

# CLUSTERPRO

## MC ProcessSaver 2.9 for Linux

### ユーザーズガイド

© 2024(Apr) NEC Corporation

- 製品の概要について
- プロセス監視の機能
- プロセス消滅監視の導入手順について
- 高度な運用手順について
- 操作・運用手順
- 注意・制限事項
- リファレンス

## 改版履歴

版数	改版	内容
1.0	2015.03	新規作成
2.0	2016.03	MC 2.1 に対応
3.0	2017.04	MC 2.2 に対応
4.0	2018.04	MC 2.3 に対応
5.0	2018.06	商標の記載の修正
6.0	2019.04	MC 2.4 に対応
7.0	2020.04	MC 2.5 に対応
8.0	2021.04	MC 2.6 に対応
9.0	2022.04	MC 2.7 に対応
10.0	2023.04	MC 2.8 に対応
11.0	2023.08	3.6. 起動、終了ファイルの導入について Unit ファイルの配置先を訂正
12.0	2024.04	MC 2.9 に対応

# はしがき

本書は、CLUSTERPRO MC ProcessSaver 2.9 for Linux (以後 ProcessSaver と記載します) のプロセス監視に関する設定、運用について記載したものです。

## (1) 商標および登録商標

- ✓ Linux は、米国およびその他の国における Linus Torvalds の登録商標です。
- ✓ Red Hat、Red Hat Enterprise Linux、OpenShift は、米国およびその他の国における Red Hat, Inc. およびその子会社の商標または登録商標です。
- ✓ Oracle は、Oracle Corporation 及びその子会社、関連会社の米国及びその他の国における登録商標です。文中の社名、商品名等は各社の商標または登録商標である場合があります。
- ✓ CLUSTERPRO、ProcessSaver は、日本電気株式会社の登録商標です。
- ✓ その他記載の製品名および会社名は、すべて各社の商標または登録商標です。
- ✓ なお、本書では®、TM マークを明記しておりません。

## (2) 本リリースの強化点について

ProcessSaver 2.9 (2024.4 月出荷版) では、下記の機能を強化しています。

- ・サポート OS の記載を更新しました。  
ProcessSaver がサポートする動作環境の詳細については、別紙『CLUSTERPRO MC ProcessSaver for Linux リリースメモ』の「動作環境」の章を参照してください。
- ・同一名プロセスの個数監視機能を強化しました。  
同一名プロセスの個数監視機能において、障害条件の変更が行えるようになりました。  
本機能の詳細については、「4.2.1. プロセス個数監視時に監視対象プロセス自身の消滅を障害条件から除外する手順」を参照してください。
- ・障害解析情報収集機能を追加しました。  
障害発生時に手動で採取していた障害解析情報をツールを利用することで自動的に採取できるようになりました。

## (3) これまでの強化履歴について

ProcessSaver 1.1 (2013.10 月出荷版) では、下記の機能を強化しています。

- ・マニュアルの改善  
各種マニュアルの記載について見直しを行い更新しました。

ProcessSaver 1.2 (2014.4 月出荷版) では、下記の機能を強化しています。

- ・リモート制御機能  
これまで各サーバーにて個別に実施する必要のあったプロセス監視の一時停止・再開などの制御を外部の運用端末などから遠隔でまとめて実施できるようになりました。  
本機能の詳細については、別紙『CLUSTERPRO MC ProcessSaver ユーザーズガイド (リモート制御機能)』を参照してください。

ProcessSaver 2.0 (2015.4 月出荷版) では、下記の機能を強化しています。

・リモート制御機能

これまでは監視機能のみが遠隔で制御できましたが、任意の業務スクリプトの遠隔実行に対応することで、業務 AP と監視の起動・終了を遠隔で一括制御できるようになりました。本機能の詳細については、別紙『CLUSTERPRO MC ProcessSaver ユーザーズガイド (リモート制御機能)』を参照してください。

ProcessSaver 2.1 (2016.4 月出荷版) では、下記の機能を強化しています。

・カーネルデーモン監視機能

これまで監視対象に指定できなかった nfsd などのカーネルデーモンプロセスを監視できるようになりました。

・プロセス個数上限監視機能

同一名プロセスの個数監視機能において、これまでの下限監視に加え上限監視を行えるようになりました。

・リソース情報収集機能

監視対象プロセス単位での使用リソースの収集が行えるようになりました。本機能の詳細については、別紙『CLUSTERPRO MC ProcessSaver ユーザーズガイド (リソース情報収集機能)』を参照してください。

ProcessSaver 2.2 (2017.4 月出荷版) では、下記の機能を強化しています。

・プロセス自動監視機能

本機能を有効化することで、稼働している常駐デーモンを自動的に死活監視できるようになりました。

本機能の詳細については、別紙『CLUSTERPRO MC ProcessSaver for Linux ユーザーズガイド(プロセス自動監視機能)』を参照してください。

ProcessSaver 2.3 (2018.4 月出荷版) では、下記の機能を強化しています。

・VMware vMotion に対応しました。(vSphere6.0 update3、vSphere6.5 update1)

以下、vMotion が実施されても、継続して ProcessSaver で監視が行えるよう対応しました。

特に設定などを行う必要はありません。

- ① コンピューティング vMotion
- ② ストレージ vMotion
- ③ クロスホスト vMotion

・コンテナ監視機能

本機能を使用することで、Docker コンテナ上で稼働しているコンテナの死活監視を行えるようになりました。

本機能の詳細については、別紙『CLUSTERPRO MC ProcessSaver for Linux ユーザーズガイド(コンテナ監視機能)』を参照してください。

ProcessSaver 2.4 (2019.4 月出荷版) では、下記の機能を強化しています。

- ・リモート監視機能

本機能を使用することで、リモートサーバーで稼働しているプロセスの死活監視を行えるようになりました。

本機能の詳細については、別紙『CLUSTERPRO MC ProcessSaver ユーザーズガイド(リモート監視機能)』を参照してください。

- ・コードワードが未登録でもインストールできるようになりました。

これまではインストール時にライセンスをチェックするため、事前にコードワードの登録が必要でした。

本リリースからは、コードワードが未登録でもインストールが可能になりました。

なお、コードワードはインストール後に登録してください。

ProcessSaver 2.5 (2020.4 月出荷版) では、下記の機能を強化しています。

- ・ストール監視機能

監視対象プロセスのプロセス状態によるストール監視(プロセス状態監視)機能を使用することで被監視対象側の作り込みが不要になり、これまでより容易にストール監視を行えるようになりました。

本機能の詳細については、『4.6.5 対象プロセスのプロセス状態によるストール監視』を参照してください。

ProcessSaver 2.6 (2021.4 月出荷版) では、下記の機能を強化しています。

- ・監視対象プロセスの状態チェック機能

運用管理コマンドに監視対象プロセスの状態チェックオプションを追加しました。

本オプションを使用することで、プロセスの死活監視、ストール監視の状態を取得することが可能となりました。

OpenShift 環境のコンテナのヘルスチェックで ProcessSaver を使用する場合は、本オプションを指定してください。

OpenShift 環境で ProcessSaver を使用する場合の詳細については、別紙『CLUSTERPRO MC ProcessSaver for Linux OpenShift 連携ガイド』を参照してください。

ProcessSaver 2.7 (2022.4 月出荷版) では、下記の機能を強化しています。

- ・Oracle VM に対応しました

ProcessSaver がサポートする動作環境の詳細については、別紙『CLUSTERPRO MC ProcessSaver for Linux リリースメモ』の「動作環境」の章を参照してください。

ProcessSaver 2.8 (2023.4 月出荷版) では、下記の機能を強化しています。

- ・サポート OS の記載を更新しました。

ProcessSaver がサポートする動作環境の詳細については、別紙『CLUSTERPRO MC ProcessSaver for Linux リリースメモ』の「動作環境」の章を参照してください。

- ・SELinux に対応しました

Linux システム用のセキュリティ・アーキテクチャである SELinux (Security-Enhanced Linux) を有効化した環境で本製品を利用可能になりました。

# 目次

1. 製品の概要について .....	1
1.1. 製品の提供する主な機能について.....	1
1.2. 製品を導入する前に .....	2
2. プロセス監視の機能.....	3
2.1. 本製品で提供するプロセス監視とは.....	3
2.2. プロセスの消滅監視と自動再開機能.....	4
2.3. グループ監視とは .....	7
2.3.1. pfile にグループを指定したグループ監視.....	8
2.3.2. pcheckを階層的に使用したグループ監視 .....	9
2.4. ストール監視とは .....	10
3. プロセス消滅監視の導入手順について .....	19
3.1. pfile、環境ファイルについて .....	19
3.2. プロセス監視の導入 .....	21
3.3. pfileファイルについて .....	23
3.4. 再起動スクリプトについて .....	31
3.5. リトライオーバースクリプトについて .....	34
3.6. 起動、終了ファイルの導入について.....	35
3.7. OS標準デーモンの監視について.....	38
4. 高度な運用手順について .....	42
4.1. グループ監視の導入手順 .....	42
4.2. 同一名プロセス監視の導入手順 .....	48
4.2.1. プロセス個数監視時に監視対象プロセス自身の消滅を障害条件から除外する手順.....	53
4.3. javaのプロセス監視の導入手順.....	55
4.4. 監視間隔を短くした運用.....	60
4.5. 一般ユーザーでのプロセス監視の実行 .....	61
4.6. ストール監視の導入手順 .....	62
4.6.1. ファイルの更新時刻によるストール監視.....	62
4.6.2. ファイルの出力メッセージによるストール監視 .....	68
4.6.3. 対象プロセスのオープンファイルによるストール監視.....	74
4.6.4. 対象プロセスの起動スレッドによるストール監視 .....	78
4.6.5. 対象プロセスのプロセス状態によるストール監視.....	82
4.7. ストール監視のユーザー組み込みライブラリの導入手順.....	90
4.8. サイレントモードでの運用手順 .....	92
4.9. pcheck 終了時に子プロセスを回収する手順.....	93
4.10. pcheck 起動時の自動待ち合わせ時間を変更する手順 .....	94
4.11. pcheck 起動時のサマリー情報を syslog 出力する手順.....	96
4.12. 監視対象選択時のプロセス情報取得量を変更する手順.....	98
4.13. 監視対象選択時のリトライ回数を変更する手順.....	99
4.14. プロセス個数監視時のリトライ回数を変更する手順.....	100
4.15. Serviceguard とのクラスターウェア連携手順 .....	102
5. 操作・運用手順 .....	110

5.1.	プロセスの状態監視について.....	110
5.2.	プロセスの運用管理について.....	120
5.3.	障害解析手順.....	124
5.4.	デバッグ支援機能.....	125
5.5.	機能制限について .....	126
6.	注意・制限事項 .....	127
7.	リファレンス.....	131

# 1. 製品の概要について

## 1.1. 製品の提供する主な機能について

本製品は、プロセスの状態監視、障害時の自動再開を行うことでシステムの高可用性を実現します。

### ・プロセスの死活監視と再開機能

Linux プラットフォームで運用される業務アプリケーションやシステムプロセスの動作状態を監視し、予期せぬ障害で異常終了した場合や動作不能状態を検出した場合に、プロセスの自動再開を行うフレームワークを提供します。

### ・グループ監視機能

特定の依存関係をもつプロセス群をグループとして定義することで、グループ単位でのプロセスの監視、再開を行うことができます。

親子関係があるプロセスや、起動、終了時に依存関係を持つプロセスを監視する場合に有効です。

### ・ストール監視機能

業務アプリケーションやシステムプロセスの動作状態を監視し、ストールなどの動作不能状態を検出した場合にプロセスの自動再開を行うフレームワークを提供します。

### ・同一名プロセスの監視機能

同一の名称をもつプロセス群を、プロセス単位に細かな監視機能を提供します。

対象のプロセスを、プロセスの起動数や引数、uid などによって識別することが可能です。

### ・管理コマンドによるメンテナンス

プロセスごとの状態監視や動的な監視停止、再開をコマンドインターフェースで操作可能です。



## 1.2. 製品を導入する前に

(1) 製品の構成について

下記のコマンドにより構成されます。

•pcheck	プロセス監視コマンド
•padmin	運用管理コマンド

下記のディレクトリを使用します。

•/opt/HA/PS	実行形式ファイルなどの格納ディレクトリ
•/var/opt/HA/PS	設定ファイル、トレースファイルなどの格納ディレクトリ

(2) オンラインマニュアルの参照について

端末の画面上で日本語環境を設定してください。

•sh の場合

```
# LANG=ja_JP.utf8
# export LANG
```

(注) 上記の設定を行っても、オンラインマニュアルが正しく表示されない場合は、

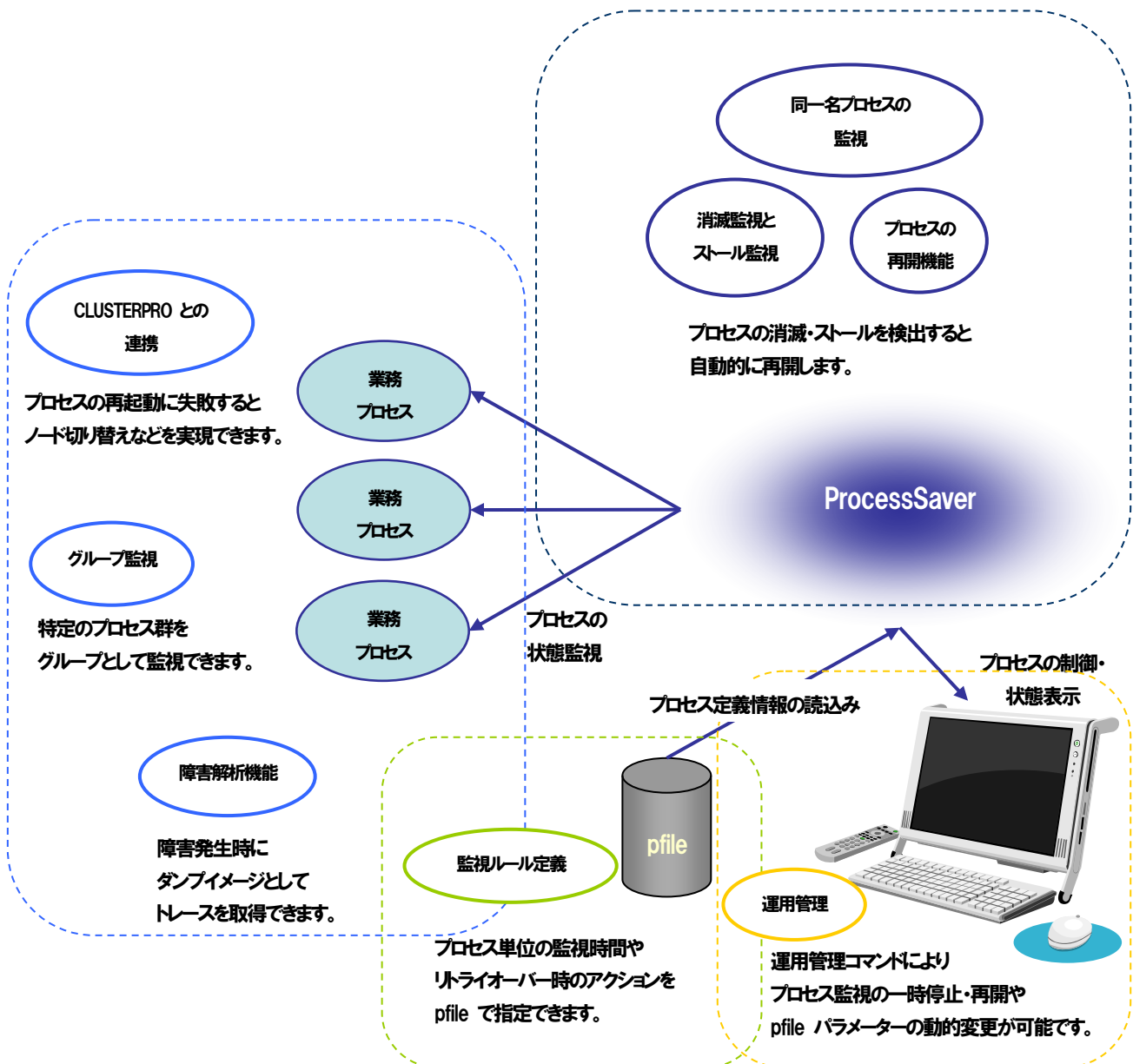
/var/opt/HA/PS/doc/man1 配下にテキストファイルがありますので、そちらを参照してください。

## 2. プロセス監視の機能

### 2.1. 本製品で提供するプロセス監視とは

本製品ではプロセス監視のフレームワークを提供します。  
本フレームワークにより、以下の機能を使用して監視を行うことが可能です。

- ・プロセスの消滅監視と自動再開機能
- ・グループ監視機能
- ・プロセスのストール監視機能
- ・同一名プロセス監視機能



## 2.2. プロセスの消滅監視と自動再開機能

プロセスの実行状態を定期的に監視することにより、消滅監視、プロセスの再開を実現します。

### (1) フレームワーク

- ① プロセス監視を実行する pcheck を起動します。
- ② pcheck は、対象プロセス群をそれぞれ一定間隔でポーリングし、プロセスの動作状態をチェックします。
  - 対象プロセスが初期起動されていなければ自動起動します。
  - 対象プロセスが正常状態であれば監視を続行します。
  - 対象プロセスの消滅またはゾンビ状態を検出すると強制終了させ指定された再起動スクリプトを実行します。
- ③ 再起動スクリプトにより、対象プロセスを再起動します。
  - 正常に再開できれば監視を続行します。
  - 一定回数リトライしても再開できない場合は、リトライオーバーアクションで指定された動作を実行します。

### (2) リトライオーバーアクション

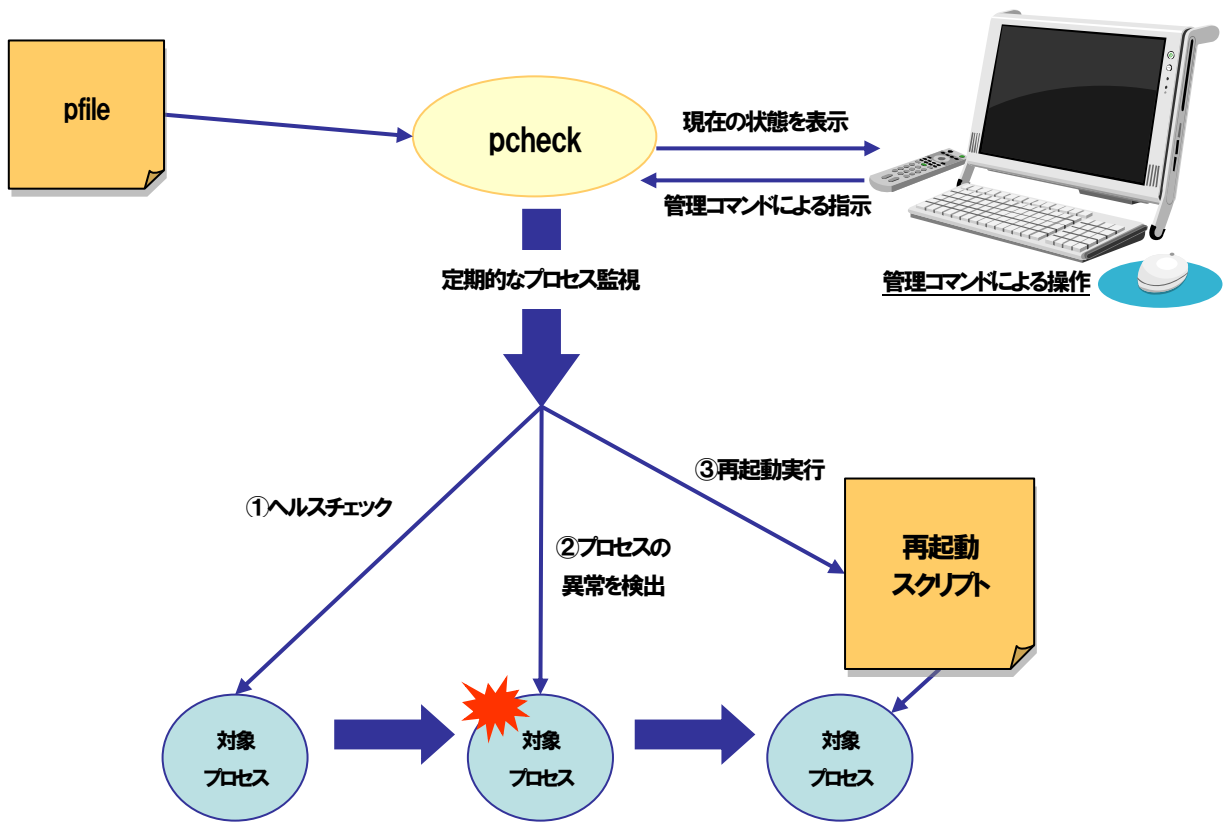
プロセスの再起動に失敗した場合に選択できるアクションは以下のとおりです。

- continue  
再起動できないプロセスのみ監視対象から外して pcheck は継続し、他のプロセスは監視を続けます。
- shutdown  
監視対象プロセスの再起動に失敗し、最大リトライ回数に達した場合に pcheck を終了します。(終了コード 0)  
たとえば pcheck に親子関係を持たせる場合、子の pcheck にはこのパラメータを指定します。
- exit  
監視対象プロセスの再起動に失敗し、最大リトライ回数に達した場合に、pcheck を終了します。(終了コード -1)  
CLUSTERPRO などのクラスターウェア製品と連携し、フェールオーバーを行う場合はこの値を指定してください。

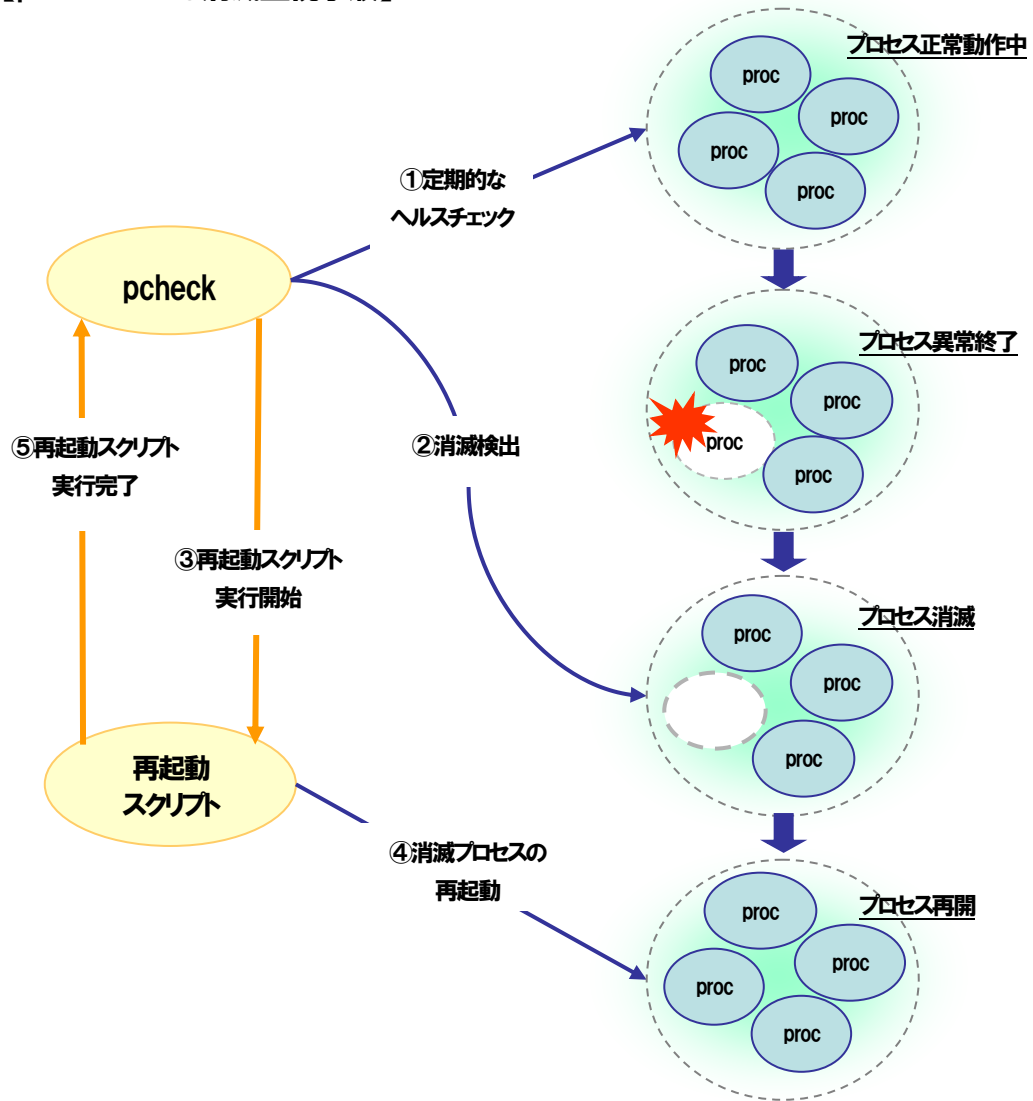
プロセスの状態監視には、/proc ファイルシステムを使用するため ps や grep などの外部コマンドを起動することはありません。

そのため、一般に行われるシェルスクリプトによるプロセス監視のプログラムに比べ CPU 負荷を軽減します。

# 【消滅監視の構成】



# 【pcheck による消滅監視手順】



## 2.3.グループ監視とは

依存関係をもつ複数の監視対象プロセスをグループとして監視することができます。

これは、pcheck が監視する特定のプロセスで障害が発生した場合に、グループ設定した監視対象プロセスすべての監視を一時停止します。

再起動スクリプトを統一することにより、グループ内の監視対象プロセスの障害発生時に、まとめて再起動を行うことができます。

設定手順の詳細については「4.1 グループ監視の導入手順」を参照してください。

グループ監視を行う場合の方式は、以下の2種類があります。

- (1) pfile の定義にて監視対象プロセスごとにグルーピングを行う方式 (2.3.1. 参照)
- (2) pcheck に親子関係を持たせ、階層的に連携して使用する方式 (2.3.2. 参照)

また、グループ監視は以下のようなプロセス群に対して有効です。

- ・親子関係のあるプロセス
- ・起動、終了に依存関係のあるプロセス
- ・共有するリソースに依存関係のあるプロセス

### 2.3.1. pfile にグループを指定したグループ監視

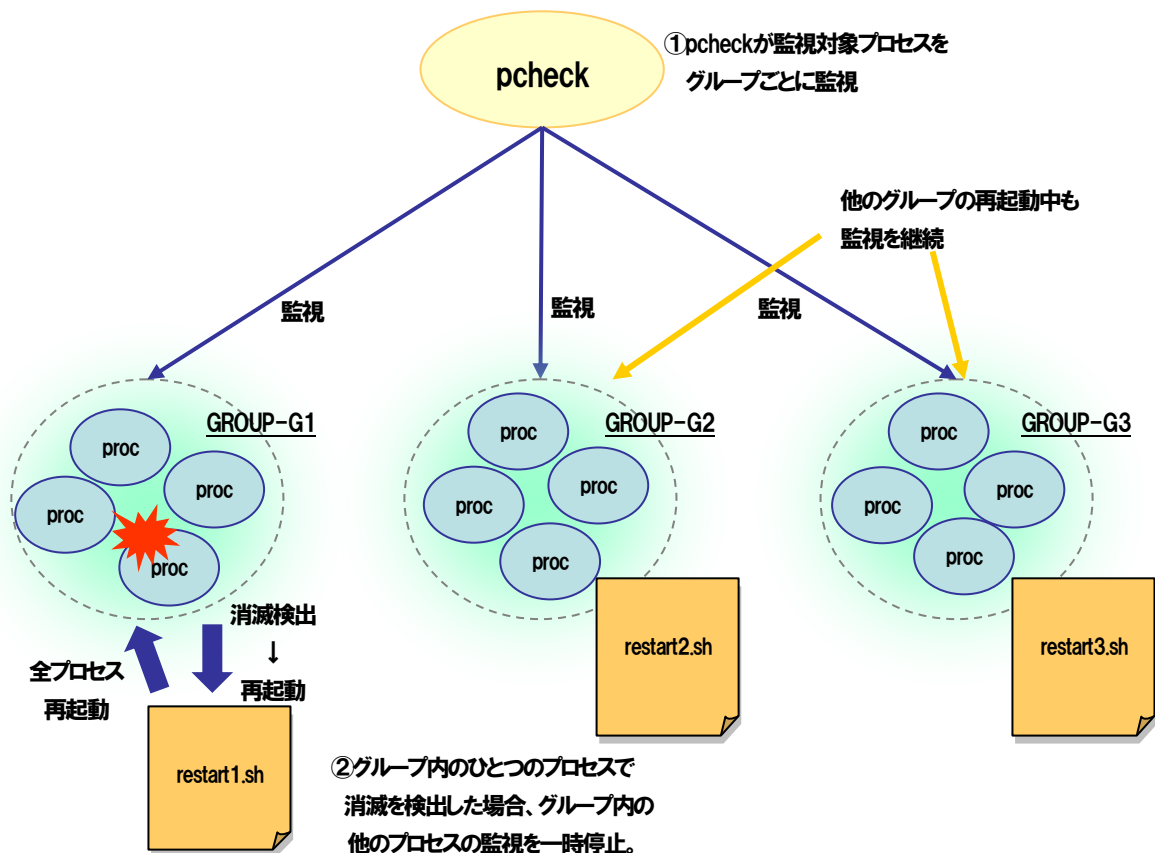
本方式の場合のグループ監視の流れは以下のとおりです。

- ① pfile のオプション情報に、関連のあるグループごとに grouptag を指定します。  
grouptag の記述方式については、「3.3. pfile ファイルについて」および「4.1. グループ監視の導入手順」を参照してください。
- ② グループ内のプロセスがダウンした場合、再起動スクリプトを実行しグループ内のすべてのプロセスを再起動します。
- ③ プロセス再起動中は、同じグループのプロセスのみ監視を一時的に中断し、再起動完了後、監視を再開します。また、グループ再起動中も、他のグループおよび従来の単独プロセスの監視は継続されます。
- ④ プロセス再起動後、すべての監視を再開します。

本グループ監視機能を使用した場合のメリットは、以下のようなものがあります。

- ・従来の pcheck を階層的に使用したグループ監視では、pcheck を複数起動する必要がありましたが、本方式では 1 つの pcheck でグループ監視を行うことができるため、使用リソースを削減できます。
- ・pfile を階層的に構成する必要がなくなり、構成がシンプルになるため、設定ミスが低減されます。

#### 【pfile にグループを設定したグループ監視】

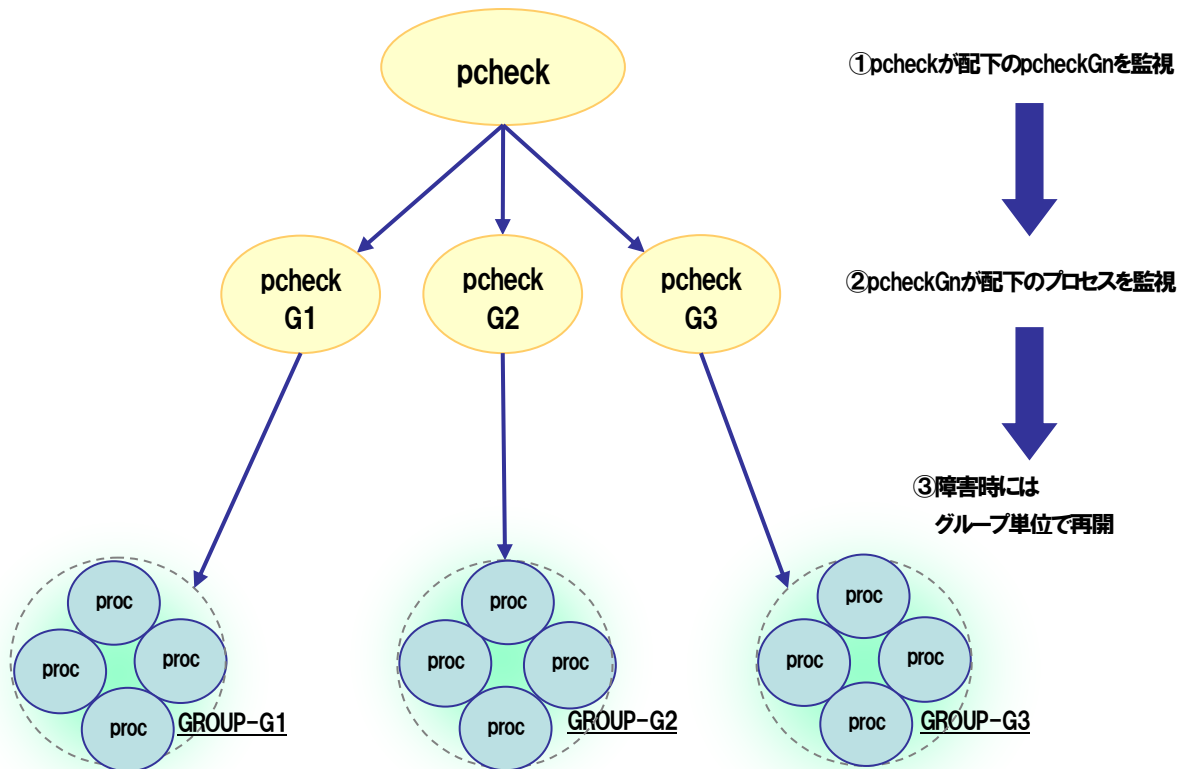


## 2.3.2. pcheck を階層的に使用したグループ監視

本方式の場合のグループ監視の流れは以下のとおりです。

- ① グループの属するプロセスを監視するために pcheckGn というグループ専用の pcheck を起動します。  
※pcheckGn は pcheck を任意の名前でコピーしたプロセスです
- ② pcheck (親) は他の監視対象プロセスと同様に pcheckGn (子) を監視します。
- ③ グループ内のあるプロセスがダウンした場合、pcheckGn は異常を検出して終了します。  
pcheck は pcheckGn の終了を検出し再起動スクリプトを実行します。
- ④ 再起動スクリプトは、グループ内のすべてのプロセスを終了、再起動した後に pcheckGn を起動します。
- ⑤ 起動された pcheckGn は、グループ内のプロセス監視を再開し、pcheck は pcheckGn の監視を再開します。

