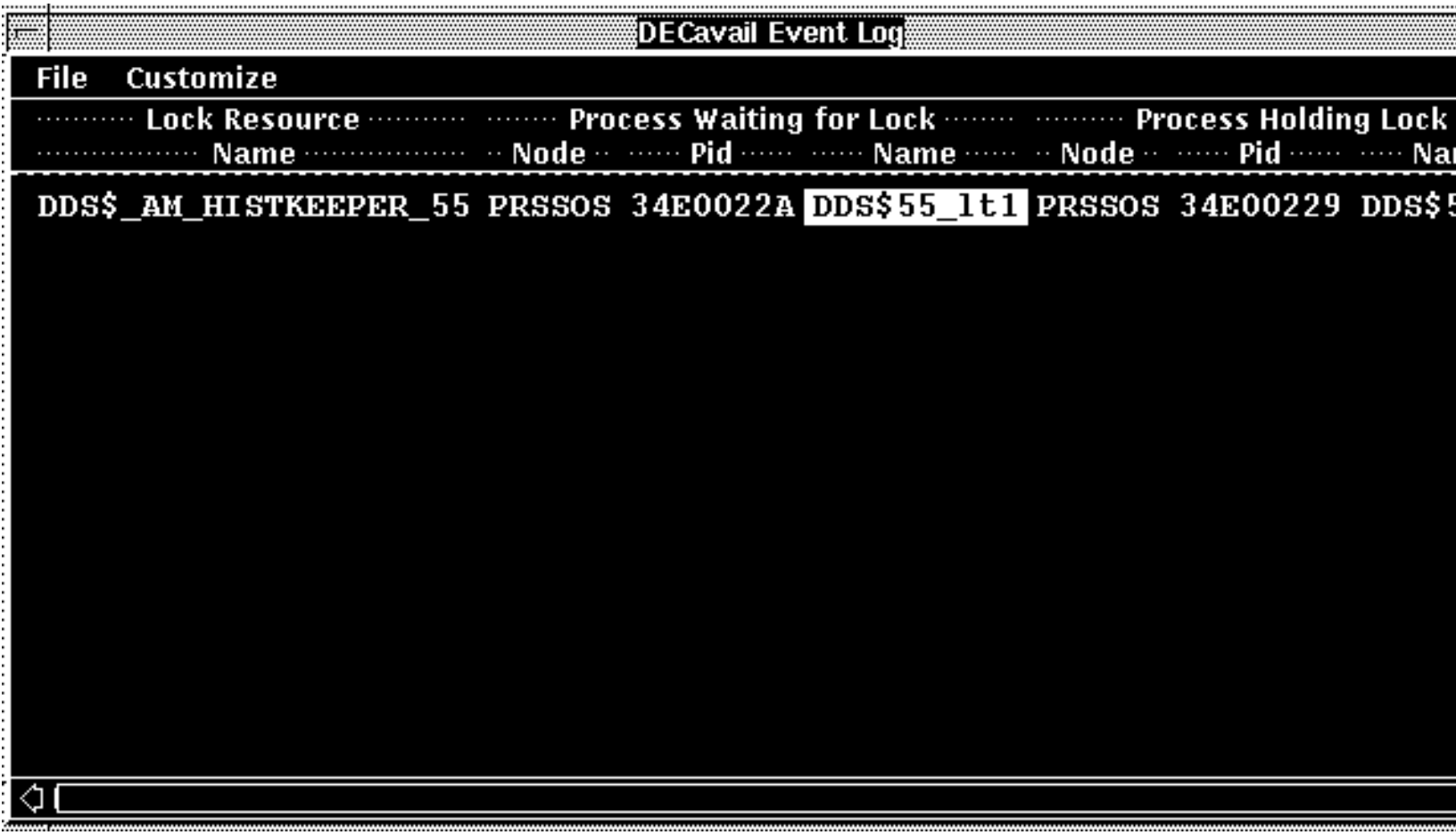
# Exemple

DECavail Event Log

File Customize

| Device Name | Path | Volume Name | Disk Space (blocks) Used | Free | Queue | OpRa |
|---|---|---|---|---|---|---|
| MALTOM$DUA0 | MALTOM | STA0 | 138405 | 267 | 0.00 | 0.0 |
| MALTOM$DUA1 | MALTOM | STA1 | 138405 | 267 | 0.00 | 0.0 |
| $1$DUA15 | PHOQUE | SPECS2 | 889050 | 2022 | 0.00 | 0.0 |
| $1$DUA11 | PHOQUE | SPECS1 | 888444 | 2628 | 0.00 | 0.0 |
| $1$DUA22 | PHOQUE | DIVERS | 1209522 | 7143 | 0.00 | 0.0 |
| $1$DJA32 | PHOQUE | VAXVMSOLD | 387549 | 12627 | 0.00 | 0.0 |
| $1$DJA33 | PHOQUE | COMMS_WORK | 386655 | 13521 | 0.00 | 0.0 |
| MALTEX$DUA5 | MALTEX | USER10 | 293848 | 17352 | 0.00 | 0.0 |
| MALTEX$DUA4 | MALTEX | MALTEX_SYS | 288013 | 23187 | 0.00 | 0.0 |
| $1$DUA27 | PHOQUE | DATABASE | 2349345 | 26808 | 0.00 | 0.0 |
| MALTEX$DUA7 | MALTEX | MALTEX_DATA | 101916 | 36756 | 0.00 | 0.0 |
| MALGRA$DUA0 | MALGRA | MALGRA_SYS | 266473 | 44727 | 0.00 | 0.0 |
| MALTEX$DUA6 | MALTEX | USER2 | 259879 | 51321 | 0.00 | 0.0 |
| PRSARL$DJB7 | PRSARL | APPLIC2 | 348720 | 51456 | 0.00 | 0.0 |
| $1$DJA24 | SOS6 | DOCD$ | 337089 | 63087 | 0.00 | 0.0 |
| $1$DJA25 | SOS6 | DECW_BOOK | 321276 | 78900 | 0.00 | 0.0 |

# Exemple


DECavail Event Log
File
Customize
Lock Resource
Process Waiting for Lock
Process Holding Lock
Name
Node
Pid
Name
Node
Pid
DDS$_AM_HISTKEEPER_55 PRSSOS 34E0022A DDS$55_lt1 PRSSOS 34E00229

# Exemple

DECavail Event Log

File Customize Help

| Device Name | Path | Volume Name | Status | Errors |
|---|---|---|---|---|
| $1$DIA0 | SOS3 | | Online | 0 |
| $1$DJA24 | SOS6 | DOCD$ | Mounted | 0 |
| $1$DJA25 | SOS6 | DECW_BOOK | Mounted | 0 |
| $1$DJA32 | PRSSOS | VAXVMSOLD | Mounted | 0 |
| $1$DJA33 | PRSSOS | COMMS_WORK | Mounted | 0 |
| $1$DUA0 | SOS6 | VAXVMSFMR | Mounted | 0 |
| $1$DUA1 | SOS6 | | Online | 0 |
| $1$DUA11 | PRSSOS | SPECS1 | Mounted | 0 |