### 【pcheck を階層的に使用したグループ監視】





## 2.4. ストール監視とは

対象プロセスと連動することで、プロセスのストール監視機能を実現できます。  
この機能はプロセスが無限ループ状態や I/O ストール状態に突入した場合に、これを早期検出しプロセスの再開を支援するためのフレームワークです。

### (1) フレームワーク

プロセスのストール監視機能のフレームワークとして以下の 5 種類を提供します。

#### ① ファイルの更新時刻によるストール監視

定期的に固定名のレギュラーファイル(トレースファイルなど)を更新するプロセスを監視する場合に利用することができます。

監視対象プロセスが使用する特定のファイルの更新時刻を定期的に監視し、更新が停止した場合に、監視対象プロセスをストール状態と判定します。

このファイルは、以下の 2 種類の方法で登録できます。

- ・監視対象プロセスが定期的に更新する保証のあるファイルを利用者が監視対象ファイルとして指定する。
- ・本製品の提供するストール監視のユーザー組み込みライブラリを監視対象プロセスに組み込み、特定のファイルを監視対象ファイルとして指定する。

#### ② ファイルの出力メッセージによるストール監視

定期的に固定名のレギュラーファイル(syslog、トレースファイルなど)に特定のメッセージを出力するプロセスを監視する場合に利用することができます。

対象プロセスが出力する特定のファイル(たとえばログファイル)の特定メッセージが出力されているか否かを定期的に監視し、設定ファイルに指定された文字列に一致するメッセージの出力有無により、プロセスをストール状態と判定します。

pfile に指定された「判定文字列」の条件で指定されたメッセージがエラーかどうかを以下のように判定します。

- ・判定文字列が「あれば NG」の場合、特定メッセージが出力された場合に、異常と判定します。
- ・判定文字列が「あれば OK」の場合、特定のメッセージが出力されていない場合に、異常と判定します。

#### ③ 対象プロセスのオープンファイルによるストール監視

定期的にファイル名が不定なファイルを更新するようなプロセスを監視する場合に利用することができます。

対象プロセスがオープンするファイルのファイル名、アクセス日時、オープンファイル数を定期的に監視し、対象プロセスがオープンしているファイルのオープン情報が更新されない場合に、プロセスをストール状態と判定します。

#### ④ 対象プロセスの起動スレッドによるストール監視

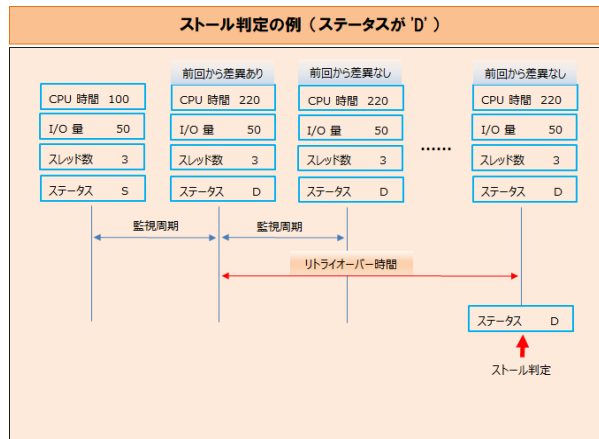
定期的にスレッドの起動、終了を繰り返すようなプロセスを監視する場合に利用することができます。

対象プロセスが起動するスレッドの数、スレッド ID を定期的に監視し、スレッドの起動、終了が行われなくなった場合に、プロセスをストール状態と判定します。

- ⑤ 対象プロセスのプロセス状態によるストール監視  
 一定時間プロセス状態に変化のない時間が続き、特定のファンクションコールで待機状態となるプロセスを監視する場合に利用することができます。  
 プロセス状態としてつぎの値を proc ファイルシステムから収集します。
- ・ CPU 使用時間
  - ・ I/O 量
  - ・ スレッド数(コンテキストスイッチ数)
  - ・ ステータス

値に変化があれば正常状態と判断し、監視を続行します。

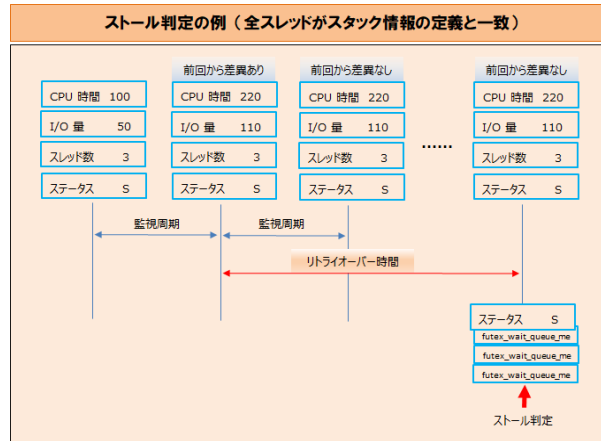
値に変化がなければ上記のステータスを再確認し、ステータスが「D' (割り込み不可能なスリープ状態)」であった場合には、プロセスをストール状態と判定します。



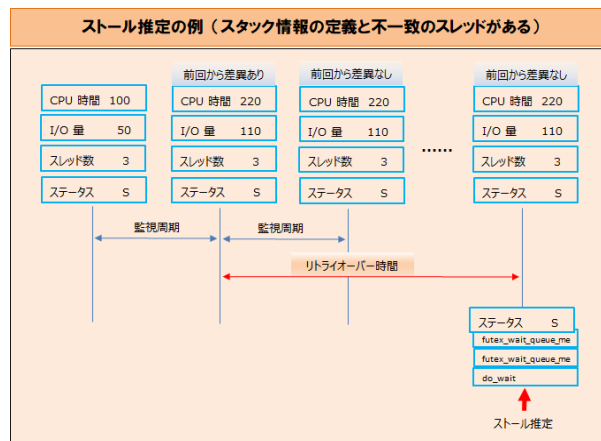
上記以外のステータスのときは、プロセスのスタック(全スレッド)から待機状態のファンクションコール名を収集します。

デフォルト(pfile に指定された **プロセスストールの判断規定** が 0 )では、収集した待機ファンクションコール名からプロセスの状態を以下のように判定します。

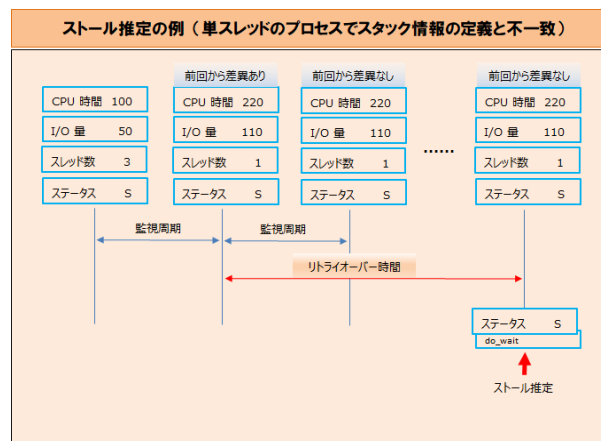
- 対象プロセス内のすべてのスレッドで待機ファンクションコール名と検索名称 **futex\_wait\_queue\_me** と部分一致する場合「ストール状態」と判断します。



- 対象プロセス内で検索名称と部分一致するスレッドと、部分一致しないスレッドがある場合「ストール推定」と判断します。

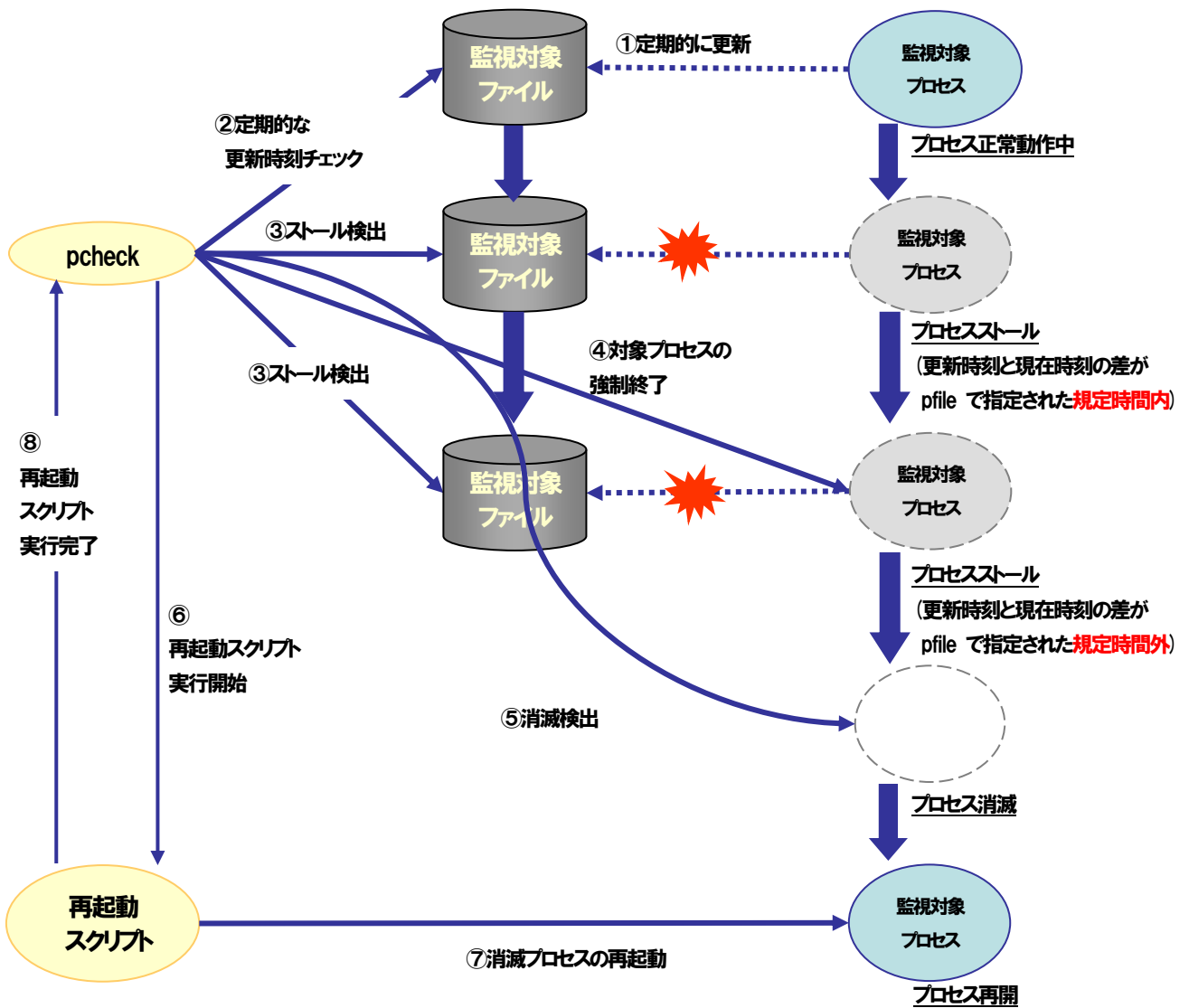


- 対象プロセス内のすべてのスレッドで待機ファンクションコール名と検索名称が部分一致しない場合「ストール推定」と判断します。

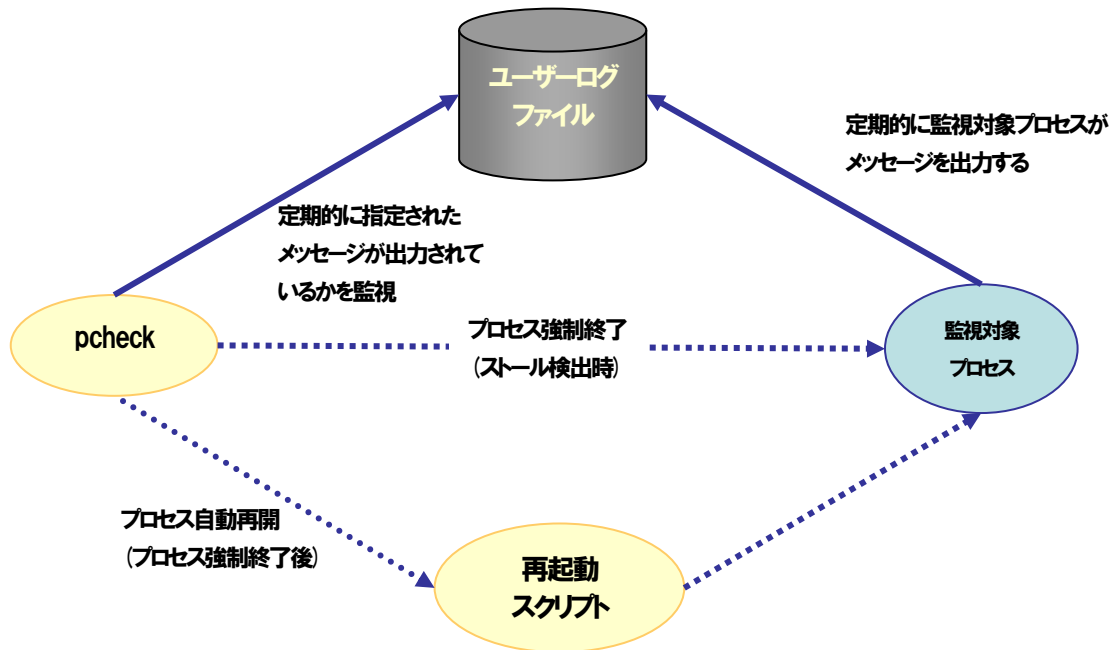


上記のほか、プロセスストールの判断規定が 1 の「ブラックリスト判定」、プロセスストールの判断規定が 2 の「ホワイトリスト判定」があります。  
詳細については 4.6.5. 対象プロセスのプロセス状態によるストール監視  
→ (5) プロセスの待機ファンクションコール名によるストール判定  
→ ② 待機ファンクションコール名でのストール判定  
を参照してください。

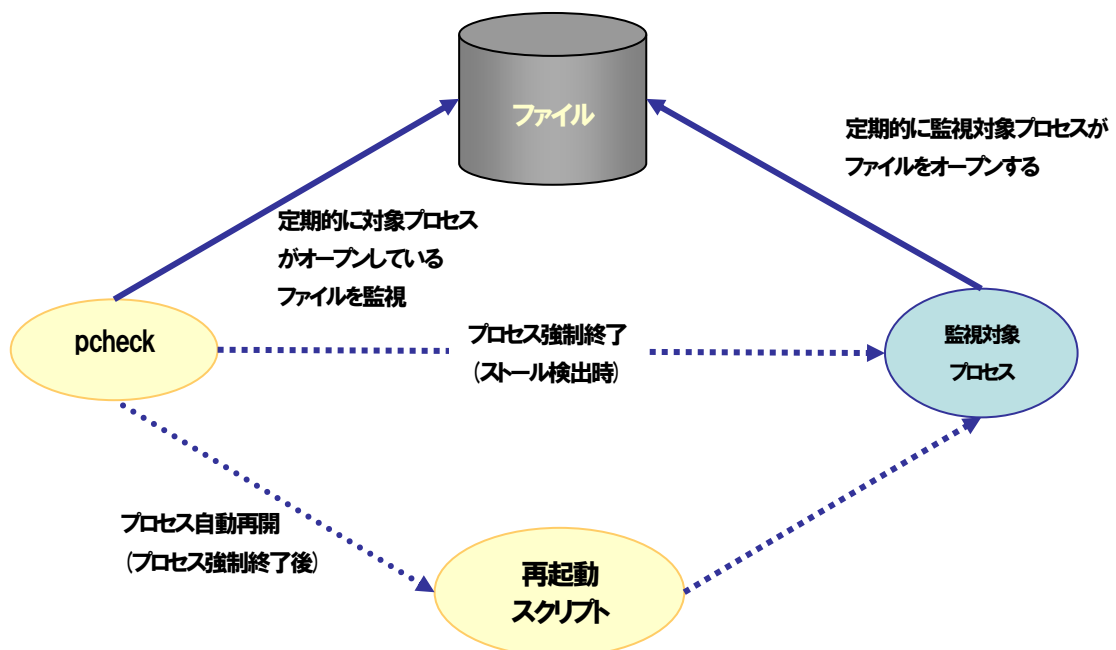
【①ファイルの更新時刻によるストール監視の構成】



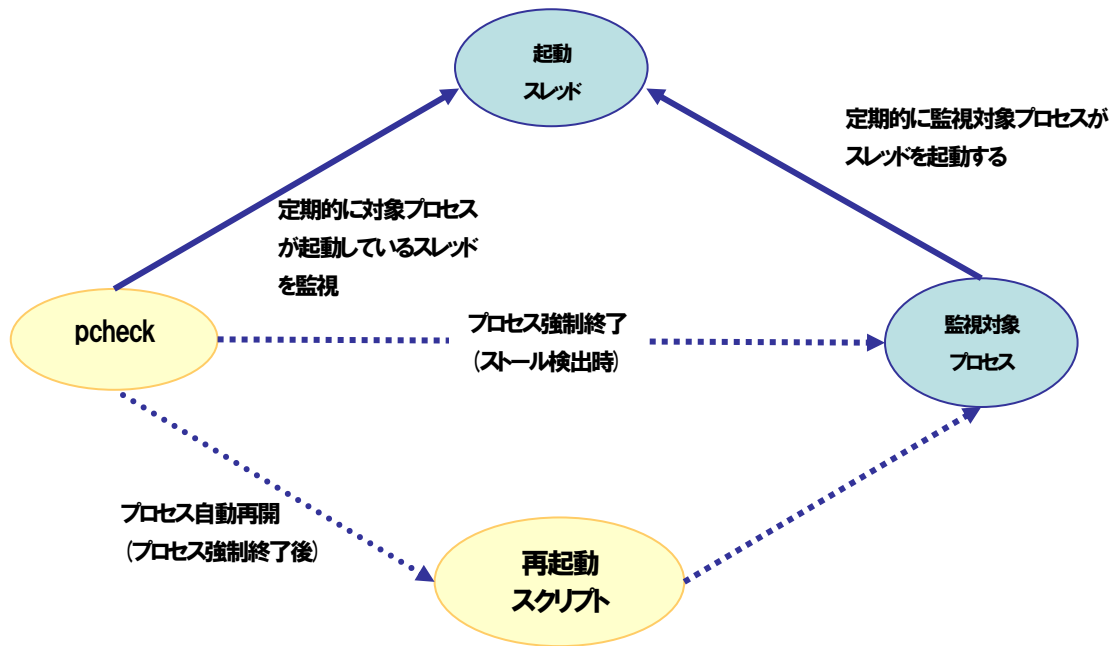
【②ファイルの出力メッセージによるメッセージ監視の構成】



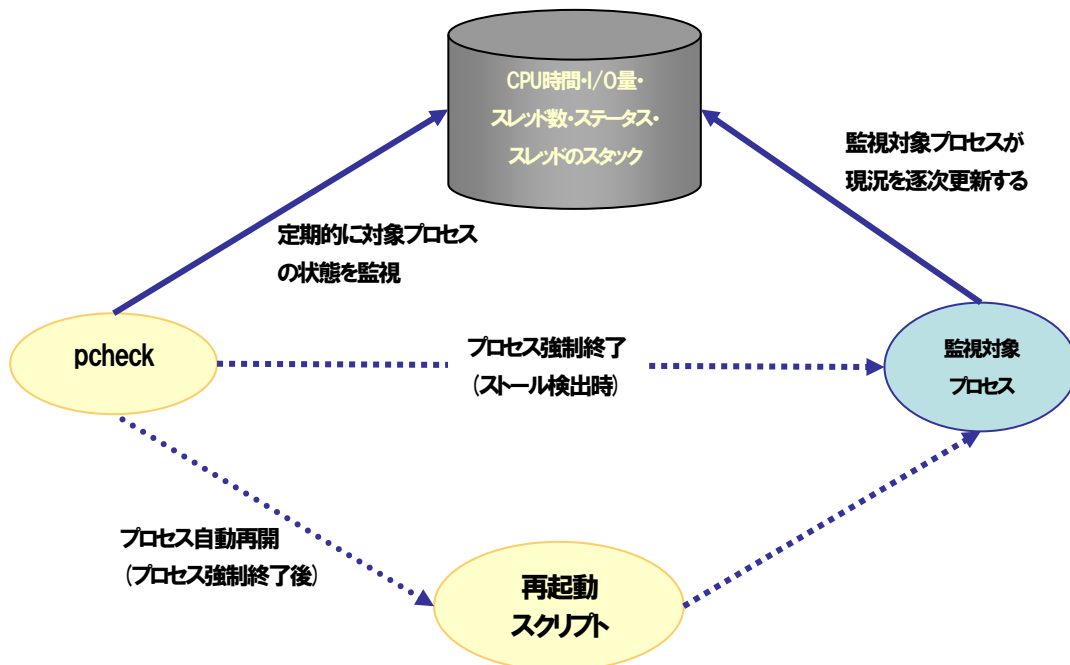
【③対象プロセスのオープンファイルによるストール監視の構成】



【④対象プロセスの起動スレッドによるストール監視の構成】



【⑤対象プロセスのプロセス状態によるストール監視の構成】



## (2) 監視手順

- ① pcheck はストール監視のために指定されたファイルの更新時刻  
(または、出力メッセージ、オープンファイル、スレッド、proc ファイル)を定期的にチェックします。

### (ファイルの更新時刻によるストール監視の場合)

- ・ファイルの更新時刻と現在時刻の差が pfile で指定された規定時間以内であれば正常状態と判断し、監視を続行します。
- ・ファイルの更新時刻と現在時刻の差が pfile で指定された規定時間を越えていればストール状態と判断し、対象プロセスを強制終了させます。
- ・ファイルが存在しない場合は、ストール監視を停止します。  
また、ファイルが生成された時点からストール監視を再開します。

### (ファイルの出力メッセージによるストール監視の場合)

- ・メッセージの未出力時にエラーと判定する場合
  - pfile で指定された規定時間内に出力された場合、正常状態と判断し、監視を続行します。
  - pfile で指定された規定時間内にメッセージが出力されなかった場合、ストール状態と判断し、対象プロセスを強制終了させます。
- ・メッセージの出力時にエラーと判定する場合
  - pfile で指定された規定時間内に出力されなかった場合、正常状態と判断し、監視を続行します。
  - pfile で指定された規定時間内にメッセージが出力された場合、ストール状態と判断し、対象プロセスを強制終了させます。

### (対象プロセスのオープンファイルによるストール監視の場合)

- ・対象プロセスがオープンしているファイルの状態(ファイル名、アクセス日時、オープンファイル数)が更新された時間が pfile で指定された規定時間以内であれば、正常状態と判断し、監視を続行します。
- ・対象プロセスがオープンしているファイルの状態(ファイル名、アクセス日時、オープンファイル数)が更新された時間が pfile で指定された規定時間を越えていれば、ストール状態と判断し、対象プロセスを強制終了させます。

### (対象プロセスの起動スレッドによるストール監視の場合)

- ・対象プロセスが起動したスレッド状態(スレッド数、スレッド ID)が更新された時間が pfile で指定された規定時間以内であれば、正常状態と判断し、監視を続行します。
- ・対象プロセスが起動したスレッド状態(スレッド数、スレッド ID)が更新された時間が pfile で指定された規定時間を越えていれば、ストール状態と判断し、対象プロセスを強制終了させます。

### (対象プロセスのプロセス状態によるストール監視の場合)

- ・対象プロセスの状態(CPU 使用時間、I/O 量、スレッド数(コンテキストスイッチ数)、ステータス)が更新された時間が pfile で指定された規定時間以内であれば正常状態と判断し、監視を続行します。
- ・対象プロセスの状態(CPU 使用時間、I/O 量、スレッド数(コンテキストスイッチ数)、ステータス) が更新された時間が pfile で指定された規定時間を越えていればステータスを再確認します。  
ステータスが「D (割り込み不可能なスリープ状態)」であればストール状態と判断し、対象プロセスを強制終了させます。  
ステータスが「D」以外の場合は判断を保留して、プロセスのスタック(全スレッド)を確認します。



- ・対象プロセスのスタック(全スレッド)の待機中のファンクションコール名を(pfile で指定された「プロセス ストールの判断規定」、および「スタック情報ファイル」の内容に沿って)確認します。  
確認の結果、ストール状態と判断したときは、対象プロセスを強制終了させます。  
ストール推定状態と判断したときは、pfile で指定された「ストール推定発生時の動作規定」に沿って、監視を続行、あるいはストール状態と判断したときと同様の手順で対象プロセスを強制終了させます。  
正常状態と判断したときは、監視を続行します。

- ② ストール状態を検出すると強制終了手順を実行します。
  - ・対象プロセスに SIGTERM を送信します。
  - ・対象プロセスが SIGTERM で終了しなければ、SIGKILL を送信します。
  - ・SIGKILL を送信しても対象プロセスが終了しなければ、SIGKILL をさらに 1 秒間隔で最大 10 回まで送信します。
  - ・それでも終了しない場合は、そのプロセスの終了を中止します。
- ③ 対象プロセスの消滅を検出し、再起動スクリプトを実行することでプロセスを再起動します。

(注) pfile の設定によっては、SIGTERM 以外の特定の signal を送信します。  
また、特定の signal で終了しない場合に実行する SIGKILL 送信による強制終了回数は、pfile の設定により変更することができます。

(注) ストール監視を導入したシステムで、date コマンドなどでサーバーのシステム日付を変更すると、ストール監視が誤動作し、不正にストール状態と判定する場合があります。  
システム日付を変更する場合は、pcheck を終了または監視停止してから実施してください。

## 3. プロセス消滅監視の導入手順について

### 3.1. pfile、環境ファイルについて

#### (1) ファイルの配置

pfile および各種環境ファイルは以下のディレクトリで管理されます。

- ・テンプレートディレクトリ  
**/var/opt/HA/PS/conf/src**
- ・実行環境ディレクトリ  
**/var/opt/HA/PS/conf/bin**

(注) 上記の実行環境ディレクトリはデフォルトであり、運用環境に合わせて任意のディレクトリを使用することができます。

#### (2) プロセス監視のテンプレート

テンプレートディレクトリには、pfile などのテンプレートが用意されています。  
必要なファイルを実行環境ディレクトリにコピーして、カスタマイズしてください。

- pfile の各種テンプレート  
**/var/opt/HA/PS/conf/src/SG\_sample** 配下
  - ・pfile  
消滅監視を定義した標準的な pfile テンプレート(デフォルト)
  - ・pfile\_syslog  
syslogd の消滅監視を定義した pfile テンプレート
  - ・sample1.sh, sample2.sh  
再起動スクリプトのテンプレート
  - ・pfile\_group  
pfile にグループ名を設定するグループ監視機能を定義したテンプレート
  - ・restart\_group01.sh, restart\_grp02.sh  
pfile にグループ名を設定するグループ監視で使用する再起動スクリプトのテンプレート群
- ユーザー組み込みスクリプトを使用するプロセス監視テンプレート  
**/var/opt/HA/PS/conf/src/SG\_linux\_sample** 配下
  - ・pfile  
ユーザー組み込みスクリプトを使用した pfile テンプレート
  - ・check.sh  
ユーザー組み込みスクリプトのテンプレート
  - ・lib\_checkproc  
監視対象プロセスのプロセス番号を出力するスクリプトのテンプレート
  - ・stallchk\_pfile  
プロセスのストール監視を指定した pfile テンプレート
  - ・stack\_def  
スタック情報ファイルのテンプレート(プロセス状態によるストール監視で使用)

- rc ファイルのテンプレート  
**/var/opt/HA/PS/conf/src/RC\_sample** 配下
  - ・rcfile  
 標準的な rc ファイルのテンプレート
- systemd ファイルのテンプレート  
**/var/opt/HA/PS/conf/src/systemd\_sample** 配下
  - ・psaver.service  
 標準的な systemd ファイルのテンプレート

また、CD-ROM のサンプルディレクトリには、以下のテンプレートが用意されています。

- OS 標準デーモン監視のテンプレート  
 使用するサンプルはディストリビューションにより選択してください。
  - ・**Red Hat Enterprise Linux 9.x 、Oracle Linux 9.x 使用時**  
 /Linux/template/pcheck\_os\_RHEL9\_sample.tar
  - ・**Red Hat Enterprise Linux 8.x 、Oracle Linux 8.x 使用時**  
 /Linux/template/pcheck\_os\_RHEL8\_sample.tar
  - ・**Red Hat Enterprise Linux 7.x 、Oracle Linux 7.x 使用時**  
 /Linux/template/pcheck\_os\_RHEL7\_sample.tar
  - ・**Red Hat Enterprise Linux 6.x 、Oracle Linux 6.x 使用時**  
 /Linux/template/pcheck\_os\_RHEL6\_sample.tar
- 自動リブート機能のテンプレート
  - ・**Red Hat Enterprise Linux 7.0 以降、Oracle Linux 7.0 以降使用時**  
 /Linux/template/haps\_AutoReboot\_sample.tar
  - ・**Red Hat Enterprise Linux 6.x 、Oracle Linux 6.x 使用時**  
 /Linux/template/haps\_AutoReboot\_RHEL6\_sample.tar

## 3.2. プロセス監視の導入

### (1) pfile、再起動スクリプトの作成

pcheck を導入する場合は、下記の手順が必要です。

#### ① 再起動スクリプトの作成

プロセス消滅時の再起動手順を再起動スクリプトに記述します。  
再起動スクリプトは通常のシェルスクリプトです。

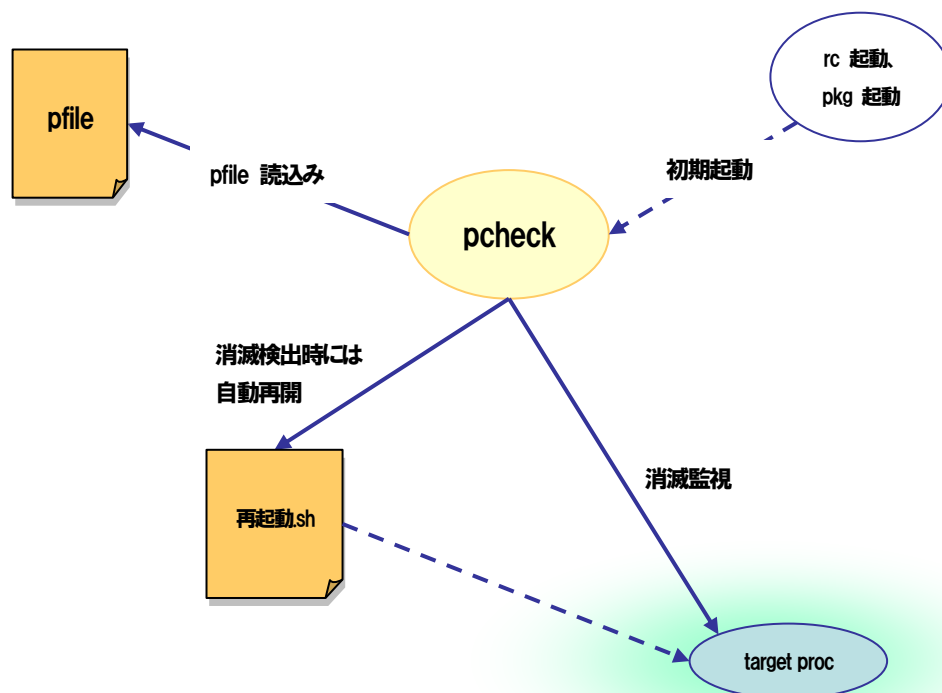
#### ② pfile の設定

pfile には以下の設定が必要です。

- プロセスグループ全体の監視ルールを定義する共通部情報の設定(必須/パラメーター)
- プロセス単位の消滅監視を定義する個別部情報の設定(必須/パラメーター)
- プロセス単位の任意の監視ルールを定義するブロック情報の設定(オプション/パラメーター)

これらのファイルは、テンプレートディレクトリにサンプルがありますので、実行環境にコピーしてからカスタマイズしてください。

なお、ひとつの pfile で複数のプロセスを監視できますが、再起動スクリプトは対象プロセスごとに定義してください。



(2) pcheck の起動、終了方式について

pcheck の起動、終了手順は運用環境によって、選択することができます。

- ① rc スクリプト、systemd から起動、終了  
rc スクリプトまたは systemd に pcheck コマンドを登録することで、OS の起動、終了と連動して pcheck の起動・終了を行うことができます。  
導入手順の詳細については「3.6 起動、終了ファイルの導入について」を参照してください。

- ② コマンドラインからの起動、終了  
コマンドラインからの起動、終了手順は以下のとおりです。

- pcheck 起動  
**`#/opt/HA/PS/bin/pcheck -f pfile &`**
  
- pcheck 終了  
**`#/opt/HA/PS/bin/padmin -f pfile -c shutdown`**
  
- プロセス監視の開始を 30 秒間待ち合わせたの pcheck 起動  
**`#/opt/HA/PS/bin/pcheck -f pfile -w 30 &`**

対象プロセスの起動に時間がかかる場合に sleep などで待ち合わせする必要があります。

(注) 上記コマンドの pfile 名は絶対パスを指定してください。

### 3.3.pfile ファイルについて

#### (1) pfile の構成について

pfile の構成は pcheck 全体の動作を定義する共通部情報と、プロセス単位の管理情報を定義する個別部情報で構成されます。

(注) pfile は pcheck プロセス単位に定義します。

複数の pcheck を起動する場合は pcheck ごとに pfile を用意してください。

このとき、pfile の **IPCKEY** と **SHM\_DUMP\_FILE** はシステムで一意的な値を設定してください。

以下のフォーマットで記述します。

```
# ProcessSaver configuration
##### PARAM #####
IPCKEY                               0x1f000001
MSG_CHECK_INTERVAL                   5
MONITOR_INTERVAL                     10
SHM_DUMP_FILE                        /var/opt/HA/PS/log/pcheck_dump

##### PENT #####
/usr/sbin/proc1:/var/opt/HA/PS/conf/src/rst1.sh:86400:3:continue
/usr/sbin/proc2:/var/opt/HA/PS/conf/src/rst2.sh:86400:3:continue
/usr/sbin/proc3:/var/opt/HA/PS/conf/src/rst3.sh:86400:3:continue
```

#### (2) 共通部情報(PARAM)の設定

共通部情報のフォーマットは以下のとおりです。

<b>IPCKEY</b>	<b><i>ipckey</i></b>
<b>MSG_CHECK_INTERVAL</b>	<b><i>msg_check_interval</i></b>
<b>MONITOR_INTERVAL</b>	<b><i>monitor_interval</i></b>
<b>SHM_DUMP_FILE</b>	<b><i>shm_dump_file</i></b>

共通部情報の設定値について以下に説明します。

#### ***ipckey***

共有メモリの key を指定します。OS 内で一意となる 1 ~ 0x7ffffff の数値を指定してください。

本製品では、pfile 情報を共有メモリに展開して監視状態の管理などを行っています。

pcheck コマンドを複数起動する設定を行っている場合、***ipckey*** を OS 内で一意にするために `ipcs(1)` コマンドを実行し、key に pfile 指定の ***ipckey*** が存在しないことを確認してください。

また、OS や他製品が使用する共有メモリの key と競合しないように比較的大きな値 (一般的には 0x1f000000 以降) を指定することを推奨しています。

### ***msg\_check\_interval***

内部イベントをスケジュールするタイマー値を設定します。  
指定値は 1 秒～60\*60\*24 秒(24 時間)の範囲です。  
デフォルト値は 5 秒です。デフォルト値を使用することを推奨します。  
***monitor\_interval*** 以下の値を設定してください。

padmin から実行されるプロセス監視停止、再開といったメッセージを受け付ける  
間隔に使用されるため、padmin コマンドからのオペレーションは、  
最大でこのタイマー値だけ遅延する可能性があります。

### ***monitor\_interval***

プロセス監視を行うタイマー値を設定します。  
指定値は 1 秒～60\*60\*24 秒(24 時間)の範囲です。デフォルト値は 10 秒です。  
デフォルト値を使用することを推奨します。  
***msg\_check\_interval*** 以上の値で、***msg\_check\_interval*** の正の整数倍の値  
を設定してください。

### ***shm\_dump\_file***

プロセス監視の内部トレースを出力するファイル名を絶対パスで設定します。  
デフォルトのファイル名は、***/var/opt/HA/PS/log/pcheck\_dump*** です。  
ファイル名は 128 文字未満で設定してください。  
複数の pcheck を起動する場合、重複しないように設定する必要があります。

なお、トレースファイルは約 3MB 程度のサイクリックログとなり、pcheck の終了時、  
padmin の reload コマンド実行時、リトライオーバーを契機として  
pfile の SHM\_DUMP\_FILE に指定した場所へ出力します。  
トレースファイルは開発元で障害時などの情報解析を行う際に必要のため、  
通常運用中は利用者が特に意識する必要はありません。  
また、ファイルは 2 世代までバックアップされます。

## (3) 個別部情報(PENT)の設定

個別部情報のフォーマットは以下のとおりです。  
ひとつの pfile において PENT は最大 256 まで記述できます。

***process\_name:shell\_path:grace\_time:retry\_num:retry\_over\_action***

または

***process\_name:shell\_path:grace\_time:retry\_num:retry\_over\_action:option***

個別部情報の設定値について以下に説明します。

## **process\_name**

対象となるプロセス名を設定します。

プロセス名はプロセス起動時に指定したコマンド名(引数は省略可能)をそのまま指定してください。登録できる文字列の長さは引数を含めて 1020 文字までとなります。プロセス名が 1020 文字を越える場合は、1020 文字までを指定するようにしてください。

この文字列を使用して対象プロセスのマッチング処理を行いますので、正確に設定してください。

また、プロセス名は通常 `ps -ef` コマンドで出力されるプロセス名をそのまま指定し、引数を付与した形式で指定してもかまいません。

引数を省略した場合は、プロセス名が完全に一致する場合のみ監視対象とします。引数を含めて指定した場合は、プロセス名は完全一致、引数部分については前方一致にて監視対象を選択します。

(注) 引数を含めて指定し、かつ完全一致によって監視対象を選択したい場合には、`pname_full_match` オプションを使用する必要があります。

**process\_name** に指定した文字列に対応するプロセスが複数存在する場合は、そのプロセスに親子関係があれば、大元の親プロセスを監視します。また、親子関係がなければ、最初にマッチしたプロセスを対象とします。同一名のプロセスを監視する場合は、特別な設定が必要になる可能性があります。設定手順の詳細については「4.2. 同一名プロセス監視の導入手順」を参照してください。

プロセス名を指定する上で、下記のような特殊なケースでは注意が必要です。

### (a) : (コロン)の付くプロセス名

パラメーターのセパレーターとして : (コロン) を使用していますので : (コロン) のつくプロセス名はそのままでは指定できません。 : (コロン) は、¥:(バックスラッシュ コロン) と指定してください。

### (b) : (コロン)の付く引数を持つプロセス

プロセスの引数には : (コロン) が付いている場合がありますが、 : (コロン) のつく引数はそのままでは指定できません。 : (コロン) は、¥:(バックスラッシュ コロン) と指定してください。

### (c) 0 番のプロセス ID をもつプロセス

ProcessSaver では 0 番のプロセス ID は監視することはできません。

### (d) [ ] (カッコ)表記のプロセス

カッコ表記のプロセスはカーネルデーモンプロセス(たとえば `[nfsd]`)を意味します。監視対象とするためには [ ] (カッコ) を除いたプロセス名を指定してください。



プロセス名の指定例です。

<syslogd の場合>

```
# ps -ef | grep syslogd
root      1925      1  0 08:27 ?          00:00:00 syslogd -m 0
```

上記の場合は、起動時のコマンド名が“syslogd -m 0”ですので、**process\_name** は syslogd または syslogd -m 0 を指定してください。

<シェルスクリプトの場合>

```
# ps -ef | grep monitor.sh
root    22313      1  0 May 24 ?          0:00          /bin/sh monitor.sh
```

シェルスクリプトの場合は、プロセス名の先頭にシェルコマンド名が付与されます。起動時のコマンド名に“monitor.sh”を指定しても、ps コマンドで確認するとシェルコマンド名が付与されているケースがあります。このような場合は、**process\_name** に /bin/sh monitor.sh を指定してください。

### **shell\_path**

監視対象プロセスの再開時に実行するシェルスクリプト(再起動スクリプト)のファイル名を設定します。ファイル名は絶対パスで 256 文字未満を指定してください。ファイル名には引数付のシェルスクリプトを指定することが可能です。また、複数のシェルスクリプトを列記する場合は、;(セミコロン)区切りで指定してください。;(セミコロン)区切りでスクリプトを指定した場合は、/bin/sh -c の引数として指定されたスクリプトを実行します。そのため、/bin/sh -c の引数として指定することのできないスクリプトおよびコマンドについては指定することができません。プロセスの再開を実行しない場合、本パラメーターは省略可能です。再起動スクリプトを指定しない場合は 0 (ゼロ) または - (ハイフン) を指定してください。

### **grace\_time**

システムが安定稼働しているとみなし、これまでのリトライ回数をリセットするまでの時間を設定します。指定値は 0 秒~31104000 秒 (60\*60\*24\*360 秒 = 約 1 年) の範囲です。一般的な例では、86400 秒 (60\*60\*24 秒 = 1 日)を指定します。この値に設定した時間内に、最大リトライ回数に達した場合はリトライオーバーアクションの設定にしたがって処理を行います。最大リトライ回数に達しなかった場合はこの値を超えた後のプロセス消滅検出時を 1 回目のリトライとして処理を行います。この際に grace\_time のカウントはリセットされます。

また、この値のカウントは最初のプロセス消滅検出時から開始されます。なお、再起動が不要な場合は、0 (ゼロ) を指定してください。本パラメーターを省略した場合、0 (ゼロ) が設定されます。

## **retry\_num**

プロセス消滅検出時に行う最大リトライ回数を設定します。  
指定値は 0 回~1024 回の範囲です。  
一般的な例では、3 回を指定します。  
なお、再起動が不要な場合は、0 (ゼロ)を指定してください。  
再起動の無限リトライを指定する場合、**grace\_time** に0(ゼロ)を指定し、**retry\_num** に0 以外の値(たとえば 1)を指定してください。  
本パラメーターを省略した場合、0 (ゼロ)が設定されます。

## **retry\_over\_action**

最大リトライ回数を超過してプロセス消滅を検出した場合の動作を設定します。  
リトライオーバーアクションには、以下のパラメーターを設定してください。

<b>continue</b>	最大リトライ回数に達したプロセスを監視対象からはずし、他のプロセスの監視は継続して行います。
<b>shutdown</b>	監視対象プロセスの再起動に失敗し、最大リトライ回数に達した場合に、pcheck を終了します。(終了コード 0) たとえば pcheck に親子関係を持たせる場合に子の pcheck にはこのパラメーターを指定します。
<b>exit</b>	監視対象プロセスの再起動に失敗し、最大リトライ回数に達した場合に、pcheck を終了します。(終了コード -1)) CLUSTERPRO などのクラスターウェア製品と連携し、フェールオーバーを行う場合はこの値を設定してください。

<リトライオーバーアクションの shutdown と exit の違いについて>

本バージョンでは終了コード以外の処理の違いや動作の差分はありません。  
そのため pcheck の終了コードを意識しないクラスターウェア製品と連携する場合は、どちらを使用しても問題ありませんが、今後はクラスターウェア製品への対応のため exit の処理を強化する可能性がありますので、基本的にクラスターウェア製品と連携する場合は exit を指定してください。

## **option**

対象プロセスを特定するためのキーを指定します。本パラメーターは省略可能です。  
オプション情報には、以下のパラメーターのいずれかを設定してください。  
複数のオプションを指定する場合には、オプション情報を “,” で区切ることで指定できます。  
オプションは最大で 20 個まで同時に指定できます。また、指定できる文字数はオプション情報全体で最大 255 文字以内です。

### **uid=xxx**

**xxx**には対象プロセスの uid を指定します。  
対象プロセスが複数存在し uid が異なる場合に有効です。

### **uname=xxx**

**xxx**には対象プロセスのユーザー名を指定します。  
対象プロセスが複数存在しユーザー名が異なる場合に有効です。  
また、ユーザー名が 9 文字以上の場合でも指定可です。

### **clear\_cmd=xxx**

**xxx**にはリトライオーバー時に実行するシェルスクリプトファイル(リトライオーバースクリプト)のファイル名を設定します。  
ファイル名に引数は指定できません。

ファイル名は絶対パスで 255 文字までを指定してください。

#### **grouptag=xxx**

**xxx** にはグループ監視を行う場合のグループ名を指定します。グループ名は 63 文字までとし、英数字で指定します。また数字のみの **grouptag** 名は指定することができません。英字のみまたは英数字を組み合わせた方式で指定してください。

(例) 指定可:grouptag=oracle\_group grouptag= group1

指定不可:grouptag=1 grouptag=8

プロセスの消滅を検出した場合に、同一のグループ名が指定されているプロセスをすべて再起動します。

#### **include\_strings=xxx または**

#### **include\_strings=xxx&yyy&zzz**

**xxx** , **yyy** , **zzz** にはプロセス名の検索条件となる文字列を指定します。

(注) プロセス名が 1020 文字を越える場合、1020 文字までを検索条件として指定してください。1021 文字以降のプロセス名は検索対象に含まれません。

複数指定する場合はアンパサンド(&)で区切ります。

本パラメーターに指定できる文字列は 255 文字までです。

なお、指定可の文字列は英数字とハイフン("-")、アンダースコア("\_")、スラッシュ("/")、ドット(".") のみです。半角スペースやイコール("=")は指定することはできません。

#### **min\_proc\_count=xxx**

**xxx** には同一名プロセスの個数を監視する場合に指定し、最小限起動しておきたいプロセス数を指定します。

指定値は 1 ~ 64(個)の範囲です。

指定された数未満となった場合にプロセス個数異常を検出します。

ただし、この設定がある場合でも監視対象として特定したプロセスの PID が消滅した場合は個数を下回っていない場合でもプロセス消滅を検出します。

監視対象として特定したプロセスの PID が消滅した場合でも、プロセス消滅を検出たくない場合には、

「4.2.1. プロセス個数監視時に監視対象プロセス自身の消滅を障害条件から除外する手順」を参照してください。

#### **max\_proc\_count=xxx**

**xxx** には同一名プロセスの個数を監視する場合に指定し、最大限起動しておきたいプロセス数を指定します。

指定値は 1 ~ 64(個)の範囲です。

指定された数を超えた場合にプロセス個数異常を検出します。

ただし、この設定がある場合でも監視対象として特定したプロセスの PID が消滅した場合は個数を上回っていない場合でもプロセス消滅を検出します。

監視対象として特定したプロセスの PID が消滅した場合でも、プロセス消滅を検出たくない場合には、

「4.2.1. プロセス個数監視時に監視対象プロセス自身の消滅を障害条件から除外する手順」を参照してください。

### **restart\_waittime=xxx**

**xxx** には再起動スクリプトによるプロセス再起動が失敗した場合にリトライにてもう一度再起動スクリプトを実行するまでの待機時間を指定します。

指定値は 1 秒～60\*60\*24 秒 (24 時間)の範囲です。

監視対象プロセスの再起動の準備およびプロセス再起動に時間がかかるような場合に有効です。

(注) **monitor\_interval** の正の整数倍の値を設定してください。

正の整数倍でない値を指定した場合、この値は

**monitor\_interval** の設定値をもとに切り上げられる場合があります。

### **restart\_timeout=xxx**

**xxx** には再起動スクリプトの実行タイムアウトを設定します。

指定値は 1 秒～60\*60\*24 秒 (24 時間)の範囲です。

再起動スクリプトが指定された時間を経過しても終了しない場合に再起動スクリプトを強制終了します。

再起動スクリプトの処理が長時間終了しなくなった場合に処理を継続させたい場合に有効です。

(注) **monitor\_interval** の正の整数倍の値を設定してください。

正の整数倍でない値を指定した場合、この値は

**monitor\_interval** の設定値をもとに切り上げられる場合があります。

### **pname\_full\_match=enable または**

### **pname\_full\_match=disable**

**enable** を指定した場合は、**process\_name** に引数を含めて指定した場合であっても、完全一致で監視対象のプロセスを選択します。

**disable** を指定した場合や本オプションを指定しない場合、下記の条件で選択されます。

- ・ **process\_name** に引数を含めて指定した場合は、プロセス名は完全一致、引数は前方一致によって監視対象のプロセスを選択します。
- ・ **process\_name** の引数を省略して指定した場合は、本オプションにかかわらず、従来どおり完全一致によって監視対象のプロセスを選択します。

(注) **enable**、**disable** 以外を指定することはできません。

#### (4) 一般的な pfile のサンプル

・OS 標準デーモンプロセスを監視する pfile の事例

(注) 下記は Red Hat Enterprise Linux 6.x の事例です。

```
# ProcessSaver for Red Hat Enterprise Linux 6.x configuration file
##### PARAM #####
IPCKEY                0x1f000101
MSG_CHECK_INTERVAL   5
MONITOR_INTERVAL     10
SHM_DUMP_FILE        /var/opt/HA/PS/log/pcheck_OS_dump

##### PENT #####
## pname:restart shell:grace:retry_count_max:retry_over_action
/sbin/init:-:0:0:continue
/sbin/rsyslogd:/etc/init.d/rsyslog start:86400:3:continue
crond:/etc/init.d/crond start:86400:3:continue
/usr/sbin/sshd:/etc/init.d/sshd start:86400:3:continue
rpcbind:/etc/init.d/rpcbind start:86400:3:continue
```

・ pfile に grouptag および restart\_timeout を指定する場合の pfile の事例

```
# ProcessSaver configuration file for group
##### PARAM #####
IPCKEY                0x1f000002
MSG_CHECK_INTERVAL   5
MONITOR_INTERVAL     10
SHM_DUMP_FILE        /var/opt/HA/PS/log/pcheck_dump_group

##### PENT #####
# Oracle Process #
ora_dbw0_db00:/opt/oracle/bin/restart_ora.sh:86400:3:exit:grouptag=oracle,restart_timeout=600
ora_lgwr_db00:/opt/oracle/bin/restart_ora.sh:86400:3:exit:grouptag=oracle,restart_timeout=600
ora_ckpt_db00:/opt/oracle/bin/restart_ora.sh:86400:3:exit:grouptag=oracle,restart_timeout=600
ora_pmon_db00:/opt/oracle/bin/restart_ora.sh:86400:3:exit:grouptag=oracle,restart_timeout=600
ora_smon_db00:/opt/oracle/bin/restart_ora.sh:86400:3:exit:grouptag=oracle,restart_timeout=600
ora_reco_db00:/opt/oracle/bin/restart_ora.sh:86400:3:exit:grouptag=oracle,restart_timeout=600

# Oracle Listener
/opt/app/oracle/product/8.1.6/bin/tnslsnr:/opt/oracle/restart_lsnr.sh:86400:3:exit

# AP 01
ap01_proc1:/opt/ap01/bin/restart_ap01.sh:86400:3:exit:grouptag=ap01
ap01_proc2:/opt/ap01/bin/restart_ap01.sh:86400:3:exit:grouptag=ap01

# AP 02
ap02_proc1:/opt/ap02/bin/restart_ap02.sh:86400:3:exit
```

### 3.4.再起動スクリプトについて

#### (1) 再起動スクリプトとは

pcheck がプロセスの消滅を検出した場合に、プロセスを再開するための手続きを記述した実行ファイルを再起動スクリプトと呼びます。

再起動スクリプトには、対象プロセスが終了した場合に必要なガベージ処理と、再起動のために必要な処理を記述してください。フォーマットは通常のシェルスクリプトです。

#### (2) 再起動スクリプト作成時の注意事項

再起動スクリプトは、対象プロセスの振る舞いに影響を受けますので、十分な注意のもとで作成してください。

- ・ 対象プロセスが使用する共有メモリ、ロックファイル、レギュラーファイルなどの資源が残っている場合、プロセスの再起動が失敗する可能性があります。  
再起動スクリプトでこれらの資源をガベージしてから、プロセスを再開するようにしてください。
- ・ 対象プロセスを起動した後に /bin/sleep などによって一定時間の猶予をとってください。  
対象プロセスの起動に時間がかかると、pcheck が先に起動される可能性があります。
- ・ pcheck は、プロセス再開時に再起動スクリプトの終了を待ち合わせてからプロセスの監視を再開します。  
再起動スクリプト内で長時間 sleep により待ち合わせを行った場合や、処理に時間がかかる場合、プロセス監視が効果的に作用しない場合があります。
- ・ デーモン化されていないプロセスを起動する場合は、コマンドラインの最後に & を付与し、バックグラウンドで起動してください。再起動スクリプトは必ず終了する必要があります。
- ・ 再起動スクリプトで実行するコマンドは絶対パスで呼び出してください。  
コマンドへのパスが張られていないとコマンドの実行に失敗する場合があります。
- ・ 再起動スクリプトが異常終了のステータス(0 以外)を返却すると、pcheck はプロセスの再起動に失敗したとみなします。
- ・ 再起動スクリプトから起動するプロセスが環境変数に依存している場合は、その環境変数を設定してからプロセスを呼び出してください。  
再起動に失敗した場合は、環境変数 PATH に /usr、/usr/bin、/bin などを追加することで回避できる場合があります。
- ・ 各監視対象の製品が提供している終了・起動・再起動コマンドやスクリプトは、終了できなかったプロセスを終了するために、SIGKILL などを送信して強制終了させる仕様のものがあります。  
再起動スクリプト名に監視対象プロセス名自体が含まれるような形で作成していた場合、強制終了の対象と誤認されて、再起動スクリプトが異常終了する場合があります。  
pcheck 名や再起動スクリプト名は監視対象プロセス名を含まないように作成することを推奨します。

- 再起動スクリプトを;(セミコロン)区切りで複数指定した場合、/bin/sh -c の引数として指定されたスクリプトを実行します。そのため、/bin/sh -c の引数として指定することのできないスクリプトやコマンドについては指定することができません。  
また、記載された構文のまま実行されますので、シェルの構文に従った方式で指定する必要があります。
- 再起動スクリプトを;(セミコロン)区切りで複数指定した場合には、最後に指定されたスクリプトの終了ステータスでプロセスの起動・再起動に失敗したかどうかを判定します。
- 再起動スクリプトを;(セミコロン)区切りで複数指定した場合には、restart\_timeout オプションは使用できません。

### (3) 再起動スクリプトのサンプル

#### ◆ 単一プロセスのサンプル (sample1.sh)

プロセス samplep が消滅した場合に、再起動する記述例。

```
#!/bin/sh
/usr/xxx/samplep > /dev/null 2>&1 &
```

プロセス samplep の起動に時間がかかるような場合は、一定時間待ち合わせる。

```
#!/bin/sh
/usr/xxx/samplep > /dev/null 2>&1 &
/bin/sleep 10
```

#### ◆ 依存関係のある複数プロセスのサンプル (sample2.sh)

プロセス samplep1 が消滅した場合に、依存関係のある samplep2 、samplep3 を強制終了させてから、再起動する記述例。

```
#!/bin/sh
PS_CMD="/bin/ps"
GREP_CMD="/bin/grep"
AWK_CMD="/bin/awk"
PROC_LIST="samplep2 samplep3"
for i in $PROC_LIST
do
    # 以下の PID の検索方法は一例です。
    pid=`${PS_CMD} -ef | ${GREP_CMD} "${i}" | ${GREP_CMD} -v ¥
"${GREP_CMD} | ${AWK_CMD} '{printf("%s ",$2)}END{printf("¥n")}'`
    if [ -n "$pid" ]
    then
        /bin/kill -9 $pid
    fi
done
/usr/xxx/samplep1 > /dev/null 2>&1 &
/usr/xxx/samplep2 > /dev/null 2>&1 &
/usr/xxx/samplep3 > /dev/null 2>&1 &
/bin/sleep 10
exit 0
```

- ◆ 再起動スクリプト複数指定のサンプル (sample\_stop.sh、sample\_start.sh)  
プロセス samplep が消滅した場合に、関連プロセスを終了して再起動する記述例。

<pfile の記述例>

```
##### PENT #####
```

```
samplep:/var/opt/HA/PS/conf/src/bin/stop.sh;/var/opt/HA/PS/conf/bin/start.sh:86400:3:continue
```

再起動スクリプトを実行する順に、;(セミコロン)区切りで指定します。

プロセス samplep と関連プロセスを終了するスクリプト(sample\_stop.sh)

```
#!/bin/sh

/etc/init.d/samplep_rc stop
/bin/sleep 10
```

プロセス終了に時間がかかるような場合は、一定時間の待ち合わせを設定します。

プロセス samplep と関連プロセスを起動するスクリプト(sample\_start.sh)

```
#!/bin/sh

/etc/init.d/samplep_rc start
/bin/sleep 10
```

起動に時間がかかるような場合は、起動するまで一定時間の待ち合わせを設定します。



## 3.5. リトライオーバースクリプトについて

### (1) リトライオーバースクリプト

pcheck が、対象プロセスの再起動に一定回数失敗するとリトライオーバーとなり監視を停止します。リトライオーバースクリプトを定義すると、プロセス再開に失敗した後のさまざまな後処理を実行できます。

フォーマットは通常のシェルスクリプトです。

なお、clear\_cmd で指定するスクリプト名に引数は指定できません。

また、リトライオーバーアクションが shutdown または exit の場合、pcheck プロセスはスクリプトの終了を待ち受けませんので、pcheck プロセスはスクリプトの実行後、即座に終了します。

そのためクラスターウェアと連携し、pcheck の消滅を契機としたフェールオーバーを行う場合は正しくリトライオーバースクリプトの実行ができない可能性がありますのでご注意ください。

```
# ProcessSaver configuration file
##### PARAM #####
IPCKEY                                0x10000001
MSG_CHECK_INTERVAL                   5
MONITOR_INTERVAL                     10
SHM_DUMP_FILE                        /var/opt/HA/PS/log/pcheck_dump

##### PENT #####
test1::-0:0:shutdown:clear_cmd=/tmp/getlog.sh
```

### (2) 運用事例

以下のような用途に使用可能です。

- ・ 業務がプロセス再起動で復旧しない場合にトレースファイルなどをセーブする。
- ・ シングルノードにおいて業務がプロセス再起動で復旧しない場合に OS 再起動を実行する。

### 3.6. 起動、終了ファイルの導入について

起動、終了ファイルの導入手順について説明します。

- Red Hat Enterprise Linux 7.0 以降、Oracle Linux 7.0 以降の場合

OS の起動、終了と同期をとって pcheck を起動、終了する場合は、systemd への設定が必要です。サンプルの Unit ファイルを必要に応じてカスタマイズし、systemd 管理ディレクトリにコピーします。ファイル名は任意の名前に変更可能です。

```
# cp -Rp /var/opt/HA/PS/conf/src/systemd_sample/psaver.service  
/etc/systemd/system/psaver.service  
※実際には一行で入力してください。
```

次に、以下のコマンドを実行して systemd へ登録してください。

```
# systemctl enable psaver.service
```

登録後、以下のコマンドを実行して登録内容を反映し、pcheck を起動してください。

```
# systemctl daemon-reload  
# systemctl start psaver.service
```

上記を設定した場合、pcheck が監視するプロセスも systemd への設定が必要です。systemd に登録した pcheck から監視対象プロセスの再起動を行う場合は、Unit ファイルに以下の記載が必要です。

```
KillMode=process
```

systemd に登録した pcheck の起動、終了は systemctl コマンドを使用してください。設定の際は、pcheck との依存関係を正しく設定してください。依存関係の設定方法の詳細については、systemctl のマニュアルを参照してください。

< Unit ファイルのサンプル >

```
[Unit]  
Description=ProcessSaver  
After=syslog.target  
  
[Service]  
ExecStart=/opt/HA/PS/bin/pcheck -f /var/opt/HA/PS/conf/bin/pfile  
ExecStop=/opt/HA/PS/bin/padmin -f /var/opt/HA/PS/conf/bin/pfile -c shutdown  
KillMode=process  
  
[Install]  
WantedBy=multi-user.target
```

・ Red Hat Enterprise Linux 6.x Oracle Linux 6.x の場合

OS の起動、終了と同期をとって pcheck を起動、終了する場合は、rc スクリプトの設定が必要です。サンプルの rc スクリプトを必要に応じてカスタマイズし、/etc/init.d/ 配下にコピーします。ファイル名は任意の名前に変更可能です。

```
# cp /var/opt/HAPS/conf/src/RC_sample/rcfile /etc/rc.d/init.d/psaver
```

chkconfig --add コマンドを実行し、/etc/rc.d/rc\*.d 配下に rc スクリプトのシンボリックリンクを作成するか、手動で /etc/rc.d/rc\*.d 配下に rc スクリプトのシンボリックリンクを作成します。また、上記を設定した場合、pcheck が監視するプロセスも rc スクリプトの設定が必要です。設定の際は、pcheck との依存関係を正しく設定してください。

(1) chkconfig --add コマンドを実行する手順

chkconfig --add コマンドを実行して /etc/rc.d/rc\*.d 配下に rc スクリプトのシンボリックリンクを作成します。

(例)

```
# chkconfig --add psaver
```

シンボリックリンクは、指定された rc スクリプトのコメントブロック(chkconfig ブロック)の設定値にしたがい作成されます。

(2) 手動でシンボリックリンクを作成する手順

ln(1) コマンドを実行し、シンボリックリンクを作成します。

S, K の後に続く番号は、何番でもかまいませんが、番号の若い順に実行されるため、依存関係のある rc スクリプトが存在する場合は、注意が必要です。

また、pcheck の終了処理は、必ず監視対象プロセスの終了処理より前に実行する必要があります。

このため rc スクリプトの pcheck 終了時のリンクファイルは、最初に行われる K01 で設定することを推奨します。なお、番号(K01 など)を除くリンクファイル名(psaver など)は実体の rc スクリプト名と同じである必要があります。

(例)

```
<rc0.d>
# ln -s /etc/init.d/psaver /etc/rc.d/rc0.d/K01psaver
<rc1.d>
# ln -s /etc/init.d/psaver /etc/rc.d/rc1.d/K01psaver
<rc2.d>
# ln -s /etc/init.d/psaver /etc/rc.d/rc2.d/S99psaver
<rc3.d>
# ln -s /etc/init.d/psaver /etc/rc.d/rc3.d/S99psaver
<rc4.d>
# ln -s /etc/init.d/psaver /etc/rc.d/rc4.d/K01psaver
<rc5.d>
# ln -s /etc/init.d/psaver /etc/rc.d/rc5.d/S99psaver
<rc6.d>
# ln -s /etc/init.d/psaver /etc/rc.d/rc6.d/K01psaver
```

< rc スクリプトのサンプル >

```
#!/bin/sh
#
# psaver          Start/Stop the ProcessSaver daemon.
#
# chkconfig: 235 99 01
# description: ProcessSaver Sample File (Red Hat Linux)
# processname: pcheck

LANG=C
export LANG

trap "" 1 2 3 13 15

#
# Environment
#
PCHECK=/opt/HA/PS/bin/pcheck
PADMIN=/opt/HA/PS/bin/padmin
PFILE=/var/opt/HA/PS/conf/bin/pfile

case $1 in
'start')
    echo "Starting ProcessSaver"
    ulimit -c unlimited
    ## You must execute on background.
    # (${PCHECK} -f ${PFILE} &) >/dev/null 2>&1 ★先頭の # を削除します。
    touch /var/lock/subsys/psaver
    ;;

'stop')
    echo "Stopping ProcessSaver"
    ${PADMIN} -f ${PFILE} -c shutdown
    rm -f /var/lock/subsys/psaver
    ;;

'restart')
    echo "Stopping ProcessSaver"
    ${PADMIN} -f ${PFILE} -c shutdown
    rm -f /var/lock/subsys/psaver
    /bin/sleep 10

    echo "Starting ProcessSaver"
    ulimit -c unlimited
    ## You must execute on background.
    # (${PCHECK} -f ${PFILE} &) >/dev/null 2>&1 ★先頭の # を削除します。
    touch /var/lock/subsys/psaver
    ;;
```

### 3.7.OS 標準デーモンの監視について

Linux における OS 標準デーモンの死活監視を実現します。

#### (1) フレームワーク

OS 標準デーモン監視を実行する pcheck を起動します。

- ① pcheck は、対象プロセス群をそれぞれ一定間隔で監視しプロセスの動作状態をチェックします。
  - 対象プロセスが正常状態であれば監視を続行します。
  - 対象プロセスの消滅またはゾンビ状態を検出すると、指定された再起動スクリプトを実行します。
- ② 再起動スクリプトにより、対象プロセスを再起動します。
  - 正常に再開できれば監視を続行します。
  - 一定回数リトライしても再開できない場合は、再起動できないプロセスを監視対象から外し、他のプロセスは監視を継続します。