| $1$DUA12 | PRSSOS | OLDTMR | Mounted | 0 |
| $1$DUA15 | PRSSOS | SPECS2 | Mounted | 0 |
| $1$DUA19 | PRSSOS | SPECS3 | Mounted | 0 |
| $1$DUA2 | PRSSOS | SWAPAGE | Mounted | 0 |
| $1$DUA20 | PRSSOS | VAXVMSSDC | Mounted | 0 |
| $1$DUA21 | PRSSOS | DIV1 | Mounted | 0 |
| $1$DUA22 | SOS6 | DIVERS | Mounted | 0 |
| $1$DUA23 | PRSSOS | DIV2 | Mounted | 0 |

# Performances

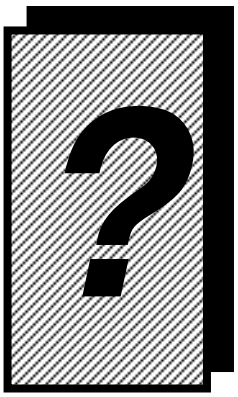
- DECavail a été conçu avec l'idée de pouvoir dédier une VAXstation comme "console DECavail"
- Cela n'est PAS obligatoire : il est possible d'utiliser DECavail dans un environnement partagé en utilisant les capacités client-serveur de DECwindows
- DECavail a été conçu pour pouvoir tourner sur un LAN étendu : il est possible d'utiliser des *bridges* Ethernet sous réserve que le temps de latence ne soit pas trop long : 99.8% des messages de type *round-trip* (DECavail<->noeud *remote*) doivent être exécutés en moins de 2s

- DECavail n'utilise pas plus de 1% du temps CPU sur le noeud *remote* analysé dans le cas d'un MicroVAX 3100 si l'on utilise les fréquences d'analyse par défaut; il faut prendre en compte que :
  - -- la majorité de ce temps CPU est passé à IPL élevé (IPL 6)
  - -- il faut modifier les férquences de *monitoring* avec discernement
- DECavail peut surveiller avec un "temps de réponse souris" inférieur à 1 seconde 64 noeuds sur une VAXstation 3100 avec 16 Mbytes
- La consommation mémoire sur les noeuds analysés par RMDRIVER est de 10 Kbytes
- Sur un MicroVAX 3600, le temps d'un *round-trip* est de 200 à 400 milli-secondes

# Le "status" de DECavail

- Pour le moment DECavail n'est PAS un produit commercialisé par DEC
- Il sera commercialisé sous la forme d'un **service** dans un premier temps vers la fin de cette année
- Il sera ensuite commercialisé comme un produit dans le courant de l'année prochaine

Ces informations sont à prendre avec

réserve

# Questions