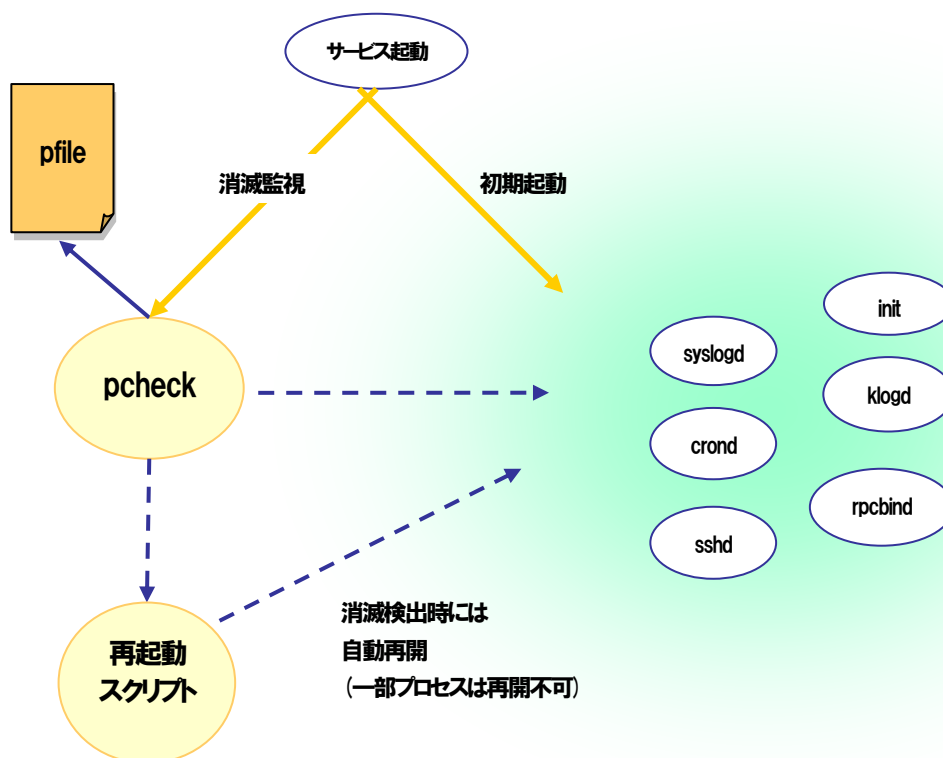
OS の起動、終了と同期をとって pcheck を起動、終了する場合は、以下の設定が必要です。詳細については、「3.6. 起動、終了ファイルの導入について」を参照してください。

・Red Hat Enterprise 7.0 以降、Oracle Linux 7.0 以降の場合

systemd による自動起動

・Red Hat Enterprise 6.x、Oracle Linux 6.x の場合

chkconfig を使用したサービス起動



(2) OS 標準デーモン監視を実装したサンプルについて

OS 標準デーモン監視のサンプルが CD-ROM に添付されています。  
使用するサンプルはディストリビューションにより選択してください。  
添付されている手順書、導入手順に従ってご利用ください。

**Red Hat Enterprise Linux 9.x、Oracle Linux 9.x 使用時**

サンプルファイル: pcheck\_os\_RHEL9\_sample.tar  
手順書: README\_pcheck\_os(RHEL9)\_euc.txt  
          README\_pcheck\_os(RHEL9)\_sjs.txt

**Red Hat Enterprise Linux 8.x、Oracle Linux 8.x 使用時**

サンプルファイル: pcheck\_os\_RHEL8\_sample.tar  
手順書: README\_pcheck\_os(RHEL8)\_euc.txt  
          README\_pcheck\_os(RHEL8)\_sjs.txt

**Red Hat Enterprise Linux 7.x、Oracle Linux 7.x 使用時**

サンプルファイル: pcheck\_os\_RHEL7\_sample.tar  
手順書: README\_pcheck\_os(RHEL7)\_euc.txt  
          README\_pcheck\_os(RHEL7)\_sjs.txt

**Red Hat Enterprise Linux 6.x、Oracle Linux 6.x 使用時**

サンプルファイル: pcheck\_os\_RHEL6\_sample.tar  
手順書: README\_pcheck\_os(RHEL6)\_euc.txt  
          README\_pcheck\_os(RHEL6)\_sjs.txt

以下に、サンプルを使用した場合に監視される OS 標準デーモンの一覧を示します。  
なお、一部プロセス消滅の可能性がないカーネルプロセスの監視を含んでおります。

【pfile\_os の設定例 (Red Hat Enterprise Linux 9.x、Oracle Linux 9.x 用)】

```
##### PARAM #####
IPCKEY                0x1f000101
MSG_CHECK_INTERVAL   5
MONITOR_INTERVAL     10
SHM_DUMP_FILE        /var/opt/HA/PS/log/pcheck_OS_dump

##### PENT #####
## pname:restart shell:grace:retry_count_max:retry_over_action
/usr/lib/systemd/systemd:-:0:0:continue
/sbin/auditd:systemctl start auditd.service:86400:3:continue
/usr/sbin/rsyslogd:systemctl restart rsyslog.service:86400:3:continue
sshd: /usr/sbin/sshd:systemctl restart sshd.service:86400:3:continue
/usr/sbin/crond:systemctl restart crond.service:86400:3:continue
```

【pfile\_os の設定例 (Red Hat Enterprise Linux 8.x、Oracle Linux 8.x 用)】

```
##### PARAM #####
IPCKEY                0x1f000101
MSG_CHECK_INTERVAL   5
MONITOR_INTERVAL     10
SHM_DUMP_FILE        /var/opt/HA/PS/log/pcheck_OS_dump

##### PENT #####
## pname:restart shell:grace:retry_count_max:retry_over_action
/usr/lib/systemd/systemd:-:0:0:continue
/usr/sbin/rsyslogd:systemctl restart rsyslog.service:86400:3:continue
/usr/sbin/crond:systemctl restart crond.service:86400:3:continue
/usr/sbin/sshd:systemctl restart sshd.service:86400:3:continue
```

【pfile\_os の設定例 (Red Hat Enterprise Linux 7.x、Oracle Linux 7.x 用)】

```
##### PARAM #####
IPCKEY                0x1f000101
MSG_CHECK_INTERVAL   5
MONITOR_INTERVAL     10
SHM_DUMP_FILE        /var/opt/HA/PS/log/pcheck_OS_dump

##### PENT #####
## pname:restart shell:grace:retry_count_max:retry_over_action
/usr/lib/systemd/systemd:-:0:0:continue
/usr/sbin/rsyslogd:systemctl start rsyslog.service:86400:3:continue
/usr/sbin/crond:systemctl start crond.service:86400:3:continue
/usr/sbin/sshd:systemctl start sshd.service:86400:3:continue
/sbin/rpcbind:systemctl start rpcbind.service:86400:3:continue
```

【pfile\_os の設定例 (Red Hat Enterprise Linux 6.x、Oracle Linux 6.x 用)】

```
##### PARAM #####
IPCKEY                0x1f000101
MSG_CHECK_INTERVAL   5
MONITOR_INTERVAL     10
SHM_DUMP_FILE        /var/opt/HAVPS/log/pcheck_OS_dump

##### PENT #####
## pname:restart shell:grace:retry_count_max:retry_over_action
/sbin/init:-:0:0:continue
/sbin/rsyslogd:/etc/init.d/rsyslog start:86400:3:continue
crond:/etc/init.d/crond start:86400:3:continue
/usr/sbin/sshd:/etc/init.d/sshd start:86400:3:continue
rpcbind:/etc/init.d/rpcbind start:86400:3:continue
```

(3) 運用上の注意・制限事項

- ・systemd、init プロセスは消滅することがないため、監視は必須ではありません。  
不要であれば、削除してください。
- ・各ディストリビューションに付属する各種の「サービス管理ツール」(GUI)からサービス起動の操作(起動、終了、再起動)には対応していません。

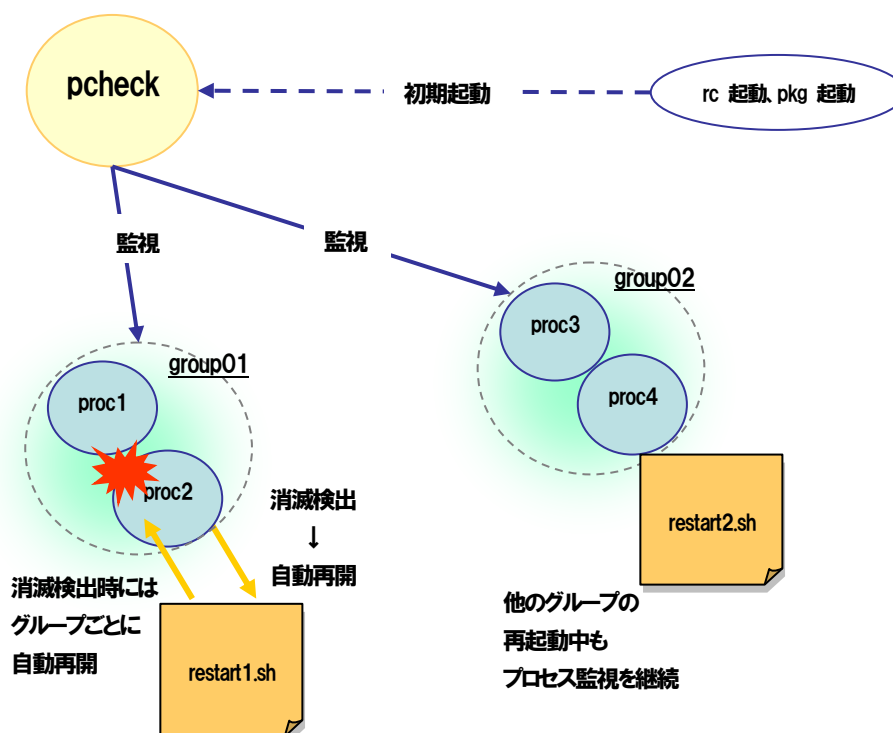


## 4. 高度な運用手順について

### 4.1. グループ監視の導入手順

(1) pfile にグループ名を設定したグループ監視の導入手順

本機能を導入するには、pfile にグループ名を指定する手続きが必要となります。  
ここでは、group01、group02 の2種類のグループを監視する pcheck を想定して説明します。  
pfile を pfile\_group、group01 を再起動するスクリプトを restart\_group01.sh、  
group02 を再起動するスクリプトを restart\_group02.sh とします。



#### ① pcheck

pcheck は指定されたグループごとにプロセスの消滅監視および再開を行います。

pfile で定義する再起動スクリプトには、グループごとに監視するすべてのプロセスを終了させた後で再起動するような処理を必要に応じて作成してください。

このとき、以下の点に注意してください。

- ・ pfile で定義する再起動スクリプトは、同一グループ内で同じ再起動スクリプトを使用してください。
- ・ pfile で定義する retry\_over\_action、grace 値、retry\_count などの各設定値は同一グループ内で同じ値を指定してください。

## グループ監視における pcheck のサンプル (pfile\_group)

```
##### PARAM #####
IPCKEY                0x10000001
MSG_CHECK_INTERVAL   5
MONITOR_INTERVAL     10
SHM_DUMP_FILE        /var/opt/HA/PS/log/pcheck_group_dump

##### PENT (group01) #####
/bin/proc1:/bin/restart_group01.sh:86400:3:continue:grouptag=group01
/bin/proc2:/bin/restart_group01.sh:86400:3:continue:grouptag=group01

##### PENT (group02) #####
/bin/proc3:/bin/restart_group02.sh:86400:3:continue:grouptag=group02
/bin/proc4:/bin/restart_group02.sh:86400:3:continue:grouptag=group02
```

## ② 再起動スクリプト

### グループ監視における再起動スクリプトのサンプル 1 (restart\_group01.sh)

```
#!/bin/sh

export LANG=C

# target process killed
PS_CMD="/bin/ps"
GREP_CMD="/bin/grep"
AWK_CMD="/bin/awk"
PROC_LIST="/bin/proc1 /bin/proc2"
for i in $PROC_LIST
do
    # 以下の PID の検索方法は一例です。
    pid=$( ${PS_CMD} -ef | ${GREP_CMD} "${i}" | ${GREP_CMD} -v "${GREP_CMD}" ¥
    | ${AWK_CMD} '{printf("%s ",$2)}END{printf("¥n")}'
    if [ -n "$pid" ]
    then
        /bin/kill -9 $pid
    fi
done

# target process start
/bin/proc1 > /dev/null 2>&1 &
/bin/proc2 > /dev/null 2>&1 &
/bin/sleep 10

exit 0
```

グループ監視における再起動スクリプトのサンプル2 (restart\_group02.sh)

```
#!/bin/sh

export LANG=C

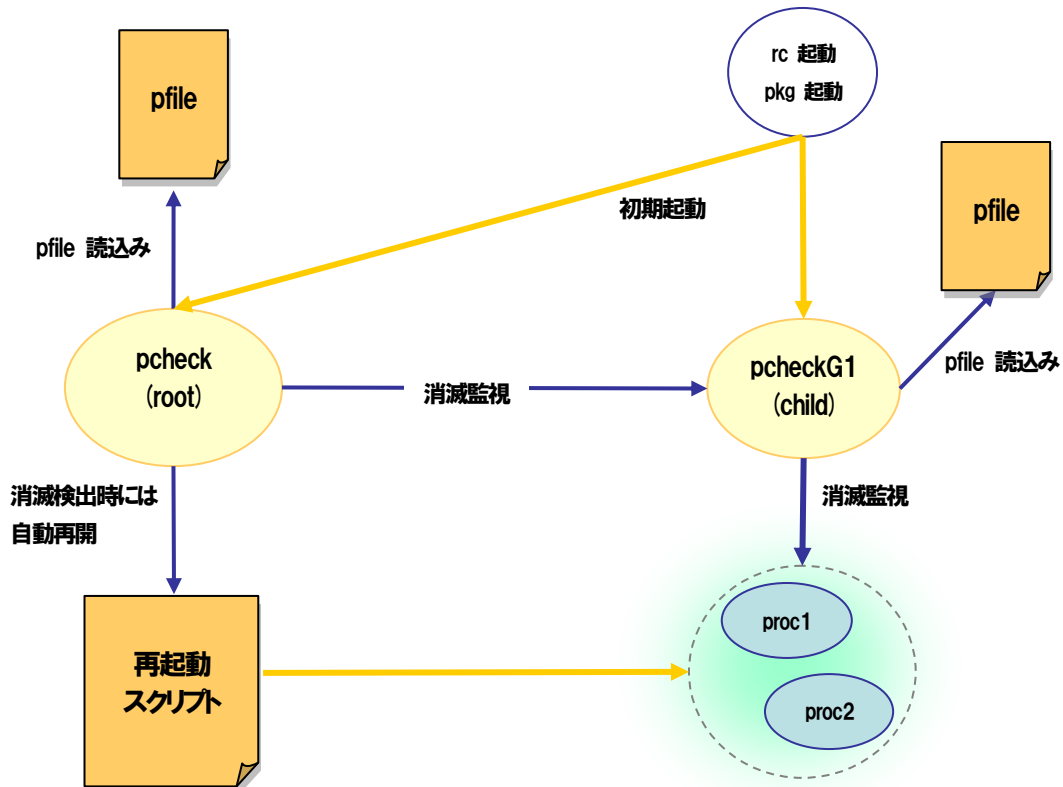
# target process killed
PS_CMD="/bin/ps"
GREP_CMD="/bin/grep"
AWK_CMD="/bin/awk"
PROC_LIST="/bin/proc3 /bin/proc4"
for i in $PROC_LIST
do
    # 以下のPIDの検索方法は一例です。
    pid=`${PS_CMD} -ef | ${GREP_CMD} "${i}" | ${GREP_CMD} -v "${GREP_CMD}" ¥
    | ${AWK_CMD} '{printf("%s ",$2)}END{printf("¥n")}'`
    if [ -n "$pid" ]
    then
        /bin/kill -9 $pid
    fi
done

# target process start
/bin/proc3 > /dev/null 2>&1 &
/bin/proc4 > /dev/null 2>&1 &
/bin/sleep 10

exit 0
```

## (2) pcheck を階層的に使用したグループ監視の導入手順

本機能を導入するには、pcheck を階層的に監視、運用する手続きが必要となります。  
ここでは、ユーザープロセスを監視する pcheck のことを pcheckGn(子の pcheck)と規定し、  
また pcheckGn を監視する pcheck を root\_pcheck(親の pcheck)として説明します。



### ① 親の pcheck(root\_pcheck)

親の pcheck は子の pcheck の消滅監視および再開を行います。

親の pcheck が子の pcheck を監視する場合は、監視ルールに消滅監視しか適用できません。

親の pfile で定義する再起動スクリプトには、子の pcheck が監視するすべてのプロセスを終了させた後で再起動するような処理と、子の pcheck 自体を再起動するような処理を必要に応じて作成してください。

このとき、以下の点に注意願います。

(注) 対象プロセスを起動した後に sleep など一定時間の猶予をとって、子の pcheck を起動してください。

対象プロセスの起動に時間がかかり、子の pcheck が先に起動されると、誤動作する可能性があります。

```
##### root pcheck pfile #####
##### PARAM #####
IPCKEY                0x10000001
MSG_CHECK_INTERVAL   5
MONITOR_INTERVAL     10
SHM_DUMP_FILE        /var/opt/HA/PS/log/pcheck_dump

##### PENT #####
#process_name:shell_path:grace_time:retry_num:retry_over_action
/opt/HA/PS/bin/pcheckG1:/var/opt/HA/PS/conf/restart_G1.sh:86400:3:continue
```

## ② 子の pcheck(pcheckGn)

子の pcheck の実行形式ファイルは、pcheck を別名でリンク(または、コピー)してください。  
 子の pcheck には、pfile の設定、ダミーの再起動スクリプトの作成が必要となりますので、  
 テンプレートファイルから作成し、これらを実行環境にコピーします。

対象プロセスの消滅を検出し、グループ全体で再起動する場合は、以下の設定が必要です。

- shell\_path は省略または、ダミーのスクリプトを登録するか、省略値を指定してください。  
省略値は 0 (ゼロ)または - (ハイフン)です。
- retry\_num には 0(ゼロ)を指定してください。
- retry\_over\_action には親の pcheck に異常を通知するために shutdown を指定してください。

```
##### child pcheckG1 pfile#####
##### PARAM #####
IPCKEY                0x10000002
MSG_CHECK_INTERVAL   5
MONITOR_INTERVAL     10
SHM_DUMP_FILE        /var/opt/HA/PS/log/pcheckG1_dump

##### PENT #####
#process_name:shell_path:grace_time:retry_num:retry_over_action
/usr/bin/test1:-:0:0:shutdown
/usr/bin/test2:-:0:0:shutdown
```

### ③ 再起動スクリプト

グループ監視における親 pcheck のサンプル (restart\_G1.sh)

```
<root pcheck restart shell>
```

```
#!/bin/sh
# target process killed
PS_CMD="/bin/ps"
GREP_CMD="/bin/grep"
AWK_CMD="/bin/awk"
PROC_LIST="/bin/test1 /bin/test2"
for i in $PROC_LIST
do
    # 以下の PID の検索方法は一例です。
    pid=`${PS_CMD} -ef | ${GREP_CMD} "${i}" | ${GREP_CMD} -v "${GREP_CMD}" ¥
    | ${AWK_CMD} '{printf("%s ",$2)}END{printf("¥n")}'`
    if [ -n "$pid" ]
    then
        /bin/kill -9 $pid
    fi
done

# target process start
/bin/test1 > /dev/null 2>&1 &
/bin/test2 > /dev/null 2>&1 &
/bin/sleep 10

# child pcheckG1 start
/opt/HA/PS/bin/pcheckG1 -f <child_pcheck_pfile> &
/bin/sleep 10

exit 0
```

## 4.2. 同一名プロセス監視の導入手順

pfile の対象エントリに指定したプロセスが複数存在する場合、その中で1つのプロセスを選択して対象として組み込みます。選択ルールは以下のとおりです。

- プロセス間に親子関係がある場合は、大元の親プロセスを監視します。
- プロセス間に親子関係がない場合、プロセスの起動時刻の最も古いものを選択します。
- プロセスの起動時刻が同じ場合、最もPIDの小さいものを監視します。

同一名のプロセスが複数同時に存在する環境で、すべてのプロセスを監視するには、以下の方法があります。

- 引数で区別する方式
- プロセス名に含まれる任意の文字列で区別する方式
- uid で区別する方式
- ユーザー名で区別する方式
- 起動するプロセス数で監視する方式
- シェルスクリプトでプロセス数をカウントする方式

### (1) 引数で区別する方式

pfile の対象プロセス名に引数を指定することで、引数によってプロセス名を区別します。

引数はすべての文字列を指定する必要はありません。

プロセス名の特定が可能であればプロセス名に含まれる文字列の一部でもかまいません。

この方式であれば、対象プロセスを確実に識別できるため、再起動スクリプトや各種パラメーターをプロセスごとに登録することが可能となります。

```
##### PENT #####  
## pname:restart shell:grace:retry_count_max:retry_over_action  
/usr/xxxx/sample -A -n nodeA: /usr/xxxx/sampleA.sh:86400:3:continue  
/usr/xxxx/sample -B -n nodeB: /usr/xxxx/sampleB.sh:86400:3:continue  
/usr/xxxx/sample -C -n nodeC: /usr/xxxx/sampleC.sh:86400:3:continue
```

### (2) プロセス名に含まれる任意の文字列で区別する方式

pfile のプロセス監視エントリに include\_strings オプションを指定することで、プロセス名に含まれる文字列をキーにプロセスを区別することができます。

この方式であれば、対象プロセスを確実に識別できるため、再起動スクリプトや各種パラメーターをプロセスごとに登録することが可能となります。

```
##### PENT #####  
## pname:restart shell:grace:retry_count_max:retry_over_action:option  
./sample:./sampleA.sh:86400:3:continue:include_strings=keyword1  
./sample:./sampleB.sh:86400:3:continue:include_strings=keyword2&keyword3  
./sample:./sampleC.sh:86400:3:continue:include_strings=keyword4
```

(3) uid で区別する方式

pfile のプロセス監視エンタリに uid オプションを指定することで、uid によってプロセス名を区別します。この方式であれば、対象プロセスを確実に識別できるため、再起動スクリプトや各種パラメーターをプロセスごとに登録することが可能となります。

```
##### PENT #####
## pname:restart shell:grace:retry_count_max:retry_over_action:option
/usr/xxxx/sample: /usr/xxxx/sampleA.sh:86400:3:continue:uid=510
/usr/xxxx/sample: /usr/xxxx/sampleB.sh:86400:3:continue:uid=520
/usr/xxxx/sample: /usr/xxxx/sampleC.sh:86400:3:continue:uid=530
```

(4) ユーザー名で区別する方式

pfile のプロセス監視エンタリに uname オプションを指定することで、ユーザー名によってプロセス名を区別します。

この方式であれば、対象プロセスを確実に識別できるため、再起動スクリプトや各種パラメーターをプロセスごとに登録することが可能となります。

```
##### PENT #####
## pname:restart shell:grace:retry_count_max:retry_over_action:option
/usr/xxxx/sample:/usr/xxxx/sampleA.sh:86400:3:continue:uname=guest
/usr/xxxx/sample:/usr/xxxx/sampleB.sh:86400:3:continue:uname=oracle
/usr/xxxx/sample:/usr/xxxx/sampleC.sh:86400:3:continue:uname=root
```

ユーザー名を指定してプロセスを起動する(再開する)場合は、su(1) コマンドが利用できます。再起動スクリプトの記述例は以下のとおりです。

<pcheck restart shell for su>

```
#!/bin/sh
# target process killed
 省略

# target process start for guest-user
/bin/su - guest -c "proc-name &" &
```

※pfile の再起動スクリプトには、基本的にスクリプトを指定することが前提であるため、su コマンドを用いた再起動を行う際は上記例のように su コマンドを記述したスクリプトを作成し、再起動スクリプトに登録してください。



(5) 起動するプロセス数で下限監視する方式

pfile のプロセス監視エントリに min\_proc\_count オプションを指定することで、監視対象プロセスを含め、同一名プロセスが指定した数を下回ると プロセスの Down を検出します。

ただし、監視対象プロセス自身が消滅した場合は、プロセス数が min\_proc\_count 以上であっても Down となります。

なお、環境変数 HAPS\_NPROC\_CHECK\_NOPPROC を設定することで、障害条件から監視対象プロセス自身が消滅した場合を外すことができます。

設定手順の詳細は、下記の手順を参照してください。

「4.2.1. プロセス個数監視時に監視対象プロセス自身の消滅を障害条件から除外する手順」

httpd など、同一名称のプロセスが複数存在するプロセスの監視に有効です。

以下の例では、"/usr/sbin/httpd" が 4 個未満になるとプロセスの消滅を検出し、再起動スクリプト "sampleA.sh" を実行します。

```
##### PENT #####  
## pname:restart shell:grace:retry_count_max:retry_over_action:option  
/usr/sbin/httpd:./sampleA.sh: 86400:3:continue:min_proc_count=4
```

(6) 起動するプロセス数で上限監視する方式

pfile のプロセス監視エントリに max\_proc\_count オプションを指定することで、監視対象プロセスを含め、同一名プロセスが指定した数を上回ると プロセスの Down を検出します。

ただし、監視対象プロセス自身が消滅した場合は、プロセス数が max\_proc\_count 以下であっても Down となります。

なお、環境変数 HAPS\_NPROC\_CHECK\_NOPPROC を設定することで、障害条件から監視対象プロセス自身が消滅した場合を外すことができます。

設定手順の詳細は、下記の手順を参照してください。

「4.2.1. プロセス個数監視時に監視対象プロセス自身の消滅を障害条件から除外する手順」

httpd など、同一名称のプロセスが複数存在するプロセスの監視に有効です。

以下の例では、"/usr/sbin/httpd" が 4 個を越えるとプロセスの消滅を検出し、再起動スクリプト "sampleA.sh" を実行します。

```
##### PENT #####  
## pname:restart shell:grace:retry_count_max:retry_over_action:option  
/usr/sbin/httpd:./sampleA.sh: 86400:3:continue:max_proc_count=4
```

(7) 起動するプロセス数の範囲で監視する方式

pfile のプロセス監視エントリに min\_proc\_count オプションと max\_proc\_count オプションを同時に指定することで、監視対象プロセスを含め、同一名プロセスが指定した範囲外になると プロセスの Down を検出します。

ただし、監視対象プロセス自身が消滅した場合は、プロセス数が指定された範囲内であっても プロセスの Down を検出します。

なお、環境変数 HAPS\_NPROC\_CHECK\_NOPPROC を設定することで、障害条件から監視対象プロセス自身が消滅した場合を外すことができます。

設定手順の詳細は、下記の手順を参照してください。

「4.2.1. プロセス個数監視時に監視対象プロセス自身の消滅を障害条件から除外する手順」

httpd など、同一名称のプロセスが複数存在するプロセスの監視に有効です。  
max\_proc\_count には、min\_proc\_count 以上の値を設定する必要があります。  
以下の例では、"/usr/sbin/httpd" が 4~8 以外の個数になるとプロセスの消滅を検出し、再起動スクリプト "sampleA.sh" を実行します。

```
##### PENT #####  
## pname:restart shell:grace:retry_count_max:retry_over_action:option  
/usr/sbin/httpd:./sampleA.sh:86400:3:continue:min_proc_count=4,max_proc_count=8
```

(8) 起動するプロセス数で監視する方式

pfile のプロセス監視エントリに対象プロセスの個数分のエントリを列記することで、個数による監視を実現します。

ただし、対象プロセスの依存関係を識別できないため、対象プロセスが同時に起動され同時に終了し、かつ、親子関係をもたないことが前提となります。

さらに、プロセス監視エントリに記述する対象プロセス名、再起動スクリプト、各種パラメーターは同一である必要があります。

対象プロセスが同時に起動、終了するようなケースは少ないため、本方式より(5)~(7)の方式が有効です。また導入の際は慎重にご検討ください。

```
##### PENT #####  
## pname:restart shell:grace:retry_count_max:retry_over_action  
/usr/xxxx/sample: /usr/xxxx/sampleA.sh:86400:3:continue  
/usr/xxxx/sample: /usr/xxxx/sampleA.sh:86400:3:continue  
/usr/xxxx/sample: /usr/xxxx/sampleA.sh:86400:3:continue
```

(9) シェルスクリプトでプロセス数をカウントする方式

プロセス数をカウントするシェルスクリプトを用意し、これを pcheck から監視することで柔軟な同一名プロセス監視を実現します。

<pfile>

```
##### PENT #####  
## pname:restart shell:grace:retry_count_max:retry_over_action  
# restart script for Multi proc sample  
/bin/sh /var/opt/HA/PS/conf/bin/MULTI_PROC/multi_check.sh:  
/var/opt/HA/PS/conf/bin/MULTI_PROC/multi_rst.sh:86400:3:continue
```

プロセス数をカウントするシェルスクリプトのサンプルは以下のとおりです。

以下の例は smbд プロセスの起動済みのプロセス数をカウントし、規定のプロセス数(ここでは 5)と一致なくなるとシェルスクリプトが異常終了します。

<multi\_check.sh>

```
#!/bin/sh

# restart shell for Multi proc sample

# process name
DAEMON_NAME="smbd"
# process alive count
DAEMON_COUNT=5
# process check interval sec
SLEEP_TIME=60

STATUS=0

while [ ${STATUS} -eq 0 ]
do
    daemonfname=`/bin/basename ${DAEMON_NAME}`
    num_daemon=`/bin/ps -e | /bin/awk -v fname=${daemonfname} '$4 == fname
{print $1}' | /usr/bin/wc -l`
    if [ ${num_daemon:=0} -ne ${DAEMON_COUNT} ]
    then
        STATUS=1
    else
        /bin/sleep ${SLEEP_TIME}
    fi
done

return 0
```

シェルスクリプトが異常終了すると pcheck が消滅を検出し、再起動スクリプトを呼び出します。

<multi\_rst.sh>

```
#!/bin/sh
# restart shell for Multi proc sample

.....

# waiting for proc wakeup
/bin/sleep 30

# multi process checker start
/var/opt/HA/PS/conf/bin/MULTI_PROC/multi_check.sh &
```

#### 4.2.1. プロセス個数監視時に監視対象プロセス自身の消滅を障害条件から除外する手順

pfile のプロセス監視エントリに、以下のオプションを指定することで、プロセス個数監視(下限監視、上限監視、範囲監視)を実施することが可能です。

##### 【 下限監視 】

・min\_proc\_count オプション

##### 【 上限監視 】

・max\_proc\_count オプション

##### 【 範囲監視 】

・min\_proc\_count オプションと max\_proc\_count オプション

上記の指定を実施することで、プロセス数の下限監視、上限監視、範囲監視をおこないますが、その障害条件は次のようになります。

##### 【 下限監視 】

以下のいずれかの場合

- ・監視対象プロセス自身が消滅した場合
- ・監視対象プロセスを含め、同一名プロセスが指定した数を下回った場合

##### 【 上限監視 】

以下のいずれかの場合

- ・監視対象プロセス自身が消滅した場合
- ・監視対象プロセスを含め、同一名プロセスが指定した数を上回った場合

##### 【 範囲監視 】

以下のいずれかの場合

- ・監視対象プロセス自身が消滅した場合
- ・監視対象プロセスを含め、同一名プロセスが指定した範囲外となった場合

そのため、下限、上限、範囲監視において、監視対象プロセス数が下限、上限、範囲外となった場合以外でも監視対象プロセス自身が消滅した場合に障害を検出します。

障害条件から監視対象プロセス自身の消滅を除外したい場合には、

環境変数 HAPS\_NPROC\_CHECK\_NOPPROC を設定します。

同環境変数に 1 を設定することで、障害条件から監視対象プロセス自身の消滅を除外します。

プロセス個数監視時に監視対象プロセス自身の消滅を障害条件から除外する手順は以下のとおりです。

(1) プロセス個数監視時に監視対象プロセス自身の消滅を障害条件から除外する手順

- ① 環境変数 HAPS\_NPROC\_CHECK\_NOPPROC に 1 を設定します。

・sh の場合

```
# HAPS_NPROC_CHECK_NOPPROC=1  
# export HAPS_NPROC_CHECK_NOPPROC
```

・csh の場合

```
% setenv HAPS_NPROC_CHECK_NOPPROC 1
```

※ 上記のように環境変数を設定した状態で起動した pcheck については、  
下限、上限、範囲監視の障害条件が変更されます。

- ② pcheck を起動し、プロセス監視を開始します。

```
#/opt/HA/PS/bin/pcheck -f <pfile> &
```

- ③ 確認手順

動作中の pcheck で設定されている環境変数は、padmin コマンドで  
確認することができます。

```
#/opt/HA/PS/bin/padmin -f <pfile 名> -c show env  
HAPS_NPROC_CHECK_NOPPROC = 1 ★
```

環境変数 HAPS\_NPROC\_CHECK\_NOPPROC を設定していない場合は、  
デフォルトの 0 が表示されます。

(2) 運用上の注意・制限事項

- すでに起動中の pcheck の障害条件を変更することはできません。  
一度 pcheck を padmin -c shutdown で終了させてから、環境変数を設定し  
pcheck を再起動してください。

### 4.3.java のプロセス監視の導入手順

java のプロセス監視の導入手順について説明します。

java プロセスが複数動作するようなシステム上では、java プロセスのプロセス名は同じになります。このような環境では、次の 2 つの方法で java プロセスの監視を実現します。

- (a) pfile の process\_name に java プロセス名を引数をすべて含めて指定する
- (b) pfile の process\_name に java プロセス名を指定し、include\_strings オプションにその java プロセス名に含まれ、他の java プロセス名に含まれない文字列を指定する

ここでは例として、WebOTX の java プロセスと、Tomcat の java プロセス監視の導入手順について説明します。

また、java のようにプロセス名が長い場合の pfile の設定ミスを防止するために、putil コマンドを使用することにより pfile を自動生成することができます。

#### (1) プロセス名の調査

ps コマンドを実行し、プロセス名を出力します。

```
# ps -ef | grep java
root      1351      1  0 13:41 pst/1    09:07:06 ? /usr/java/jdk1.5.0_04/bin/java -Xbootclass
path/p:/opt/WebOTX/WebCont/../../share.nec/jars/xercesImpl.jar:/opt/WebOTX/WebCont/../../share.nec/jars/xmlParserAPIs.jar:/opt/WebOTX/WebCont/../../share.nec/jars/crimson.jar:/opt/WebOTX/WebCont/../../share.nec/jars/xalan.jar -Dorg.omg.CORBA.ORBClass=jp.co.nec.orb.OSPORB -Dorg.omg.CORBA.ORBSingletonClass=jp.co.nec.orb.OSPORBSingleton -Djavax.rmi.CORBA.UtilClass=jp.co.nec.orb.rmi.UtilDelegatImpl -Djavax.rmi.CORBA.PortableRemoteObjectClass=jp.co.nec.orb.rmi.PortableRemoteObjectDelegatImpl -Djavax.rmi.CORBA.StubClass=jp.co.nec.orb.rmi.StubDelegatImpl -Dwebotx.uddi.home=/opt/WebOTX/WebCont/./UDDIReg/conf -classpath ./:/opt/java1.3/lib/tools.jar:/opt/WebOTX/WebCont/bin/bootstrap.jar:/opt/WebOTX/WebCont/./TS/javalib/wots51.jar:/opt/WebOTX/WebCont/../../share.nec/jars/wojdbc42.jar:/opt/WebOTX/WebCont/../../ObjectSpinner/lib/ospiorb50.jar:/opt/WebOTX/WebCont/../../ObjectSpinner/lib/ospiname50.jar:/opt/WebOTX/WebCont/../../ObjectSpinner/lib/osp
root      12054     1  0 13:40 pts/1    00:00:17 /usr/java/jdk1.5.0_04/bin/java -Djava.endorsed
.dirs=/opt/jakarta-tomcat-4.1.31/common/endorsed -classpath /usr/java/jdk1.5.0_04/lib/tools.jar:/opt/jakarta-tomcat-4.1.31/bin/bootstrap.jar -Dcatalina.base=/opt/jakarta-tomcat-4.1.31 -Dcatalina.home=/opt/jakarta-tomcat-4.1.31 -Djava.io.tmpdir=/opt/jakarta-tomcat-4.1.31/temp org.apache.catalina.startup.Bootstrap start
```

## (2) pfile の例

- ① process\_name に java プロセス名を、引数をすべて含めて指定する場合

pfile の PENT 部には、以下のような形式で指定します。

**<ps -ef | grep で出力されたプロセス名>:restart shell:grace:retry\_count\_max:retry\_over\_action**

<pfile>

```
##### PENT #####
```

```
## pname:restart shell:grace:retry_count_max:retry_over_action
```

```
# WebOTX java process
```

```
/usr/java/jdk1.5.0_04/bin/java -Xbootclasspath/p¥/opt/WebOTX/WebCont/../../share.nec/jars/xerc  
esImpl.jar¥/opt/WebOTX/WebCont/../../share.nec/jars/xmlParserAPIs.jar¥/opt/WebOTX/WebCon  
t/../../share.nec/jars/crimson.jar¥/opt/WebOTX/WebCont/../../share.nec/jars/xalan.jar -Dorg.omg.C  
ORBA.ORBClass=jp.co.nec.orb.OSPORB -Dorg.omg.CORBA.ORBSingletonClass=jp.co.nec.or  
b.OSPORBSingleton -Djavax.rmi.CORBA.UtilClass=jp.co.nec.orb.rmi.UtilDelegatImpl -Djavax.r  
mi.CORBA.PortableRemoteObjectClass=jp.co.nec.orb.rmi.PortableRemoteObjectDelegatImpl -  
Djavax.rmi.CORBA.StubClass=jp.co.nec.orb.rmi.StubDelegatImpl -Dwebotx.uddi.home=/opt/W  
ebOTX/WebCont/../../UDDIReg/conf -classpath .¥¥/opt/java1.3/lib/tools.jar¥/opt/WebOTX/WebC  
ont/bin/bootstrap.jar¥/opt/WebOTX/WebCont/../../TS/javalib/wots51.jar¥/opt/WebOTX/WebCont/..  
/share.nec/jars/wojdbc42.jar¥/opt/WebOTX/WebCont/../../ObjectSpinner/lib/ospiorb50.jar¥/opt/W  
ebOTX/WebCont/../../ObjectSpinner/lib/ospiname50.jar¥/opt/WebOTX/WebCont/../../ObjectSpinn  
er/lib/osp:-:86400:3:shutdown
```

```
# Tomcat java process
```

```
/usr/java/jdk1.5.0_04/bin/java -Djava.endorsed.dirs=/opt/jakarta-tomcat-4.1.31/common/endorsed  
-classpath /usr/java/jdk1.5.0_04/lib/tools.jar¥/opt/jakarta-tomcat-4.1.31/bin/bootstrap.jar -Dcatali  
na.base=/opt/jakarta-tomcat-4.1.31 -Dcatalina.home=/opt/jakarta-tomcat-4.1.31 -Djava.io.tmpdir=  
/opt/jakarta-tomcat-4.1.31/temp org.apache.catalina.startup.Bootstrap start:-:86400:3:shutdown
```

(1) のプロセス名調査で出力された最大のプロセス名と引数を、すべて pfile の process\_name に指定します。

指定するプロセス名に:(コロン)が含まれる場合は、直前に¥(バックスラッシュ)を挿入してください。

putil コマンドを使用して pfile を自動生成した場合は、自動で:(コロン)の直前に¥(バックスラッシュ)が挿入されます。

- ② process\_name に java プロセス名を指定し、include\_strings オプションにその java プロセス名に含まれ、他の java プロセス名に含まれない文字列を指定する場合

pfile の PENT 部には、以下のような形式で指定します。

**<ps -ef | grep で出力されたプロセス名>:restart shell:grace:retry\_count\_max:retry\_over\_action:include\_strings=<ps -ef |grep で出力されたプロセス名に含まれる文字列>**

```
<pfile>
##### PENT #####
## pname:restart shell:grace:retry_count_max:retry_over_action
# restart script for Multi proc sample
# WebOTX java process
/usr/java/jdk1.5.0_04/bin/java::-86400:3:shutdown:include_strings=/opt/WebOTX/WebCont/..J
DDIReg/conf
# Tomcat java process
/usr/java/jdk1.5.0_04/bin/java::-86400:3:shutdown:include_strings=/opt/jakarta-tomcat-4.1.31
```

- (1)のプロセス名調査で出力された java プロセス名を pfile の process\_name に指定します。さらに pfile の include\_strings オプションに引数の文字列に含まれ、他の java プロセス名に含まれない文字列を指定します。

include\_strings は以下のように "&" で区切って複数指定することで、すべての文字列が含まれるプロセスを指定することができます。

```
/opt/java1.3/jre/bin/./bin/PA_RISC2.0/native_threads/java::-86400:3:shutdown:include_strings=-
Djava.endorsed.dirs&/opt/jakarta-tomcat-4.1.31/common/endorsed&-classpath&/usr/java/j
dk1.5.0_04/lib/tools.jar&/opt/jakarta-tomcat-4.1.31/bin/bootstrap.jar&-Dcatalina.base&/opt/j
akarta-tomcat-4.1.31&-Dcatalina.home&/opt/jakarta-tomcat-4.1.31&-Djava.io.tmpdir&/opt/ja
karta-tomcat-4.1.31/temp&org.apache.catalina.startup.Bootstrap&start
```

include\_strings オプションの詳細については「3.3 pfile ファイルについて」を参照してください。



(3) pcheck 開始後の確認

/opt/HA/PS/bin/padmin -f <pfile 名> -c show pent で、pid が指定したプロセスの PID と一致していることを確認してください。

① process\_name に java プロセス名を引数をすべて含めて指定する場合

```
# /opt/HA/PS/bin/padmin -f /tmp/pfile_java -c show pent
pname          = /usr/java/jdk1.5.0_04/bin/java -Xbootclasspath/p:/opt/WebOTX/WebCont/././share.nec/jars/xercesImpl.jar:/opt/WebOTX/WebCont/././share.nec/jars/xmlParserAPIs.jar:/opt/WebOTX/WebCont/././share.nec/jars/crimson.jar:/opt/WebOTX/WebCont/././share.nec/jars/xalan.jar -Dorg.omg.CORBA.ORBClass=jp.co.nec.orb.OSPORB -Dorg.omg.CORBA.ORBSingletonClass=jp.co.nec.orb.OSPORBSingleton -Djavax.rmi.CORBA.UtilClass=jp.co.nec.orb.rmi.UtilDelegatImpl -Djavax.rmi.CORBA.PortableRemoteObjectClass=jp.co.nec.orb.rmi.PortableRemoteObjectDelegatImpl -Djavax.rmi.CORBA.StubClass=jp.co.nec.orb.rmi.StubDelegatImpl -Dwebotx.ddi.home=/opt/WebOTX/WebCont/./UDDIReg/conf -classpath ../opt/java1.3/lib/tools.jar:/opt/WebOTX/WebCont/bin/bootstrap.jar:/opt/WebOTX/WebCont/./TS/javalib/wots51.jar:/opt/WebOTX/WebCont/././share.nec/jars/wojdbc42.jar:/opt/WebOTX/WebCont/././ObjectSpinner/lib/ospiorb50.jar:/opt/WebOTX/WebCont/././ObjectSpinner/lib/ospiname50.jar:/opt/WebOTX/WebCont/././ObjectSpinner/lib/osp
pid          = 1351
retry_count    = 0
restart_count  = 0
proc_sts      = AVAIL
retry_over_act = continue
rerun_time    = -----

pname          = /usr/java/jdk1.5.0_04/bin/java -Djava.endorsed.dirs=/opt/jakarta-tomcat-4.1.31/common/endorsed -classpath /usr/java/jdk1.5.0_04/lib/tools.jar:/opt/jakarta-tomcat-4.1.31/bin/bootstrap.jar -Dcatalina.base=/opt/jakarta-tomcat-4.1.31 -Dcatalina.home=/opt/jakarta-tomcat-4.1.31 -Djava.io.tmpdir=/opt/jakarta-tomcat-4.1.31/temp org.apache.catalina.startup.Bootstrap start
pid          = 12054
retry_count    = 0
restart_count  = 0
proc_sts      = AVAIL
retry_over_act = continue
rerun_time    = -----
```

- ② process\_name に java プロセス名を指定し、include\_strings オプションに、その java プロセス名に含まれ、他の java プロセス名に含まれない文字列を指定する場合

```
# padmin -f /tmp/pfile_java -c show pent
pname          = /usr/java/jdk1.5.0_04/bin/java
pid           = 1351
retry_count    = 0
restart_count  = 0
proc_sts       = AVAIL
retry_over_act = continue
rerun_time     = -----
include_strings = /opt/WebOTX/WebCont/././share.nec/jars/crimson.jar
```

```
pname          = /usr/java/jdk1.5.0_04/bin/java
pid           = 12054
retry_count    = 0
restart_count  = 0
proc_sts       = AVAIL
retry_over_act = continue
rerun_time     = -----
include_strings = /opt/jakarta-tomcat-4.1.31
```

## 4.4. 監視間隔を短くした運用

pfile の共通部情報(PARAM)に設定する MSG\_CHECK\_INTERVAL (内部イベントをスケジュールするタイマー値)、MONITOR\_INTERVAL(プロセス監視間隔)を短く設定することによって、プロセス消滅の検出とリカバリを早期に実現できるようにすることが可能となります。

次の pfile のサンプルでは、MSG\_CHECK\_INTERVAL=1(秒)、MONITOR\_INTERVAL=1(秒) 間隔でプロセス監視を行います。

```
# ProcessSaver configuration
##### PARAM #####
IPCKEY                0x1f000001
MSG_CHECK_INTERVAL    1
MONITOR_INTERVAL      1
SHM_DUMP_FILE         /var/opt/HA/PS/log/pcheck_dump

##### PENT #####
syslogd:/etc/init.d/syslog restart:86400:3:continue
```

ただし、上記のような運用事例もありますが、システムによっては負荷が高くなり業務運用に影響する場合がありますので、事前に性能的に問題がないか動作確認を行ってください。

## 4.5. 一般ユーザーでのプロセス監視の実行

pcheck、padmin は通常 root ユーザーで実行しますが、各コマンドに一般ユーザーの実行権を付与することで、一般ユーザーでも実行可能にすることができます。

```
一般ユーザーへの実行権の付与(root ユーザーで実行)
# /bin/chmod 555 /opt/HAPS/bin/pcheck
# /bin/chmod 555 /opt/HAPS/bin/padmin
```

ただし、再起動スクリプトなど、pcheck から実行されるコマンド、スクリプトなどで、root 権限を必要とするコマンドを使用することはできません。

### 運用上の注意・制限事項

- ・一般ユーザーでコマンドを実行する場合には、padmin、pcheck の両方を一般ユーザーで実行してください。  
pcheck コマンドを root ユーザー、padmin コマンドを一般ユーザーで実行すると正しく動作しない場合があります。

## 4.6. ストール監視の導入手順

### 4.6.1. ファイルの更新時刻によるストール監視

ファイルの更新時刻によるストール監視機能の導入手順および pfile 登録手順について説明します。

#### (1) ストールを検出するためのファイルの登録について

pcheck でプロセスストールを検出するには、対象プロセスが特定のファイルを登録し、定期的に更新する仕掛けを実装する必要があります。

監視対象ファイルは、以下の 2 種類の方法で登録できます。

- 利用者があらかじめ専用のファイルを用意
- ユーザー提供ライブラリを対象プロセスに組み込む

#### (2) pfile の登録手順

pfile のプロセスエントリに続く行において、"\_bi\_stall" という文字で囲まれたブロックを、ストール監視の設定として解釈します。

##### ・エントリの定義

ストール監視のエントリには INIT エントリ と EXEC エントリ と ACTION エントリ が存在します。

```
_bi_stall { INIT エントリ, EXEC エントリ, ACTION エントリ }
```

##### ・INIT エントリの定義

プロセスのストール監視開始時の前処理を定義します。

本エントリには、対象ファイル名を touch する設定をします。

**監視対象ファイル名**には、対象プロセスが定期的に更新するファイル名を絶対パスで指定してください。

たとえば、「file に /bin/touch /tmp/stall.file 」と記述されていれば、プロセスのストール監視起動時に /tmp/stall.file ファイルの作成時間を更新します。

また起動時にファイルが存在しない場合は、空ファイルを新規作成します。

```
entry type:file:call:timeout:interval: error:argc:...
```

その他のパラメータはデフォルト値を使用してください。(変更不可)

パラメーター	設定値	説明
entry	PS_INIT	エントリの種類
type	AP	部品の属性
file	<b>/bin/touch filename</b>	<b>touch 監視対象ファイル名</b>
call	DIRECT	呼び出し方式
timeout	-(省略)	タイムアウト時間(秒)
interval	-(省略)	監視間隔(秒)
error	-(省略)	アクション失敗時の動作規定
argc	0	エントリ呼び出し時の引数 (argv) の数

・EXEC エントリの定義

プロセスのストール監視の手順を定義します。

本エントリには、異常検出時の動作規定、監視対象ファイル名、リトライオーバー時間(秒)を設定します。

**異常検出時の動作規定** には以下の指定が可能です。

- 0: ストールを検出した際、以降継続してストール監視を実行します。  
ストール状態が解消されるまで、ストール検知の syslog メッセージを継続して出力します。
- 1: ストールを検出した際、以降ストール監視を実行しません。  
(注) ストール監視は停止しますが、プロセスの消滅監視は継続します。  
ストール検出を一度のみとする必要がある場合には、本パラメーターに 1 を指定してください。
- 2: ストールを検出した際、以降継続してストール監視を実行します。  
ストール状態が解消されるまで、ストール検知の syslog メッセージを抑制します。  
(注) OpenShift 環境のコンテナのヘルスチェック(readiness チェック)にストール監視を使用する場合は、本パラメーターに 2 を指定してください。

**監視対象ファイル名**は、対象プロセスが定期的に更新するファイル名を絶対パスで指定してください。

**リトライオーバー時間**にはプロセスをストール状態と解釈する時間(秒)を指定します。

60 秒以上を指定してください。デフォルト値は 300 秒指定となります。

300 秒指定の場合は、/tmp/stall.file ファイルの更新時刻を 60 秒ごとにチェックし、

300 秒経過しても更新されていないければ、ストール状態と解釈します。

```
entry type:file:call:timeout:interval: error:argc:argv[0]:...
```

その他のパラメーターはデフォルト値を使用してください。(変更不可)

パラメーター	設定値	説明
entry	PS_EXEC	エントリの種類
type	DLL	部品の属性
file	/opt/HA/PS/lib/lib_bi_stall.so	ライブラリのファイル名
call	DIRECT	呼び出し方式
timeout	-(省略)	タイムアウト時間(秒)
interval	60	監視間隔(秒)
error	0 ( or 1 or 2)	<b>異常検出時の動作規定</b>
argc	2	エントリ呼び出し時の引数 (argv) の数
argv[0]	filename	<b>監視対象ファイル名</b>
argv[1]	300	<b>リトライオーバー時間</b>

・ACTION エントリの定義

プロセスストールを検出した場合の後処理を定義します。  
本エントリでは、アクション失敗時の動作規定の設定が必要です。  
その他のパラメーターはデフォルト値を使用してください。(変更不可)

ストール状態を検出すると ACTION エントリによって対象プロセスを強制的に終了させますが  
プロセスの強制終了に失敗した場合は、ストール監視を停止させた後にアクション失敗時の  
動作規定の設定に従った後処理を実行します。

**アクション失敗の動作規定**には以下の指定が可能です。

0 : 後処理を何もしません。

1 : `retry_over_action` で定義される動作(`continue / shutdown / exit` のいずれか)  
を実行します。

(注) CLUSTERPRO とのクラスターウェア連携など、`retry_over_action` で定義される  
動作を行う必要がある場合には、本パラメーターに 1 を指定してください。

パラメーター	設定値	説明
<code>entry</code>	PS_ACTION	エントリの種類
<code>type</code>	DLL	部品の属性
<code>file</code>	/opt/HA/PS/lib/lib_bi_stall.so	ライブラリのファイル名
<code>call</code>	DIRECT	呼び出し方式
<code>timeout</code>	-(省略)	タイムアウト時間(秒)
<code>interval</code>	-(省略)	監視間隔(秒)
<code>error</code>	0	<b>アクション失敗の動作規定</b>
<code>argc</code>	0 (or 1 or 2)	<b>エントリ呼び出し時の引数 (argv) の数</b>
<code>argv[0]</code>	-(省略)	<b>ストール検出時に送信する signal (*)</b>
<code>argv[1]</code>	-(省略)	<b>SIGKILL リトライ回数 (*)</b>

(\*) シグナル設定の詳細については、(4)ストール検出時の signal 送信についてを  
参照してください。

・QUIT エントリの定義

ファイルの更新時刻によるストール監視の場合、本定義は不要です。

- (3) ファイルの更新が 5 分間ないときストールと判断し対象プロセス再起動後、監視を継続する場合の pfile の例  
以下のように INIT エントリの指定をします。

**監視対象ファイル名(新規作成込み)** : /bin/touch **監視対象ファイルの絶対パス**

また以下のように EXEC エントリの指定をします。

**異常検出時の動作規定** : 0  
**監視対象ファイル名** : **監視対象ファイルの絶対パス**  
**リトライオーバー時間** : 300

```
##### entry type:file:call:timeout:interval: error:argv[0]:...#####  
##### stall checker #####  
_bi_stall {  
  PS_INIT      AP:/bin/touch /tmp/stall.file:DIRECT:-::-:0  
  PS_EXEC      DLL:/opt/HA/PS/lib/lib_bi_stall.so:DIRECT:-:60:0:2:/tmp/stall.file:300  
  PS_ACTION    DLL:/opt/HA/PS/lib/lib_bi_stall.so:DIRECT:-::-:0:0  
}
```



(4) ストール検出時の signal 送信について

プロセスのストールを検出すると、ACTION エントリのデフォルトでは SIGTERM(および SIGKILL)を送信しプロセスを強制終了させますが、**ストール検出時に送信する signal** パラメーターを設定することで SIGTERM の代わりに送信する signal を指定できます。

以下のように ACTION エントリの指定をします。

**エントリ呼び出し時の引数 (argv) の数** : 1  
**ストール検出時に送信する signal** : **SIGTERM ( or SIGKILL or SIGABRT )**

たとえば、SIGABRT を送信することでプロセスの core ファイルを強制出力することができます。

```
##### entry type:file:call:timeout:interval: error:argc:argv[0]:...#####  
##### stall checker #####  
_bi_stall {  
  PS_INIT      AP:/bin/touch /tmp/stall.file:DIRECT:-::-:0  
  PS_EXEC      DLL:/opt/HA/PS/lib/lib_bi_stall.so:DIRECT:-:60:0:2:/tmp/stall.file:300  
  PS_ACTION    DLL:/opt/HA/PS/lib/lib_bi_stall.so:DIRECT:-::-:0:1:SIGABRT  
}
```

また、特定の signal を送信してもプロセスを強制終了できなかった場合、デフォルトではプロセスが終了するまで SIGKILL を最大 10 回送信しますが、**SIGKILL リトライ回数** パラメーターを設定することで SIGKILL を送信する回数を指定できます。

たとえば、SIGTERM を 1 回だけ送信して SIGKILL によるプロセスの強制終了を行わないようにする場合は、以下のように ACTION エントリの指定をします。

**エントリ呼び出し時の引数 (argv) の数** : 2  
**ストール検出時に送信する signal** : **SIGTERM**  
**SIGKILL リトライ回数** : 0

```
##### entry type:file:call:timeout:interval: error:argc:argv[0]:...#####  
##### stall checker #####  
_bi_stall {  
  PS_INIT      AP:/bin/touch /tmp/stall.file:DIRECT:-::-:0  
  PS_EXEC      DLL:/opt/HA/PS/lib/lib_bi_stall.so:DIRECT:-:60:0:2:/tmp/stall.file:300  
  PS_ACTION    DLL:/opt/HA/PS/lib/lib_bi_stall.so:DIRECT:-::-:0:2:SIGTERM.0  
}
```

- (5) OpenShift 環境のコンテナのヘルスチェックに本製品のコマンドを使用する場合の pfile の例  
OpenShift 環境のコンテナのヘルスチェック(readiness チェック) にストール監視を設定する場合は、  
以下のように EXEC エントリを指定します。

**異常検出時の動作規定 : 2**

この場合、ACTION エントリの定義は不要です。  
定義の有無にかかわらず ACTION を実行しません。

```
##### entry type:file:call:timeout:interval: error:argc:argv[0]:...#####  
##### stall checker #####  
_bi_stall {  
    PS_INIT      AP:/bin/touch /tmp/stall.file:DIRECT:-::-:0  
    PS_EXEC      DLL:/opt/HA/PS/lib/lib_bi_stall.so:DIRECT:-:60:2:2:/tmp/stall.file:300  
}
```

OpenShift 環境で ProcessSaver を使用する場合は詳細については、別紙『CLUSTERPRO MC ProcessSaver for Linux OpenShift 連携ガイド』を参照してください。

## 4.6.2. ファイルの出力メッセージによるストール監視

特定のファイルに出力されるメッセージ内容によるストール監視機能(メッセージ監視)の導入手順および pfile 登録手順について説明します。

### (1) pfile の登録手順

pfile のプロセスエントリに続く行において、"\_bi\_stall\_message" という文字で囲まれたブロックを、メッセージ監視の設定として解釈します。

#### ・エントリの定義

ストール監視のエントリには INIT エントリ と EXEC エントリ と ACTION エントリ \*1 と QUIT エントリ が存在します。

```
_bi_stall_message { INIT エントリ, EXEC エントリ, ACTION エントリ, QUIT エントリ }
```

\*1 ACTION エントリは省略できます。

#### ・INIT エントリの定義

メッセージ監視開始時の前処理を定義します。

本エントリには、メッセージファイル名、エラーメッセージ(error) か必須メッセージ(required)の指定と監視対象メッセージを正規表現で指定します。

**監視対象ファイル名**には、対象プロセスがメッセージを更新するファイル名を絶対パスで指定してください。

**監視種別**には以下の指定が可能です。

- error : 対象メッセージが出力された際にストールと判断する場合  
※監視間隔ごとに継続してメッセージが出力し、リトライオーバー時間経過後も継続出力していれば、ストールと判断します。
- required : 対象メッセージが出力されなかった際にストールと判断する場合

**監視メッセージ**には、監視対象とするメッセージを正規表現で指定してください。

その他のパラメーターはデフォルト値を使用してください。(変更不可)

たとえば、「監視対象ファイル名に /tmp/log.file」と記述されていれば/tmp/log.file を読み込んで指定されたメッセージの検出を行います。

パラメーター	設定値	説明
entry	PS_INIT	エントリの種類
type	DLL	部品の属性
lib	<b>/opt/HA/PS/lib/lib_bi_stall_message.so</b>	ライブラリのファイル名
call	DIRECT	呼び出し方式
timeout	-(省略)	タイムアウト時間(秒)
interval	-(省略)	監視間隔(秒)
error	-(省略)	アクション失敗時の動作規定
argc	3	エントリ呼び出し時の引数 (argv) の数
<b>argv[0]</b>	<b>/tmp/logdat</b>	<b>監視対象ファイル名:</b>
<b>argv[1]</b>	<b>error または required</b>	<b>監視種別</b>
<b>argv[2]</b>	<b>message</b>	<b>監視メッセージ</b>

・EXEC エントリの定義

メッセージ監視の手順を定義します。

**異常検出時の動作規定**には以下の指定が可能です。

- 0: ストールを検出した際、以降継続してストール監視を実行します。  
ストール状態が解消されるまで、ストール検知の syslog メッセージを継続して出力します。
- 1: ストールを検出した際、以降ストール監視を実行しません。  
(注) ストール監視は停止しますが、プロセスの消滅監視は継続します。  
ストール検出を一度のみとする必要がある場合には、本パラメーターに 1 を指定してください。
- 2: ストールを検出した際、以降継続してストール監視を実行します。  
ストール状態が解消されるまで、ストール検知の syslog メッセージを抑制します。  
(注) OpenShift 環境のコンテナのヘルスチェック(readiness チェック)にストール監視を使用する場合は、本パラメーターに 2 を指定してください。

**リトライオーバー時間**にはプロセスをストール状態と解釈する時間(秒)を指定します。

60 秒以上を指定してください。デフォルト値は 600 秒指定となります。

600 秒指定の場合は、/tmp/log.file ファイルを 60 秒ごとにチェックし、指定メッセージの検出が 600 秒経過しても継続的に発生していれば、ストール状態と解釈します。

```
entry type:file:call:timeout:interval: error:argc:argv[0]:...
```

その他のパラメーターはデフォルト値を使用してください。(変更不可)

パラメーター	設定値	説明
entry	PS_EXEC	エントリの種類
type	DLL	部品の属性
file	/opt/HA/PS/lib/lib_bi_stall_message.so	ライブラリのファイル名
call	DIRECT	呼び出し方式
timeout	-(省略)	タイムアウト時間(秒)
interval	60	監視間隔(秒)
<b>error</b>	<b>0 ( or 1 or 2)</b>	<b>異常検出時の動作規定</b>
argc	1	エントリ呼び出し時の引数(argv)の数
<b>argv[0]</b>	<b>600</b>	<b>リトライオーバー時間</b>

・ACTION エントリの定義

ACTION エントリは、既存ライブラリ(lib\_bi\_stall.so)を使用します。

詳細については、「4.6.1.ファイルの更新日時によるストール監視」の(2) pfile の登録手順 ACTION エントリの定義を参照してください。

パラメーター	設定値	説明
entry	PS_ACTION	エントリの種類
type	DLL	部品の属性
file	/opt/HA/PS/lib/lib_bi_stall.so	ライブラリのファイル名
call	DIRECT	呼び出し方式
timeout	-(省略)	タイムアウト時間(秒)
interval	-(省略)	監視間隔(秒)
error	0 ( or 1)	アクション失敗の動作規定
argc	0 (or 1 or 2)	エントリ呼び出し時の引数 (argv) の数
argv[0]	-(省略)	ストール検出時に送信する signal (*)
argv[1]	-(省略)	SIGKILL リトライ回数 (*)

(\*) 詳細については、「4.6.1.ファイルの更新時刻によるストール監視」の

(4) ストール検出時の signal 送信についてを参照してください。

・QUIT エントリの定義

メッセージ監視の終了手順を定義します。

パラメーター	設定値	説明
entry	PS_QUIT	エントリの種類
type	DLL	部品の属性
file	/opt/HA/PS/lib/lib_bi_stall_message.so	ライブラリのファイル名
call	DIRECT	呼び出し方式
timeout	-(省略)	タイムアウト時間(秒)
interval	-(省略)	監視間隔(秒)
error	-(省略)	アクション失敗時の動作規定
argc	0	エントリ呼び出し時の引数(argv)の数

- (2) エラーメッセージが出力されたときストールと判断し対象プロセスを終了する場合の pfile の例  
 以下のように INIT エントリの指定をします。

**監視種別** : **error**  
**監視メッセージ** : **エラーメッセージ**

あわせて ACTION エントリも指定します。

```
##### entry type:file:call:timeout:interval: error:argc:argv[0]:...#####
##### stall checker #####
_bi_stall_message {
  PS_INIT      DLL:/opt/HA/PS/lib/lib_bi_stall_message.so:DIRECT:-::-:
                                     3:/tmp/logdat:error:Error Message
  PS_EXEC      DLL:/opt/HA/PS/lib/lib_bi_stall_message.so:DIRECT:-:60:0:1:600
  PS_ACTION  DLL:/opt/HA/PS/lib/lib_bi_stall.so:DIRECT:-::-:0:0
  PS_QUIT      DLL:/opt/HA/PS/lib/lib_bi_stall_message.so:DIRECT:-::-:0
}
```

- (3) 監視メッセージが出力されないときにストールと判断し対象プロセスを終了する場合の pfile の例  
 以下のように INIT エントリの指定をします。

**監視種別** : **required**  
**監視メッセージ** : **期待メッセージ**

あわせて ACTION エントリも指定します。

```
##### entry type:file:call:timeout:interval: error:argc:argv[0]:...#####
##### stall checker #####
_bi_stall_message {
  PS_INIT      DLL:/opt/HA/PS/lib/lib_bi_stall_message.so:DIRECT:-::-:
                                     3:/tmp/logdat:required:Normal Message
  PS_EXEC      DLL:/opt/HA/PS/lib/lib_bi_stall_message.so:DIRECT:-:60:0:1:600
  PS_ACTION  DLL:/opt/HA/PS/lib/lib_bi_stall.so:DIRECT:-::-:0:0
  PS_QUIT      DLL:/opt/HA/PS/lib/lib_bi_stall_message.so:DIRECT:-::-:0
}
```

- (4) ストール検出後、メッセージ監視を停止する pfile の例  
 以下のように EXEC エントリの指定をします。

**異常検出時の動作規定** : **1**

ACTION エントリの指定を行わない場合は特定の signal の送信は行わず、メッセージのみ出力します。

```
##### entry type: call:timeout:interval: error:argc:argv[0]:...#####
##### stall checker #####
_bi_stall_message {
  PS_INIT      DLL:/opt/HA/PS/lib/lib_bi_stall_message.so:DIRECT:-::-:
                                     3:/tmp/logdat:required:Normal Message
  PS_EXEC      DLL:/opt/HA/PS/lib/lib_bi_stall_message.so:DIRECT:-:60:1:1:600
  PS_QUIT      DLL:/opt/HA/PS/lib/lib_bi_stall_message.so:DIRECT:-::-:0
}
```

- (5) ストール検出時の signal 送信について  
「4.6.1.ファイルの更新時刻によるストール監視」の「(4)ストール検出時の signal 送信について」を参照してください。
  
- (6) OpenShift 環境のコンテナのヘルスチェックに本製品のコマンドを使用する場合の pfile の例  
「4.6.1.ファイルの更新時刻によるストール監視」の「(5) OpenShift 環境のコンテナのヘルスチェックに本製品のコマンドを使用する場合の pfile の例」を参照してください。



### 4.6.3. 対象プロセスのオープンファイルによるストール監視

監視対象プロセスのオープンファイル数によるストール監視(オープンファイル監視)機能の導入手順および pfile 登録手順について説明します。

#### (1) pfile の登録手順

pfile のプロセスエントリに続く行において、"\_bi\_stall\_openfile" という文字で囲まれたブロックを、オープンファイル監視の設定として解釈します。

##### ・エントリの定義

オープンファイル監視のエントリには INIT エントリ と EXEC エントリ と ACTION エントリ \*1 と QUIT エントリ が存在します。

```
_bi_stall_openfile { INIT エントリ, EXEC エントリ, ACTION エントリ, QUIT エントリ }
```

\*1 ACTION エントリは省略できます。

##### ・INIT エントリの定義

オープンファイル監視開始時の前処理を定義します。

**監視対象となる最大オープンファイル数**は 1(最小値)~1024(最大値)の間で指定してください。  
1 未満を指定すると、デフォルト値である 256 に補正します。

その他のパラメーターはデフォルト値を使用してください。(変更不可)

パラメーター	設定値	説明
entry	PS_INIT	エントリの種類
type	DLL	部品の属性
file	/opt/HAPS/lib/lib_bi_stall_openfile.so	ライブラリのファイル名
call	DIRECT	呼び出し方式
timeout	-(省略)	タイムアウト時間(秒)
interval	-(省略)	監視間隔(秒)
error	-(省略)	アクション失敗時の動作規定
argc	1	エントリ呼び出し時の引数(argv)の数
argv[0]	256	<b>監視対象となる最大オープンファイル数</b>

- EXEC エントリの定義  
オープンファイル監視の手順を定義します。

**異常検出時の動作規定**には以下の指定が可能です。

- 0: ストールを検出した際、以降継続してストール監視を実行します。  
ストール状態が解消されるまで、ストール検知の syslog メッセージを継続して出力します。
- 1: ストールを検出した際、以降ストール監視を実行しません。  
(注) ストール監視は停止しますが、プロセスの消滅監視は継続します。  
ストール検出を一度のみとする必要がある場合には、本パラメーターに 1 を指定してください。
- 2: ストールを検出した際、以降継続してストール監視を実行します。  
ストール状態が解消されるまで、ストール検知の syslog メッセージを抑制します。  
(注) OpenShift 環境のコンテナのヘルスチェック(readiness チェック)にストール監視を使用する場合は、本パラメーターに 2 を指定してください。

**リトライオーバー時間**にはプロセスをストール状態と解釈する時間(秒)を指定します。

60 秒以上を指定してください。デフォルト値は 600 秒指定となります。

600 秒指定の場合は、/proc/<対象プロセスのPID>/fd 配下のファイルを 60 秒ごとにチェックし、600 秒経過しても更新されていないければ、ストール状態と解釈します。一定期間総オープンファイル数が同じでもオープンしているファイルが異なる場合はストール状態とは解釈しません。

その他のパラメーターはデフォルト値を使用してください。(変更不可)

パラメーター	設定値	説明
entry	PS_EXEC	エントリの種類
type	DLL	部品の属性
file	/opt/HA/PS/lib/lib_bi_stall_openfile.so	ライブラリのファイル名
call	DIRECT	呼び出し方式
timeout	-(省略)	タイムアウト時間(秒)
interval	60	監視間隔(秒)
<b>error</b>	<b>0 ( or 1 or 2)</b>	<b>異常検出時の動作規定</b>
argc	1	エントリ呼び出し時の引数(argv)の数
<b>argv[0]</b>	<b>600</b>	<b>リトライオーバー時間</b>

・ACTION エントリの定義

ACTION エントリは、既存ライブラリ(lib\_bi\_stall.so)を使用します。

詳細については、「4.6.1.ファイルの更新日時によるストール監視」の(2) pfile の登録手順、ACTION エントリの定義を参照してください。

パラメーター	設定値	説明
entry	PS_ACTION	エントリの種類
type	DLL	部品の属性
file	/opt/HA/PS/lib/lib_bi_stall.so	ライブラリのファイル名
call	DIRECT	呼び出し方式
timeout	-(省略)	タイムアウト時間(秒)
interval	-(省略)	監視間隔(秒)
error	0 ( or 1)	アクション失敗の動作規定
argc	0 (or 1 or 2)	エントリ呼び出し時の引数 (argv) の数
argv[0]	-(省略)	ストール検出時に送信する signal (*)
argv[1]	-(省略)	SIGKILL リトライ回数 (*)

(\*) 詳細については、「4.6.1.ファイルの更新時刻によるストール監視」の  
(4) ストール検出時の signal 送信についてを参照してください。

・QUIT エントリの定義

オープンファイル監視の終了手順を定義します。

パラメーター	設定値	説明
entry	PS_QUIT	エントリの種類
type	DLL	部品の属性
file	/opt/HA/PS/lib/lib_bi_stall_openfile.so	ライブラリのファイル名
call	DIRECT	呼び出し方式
timeout	-(省略)	タイムアウト時間(秒)
interval	-(省略)	監視間隔(秒)
error	-(省略)	アクション失敗時の動作規定
argc	0	エントリ呼び出し時の引数(argv)の数

- (2) ストール検出後に対象プロセスを終了し、継続してオープンファイル監視を実行する pfile の例  
以下のように EXEC エントリの指定をします。

**異常検出時の動作規定** : 0

あわせて ACTION エントリも指定します。

```
##### entry type: call:timeout:interval: error:argc:argv[0]:...#####  
##### stall checker #####  
_bi_stall_openfile {  
    PS_INIT      DLL:/opt/HA/PS/lib/lib_bi_stall_openfile.so:DIRECT:-::-:1:256  
    PS_EXEC      DLL:/opt/HA/PS/lib/lib_bi_stall_openfile.so:DIRECT:-:60:0:1:600  
    PS_ACTION    DLL:/opt/HA/PS/lib/lib_bi_stall.so:DIRECT:-::-:0:0  
    PS_QUIT      DLL:/opt/HA/PS/lib/lib_bi_stall_openfile.so:DIRECT:-::-:0  
}
```

- (3) ストール検出後、オープンファイル監視を停止する pfile の例  
以下のように INIT エントリの指定をします。

**監視対象となる最大オープンファイル数** : 1024

また以下のように EXEC エントリの指定をします。

**異常検出時の動作規定** : 1

ACTION エントリの指定を行わない場合は特定の signal の送信は行わず、メッセージのみ出力します。

```
##### entry type: call:timeout:interval: error:argc:argv[0]:...#####  
##### stall checker #####  
_bi_stall_openfile {  
    PS_INIT      DLL:/opt/HA/PS/lib/lib_bi_stall_openfile.so:DIRECT:-::-:1:1024  
    PS_EXEC      DLL:/opt/HA/PS/lib/lib_bi_stall_openfile.so:DIRECT:-:60:1:1:600  
    PS_QUIT      DLL:/opt/HA/PS/lib/lib_bi_stall_openfile.so:DIRECT:-::-:0  
}
```

- (4) ストール検出時の signal 送信について  
「4.6.1.ファイルの更新時刻によるストール監視」の「(4)ストール検出時の signal 送信について」を参照してください。

- (5) OpenShift 環境のコンテナのヘルスチェックに本製品のコマンドを使用する場合の pfile の例  
「4.6.1.ファイルの更新時刻によるストール監視」の「(5) OpenShift 環境のコンテナのヘルスチェックに本製品のコマンドを使用する場合の pfile の例」を参照してください。

#### 4.6.4. 対象プロセスの起動スレッドによるストール監視

監視対象プロセスのスレッドによるストール監視(スレッド監視)機能の導入手順および pfile 登録手順について説明します。

##### (1) pfile の登録手順

pfile のプロセスエントリに続く行において、"\_bi\_stall\_thread" という文字で囲まれたブロックを、スレッド監視の設定として解釈します。

##### ・エントリの定義

スレッド監視のエントリには INIT エントリと EXEC エントリ と ACTION エントリ<sup>\*1</sup> と QUIT エントリ が存在します。

```
_bi_stall_thread { INIT エントリ, EXEC エントリ, ACTION エントリ QUIT エントリ }
```

<sup>\*1</sup> ACTION エントリは省略できます。

##### ・INIT エントリの定義

スレッド監視開始時の前処理を定義します。

**監視対象となる最大スレッド数**は 1(最小値)~4096(最大値)の間で指定してください。  
1 未満を指定すると、デフォルト値である 1024 に補正します。

その他のパラメーターはデフォルト値を使用してください。(変更不可)

パラメーター	設定値	説明
entry	PS_INIT	エントリの種類
type	DLL	部品の属性
file	/opt/HAPS/lib/lib_bi_stall_thread.so	ライブラリのファイル名
call	DIRECT	呼び出し方式
timeout	-(省略)	タイムアウト時間(秒)
interval	-(省略)	監視間隔(秒)
error	-(省略)	アクション失敗時の動作規定
argc	1	エントリ呼び出し時の引数(argv)の数
argv[0]	1024	<b>監視対象となる最大スレッド数</b>

- EXEC エントリの定義  
スレッド監視の手順を定義します。

**異常検出時の動作規定**には以下の指定が可能です。

- 0: ストールを検出した際、以降継続してストール監視を実行します。  
ストール状態が解消されるまで、ストール検知の syslog メッセージを継続して出力します。
- 1: ストールを検出した際、以降ストール監視を実行しません。  
(注) ストール監視は停止しますが、プロセスの消滅監視は継続します。  
ストール検出を一度のみとする必要がある場合には、本パラメーターに 1 を指定してください。
- 2: ストールを検出した際、以降継続してストール監視を実行します。  
ストール状態が解消されるまで、ストール検知の syslog メッセージを抑制します。  
(注) OpenShift 環境のコンテナのヘルスチェック(readiness チェック)にストール監視を使用する場合は、本パラメーターに 2 を指定してください。

**リトライオーバー時間**にはプロセスをストール状態と解釈する時間(秒)を指定します。

60 秒以上を指定してください。デフォルト値は 600 秒指定となります。

600 秒指定の場合は、/proc/<対象プロセスのPID>/task 配下のファイルを 60 秒ごとにチェックし、600 秒経過しても更新されていない場合は、ストール状態と解釈します。一定期間総スレッド数が同じでも構成するスレッド ID が異なる場合はストール状態とは解釈しません。

その他のパラメーターはデフォルト値を使用してください。(変更不可)

パラメーター	設定値	説明
entry	PS_EXEC	エントリの種類
type	DLL	部品の属性
file	/opt/HA/PS/lib/lib_bi_stall_thread.so	ライブラリのファイル名
call	DIRECT	呼び出し方式
timeout	-(省略)	タイムアウト時間(秒)
interval	60	監視間隔(秒)
<b>error</b>	<b>0 ( or 1 or 2)</b>	<b>異常検出時の動作規定</b>
argc	1	エントリ呼び出し時の引数(argv)の数
<b>argv[0]</b>	<b>600</b>	<b>リトライオーバー時間</b>

・ACTION エントリの定義

ACTION エントリは、既存ライブラリ(lib\_bi\_stall.so)を使用します。

詳細については、「4.6.1.ファイルの更新日時によるストール監視」の(2) pfile の登録手順の ACTION エントリの定義を参照してください。

パラメーター	設定値	説明
entry	PS_ACTION	エントリの種類
type	DLL	部品の属性
file	/opt/HA/PS/lib/lib_bi_stall.so	ライブラリのファイル名
call	DIRECT	呼び出し方式
timeout	-(省略)	タイムアウト時間(秒)
interval	-(省略)	監視間隔(秒)
error	0 ( or 1)	アクション失敗の動作規定
argc	0 (or 1 or 2)	エントリ呼び出し時の引数 (argv) の数
argv[0]	-(省略)	ストール検出時に送信する signal (*)
argv[1]	-(省略)	SIGKILL リトライ回数 (*)

(\*) 詳細については、「4.6.1.ファイルの更新時刻によるストール監視」の

(4) ストール検出時の signal 送信についてを参照してください。

・QUIT エントリの定義

スレッド監視の終了手順を定義します。

パラメーター	設定値	説明
entry	PS_QUIT	エントリの種類
type	DLL	部品の属性
file	/opt/HA/PS/lib/lib_bi_stall_thread.so	ライブラリのファイル名
call	DIRECT	呼び出し方式
timeout	-(省略)	タイムアウト時間(秒)
interval	-(省略)	監視間隔(秒)
error	-(省略)	アクション失敗時の動作規定
argc	0	エントリ呼び出し時の引数(argv)の数

- (2) ストール検出後に対象プロセスを終了し、継続してスレッド監視を実行する pfile の例  
以下のように EXEC エントリの指定をします。

**異常検出時の動作規定** : 0

あわせて ACTION エントリも指定します。

```
##### entry type: call:timeout:interval: error:argc:argv[0]:...#####
##### stall checker #####
_bi_stall_thread {
  PS_INIT      DLL:/opt/HA/PS/lib/lib_bi_stall_thread.so:DIRECT:-::-:1:1024
  PS_EXEC      DLL:/opt/HA/PS/lib/lib_bi_stall_thread.so:DIRECT:-:60:0:1:600
  PS_ACTION    DLL:/opt/HA/PS/lib/lib_bi_stall.so:DIRECT:-::-:0:0
  PS_QUIT      DLL:/opt/HA/PS/lib/lib_bi_stall_thread.so:DIRECT:-::-:0
}
```

- (3) ストール検出後、スレッド監視を停止する pfile の例  
以下のように EXEC エントリの指定をします。

**異常検出時の動作規定** : 1

ACTION エントリの指定を行わない場合は特定の signal の送信は行わず、メッセージのみ出力します。

```
##### entry type: call:timeout:interval: error:argc:argv[0]:...#####
##### stall checker #####
_bi_stall_thread {
  PS_INIT      DLL:/opt/HA/PS/lib/lib_bi_stall_thread.so:DIRECT:-::-:1:1024
  PS_EXEC      DLL:/opt/HA/PS/lib/lib_bi_stall_thread.so:DIRECT:-:60:1:1:600
  PS_QUIT      DLL:/opt/HA/PS/lib/lib_bi_stall_thread.so:DIRECT:-::-:0
}
```

- (4) ストール検出時の signal 送信について

「4.6.1.ファイルの更新時刻によるストール監視」の「(4)ストール検出時の signal 送信について」を参照してください。

- (5) OpenShift 環境のコンテナのヘルスチェックに本製品のコマンドを使用する場合の pfile の例

「4.6.1.ファイルの更新時刻によるストール監視」の「(5) OpenShift 環境のコンテナのヘルスチェックに本製品のコマンドを使用する場合の pfile の例」を参照してください。



#### 4.6.5. 対象プロセスのプロセス状態によるストール監視

監視対象プロセスのプロセス状態によるストール監視(プロセス状態監視)機能の導入手順および pfile 登録手順について説明します。

##### (1) pfile の登録手順

pfile のプロセスエンタリに続く行において、"\_bi\_stall\_general" という文字で囲まれたブロックを、プロセス状態監視の設定として解釈します。

##### ・エンタリの定義

プロセス状態監視のエンタリには INIT エンタリと EXEC エンタリ と ACTION エンタリ \*1 と QUIT エンタリ が存在します。

```
_bi_stall_general { INIT エンタリ, EXEC エンタリ, ACTION エンタリ, QUIT エンタリ }
```

\*1 ACTION エンタリは省略できます。

・INIT エントリの定義

プロセス状態監視開始時の前処理を定義します。

エントリ呼び出し時の引数(argv)の数は以下のとおりにします。

- 1: スタック情報ファイルを使わない場合、1 と指定します。
- 2: スタック情報ファイルを使う場合、2 と指定します。

プロセスストールの判断規定には以下の指定が可能です。

- 0: 特定のファンクションコール名<sup>\*1</sup>でプロセスが待機している場合、プロセスストールと判断します。
- 1: スタック情報ファイルに記載したファンクションコール名でプロセスが待機している場合、プロセスストールと判断します。(ブラックリスト判定)
- 2: スタック情報ファイルに記載されていないファンクションコール名でプロセスが待機している場合、プロセスストールと判断します。(ホワイトリスト判定)

本パラメーターの指定は必須となります。

本パラメーターで 1 および 2 を指定した場合には、つぎのパラメーター**スタック情報ファイル名**を指定してください。

**スタック情報ファイル名**にはプロセスの状態が一定時間変化しないときにストール判定に使用する待機ファンクションコール名を列挙したテキストファイルの絶対パスを指定します。<sup>\*2</sup>

その他のパラメーターはデフォルト値を使用してください。(変更不可)

<sup>\*1</sup> 待機ファンクションコール名が `futex_wait_queue_me` のときにストールと判断します。

<sup>\*2</sup> **スタック情報ファイル**の使用については本章の (3) **スタック情報ファイル**に記載したファンクションコールでプロセスが動きを止めた場合にストールと判断する場合の `pfile` の例、(4) **スタック情報ファイル**に記載されていないファンクションコールでプロセスが動きを止めた場合にストールと判断する場合の `pfile` の例 で説明します。

**スタック情報ファイル**の作り方、ストール判定の詳細については (5) プロセスの待機ファンクションコール名によるストール判定 内の ③**スタック情報ファイル**の作成 で説明します。

パラメーター	設定値	説明
<code>entry</code>	PS_INIT	エントリの種類
<code>type</code>	DLL	部品の属性
<code>file</code>	/opt/HA/PS/lib/lib_bi_stall_general.so	ライブラリのファイル名
<code>call</code>	DIRECT	呼び出し方式
<code>timeout</code>	-(省略)	タイムアウト時間(秒)
<code>interval</code>	-(省略)	監視間隔(秒)
<code>error</code>	-(省略)	アクション失敗時の動作規定
<code>argc</code>	1 ( or 2 )	エントリ呼び出し時の引数(argv)の数
<code>argv[0]</code>	0 ( or 1 or 2 )	プロセスストールの判断規定
<code>argv[1]</code>	/var/opt/HA/PS/conf/bin/filename	スタック情報ファイル名

・EXEC エントリの定義

プロセス状態監視の手順を定義します。

**異常検出時の動作規定**には以下の指定が可能です。

- 0: ストールを検出した際、以降継続してストール監視を実行します。  
ストール状態が解消されるまで、ストール検知の syslog メッセージを継続して出力します。
- 1: ストールを検出した際、以降ストール監視を実行しません。  
(注) ストール監視は停止しますが、プロセスの消滅監視は継続します。  
ストール検出を一度のみとする必要がある場合には、本パラメーターに 1 を指定してください。
- 2: ストールを検出した際、以降継続してストール監視を実行します。  
ストール状態が解消されるまで、ストール検知の syslog メッセージを抑制します。  
(注) OpenShift 環境のコンテナのヘルスチェック(readiness チェック)にストール監視を使用する場合は、本パラメーターに 2 を指定してください。

**リトライオーバー時間**にはプロセスをストール状態と解釈する時間(秒)を指定します。

60 秒以上を指定してください。デフォルト値は 600 秒指定となります。

600 秒指定の場合は、プロセス状態監視を 60 秒ごとにチェックし、状態が 600 秒経過しても変化なければストール状態と解釈します。

**ストール推定発生時の動作規定**はストールの疑いのあるプロセスに対する対処の指定です。

対象プロセスが動きを止めてリトライオーバー時間の経過後にカーネルスタックを確認し、ストールとは認められない(この状態を以後「ストールの推定状態」と呼ぶ)ときに、ACTION エントリの処理を実行するか否かを指定します。

以下の指定が可能です。

- 0: ACTION エントリの処理を実行しません。
- 1: ACTION エントリの処理を実行します。

**ストール推定発生時の SYSLOG 出力**はストール推定の発生、あるいはストール推定の解消を syslog に情報出力するか否かの指定です。

以下の指定が可能です。

- 0: syslog に情報出力しない。
- 1: syslog に情報出力する。

その他のパラメーターはデフォルト値を使用してください。(変更不可)

パラメーター	設定値	説明
entry	: PS_EXEC	エントリの種類
type	: DLL	部品の属性
file	: /opt/HAPS/lib/lib_bi_stall_general.so	ライブラリのファイル名
call	: DIRECT	呼び出し方式
timeout	: -(省略)	タイムアウト時間(秒)
interval	: 60	監視間隔(秒)
<b>error</b>	<b>: 0 (or 1 or 2)</b>	<b>異常検出時の動作規定</b>
argc	: 3	エントリ呼び出し時の引数(argv)の数
<b>argv[0]</b>	<b>: 600</b>	<b>リトライオーバー時間</b>
<b>argv[1]</b>	<b>: 0 (or 1)</b>	<b>ストール推定発生時の動作規定</b>
<b>argv[2]</b>	<b>: 0 (or 1)</b>	<b>ストール推定発生時の syslog 出力</b>

・ACTION エントリの定義

ACTION エントリは、既存ライブラリ(lib\_bi\_stall.so)を使用します。

詳細については、「4.6.1.ファイルの更新日時によるストール監視」の(2) pfile の登録手順

ACTION エントリの定義を参照してください。

パラメーター	設定値	説明
entry	PS_ACTION	エントリの種類
type	DLL	部品の属性
file	/opt/HA/PS/lib/lib_bi_stall.so	ライブラリのファイル名
call	DIRECT	呼び出し方式
timeout	-(省略)	タイムアウト時間(秒)
interval	-(省略)	監視間隔(秒)
error	0 (or 1)	アクション失敗の動作規定
argc	0 (or 1 or 2)	エントリ呼び出し時の引数 (argv) の数
argv[0]	-(省略)	ストール検出時に送信する signal (*)
argv[1]	-(省略)	SIGKILL リトライ回数 (*)

(\*) 詳細については、「4.6.1.ファイルの更新時刻によるストール監視」の

(4) ストール検出時の signal 送信についてを参照してください。

・QUIT エントリの定義

プロセス状態監視の終了手順を定義します。

パラメーター	設定値	説明
entry	PS_QUIT	エントリの種類
type	DLL	部品の属性
file	/opt/HA/PS/lib/lib_bi_stall_general.so	ライブラリのファイル名
call	DIRECT	呼び出し方式
timeout	-(省略)	タイムアウト時間(秒)
interval	-(省略)	監視間隔(秒)
error	-(省略)	アクション失敗時の動作規定
argc	0	エントリ呼び出し時の引数(argv)の数

pfile のテンプレートについては、「3.1. pfile、環境ファイルについて」の(2) プロセス監視のテンプレート内の /var/opt/HA/PS/conf/src/SG\_linux\_sample 配下の pfile テンプレート **stallchk\_pfile** をご利用ください。

- (2) 特定のファンクションコールでプロセスが動きを止めた場合にストールと判断する場合の pfile の例  
 以下のように INIT エントリの指定をします。

**エントリ呼び出し時の引数(argv)の数** : 1  
**プロセスストールの判断規定** : 0

またストール検出後、プロセス状態監視を停止する場合は以下のように EXEC エントリの指定をします。

**異常検出時の動作規定** : 1

ACTION エントリの指定を行わない場合は特定の signal の送信は行わず、メッセージのみ出力します。

```
##### entry type:file:call:timeout:interval: error:argc:argv[0]:...#####
##### stall checker #####
_bi_stall_general {
  PS_INIT      DLL: /opt/HA/PS/lib/lib_bi_stall_general.so:DIRECT:-:-:1:0
  PS_EXEC      DLL: /opt/HA/PS/lib/lib_bi_stall_general.so:DIRECT:-:60:1:3:600:0:0
  PS_QUIT      DLL: /opt/HA/PS/lib/lib_bi_stall_general.so:DIRECT:-:-:-:0
}
```

- (3) スタック情報ファイルに記載したファンクションコールでプロセスが動きを止めた場合にストールと判断する場合の pfile の例

以下のように INIT エントリの指定をします。

**エントリ呼び出し時の引数(argv)の数** : 2  
**プロセスストールの判断規定** : 1  
**スタック情報ファイル名:** : **ファイルの絶対パス**

またストール検出後も継続してプロセス状態監視を実行し、ストール推定状態で対象プロセスを終了する場合は以下のように EXEC エントリの指定をします。

**異常検出時の動作規定** : 0  
**ストール推定発生時の動作規定** : 1

あわせて ACTION エントリも指定します。

```
##### entry type:file:call:timeout:interval: error:argc:argv[0]:...#####
##### stall checker #####
_bi_stall_general {
  PS_INIT      DLL: /opt/HA/PS/lib/lib_bi_stall_general.so:DIRECT:-:-:-:
                2:1: /var/opt/HA/PS/conf/bin/filename
  PS_EXEC      DLL: /opt/HA/PS/lib/lib_bi_stall_general.so:DIRECT:-:60:0:3:600:1:0
  PS_ACTION    DLL: /opt/HA/PS/lib/lib_bi_stall.so:DIRECT:-:-:0:0
  PS_QUIT      DLL: /opt/HA/PS/lib/lib_bi_stall_general.so:DIRECT:-:-:-:0
}
```

- (4) スタック情報ファイルに記載されていないファンクションコールでプロセスが動きを止めた場合にストールと判断する場合の pfile の例

以下のように INIT エントリの指定をします。

*エントリ呼び出し時の引数(argv)の数* : 2  
*プロセスストールの判断規定* : 2  
*スタック情報ファイル名* : *ファイルの絶対パス*

またストール検出後も継続してプロセス状態監視を実行し、正常 → ストール推定 への状態遷移時、ならびに ストール推定 → 正常 への状態遷移時に syslog に情報を出力する場合は以下のように EXEC エントリの指定をします。

*異常検出時の動作規定* : 0  
*ストール推定発生時の動作規定* : 0  
*ストール推定発生時の SYSLOG 出力* : 1

```
##### entry type:file:call:timeout:interval: error:argc:argv[0]:...#####  
##### stall checker #####  
_bi_stall_general {  
  PS_INIT      DLL: /opt/HA/PS/lib/lib_bi_stall_general.so:DIRECT:-::-:  
                2:2: /var/opt/HA/PS/conf/bin/filename  
  PS_EXEC      DLL: /opt/HA/PS/lib/lib_bi_stall_general.so:DIRECT:-:60:0:3:600:0:1  
  PS_ACTION    DLL: /opt/HA/PS/lib/lib_bi_stall.so:DIRECT:-::-:0:0  
  PS_QUIT      DLL: /opt/HA/PS/lib/lib_bi_stall_general.so:DIRECT:-::-:0  
}
```

(5) プロセスの待機ファンクションコール名によるストール判定

プロセス状態監視では、対象プロセスの状態が一定時間変化しないとき、ストール判定に使用する待機ファンクションコール名を、対象プロセスの全スレッドのスタックファイルから収集します。

① プロセスの待機ファンクションコール名

収集する情報は対象プロセスの全スレッドのスタックファイルの先頭行のファンクションコール名です。

一例として下図に示すスタックファイルの場合は文字列 `futex_wait_queue_me` が待機ファンクションコール名となります。

```
[ /proc/[pid]/task/[tid]/stack  
  
# cat /proc/2390/task/2390/stack  
[<0>] futex_wait_queue_me+0xc1/0x120  
[<0>] futex_wait+0x13f/0x240  
[<0>] do_futex+0x147/0xac0  
[<0>] __x64_sys_futex+0x143/0x180  
[<0>] do_syscall_64+0x5b/0x1b0  
[<0>] entry_SYSCALL_64_after_hwframe+0x65/0xca  
[<0>] 0xffffffffffffffff  
#
```

② 待機ファンクションコール名でのストール判定

上記①の方法で収集した待機ファンクションコール名は、INIT エントリのパラメーター **プロセスストールの判断規定** に沿って検索名称と一致確認を行います。

**プロセスストールの判断規定 0** のとき(特定ファンクションコール名判定)

検索用のファンクションコール名(検索名称)は `futex_wait_queue_me` となります。

すべてのスレッドの待機ファンクションコール名と検索名称の一致確認を行います。

- 対象プロセス内のすべてのスレッドで待機ファンクションコール名と検索名称と部分一致する場合「ストール状態」と判断します。
- 対象プロセス内で検索名称と部分一致するスレッドと、部分一致しないスレッドがある場合「ストール推定」と判断します。
- 対象プロセス内のすべてのスレッドで待機ファンクションコール名と検索名称が部分一致しない場合「ストール推定」と判断します。

**プロセスストールの判断規定 1** のとき(ブラックリスト判定)

検索用のファンクションコール名(検索名称)は **スタック情報ファイル** から取得します。

すべてのスレッドの待機ファンクションコール名と検索名称の一致確認を行います。

- 対象プロセス内のすべてのスレッドで待機ファンクションコール名と検索名称が部分一致する場合「ストール状態」と判断します。
- 対象プロセス内で検索名称と部分一致するスレッドと、部分一致しないスレッドがある場合「ストール推定」と判断します。
- 対象プロセス内のすべてのスレッドで待機ファンクションコール名と検索名称が部分一致しない場合「ストール推定」と判断します。

**プロセスストールの判断規定 2** のとき(ホワイトリスト判定)

検索用のファンクションコール名(検索名称)は **スタック情報ファイル** から取得します。

すべてのスレッドの待機ファンクションコール名と検索名称の一致確認を行います。

- 対象プロセス内のすべてのスレッドで待機ファンクションコール名と検索名称が部分一致する場合「正常状態」と判断します。
- 対象プロセス内で検索名称と部分一致するスレッドと、部分一致しないスレッドがある場合

「ストール推定」と判断します。

- 対象プロセス内のすべてのスレッドで待機ファンクションコール名と検索名称が部分一致しない場合「ストール状態」と判断します。

### ③ スタック情報ファイルの作成

上記②のなかで登場する**スタック情報ファイル**の作成方法を説明します。

- スタック情報ファイル**は一致確認用のファンクションコール名(検索名称)を記載したテキストファイルです。
- コメント行としたい場合は先頭に `#` を書いてください。
- 検索名称は各行にひとつとしてください。各行の区切り文字(半角スペース ` `、およびタブ `␣`)で区切られた最初の文字列を検索名称として扱います。
- 重複する(既出の)検索名称が見つかった場合はその行の情報を無視します。
- 検索名称は最大 256 個まで指定可です。
- 257 個目以降の検索名称は無効となります。

### スタック情報の定義例

[ スタック情報ファイル (/var/opt/HA/PS/conf/bin/filename) ]

```
##### STACK DEF #####
futex_wait_queue_me

# futex_wait_queue_me   スレッド間でデッドロックするプロセス(mutex使用)
# hrtimer_nanosleep     プロセス間でデッドロックするプロセス(flock使用)
# hrtimer_nanosleep     時間が来るまで待ち合わせるプロセス(crond)
# poll_schedule_timeout 処理が要求されるまで待ち合わせるプロセス(sshd)
# SYSSEMtimedop        処理が要求されるまで待ち合わせるプロセス(httpd)
# __sb_start_write     ディスクへのアクセス中に停止しているプロセス
# hrtimer_nanosleep     OSの更新情報を定期的にチェックするデーモン(rhnsd)
```

#### 【参考】

スタック情報ファイルのテンプレートとして、「3.1. pfile、環境ファイルについて」の(2) プロセス監視のテンプレート内の /var/opt/HA/PS/conf/src/SG\_linux\_sample 配下の **stack\_def** をご利用ください。

#### (6) ストール検出時の signal 送信について

「4.6.1.ファイルの更新時刻によるストール監視」の「(4)ストール検出時の signal 送信について」を参照してください。

#### (7) OpenShift 環境のコンテナのヘルスチェックに本製品のコマンドを使用する場合の pfile の例

「4.6.1.ファイルの更新時刻によるストール監視」の「(5) OpenShift 環境のコンテナのヘルスチェックに本製品のコマンドを使用する場合の pfile の例」を参照してください。



## 4.7. ストール監視のユーザー組み込みライブラリの導入手順

ストール監視機能のユーザー組み込みライブラリの導入手順について説明します。

### (1) ストール監視のユーザー組み込みライブラリとは

対象プロセスはユーザー組み込みライブラリである libHAPSuser を組み込むことで pcheck との連携が容易に実現できます。  
このライブラリは以下の API を提供します。

API 仕様の詳細については、「7. リファレンス」を参照してください。

- HA\_PS\_ckstall\_init()           対象ファイルの初期化関数
- HA\_PS\_ckstall\_update()       対象ファイルの更新関数
- HA\_PS\_ckstall\_end()          対象ファイルの消去関数

(注) 本ライブラリは 64 bit アプリケーションとして作成されたプロセスのみ使用可能です。  
32 bit アプリケーションとして作成されたプロセスでは使用できません。

### (2) libHAPSuser について

libHAPSuser は、以下のファイルを提供します。

ダイナミックリンクライブラリ	<b>/opt/HA/PS/lib/libHAPSuser.so</b>
スタティックリンクライブラリ	<b>/opt/HA/PS/lib/libHAPSuser.a</b>
ヘッダーファイル	<b>/usr/include/sys/ha_ps_user.h</b>

libHAPSuser を利用する場合は、/usr/lib 配下にリンクを張ってください。

```
#ln -s /opt/HA/PS/lib/libHAPSuser.so /usr/lib/libHAPSuser.so  
#ln -s /opt/HA/PS/lib/libHAPSuser.a /usr/lib/libHAPSuser.a
```

### (3) ストール監視を実装したユーザープロセスのサンプルソース

実際に、libHAPSuser を組み込んでストール監視を実現したサンプルソースを以下に示しますので参考にしてください。

<sample.c>

```
/* sample program for ProcessSaver stall check */
/* How to compile this sample program:
 *   cc -D_LINUX -fwritable-strings ¥
 *   -I<ProcessSaver header file directory> -o sample sample.c ¥
 *   -L<ProcessSaver library directory> -IHAPSuser
 */
#include <sys/param.h>
#include <stdio.h>
#include <string.h>
#include <strings.h>
#include "sys/ha_ps_user.h"
#define STALL_CHECK_FILE    "/tmp/ha_ps_stall_check"

int
#ifdef __STDC__
main(int argc, char *argv[])
#else
main(argc,argv)
    int argc;
    char *argv[];
#endif
{
    int ret, cnt;
    char file_name[MAXPATHLEN];

    /* setup file_name */
    (void)strcpy(file_name, STALL_CHECK_FILE);
    ret = HA_PS_ckstall_init( file_name );
    if (ret != HA_PS_SUCCESS) {
        /* Do not use stall checking */
        fprintf(stderr, "Can not check process stall.¥n");
        return 1;
    }

    /* main loop */
    for (cnt = 0; ; cnt++) {
        ret = HA_PS_ckstall_update();
        if (ret != HA_PS_SUCCESS) {
            /* Do not use stall checking */
            break;
        }
        sleep(10);
    }

    /* finalize stall checking */
    (void)HA_PS_ckstall_end();

    return 0;
}
```

## 4.8. サイレントモードでの運用手順

ログメッセージを抑制して運用する手順について説明します。

pcheck、padmin は、運用中システムログにメッセージを出力します。  
システムログにメッセージを出力させたくない場合、メッセージを抑制したサイレントモードで実行することができます。

### (1) 導入手順

- ① 環境変数 HAPS\_SILENT\_MODE に 1 を設定します。

・sh の場合

```
# HAPS_SILENT_MODE=1  
# export HAPS_SILENT_MODE
```

・csh の場合

```
% setenv HAPS_SILENT_MODE 1
```

※ 上記のように環境変数を設定した状態で起動した pcheck および  
実行した padmin コマンドについては、システムログメッセージを抑制した  
状態で実行されます。

- ② pcheck を起動し、プロセス監視を開始します。  
# /opt/HAPS/bin/pcheck -f <pfile 名> &

### (2) 確認手順

動作中の pcheck のモードについては、padmin コマンドで確認することができます。

```
# /opt/HAPS/bin/padmin -f <pfile 名> -c show env  
HAPS_SILENT_MODE = on ★
```

サイレントモードで pcheck を起動している場合は "on"、  
サイレントモードでない場合は、"off" が表示されます。

### (3) 運用上の注意・制限事項

- ・ サイレントモードで運用した場合は、pcheck のメッセージは一切出力されません。  
そのため障害発生時の解析などは困難となりますので、あらかじめご了承ください。  
一般的には、通常モードでの運用を推奨します。
- ・ すでに起動中の pcheck を、サイレントモードに変更することはできません。  
一度 pcheck を padmin -c shutdown で終了させてから、環境変数を設定し pcheck を  
再起動してください。

## 4.9.pcheck 終了時に子プロセスを回収する手順

pcheck コマンドは、再起動スクリプト実行時に内部で fork(2) を使用しているため pcheck 終了時に生成された子プロセスなどが残ってしまう場合があります。

子プロセスが残ってしまうことを避けるために、pcheck 終了時に関連のプロセスを終了する

子プロセス終了モードで運用することが可能です。

子プロセス終了モードの設定手順は以下のとおりです。

### (1) 導入手順

- ① 環境変数 HAPS\_CHILDPROC\_KILL に 1 を設定します。

・sh の場合

```
#HAPS_CHILDPROC_KILL=1
# export HAPS_CHILDPROC_KILL
```

・csh の場合

```
% setenv HAPS_CHILDPROC 1
```

※ 上記のように環境変数を設定した状態で起動した pcheck については、pcheck 終了時に、子プロセスなどの関連プロセスを終了します。

- ② pcheck を起動し、プロセス監視を開始します。

```
# /opt/HAPS/bin/pcheck -f <file> &
```

### (2) 確認手順

動作中の pcheck のモードは、padmin コマンドで確認することができます。

```
# /opt/HAPS/bin/padmin -f <pfile 名> -c show env
HAPS_CHILDPROC_KILL = on ★
```

子プロセス終了モードで起動している pcheck の場合は on、  
子プロセス終了モードでない場合は、off が表示されます。

### (3) 運用上の注意・制限事項

- ・ すでに起動中の pcheck を、子プロセス終了モードに変更することはできません。一度 pcheck を padmin -c shutdown で終了させてから、環境変数を設定し pcheck を再起動してください。
- ・ 子プロセス終了モードで実行しない場合、pcheck 終了後に pcheck の子プロセスなどが残留する可能性があります。基本的な運用には特に影響はございません。

## 4.10. pcheck 起動時の自動待ち合わせ時間を変更する手順

pcheck コマンドは、起動時および reload 実行時に監視対象のプロセスが存在しない場合に、監視対象プロセスが起動するまで一定時間自動的に待ち合わせを行います。自動待ち合わせ時間内に監視対象のプロセスが起動した場合は、その時点から監視を開始します。自動待ち合わせ時間はデフォルトで 60 秒となっていますが、環境変数 HAPS\_PENDING\_TIME を設定することで待ち合わせ時間を変更することができます。

pcheck 起動時にすぐに消滅を検出したい場合や監視対象プロセスの起動に時間がかかる場合などに自動待ち合わせ時間を変更してください。通常は変更する必要はありません。設定は秒単位で指定し、設定可能な範囲は 0 ~ 86400 (=60\*60\*24 (1 日)) です。

自動待ち合わせ時間を変更する手順は以下のとおりです。

(1) pcheck 起動後すぐに監視対象プロセスの消滅を検出したい(自動待ち合わせを行わない) 場合の手順

- ① 環境変数 HAPS\_PENDING\_TIME に 0 を設定します。

•sh の場合

```
#HAPS_PENDING_TIME=0
#export HAPS_PENDING_TIME
```

•csh の場合

```
% setenv HAPS_PENDING_TIME 0
```

※ 上記のように環境変数を設定した状態で起動した pcheck については、pcheck 起動時に監視対象プロセスが存在しない場合、すぐに監視対象プロセスの消滅を検出します。

- ② pcheck を起動し、プロセス監視を開始します。

```
#/opt/HA/PS/bin/pcheck -f <pfile> &
```

- ③ 確認手順

動作中の pcheck の自動待ち合わせ時間は、padmin コマンドで確認することができます。

```
#/opt/HA/PS/bin/padmin -f <pfile 名> -c show env
HAPS_PENDING_TIME    = 0 ★
```

設定した自動待ち合わせ時間が表示されます。

環境変数 HAPS\_PENDING\_TIME を設定していない場合は、デフォルトの 60 が表示されます。

(2) 自動待ち合わせ時間を 120 秒に変更する場合の手順

- ① 環境変数 HAPS\_PENDING\_TIME に 120 を設定します。

・sh の場合

```
#HAPS_PENDING_TIME=120  
# export HAPS_PENDING_TIME
```

・csh の場合

```
% setenv HAPS_PENDING_TIME 120
```

※ 上記のように環境変数を設定した状態で起動した pcheck については、pcheck 起動時に監視対象プロセスが存在しない場合、120 秒間待ち合わせを行います。  
pcheck 起動後、120 秒経過しても監視対象プロセスが起動していない場合に監視対象プロセスの消滅を検出します。

- ② pcheck を起動し、プロセス監視を開始します。

```
#/opt/HA/PS/bin/pcheck -f <pfile> &
```

- ③ 確認手順

動作中の pcheck の自動待ち合わせ時間は、padmin コマンドで確認することができます。

```
#/opt/HA/PS/bin/padmin -f <pfile 名> -c show env  
HAPS_PENDING_TIME = 120 ★
```

設定した自動待ち合わせ時間が表示されます。

環境変数 HAPS\_PENDING\_TIME を設定していない場合は、デフォルトの 60 が表示されます。

(3) 運用上の注意・制限事項

- すでに起動中の pcheck の自動待ち合わせ時間を変更することはできません。  
一度 pcheck を padmin -c shutdown で終了させてから、環境変数を設定し pcheck を再起動してください。  
指定した時間内に監視対象のプロセスが起動した場合は、その時点で監視を開始します。

## 4.11. pcheck 起動時のサマリー情報を syslog 出力する手順

pcheck 起動時に監視を開始したプロセスの個数と、監視対象プロセスの個数を syslog に出力します。サマリーメッセージを出力するように設定することで、プロセスを大量に監視するシステムの場合であっても pcheck 起動時にすべてのプロセスの監視が正しく行われていることを容易に判断することが可能となります。

### (1) pfile の記述方式

pfile の共通部情報 (PARAM) に、UP\_MESSAGE\_REDUCE\_MODE を指定することで、pcheck 起動時の監視プロセス数を syslog に出力することが可能となります。

### 共通部情報 (PARAM) の設定

共通部情報のフォーマットは以下のとおりです。

IPCKEY	ipckey
MSG_CHECK_INTERVAL	msg_check_interval
MONITOR_INTERVAL	monitor_interval
SHM_DUMP_FILE	shm_dump_file
<b>UP_MESSAGE_REDUCE_MODE</b>	<b>up_message_reduce_mode</b> ←追加

設定値について以下に説明します。

#### **up\_message\_reduce\_mode**

enable または disable のみ設定可能です。

enable を設定した場合、pcheck 起動時および reload 実行時に監視プロセス数と、pent に記載されたプロセス数を syslog に出力します。

disable またはパラメーター未設定の場合は、従来のメッセージのみ出力し、pcheck 起動時の監視プロセス数を示すメッセージは出力されません。

pfile の記述例は以下となります。

```
# HA/ProcessSaver configuration
##### PARAM #####
IPCKEY                                0x1f000001
MSG_CHECK_INTERVAL                    5
MONITOR_INTERVAL                      10
SHM_DUMP_FILE                         /var/opt/HA/PS/log/pcheck_dump
UP_MESSAGE_REDUCE_MODE           enable

##### PENT #####
/usr/sbin/proc1:/var/opt/HA/PS/conf/src/rst1.sh:86400:3:continue
```

## (2) syslog メッセージ

syslog の facility と level は以下のとおりです。

facility : LOG\_USER

level : LOG\_INFO

pcheck 起動時の監視開始プロセス数出力機能で出力されるメッセージは以下です。

***pcheck start. (up\_proc\_num=<監視を開始したプロセス数>,  
total\_pent\_num=<PENT 記載のプロセス数>) pfile=<pfile 名>***

syslog メッセージの出力例は以下となります。

(例) pcheck 起動時、pent に記載された 3 個のプロセスのうち 2 個のプロセスの起動を確認した場合

```
May  8 16:23:45 node1 pcheck: pcheck start (up_proc_num=2, total_proc_num=3  
                                     pfile=/var/opt/HA/PS/conf/pfile_test
```

## (3) 確認手順

動作中の pcheck の設定については、padmin コマンドで確認することができます。

```
# /opt/HA/PS/bin/padmin -f <pfile 名> -c show param  
MSG_CHECK_INTERVAL      = 5  
MONITOR_INTERVAL        = 10  
MONITOR_TRY_COUNT       = 2  
SHM_DUMP_FILE           = /opt/HA/PS/log/pcheck_dump  
PFILE                    = pfile  
MESSAGE_BOX              = start  
MONITOR_STOP_COUNT      = 2  
FAIL_PROC_COUNT          = 0  
ALL_PROC_COUNT           = 4  
UP_MESSAGE_REDUCE_MODE = enable ★
```

★ UP\_MESSAGE\_REDUCE\_MODE enable 設定時に表示されます。  
UP\_MESSAGE\_REDUCE\_MODE disable 設定時または、  
UP\_MESSAGE\_REDUCE\_MODE 未指定時は表示されません。



## 4.12. 監視対象選択時のプロセス情報取得量を変更する手順

pcheck は、起動時および padmin の start や restart コマンド実行時に OS の /proc 配下から稼働している全プロセス情報を検索して、監視対象プロセスの特定を行います。通常、監視対象プロセスの特定後は PID を使用して監視を行うため、全プロセス情報の検索を行うことはありませんが、max\_proc\_count オプションを指定した場合には、プロセス数を算出するために監視タイミングごとに全プロセス情報を検索します。

デフォルトではこの全プロセス情報取得処理は 1000 プロセス単位で行いますが、環境変数 HAPS\_GETPROC\_COUNT を設定することで取得するプロセス数の単位を変更することができます。システムに稼働するプロセス数が 10000 を越えるような大規模環境の場合に、必要に応じて稼働プロセス数よりも大きなプロセス情報取得量を設定してください。通常は変更する必要はありません。設定可能な範囲は 1 ~ 100000 です。

プロセス情報取得量を変更する手順は以下のとおりです。

(1) pcheck 起動時に 15000 プロセス分を一度に取得する場合  
(システムに稼働するプロセス数が 14000 程度の場合など)

① 環境変数 HAPS\_GETPROC\_COUNT に 15000 を設定します。

・sh の場合

```
# HAPS_GETPROC_COUNT=15000
# export HAPS_GETPROC_COUNT
```

・csh の場合

```
% setenv HAPS_GETPROC_COUNT 15000
```

※ 上記のように環境変数を設定した状態で起動した pcheck については、pcheck 起動時に最大 15000 プロセス分の情報を一度に取得します。

② pcheck を起動し、プロセス監視を開始します。

```
#/opt/HA/PS/bin/pcheck -f <pfile> &
```

③ 確認手順

動作中の pcheck のプロセス情報取得量は、padmin コマンドで確認することができます。

```
#/opt/HA/PS/bin/padmin -f <pfile 名> -c show env
HAPS_GETPROC_COUNT = 15000 ★
```

設定したプロセス情報取得量が表示されます。

環境変数 HAPS\_GETPROC\_COUNT を設定していない場合は、デフォルトの 1000 が表示されます。

(2) 運用上の注意・制限事項

- ・すでに起動中の pcheck のプロセス情報取得量を変更することはできません。一度 pcheck を padmin -c shutdown で終了させてから、環境変数を設定し pcheck を再起動してください。
- ・一回の情報取得量を多くするほど、pcheck 内部で一時的に確保するメモリ量が増えることとなります。たとえば 15000 プロセス指定時は約 15 MB 程度、pcheck 起動時などの監視対象プロセス特定処理時にのみ一時的に増加します。

## 4.13. 監視対象選択時のリトライ回数を変更する手順

pcheck は、起動時および padmin の start や restart コマンド実行時に OS の /proc 配下を検索して、監視対象プロセスの特定を行います。このとき、デフォルトでは最大 6 回の情報取得のリトライを行いますが、環境変数 HAPS\_GETPROC\_RETRY\_COUNT を設定することで変更することができます。通常は変更する必要はありません。設定可能な範囲は 1 ~ 100 です。

プロセス情報取得時のリトライ回数を変更する手順は以下のとおりです。

### (1) リトライを最大 10 回行う場合

- ① 環境変数 HAPS\_GETPROC\_RETRY\_COUNT に 10 を設定します。

・sh の場合

```
# HAPS_GETPROC_RETRY_COUNT=10
# export HAPS_GETPROC_RETRY_COUNT
```

・csh の場合

```
% setenv HAPS_GETPROC_RETRY_COUNT 10
```

※ 上記のように環境変数を設定した状態で起動した pcheck については、最大 10 回のプロセス情報取得のリトライを行います。

- ② pcheck を起動し、プロセス監視を開始します。

```
# /opt/HA/PS/bin/pcheck -f <pfile> &
```

- ③ 確認手順

動作中の pcheck のプロセス情報取得時のリトライ回数は、padmin コマンドで確認することができます。

```
# /opt/HA/PS/bin/padmin -f <pfile 名> -c show env
HAPS_GETPROC_RETRY_COUNT = 10 ★
```

設定したプロセス情報取得時のリトライ回数が表示されます。

環境変数 HAPS\_GETPROC\_RETRY\_COUNT を設定していない場合は、デフォルトの 6 が表示されます。

### (2) 運用上の注意・制限事項

- ・すでに起動中の pcheck のプロセス情報取得時のリトライ回数を変更することはできません。一度 pcheck を padmin -c shutdown で終了させてから、環境変数を設定し pcheck を再起動してください。

## 4.14. プロセス個数監視時のリトライ回数を変更する手順

pcheck は、同名プロセス監視設定時にリトライ回数を設定して、設定した回数連続で異常となった場合に個数監視異常を検知します。

このとき、デフォルトでは下限監視リトライ回数は 2 回、上限監視リトライ回数は 1 回の情報取得のリトライを行います。環境変数を設定することで変更することができます。

通常は変更する必要はありません。

設定可能な範囲は 1 ~ 100 です。

下限監視リトライ回数を変更する手順は以下のとおりです。

### (1) リトライを最大 10 回行う場合

- ① 環境変数 HAPS\_MIN\_PROC\_CHECK\_RETRY\_COUNT に 10 を設定します。

・sh の場合

```
# HAPS_MIN_PROC_CHECK_RETRY_COUNT=10
# export HAPS_MIN_PROC_CHECK_RETRY_COUNT
```

・csh の場合

```
% setenv HAPS_MIN_PROC_CHECK_RETRY_COUNT 10
```

※ 上記のように環境変数を設定した状態で起動した pcheck については、最大 10 回の下限監視のリトライを行います。

- ② pcheck を起動し、プロセス監視を開始します。

```
# /opt/HAPS/bin/pcheck -f <pfile> &
```

- ③ 確認手順

動作中の pcheck の下限監視リトライ回数は、padmin コマンドで確認することができます。

```
# /opt/HAPS/bin/padmin -f <pfile 名> -c show env
HAPS_MIN_PROC_CHECK_RETRY_COUNT = 10 ★
```

設定した下限監視リトライ回数が表示されます。

環境変数 HAPS\_MIN\_PROC\_CHECK\_RETRY\_COUNT を設定していない場合は、デフォルトの 2 が表示されます。

### (2) 運用上の注意・制限事項

- ・すでに起動中の pcheck の下限監視リトライ回数を変更することはできません。一度 pcheck を padmin -c shutdown で終了させてから、環境変数を設定し pcheck を再起動してください。

上限監視リトライ回数を変更する手順は以下のとおりです。

(1) リトライを最大 10 回行う場合

- ① 環境変数 HAPS\_MAX\_PROC\_CHECK\_RETRY\_COUNT に 10 を設定します。

・sh の場合

```
# HAPS_MAX_PROC_CHECK_RETRY_COUNT=10
# export HAPS_MAX_PROC_CHECK_RETRY_COUNT
```

・csh の場合

```
% setenv HAPS_MAX_PROC_CHECK_RETRY_COUNT 10
```

※ 上記のように環境変数を設定した状態で起動した pcheck については、最大 10 回の上限監視のリトライを行います。

- ② pcheck を起動し、プロセス監視を開始します。

```
#/opt/HA/PS/bin/pcheck -f <pfile> &
```

- ③ 確認手順

動作中の pcheck の上限監視リトライ回数は、padmin コマンドで確認することができます。

```
#/opt/HA/PS/bin/padmin -f <pfile 名> -c show env
HAPS_MAX_PROC_CHECK_RETRY_COUNT = 10 ★
```

設定した上限監視リトライ回数が表示されます。

環境変数 HAPS\_MAX\_PROC\_CHECK\_RETRY\_COUNT を設定していない場合は、デフォルトの 1 が表示されます。

(2) 運用上の注意・制限事項

- すでに起動中の pcheck の上限監視リトライ回数を変更することはできません。一度 pcheck を padmin -c shutdown で終了させてから、環境変数を設定し pcheck を再起動してください。

## 4.15. Serviceguard とのクラスターウェア連携手順

Serviceguard を利用したクラスターシステムにおいて、本製品を導入する場合の手順について記述します。

パッケージ制御スクリプトにサービスコマンドとなるシェルスクリプトを登録し、そのサービスコマンドにおいて、対象プロセスおよび pcheck コマンドを起動するような設定を行うことによって、Serviceguard との連携を行います。

Serviceguard と連携するためには、各サーバーに本製品をインストールし SG ファイルや環境ファイルを登録する必要があります。

- (1) pcheck コマンドの設定ファイル(pfile)
- (2) サービスコマンド(service.sh)
- (3) Serviceguard のパッケージ制御スクリプト(pkg.sh)

以下に Serviceguard と連携するために設定が必要なファイルについて説明します。

### (1) pcheck コマンドの設定ファイル(pfile)の設定

pcheck コマンドの設定ファイルは、テンプレートを用意してあります。

Serviceguard と連携する場合、設定ファイルのプロセス監視のリトライオーバーアクションに exit を指定します。exit 以外を指定した場合、動作を保証しません。

対象となるプロセスが消滅し、設定条件(GRACE 値、リトライ回数)内で、再起動できない場合、pcheck コマンドを終了し、Serviceguard の設定に従ってパッケージ移動やノード切り替えの処理を行います。

設定ファイルの設定例を以下に記述します。

【設定例】

```
##### PARAM #####
IPCKEY                               0x10000001
MSG_CHECK_INTERVAL                   5
MONITOR_INTERVAL                     10
SHM_DUMP_FILE                        /var/opt/HA/PS/log/pcheck_dump
##### PENT #####
/usr/xxxx/sample:/var/opt/HA/PS/conf/src/sample1.sh:86400:3:exit
```

通常、クラスターパッケージに定義される oracle や AP サーバー製品は、多数の管理プロセスによって構成されますので、これらのプロセスを pcheck から監視する場合はグループ監視で実現します。

各種クラスターウェアとの連携時に、pcheck を階層的に使用したグループ監視を使用する場合は、親の pcheck のリトライオーバーアクションには exit、子の pcheck には shutdown を指定します。詳細については「2.3.2. pcheck を階層的に使用したグループ監視」を参照してください。

## (2) サービスコマンド(service.sh)の設定

サービスコマンドでは、対象プロセスを監視するために pcheck コマンドを起動します。  
このサービスコマンドはデフォルトでは用意されていませんので、運用環境に応じて作成する必要があります。作成したサービスコマンドを、Serviceguard のパッケージ制御スクリプトに登録し、Serviceguard からの起動および監視を行うことにより、本製品と Serviceguard との連携が実現できます。

対象プロセスが再起動できない場合、サービスコマンドから実行した pcheck コマンドが消滅を検出し、設定ファイルの exit 指定にしたがって終了します。  
pcheck コマンドの終了によってサービスコマンドが終了するため、サービスコマンドの終了を Serviceguard が検出し、パッケージ移動やノード切り替えなどの処理を実行します。

サービスコマンドには以下の動作を設定します。

- 対象プロセスの動作に必要な環境変数の設定
  - 対象プロセスの起動 (\*1)
  - pcheck の起動
- (\*1) … 対象プロセスの起動をパッケージ制御スクリプトで実行している場合は不要です。

サービスコマンドのシェルスクリプトを作成する場合、以下の点に注意してください。

- ・ pcheck コマンドは、監視対象プロセスを起動した後で実行してください。
- ・ 複数の pcheck コマンドを呼び出す場合は、バックグラウンドで起動し、最後の pcheck をフォアグラウンドで起動してください。
- ・ プロセス監視の対象となるプロセスを起動する場合は、pcheck コマンドが実行できるように、バックグラウンドなどで起動してください。

簡単なサービスコマンドの設定例を以下に記述します。

【設定例】

```
#!/bin/sh
# 対象プロセスの起動
/usr/xxx/sample &
# 対象プロセスが起動できるまで待ち合わせ
/bin/sleep 10
# プロセス監視を行う pcheck コマンドの起動
/opt/HA/PS/bin/pcheck -f /var/opt/HA/PS/conf/bin/pfile
```

サービスコマンドとして pcheck を定義すると、Serviceguard の仕様により環境変数がクリアされた状態で pcheck が呼び出されるため、プロセスの再開に失敗する場合があります。対象プロセスが PATH や LANG などの環境変数に依存している場合、サービスコマンドまたは再起動スクリプトにあらかじめ、必要な環境変数を設定しプロセスを再開してください。

たとえば、対象プロセスが相対パスを意識した構成の場合(相対パスでコマンドを実行したり、ファイルを参照するなど)は、サービスコマンドまたは再起動スクリプトに PATH の設定を追加してください。

**【設定例】**

```
# PATH の設定
if [ -r /etc/PATH ]
then
    PATH=`cat /etc/PATH`
    export PATH
fi
```

### (3) Serviceguard のパッケージ制御スクリプト(pkg.sh)の設定

Serviceguard において、サービスコマンドの起動および監視を行うために、Serviceguard のパッケージ制御スクリプト内の下記変数を設定します。

SERVICE_NAME	… サービス名
SERVICE_CMD	… サービスコマンドのパス名
SERVICE_RESTART	… 再起動回数

SERVICE\_CMD 変数に作成したサービスコマンドのパス名を記述します。

パッケージ制御スクリプトの設定方法の詳細については、Serviceguard のマニュアルを参照してください。

パッケージ制御スクリプトの変数の設定例を以下に記述します。

【設定例】

```
SERVICE_NAME[0]="service"  
SERVICE_CMD[0]="/home/xxxx/service.sh"  
SERVICE_RESTART[0]=""
```

パッケージの構成によっては論理ボリューム構成、仮想 IP 構成、起動および終了スクリプトなどの設定が必要です。

詳細は Serviceguard のマニュアルを参照してください。

また、クラスターの共有ディスク上に更新型のファイル(設定ファイル、ジャーナルなど)を配置している場合は、ファイルの整合性に十分注意してください。パッケージ移動契機でファイルが損傷している可能性があります。対象プロセスにおいてファイルのリカバリ(ロールバック)を実現する仕組みが必要です。



(4) 運用上の注意・制限事項

・Serviceguard と連携して、pcheck コマンドによるプロセス監視を実施している場合、pcheck コマンドを padmin コマンドの -c shutdown オプションおよび kill コマンドなどにより終了しないようにしてください。

pcheck コマンドの終了によって、パッケージ移動やノード切り替えを実行します。

・Serviceguard と連携してプロセス監視を実施している場合、プロセス監視の停止/再開/開始を行う場合は、padmin コマンドの -c start / stop / reload オプションを実行してください。

[プロセス監視を停止する場合]

```
# /opt/HA/PS/bin/padmin -f /var/opt/HA/PS/conf/bin/pfile -c stop
```

[プロセス監視を再開する場合]

```
# /opt/HA/PS/bin/padmin -f /var/opt/HA/PS/conf/bin/pfile -c reload
```

[プロセス監視を開始する場合]

```
# /opt/HA/PS/bin/padmin -f /var/opt/HA/PS/conf/bin/pfile -c start
```

・プロセス監視のリトライオーバーアクションに、continue または、shutdown を設定する場合は、Serviceguard と連携する設定は行わないようにしてください。

グループ監視の子の pcheck はこのかぎりではありません。

(5) クラスタ構成で oracle を監視する事例

- oracle を Serviceguard のパッケージとして定義する pcheck 監視の典型的な事例を紹介します。  
本事例では、pcheck はグループ監視構成(プロセス再起動なし)を採用していますが、監視方式や oracle の起動・終了手順によって設定が異なりますのでご注意願います。

<root pcheck>

```
# ProcessSaver SG file for root pcheck
# root_pcheck (pcheck)
##### PARAM #####
IPCKEY                0x10000001
MSG_CHECK_INTERVAL   5
MONITOR_INTERVAL     10
SHM_DUMP_FILE        /var/opt/HA/PS/log/dump_root

##### PENT #####
/opt/HA/PS/bin/pcheck_DB1-:0:0:exit
```

<child pcheck>

```
# ProcessSaver SG file for OracleDB
# child_pcheck: pcheck_DB1
##### PARAM #####
IPCKEY                0x10000002
MSG_CHECK_INTERVAL   5
MONITOR_INTERVAL     10
SHM_DUMP_FILE        /var/opt/HA/PS/log/dump_DB1

##### PENT #####
# instance #
ora_dbw0_jinji-:0:0:shutdown
ora_lgwr_jinji-:0:0:shutdown
ora_ckpt_jinji-:0:0:shutdown
ora_pmon_jinji-:0:0:shutdown
ora_smon_jinji-:0:0:shutdown
ora_reco_jinji-:0:0:shutdown

# listner #
/opt/app/oracle/product/8.1.6/bin/tnslsnr-:0:0:shutdown
```

<パッケージ起動スクリプト>

... 省略 ...

<サービス設定>

```
SERVICE_NAME[0]="serv1"  
SERVICE_CMD[0]="/etc/cmcluster/pkg1/service.sh"  
SERVICE_RESTART[0]=""
```

... 省略 ...

<起動コマンド>

```
function customer_defined_run_cmds  
{  
#ADD customer defined run commands.  
:# do nothing instruction, because a function must contain some command.  
    /sbin/init.d/dbora start  
    /bin/sleep 60  
    test_return 51  
}
```

<終了コマンド>

```
function customer_defined_halt_cmds  
{  
#ADD customer defined halt commands.  
:# do nothing instruction, because a function must contain some command.  
    /opt/HA/PS/bin/padmin -f /var/opt/HA/PS/conf/bin/pfile -c shutdown  
    /opt/HA/PS/bin/padmin -f /var/opt/HA/PS/conf/bin/pfile_DB1 -c shutdown  
    /bin/sleep 5  
  
    /sbin/init.d/dbora stop  
    /bin/sleep 10  
    test_return 52  
}
```

... 省略 ...

※ oracle のインスタンスが増加すると、起動に時間がかかる場合がありますので sleep の値を調整してください。

<service.sh>

```
#!/bin/sh

export SERVICENAME="$0"
export PS_PATH="/opt/HA/PS/bin"
export PFILE_PATH="/var/opt/HA/PS/conf/bin"

rval=0

set_return() {
    x=$?
    if [[ $x != 0 ]]; then
        /usr/bin/logger -p local3.error -t $SERVICENAME "ERROR : ¥$1¥
        " failed. exit with failure ($x)"
        rval=1
        exit $rval
    fi
}

Exec() {
    SrvCmd="$1"
    SrvArg="$2"
    CmdLst="$SrvCmd $SrvArg"
    echo "#####$rval# date ` $CmdLst"
    if [[ -x $SrvCmd ]]; then
        $SrvCmd $SrvArg
        set_return "$CmdLst"
    else
        /usr/bin/logger -p local3.error -t $SERVICENAME "ERROR : ¥
        "${SrvCmd}: Permission denied, or command not found.¥""
        exit 2
    fi
}

/bin/sleep 5
Exec "${PS_PATH}/pcheck_DB1" "-f ${PFILE_PATH}/pfile_DB1" &
/bin/sleep 5
Exec "${PS_PATH}/pcheck" "-f ${PFILE_PATH}/pfile"
```

## 5. 操作・運用手順

### 5.1. プロセスの状態監視について

プロセスの監視状態は/opt/HA/PS/bin/padmin コマンドで実行できます。

- (1) プロセス監視コマンド(pcheck) の一覧を表示する

```
# /opt/HA/PS/bin/padmin -l
```

```
ipckey   = 0x9192  
mypid    = 1335  
myname   = pcheck  
pfile    = /home/test/pm/pfile1  
message  = start
```

```
ipckey   = 0x1000  
mypid    = 1336  
myname   = pcheck  
pfile    = /home/test/pm/pfile2  
message  = stop
```

(2) プロセスの監視ルールを表示する

```
# /opt/HA/PS/bin/padmin -f <pfile> -c show param
MSG_CHECK_INTERVAL = 5
MONITOR_INTERVAL = 10
MONITOR_TRY_COUNT = 2
SHM_DUMP_FILE = /opt/HA/PS/test_pm
PFILE = pfile
MESSAGE_BOX = start
MONITOR_STOP_COUNT = 2
FAIL_PROC_COUNT = 0
ALL_PROC_COUNT = 4
```

```
# /opt/HA/PS/bin/padmin -p <pcheck-pid> -c show param
```

pfile 名の代わりに pcheck の pid を指定することも可能です。

MESSAGE\_BOX には、処理中のメッセージ(イベント)が表示されますが、以下の種類があります。

メッセージ	説明
start	プロセス監視の実行
resume	プロセス監視の実行 (startと同じ)
stop	プロセス監視の停止
suspend	プロセス監視の停止 (stopと同じ)
shutdown	プロセス監視の終了
change	一時的なpfile設定変更
dump	トレース情報などのファイル出力

MONITOR\_STOP\_COUNT には、監視停止中のプロセス数が表示されます。  
FAIL\_PROC\_COUNT には、リトライオーバーとなっているプロセス数が表示されます。  
ALL\_PROC\_COUNT には、監視中、停止中にかかわらず、すべての監視プロセス数が表示されます。

(3) プロセスの監視状態を表示する

```
# /opt/HA/PS/bin/padmin -f <file> -c show pent
pname                = /opt/HA/PS/test_pm
pid                  = 1883
retry_count          = 0
restart_count        = 0
proc_sts             = AVAIL
retry_over_act       = continue
rerun_time           = Thu May 20 10:56:48 1999
pent_id              = 1
monitor_sts          = on
```

proc\_sts には、プロセスの監視状態が表示されますが、以下の種類があります。

プロセス監視状態	説明
INIT	初期状態
RESTARTED	プロセス再開成功
AVAIL	プロセス正常動作中(通常状態)
RESTART_FAIL	プロセス再開失敗
UNAVAIL	プロセス異常検出
RETRY_OVER	プロセス再開リトライオーバー
RESTARTING	プロセス再開開始
UNKNOWN	状態不明

pent\_id は、pfile の pent の通番が、pcheck 起動時に pent の昇順に付与されます。  
ID は、1 ~ 256(pent の上限値)とし、pfile 単位で一意となります。

monitor\_sts には、以下のように pent 単位の監視状態が表示されます。

pent 監視状態	説明
on	監視状態
off	監視停止状態
unknown	状態不明

(4) ストール監視の状態を表示する(pfile でストール監視を指定した場合)

```
# /opt/HA/PS/bin/padmin -f <pfile> -c show pent
pname                = /opt/HA/PS/test_threadleak
pid                  = 26206
retry_count          = 0
restart_count        = 0
proc_sts             = AVAIL
stall_chk (xxx)     = AVAIL
retry_over_act       = continue
rerun_time           = -----
pent_id              = 1
monitor_sts          = on
```

stall\_chk (xxx) のカッコ内の xxx は pfile のオプション部に指定したストール監視の種別により表示が異なります。

xxx とライブラリ種別は、以下の対応となっています。

xxx	ライブラリ種別	説明
nor	_bi_stall	ファイルの更新時刻によるストール監視
msg	_bi_stall_message	ファイルの出力メッセージによるストール監視
opf	_bi_stall_openfile	対象プロセスのオープンファイルによるストール監視
thd	_bi_stall_thread	対象プロセスの起動スレッドによるストール監視
gen	_bi_stall_general	対象プロセスのプロセス状態によるストール監視

stall\_chk (xxx) には、ストール監視の状態が表示されますが、以下の種類があります。

状態表示値	意味	説明
INIT	初期状態	初期状態
AVAIL	監視中	プロセスは正常動作している ユーザーが定義したストール状態ではない
DETECT	ストール検出	プロセスストールを検出した ユーザーが定義したストール状態となった
SUSPECT *1	ストール推定	プロセスストールと推定される 定義したストール状態ではないもののプロセスの動きがない
UNAVAIL	監視失敗	内部エラーの発生により監視が失敗した
UNKNOWN	状態不明	上記のどれとも一致しない状態

\*1 種別 gen (対象プロセスのプロセス状態によるストール監視)でのみ表示されます。



(5) グループの監視状態の一覧を表示する(pfile でグループ指定した場合)

```
# /opt/HA/PS/bin/padmin -f <pfile> -c show pent
```

```

pname           = /opt/HA/PS/test_pm
pid             = 1883
retry_count     = 0
restart_count   = 0
proc_sts       = AVAIL
retry_over_act  = continue
rerun_time     = Thu May 20 10:56:48 1999
group_name     = group01
group_sts      = AVAIL
pent_id        = 1
monitor_sts    = on

```

```
# /opt/HA/PS/bin/padmin -f <pfile> -c show group
```

```

group_name      = GROUP1
group_sts      = AVAIL
restart_time    = Thu May 20 10:56:48 1999
monitor_sts    = on

```

pent\_id は、pfile の pent の通番が、pcheck 起動時に pent の昇順に付与されます。  
ID は、1 ~ 256(pent の上限値)とし、pfile 単位で一意となります。

group\_sts には、プロセスの監視状態が表示されますが、以下の種類があります。

グループ監視状態	説明
INIT	初期状態
RESTARTED	プロセス再開成功
AVAIL	プロセス正常動作中(通常状態)
RESTART_FAIL	プロセス再開失敗
UNAVAIL	プロセス異常検出
RETRY_OVER	プロセス再開リトライオーバー
RESTARTING	プロセス再開開始
UNKNOWN	状態不明

monitor\_sts には、以下のように pent 単位の監視状態が表示されます。

pent 監視状態	説明
on	監視状態
off	監視停止状態
unknown	状態不明

※ show group で表示される monitor\_sts は、grouptag 指定されたグループの中で、監視状態のプロセスが1つでも存在する場合 on となります。

(6) プロセスの再起動が発生すると

対象プロセスの消滅を検出し再起動が発生すると、`retry_count`、`restart_count` の値が加算されますが、次の再起動契機まで元に戻ることはありません。  
運用には支障はありませんが、`reload` を実行することで、初期値に戻すことができます。

`retry_count` : プロセスを再起動した回数です。

GRACE 値の範囲で再起動できれば、次回消滅を検出した時点でリセットされます。

`restart_count` : プロセスを再起動した累計です。

`pcheck` が終了するまでリセットされることはありません。

```
# /opt/HA/PS/bin/padmin -f <pfile> -c show pent
```

```
pname           = /opt/HA/PS/test_pm
pid             = 1883
retry_count     = 1
restart_count   = 1
proc_sts       = AVAIL
retry_over_act  = continue
rerun_time     = Thu May 20 10:56:48 1999
pent_id        = 1
monitor_sts    = on
```

`reload` でファイルを再読み込みすることで `retry_count`、`restart_count` の値をクリアできます。

```
# /opt/HA/PS/bin/padmin -f <pfile> -c reload
```

(7) 機能制限中の状態表示について

コードワードが未登録の状態では本製品をインストールした場合、インストールから 30 日を経過した後に ProcessSaver の機能が制限され、障害を検知なくなります。  
機能制限の詳細については、「5.5. 機能制限について」を参照してください。

④ プロセス監視コマンド(pcheck) の一覧表示

pcheck の一覧を表示した場合、機能制限中の pcheck は最終行に機能制限中を示す「restricted\_mode = on」を表示します。  
restricted\_mode は、機能制限中以外は表示されません。

```
# /opt/HA/PS/bin/padmin -l
ipckey      = 0x4f009192
mypid       = 1335
myname      = pcheck
pfile       = /var/opt/HA/PS/conf/bin/pfile1
message     = start

ipckey = 0x4f001000
mypid   = 1336
myname  = pcheck
pfile   = /var/opt/HA/PS/conf/bin/pfile2
message = start
restricted_mode = on
```

restricted\_mode には、以下のように pcheck 単位の機能制限の状態が表示されます。

機能制限の状態	説明
on	機能制限中

⑤ プロセスの監視ルールの一覧表示

機能制限中にプロセスの監視ルールの一覧を表示した場合、最終行に機能制限中を示すメッセージ「Monitoring stop until activation succeeded.」を表示します。  
各値は機能制限中のため、実際の状態と異なる場合があります。

```
# /opt/HA/PS/bin/padmin -f <pfile> -c show param
MSG_CHECK_INTERVAL      = 5
MONITOR_INTERVAL        = 10
MONITOR_TRY_COUNT       = 2
SHM_DUMP_FILE           = /var/opt/HA/PS/log/pcheck_dump
PFILE                    = pfile1
MESSAGE_BOX              = start
MONITOR_STOP_COUNT      = 2
FAIL_PROC_COUNT          = 0
ALL_PROC_COUNT           = 4
```

**Monitoring stop until activation succeeded.**

⑥ プロセスの監視状態の一覧表示

機能制限中にプロセスの監視状態の一覧を表示した場合、最終行に機能制限中を示すメッセージ「Monitoring stop until activation succeeded.」を表示します。

各値は機能制限中のため、実際の状態と異なる場合があります。

```
# /opt/HA/PS/bin/padmin -f <pfile> -c show pent
```

```
  pname          = /opt/HA/PS/test_pm
  pid            = 1883
  retry_count    = 0
  restart_count  = 0
  proc_sts       = AVAIL
  retry_over_act = continue
  rerun_time     = Thu May 20 10:56:48 1999
  pent_id        = 1
  monitor_sts    = on
```

**Monitoring stop until activation succeeded.**

⑦ グループの監視状態の一覧表示

機能制限中にグループの監視状態の一覧を表示した場合、最終行に機能制限中を示すメッセージ「Monitoring stop until activation succeeded.」を表示します。

各値は機能制限中のため、実際の状態と異なる場合があります。

```
# /opt/HA/PS/bin/padmin -f <pfile> -c show group
```

```
  group_name     = GROUP1
  group_sts      = AVAIL
  restart_time   = Thu May 20 10:56:48 1999
  monitor_sts    = on
```

**Monitoring stop until activation succeeded.**

(8) 監視対象プロセスの状態チェックについて

監視対象プロセスの状態チェック確認コマンドは、OpenShift 環境でコンテナのヘルスチェックを行う場合に有効です。

OpenShift 環境での監視対象の状態確認コマンドの活用方法については、別紙『CLUSTERPRO MC ProcessSaver for Linux OpenShift 連携ガイド』を参照してください。

① 監視対象プロセスの消滅監視状態の確認

指定された監視定義での監視対象プロセスの消滅監視状態を表示します。

また監視状態を終了ステータスで返却します。

```
# /opt/HA/PS/bin/padmin -f <pfile> -c check alive
```

(監視対象プロセス正常動作時の出力例)

```
2021-3-30 11:32:55 Alive Check OK (0) [<pfile 名>]
```

(監視対象プロセス正常動作時の出力例)

```
2021-3-30 11:32:55 Alive Check NG (1) [<pfile 名>]
```

監視対象プロセスが正常に動作しているときは Alive Check OK (0) を表示します。

監視対象プロセスが正常に動作していないときは Alive Check NG (状態コード 0 以外) を表示します。

() 内に表示される状態コードは、以下の種類があります。

状態コード	説明
0	プロセスは正常動作中(通常状態)
1	コマンド/パラメーターのエラー またはpcheckが起動していない
103	プロセスの再開失敗
104	プロセスは存在しない(停止状態)
105	内部エラーが発生した
106	プロセスの再開断念(リトライオーバー)
107	プロセスの再開中

状態コードは、複数プロセスを監視中の場合は、すべての監視対象が通常動作中であれば 0 を表示します。

複数プロセスを監視中で、通常動作ではない対象が 1 つでもあった場合は最初の問題プロセスの状況を状態コードで表示します。

表示される状態コードは、本コマンドの終了ステータスと同じ値となります。

- ② 監視対象プロセスのストール監視状態の確認  
 指定された監視定義でのストール監視状態を表示します。  
 また監視状態を終了ステータスで返却します。

```
# /opt/HA/PS/bin/padmin -f <pfile> -c check stall
```

(監視対象プロセス正常動作時の出力例)  
 2021-3-30 11:32:55 Stall Check OK (0) [<pfile 名>]

(監視対象プロセス正常動作時の出力例)  
 2021-3-30 11:32:55 Stall Check NG (1) [<pfile 名>]

監視対象プロセスが正常に動作しているときは Stall Check OK (0) を表示します。  
 監視対象プロセスが正常に動作していないときは Stall Check NG (状態コード 0 以外) を表示します。

- () 内に表示される状態コードは、以下の種類があります。

状態コード	説明
0	プロセスは準備できている(通常状態)
1	コマンドパラメーターのエラー pcheckが起動していない またはストール監視していない
102	プロセスストールを検出した (ユーザーが定義したストール状態となった)
103	プロセスストールと推定される (ユーザーが定義したストール状態ではないがプロセスの動きがない)
104	内部エラーが発生した

状態コードは、複数プロセスを監視中の場合は、すべての監視対象が通常動作中であれば 0 を表示します。

また複数プロセスを監視中、通常動作ではない対象が 1 つでもあった場合は最初の問題プロセスの状況を状態コードで表示します。

表示される状態コードは、本コマンドの終了ステータスと同じ値となります。

## 5.2. プロセスの運用管理について

- (1) プロセス監視の停止

```
# /opt/HA/PS/bin/padmin -f <pfile> -c stop
```

- (2) 停止中のプロセス監視の再開

```
# /opt/HA/PS/bin/padmin -f <pfile> -c start
```

- (3) プロセス監視の終了

```
# /opt/HA/PS/bin/padmin -f <pfile> -c shutdown
```

- (4) タイマー値の一時的な変更

監視タイマーの値を一時的に変更できます。

```
# /opt/HA/PS/bin/padmin -f <pfile> -c change XXX val
```

- (5) pfile の再読込

reload 機能を使うと、pcheck を終了せずに変更した pfile の情報を反映できます。

```
# /opt/HA/PS/bin/padmin -f <pfile> -c reload
```

(注) pfile の ipckey を変更した場合は、reload 機能で情報を反映することはできません。  
pcheck を一度終了してから再起動を行ってください。

- (6) リトライオーバー時におけるプロセス監視の再開

リトライオーバーアクションに continue を指定した環境で、対象プロセスのリカバリーに失敗すると pcheck プロセスは常駐したまま当該プロセスを監視対象から外します。

監視対象プロセス復旧後に、reload 機能を使うと pcheck を終了することなく監視の再開を行うことができます。

```
# /opt/HA/PS/bin/padmin -f <pfile> -c reload
```

- (7) 情報をクリアしてプロセス監視を再開

```
# /opt/HA/PS/bin/padmin -f <pfile> -c restart
```

(8) プロセス監視中におけるメンテナンス作業手順

pcheck を終了することなく対象プロセスのメンテナンス、再起動を行う場合には、restart、reload 機能が有効です。

プロセス監視の一時停止

```
# /opt/HA/PS/bin/padmin -f <pfile> -c stop
```

<実行形式の変更によるプロセスの再起動などのメンテナンス作業実施>

プロセス監視の再開

```
# /opt/HA/PS/bin/padmin -f <pfile> -c restart
```

または、

① プロセス監視を初期化し再開

```
# /opt/HA/PS/bin/padmin -f <pfile> -c reload
```

② プロセス監視の再開

```
# /opt/HA/PS/bin/padmin -f <pfile> -c start
```

プロセス監視の再開は、restart オプションを使用することを推奨します。

ただし、pfile を変更した場合には、reload を使用して pfile の再読み込みが必要となります。

プロセス監視の状態確認

```
# /opt/HA/PS/bin/padmin -f <pfile> -c show pent
```

(9) 監視対象プロセスの起動状況を確認

pent 部に設定されている監視対象プロセスの起動状況を確認することが可能です。

```
# /opt/HA/PS/bin/padmin -f <pfile> -c show proc
```

```
UP          = /opt/HA/PS/test_a
```

```
UP          = /opt/HA/PS/test_b
```

```
DOWN       = /opt/HA/PS/test_c
```

監視対象プロセスが起動している場合は "UP"、未起動の場合は "DOWN" が表示されます。

また、監視対象プロセスがすべて起動している場合は 0、1 つでも起動していない場合には 1 を返却しますので、戻り値によっても対象プロセスの起動状況を確認することが可能です。

機能制限中は最終行に機能制限中を示すメッセージ「Monitoring stop until activation succeeded.」が表示されます。

機能制限の詳細については、「5.5. 機能制限について」を参照してください。



(10) pcheck 実行時の環境変数を確認

pcheck 起動時に使用されている環境変数を確認することが可能です。

```
# /opt/HA/PS/bin/padmin -f <pfile> -c show env
HAPS_CHILDPROC_KILL           = off
HAPS_SILENT_MODE              = off
HAPS_PENDING_TIME             = 60
HAPS_GETPROC_COUNT            = 1000
HAPS_GETPROC_RETRY_COUNT      = 6
HAPS_MIN_PROC_CHECK_RETRY_COUNT = 2
HAPS_MAX_PROC_CHECK_RETRY_COUNT = 1
HAPS_NPROC_CHECK_NOPPROC      = 0
```

環境変数と動作中のモードおよび設定値が以下のように表示されます。

- HAPS\_CHILDPROC\_KILL = off 子プロセス終了モード OFF(デフォルト)
- HAPS\_CHILDPROC\_KILL = on 子プロセス終了モード ON
  
- HAPS\_SILENT\_MODE = off サイレントモード OFF(デフォルト)
- HAPS\_SILENT\_MODE = on サイレントモード ON
  
- HAPS\_PENDING\_TIME = xx pcheck 自動待ち合わせ時間(デフォルト=60)
  
- HAPS\_GETPROC\_COUNT = xx 監視対象選択時のプロセス情報取得量  
(デフォルト=1000)
  
- HAPS\_GETPROC\_RETRY\_COUNT = xx 監視対象選択時のリトライ回数  
(デフォルト=6)
  
- HAPS\_MIN\_PROC\_CHECK\_RETRY\_COUNT = xx 下限監視時のリトライ回数  
(デフォルト=2)
  
- HAPS\_MAX\_PROC\_CHECK\_RETRY\_COUNT = xx 上限監視時のリトライ回数  
(デフォルト=1)
  
- HAPS\_NPROC\_CHECK\_NOPPROC = 0 監視対象プロセス自身の消滅を障害条件  
に含める(デフォルト)
  
- HAPS\_NPROC\_CHECK\_NOPPROC = 1 監視対象プロセス自身の消滅を障害条件  
から除外

環境変数 HAPS\_CHILDPROC\_KILL、HAPS\_SILENT\_MODE に 1 が設定されている場合は on が、未設定時および 1 以外が設定されている場合は off が表示されます。

環境変数 HAPS\_PENDING\_TIME を設定している場合は、設定された値が表示されます。

また未設定時は、デフォルトの 60 が表示されます。

環境変数 HAPS\_GETPROC\_COUNT を設定している場合は、設定された値が表示されます。

また未設定時は、デフォルトの 1000 が表示されます。

環境変数 HAPS\_GETPROC\_RETRY\_COUNT を設定している場合は、設定された値が

表示されます。

また未設定時は、デフォルトの 6 が表示されます。

環境変数 HAPS\_MIN\_PROC\_CHECK\_RETRY\_COUNT を設定している場合は、設定された値が表示されます。

また未設定時は、デフォルトの 2 が表示されます。

環境変数 HAPS\_MAX\_PROC\_CHECK\_RETRY\_COUNT を設定している場合は、設定された値が表示されます。

また未設定時は、デフォルトの 1 が表示されます。

環境変数 HAPS\_NPROC\_CHECK\_NOPPROC を設定している場合は、設定された値が表示されます。

また未設定時は、デフォルトの 0 が表示されます。

各環境変数の設定手順および詳細については、「4.8. サイレントモードでの運用手順」および、「4.9. pcheck 終了時に子プロセスを回収する手順」、「4.10. pcheck 起動時の自動待ち合わせ時間を変更する手順」、「4.12. 監視対象選択時のプロセス情報取得量を変更する手順」、「4.13. 監視対象選択時のリトライ回数を変更する手順」、「4.14. プロセス個数監視時のリトライ回数を変更する手順」を参照してください。

なお、通常は変更する必要ありません。

機能制限中は最終行に機能制限中を示すメッセージ「Monitoring stop until activation succeeded.」が表示されます。

機能制限の詳細については、「5.5. 機能制限について」を参照してください。

## 5.3. 障害解析手順

(1) プロセス監視における異常通知について

pcheck は、プロセス監視中に致命的な異常を検出するとシステムログファイル (/var/log/messages) にエラーメッセージを出力します。

システムログメッセージの説明と対処方法については、

別紙『CLUSTERPRO MC ProcessSaver for Linux syslog メッセージ一覧』を参照してください。

(2) 共有メモリトレースファイルについて

pcheck は、共有メモリ上にプロセス監視の履歴を常時取得していますが、プロセス監視で異常を検出した場合の解析手段として有効です。

このトレースファイルは約 3Mbyte 程度のサイクリックログですが、

pcheck の終了時、reload コマンド実行時、リトライオーバー時を契機として pfile の SHM\_DUMP\_FILE に指定した場所へ出力します。

このファイルは、2 世代までバックアップされます。

なお、以下のコマンドで強制的に共有メモリトレースをファイルへ出力できます。

```
# /opt/HA/PS/bin/padmin -f <pfile> -c dump filename
```

(3) 共有メモリトレースファイルの参照手順について

本ファイルは padmin で参照できます。

```
# /opt/HA/PS/bin/padmin -f <pfile> -c unload filename
```

また、strings コマンドでもメッセージのみ参照できます。

```
# strings filename
```

(4) ストール監視のダンプファイルについて

対象プロセスのプロセス状態によるストール監視ではストール検出時の情報をログファイルへ出力しています。ログファイルは pfile の SHM\_DUMP\_FILE に指定した名前に "\_stall\_" と Pent\_ID (1~256) をつなげた名前になります。

## 5.4. デバッグ支援機能

pfile や再起動スクリプトの妥当性を確認するために、デバッグ支援機能があります。

### (1) デバッグ手順について

pcheck をトレースオプション付き(-t)で起動すると、標準出力にプロセス監視の実行履歴が表示されます。このとき、端末画面を占有しますので注意が必要です。  
他の端末画面から、padmin コマンドでの操作や、kill コマンドなどで対象プロセスを終了することで、pcheck の動作が確認できます。

本機能は、pfile をデバッグするには非常に有効です。

トレースモードでプロセス監視を起動(バックグラウンド起動である & は付与しません)  
# /opt/HA/PS/bin/pcheck -f <pfile> -t

### (2) プロセス監視の一時停止機能について

特定のファイルを生成することで、pcheck 起動時からプロセス監視の一時停止が可能です。  
対象プロセスが正常に動作しない場合でも pcheck を起動することができます。

pcheck 起動前に、一時ファイルを作成します。

```
# touch /opt/HA/PS/bin/pcheck.ignore
```

pcheck を起動すると、システムログに監視停止中のメッセージが表示されます。

```
Jun  8 16:23:45 node1 pcheck: pcheck wait... by /opt/HA/PS/bin/pcheck.ignore
```

一時ファイルを削除すると、監視を再開します。

```
# rm /opt/HA/PS/bin/pcheck.ignore
```

pcheck が監視を開始したメッセージがシステムログに表示されます。

```
Jun  8 16:23:56 node1 pcheck: pcheck start by /opt/HA/PS/bin/pcheck.ignore
```

(注) pcheck 起動後にプロセス監視の一時停止を行うには、padmin で stop を指定してください。

## 5.5. 機能制限について

CLUSTERPRO MC ProcessSaver 2.3 for Linux までは、インストール時にライセンスをチェックするため、インストール前にコードワードの登録が必要でした。

CLUSTERPRO MC ProcessSaver 2.4 for Linux より、コードワードが未登録でもインストールが可能となりました。

なお、インストールから 30 日を経過してもコードワードが未登録の場合には ProcessSaver の機能に制限がかかり、障害発生時でも障害を検知なくなります。

コードワードが未登録の場合、以下の契機で syslog にメッセージが出力されます。

1. pcheck の起動時
2. 毎日 10:00 に到達したタイミング (1 日 1 回)

出力されるメッセージは、次のとおりです。

- ・ インストールから 30 日以内  
**After YYYYMMDD, monitoring function is stopped.**  
説明: ライセンス認証に失敗しました。  
YYYYMMDD で指定された年月日までは通常どおり使用できますが、経過後は ProcessSaver の機能が制限されます。  
機能制限中は障害が検知されません。
- ・ インストールから 30 日経過後  
**Monitoring stop until activation succeeded.**  
説明: ライセンス認証に失敗しました。  
正しいコードワードの登録が確認できるまで ProcessSaver の機能が制限されています。  
機能制限中は障害が検知されません。

これらのメッセージが表示された場合は、速やかにコードワードを登録してください。

コードワードの登録手順は、『CLUSTERPRO MC ProcessSaver 2.9 for Linux リリースメモ』の「1.2.2. ソフトウェアパッケージのインストール後にコードワードを登録する方法」を参照してください。

## 6. 注意・制限事項

### (1) pcheck

- ・ ひとつの pcheck コマンドで監視できるプロセスの最大値は 256 です。  
これ以上のプロセスを監視する場合は、pcheck を複数起動してください。  
同時に起動できるプロセス数には、システム設定値の制約があります。
- ・ pfile に指定する MONITOR\_INTERVAL (プロセス監視間隔)は、以下の制約があります。
  - a) MSG\_CHECK\_INTERVAL (メッセージ処理間隔) 以上の値であること
  - b) MSG\_CHECK\_INTERVAL の正の整数倍の値であること正の整数倍の値が設定されていない場合、pcheck コマンドは、正の整数倍の値に変換して実行します。
- ・ 同名のプロセスが複数存在し、そのプロセスを pcheck コマンドで監視する場合、そのプロセスに親子関係がある場合は、大元の親プロセスを監視します。  
親子関係がなければ、最初にマッチしたプロセスを監視対象とします。  
起動時刻が同じ場合、最も PID の小さいものを監視対象とします。  
また、特殊な設定を行うと、uid や起動数、引数による識別も可能です。
- ・ pcheck に親子関係を持たせる場合で、親と子の両方とも、"pcheck" という名前の場合、正常に監視できない場合があります。  
グループ監視などで、プロセス監視コマンド名と監視対象プロセス名が同一になる場合、子の pcheck のプロセス名を変更してください。(グループ監視の導入手順を参照)
- ・ pfile に指定する IPCKEY には、サーバー内で一意の値を指定してください。  
OS や他の製品の使用する IPCKEY と重複すると動作を保証できません。
- ・ 同一プロセス名監視で min\_proc\_count オプションを指定し、起動するプロセス数で監視を行った場合、監視対象プロセスを含め、同一名プロセスが指定した数を下回るとプロセス Down を検出します。  
ただし、監視対象プロセス自身が消滅した場合は、プロセス数が min\_proc\_count 以上であっても Down となります。  
監視対象プロセスが消滅した場合にプロセス Down を検出たくない場合は、「4.2.1. プロセス個数監視時に監視対象プロセス自身の消滅を障害条件から除外する手順」を参照してください。
- ・ 同一プロセス名監視で max\_proc\_count をオプション指定し、起動するプロセス数で監視を行った場合、監視対象プロセスを含め、同一名プロセスが指定した数を上回るとプロセス Down を検出します。  
ただし、監視対象プロセス自身が消滅した場合は、プロセス数が max\_proc\_count 以下であっても Down となります。  
監視対象プロセスが消滅した場合にプロセス Down を検出たくない場合は、「4.2.1. プロセス個数監視時に監視対象プロセス自身の消滅を障害条件から除外する手順」を参照してください。

- 同一プロセス名監視で `max_proc_count` オプションを指定する場合、監視タイミングごとに OS の全プロセス情報を検索してプロセス数を解析するため、システムで稼働するプロセス数が 10000 を越えるような大規模環境では、高負荷状態となる可能性があります。このような場合は、必要に応じて全プロセス情報取得処理で一度に採取するプロセス数を環境にあわせて、大きな値に変更してください。  
 変更手順の詳細については、「4.12. 監視対象選択時のプロセス情報取得量を変更する手順」を参照してください。
- 同一プロセス名監視で `min_proc_count` オプションと `max_proc_count` オプションを同時に指定する場合、`max_proc_count` オプションには、`min_proc_count` オプションの設定値以上の値を設定してください。
- グループ監視で `grouptag` オプションを指定した場合、システムログに実際に消滅したプロセスの Down メッセージは出力されますが、同一グループの他のプロセスの再起動に伴って、再起動されたプロセスについては、Down のメッセージは出力されません (再起動完了後の Up メッセージは出力されます)。  
 また再起動回数(`restart_count`)および再起動回数(`retry_count`)も同様に加算されます。
- グループ監視で `grouptag` オプションを指定する場合、同一グループで `retry_over_action`、`restart_count` 値、`grace` 値などの各設定値は同じ値を指定してください。
- グループ監視で `grouptag` オプションを指定する場合、数字のみの `grouptag` 名は指定できません。英字のみまたは英数字を組み合わせた方式で指定してください。  
 (例)  
 指定可 : `grouptag=oracle_group` や `grouptag=group01`  
 指定不可 : `grouptag=1` や `grouptag=8` のように数字のみの `grouptag` 名
- グループ監視で `grouptag` オプションと `clear_cmd` オプションを同時に指定する場合、`clear_cmd` オプションは同一グループで 1 つのみ指定してください。  
 複数記載している場合は、リトライオーバー発生時にグループ指定されているプロセスについて、すべての `clear_cmd` が実行されます。
- `restart_waittime` および `restart_timeout` オプションを指定する場合、`monitor_interval` の正の整数倍の値を指定してください。  
 正の整数倍でない値を設定した場合、`monitor_interval` の値に切り上げられます。
- `pfile` のオプション情報に、各オプションを“,” で区切って複数指定する場合、オプションは最大で 20 個まで同時に指定できます。また、指定できる文字数はオプション情報全体で最大 255 文字以内です。  
 また、オプション情報に、“=” を指定することはできません。
- 再起動スクリプトを;(セミコロン)区切りで複数指定した場合、`/bin/sh -c` の引数として指定されたスクリプトを実行します。  
 そのため、`/bin/sh -c` の引数として指定することのできないスクリプトやコマンドについては指定することができません。  
 また、記載された構文のまま実行されますので、シェルの構文に従った方式で指定する必要があります。
- 再起動スクリプトを;(セミコロン)区切りで複数指定した場合には、最後に指定されたスクリプトの終了ステータスでプロセスの起動・再起動に失敗したかどうかを判定します。

- ・ コードワードが未登録の状態でも、インストールから 30 日までは本製品のすべての機能を通常どおり使用できます。  
 なお、インストールから 30 日を経過してもコードワードが未登録の場合には本製品の機能に制限がかかり、障害発生時でも障害を検知しなくなります。  
 詳細は「5.5. 機能制限について」を参照してください。

## (2) padmin

- ・ padmin コマンドは、pcheck コマンドを起動中でない場合は使用できません。  
 また、pcheck コマンドを kill コマンドなどで強制終了させた後に padmin を実行させた場合、異常終了します。
- ・ padmin コマンドからの処理要求は、pcheck で管理する共有メモリ上にメッセージを設定することにより動作しますので、共有メモリ上のメッセージ処理間隔(MSG\_CHECK\_INTERVAL)の設定値によって、実行に時間差が発生します。  
 特に、padmin コマンドをシェルスクリプトから連続して呼び出す場合は、一定時間の猶予(MSG\_CHECK\_INTERVAL の約 2 倍程度)を sleep で与えてください。
- ・ padmin コマンドの実行に失敗した場合、エラーメッセージは共有メモリ上のトレースまたはシステムログファイルに出力され、標準エラー出力に出力されない場合があります。
- ・ padmin コマンドを一般ユーザーで使用する場合には、pcheck コマンドも同様に一般ユーザーで起動してください。  
 pcheck を root ユーザー、padmin を一般ユーザーで実行すると失敗する場合があります。
- ・ padmin コマンドで監視の停止、再開、再起動(stop、start、restart)を実行した場合、システムログに以下のようなメッセージを出力します。  
 監視の停止時  

```
Jun 14 09:58:59 host1 pcheck[23328]:(['対象プロセス名', pent_id='xxx']) Monitor stop
```

 監視の再開、再起動時  

```
Jun 14 09:58:59 host1 pcheck[23328]:(['対象プロセス名', pent_id='xxx']) Monitor start
```
- ・ padmin コマンドを restart オプションで実行時に、該当のプロセスが起動していない場合は指定されたプロセスの監視は再開されません。  
 その場合、システムログに以下のようなメッセージを出力します。  

```
Jun 14 09:58:59 host1 pcheck[23328]: Set pid fail ('対象プロセス名')
```

```
Jun 14 09:58:59 host1 pcheck[23328]:(['対象プロセス名', pent_id='xxx']) Monitor stop
```
- ・ padmin コマンドを restart オプションで実行時には、pfile の再読み込みは行いません。  
 pfile の変更後に pfile の再読み込みを行う場合は、従来どおり reload オプションを使用してください。
- ・ padmin コマンドは、pcheck の動作を予約するコマンドのため、複数同時に実行することはできません。
- ・ pcheck 実行中に ipckey を変更した場合は、reload によって動的に変更することはできません。  
 pcheck を一度終了し再起動を行う必要があります。



### (3) システム構成における注意事項

- ・ ストール監視機能をご利用になる場合は、あらかじめ導入手順を十分確認してください。導入手順の詳細については、「4.6. ストール監視の導入手順」を参照してください。
- ・ pcheck から pcheck 自身のストール監視はサポートしていません。
- ・ マルチスレッド構成のプロセスにおいて、プロセス内部のスレッド単位で監視を行うことはできません。
- ・ 同一名プロセスを別々の pcheck から個々に監視を行うことはできません。
- ・ スワップアウトされているプロセスについては、次回スワップインされるまで監視を行うことはできません。
- ・ 各監視対象の製品が提供している起動・再起動コマンドやスクリプトは、終了できなかったプロセスを強制終了させるために、SIGKILL などを送信して終了させる仕様のものがあります。再起動スクリプト名や pcheck 名などに監視対象プロセス名自体が含まれるような形で作成していると強制終了の対象と誤認されシグナルが送信されて、再起動スクリプトが異常終了する場合があります。pcheck 名や再起動スクリプト名は監視対象プロセス名が入らないよう作成することを推奨します。

### (4) rc スクリプトの制限事項

- ・ 各ディストリビューションに付属する各種の「サービス管理ツール」(GUI)からサービス起動の操作(起動、終了、再起動)には対応していません。
- ・ コマンドラインから rc スクリプトを実行する場合、/etc/init.d に移動して rc スクリプトを実行すると、/etc/init.d 配下に nohup.out ファイルが作成され、ランレベルエディターの表示が不正になる場合がありますので、rc スクリプトは /etc/init.d 配下で実行しないでください。
- ・ pcheck の終了処理は、必ず監視対象プロセスの終了処理より前に実行する必要があります。このため rc スクリプトの pcheck 終了時のリンクファイルは、最初に実行する K01 で始まる文字で指定することを推奨します。

### (5) systemctl の制限事項

- ・ pcheck を systemd に登録している場合、pcheck の起動、終了は systemctl コマンドを使用して実施してください。

## 7. リファレンス

### 名称

padmin - プロセス監視運用管理コマンド

### 構文

```
padmin -f pfilename -c option
padmin -p pcheck-pid -c option
padmin -l
```

### 機能説明

プロセス監視の停止/再開や動的パラメーターの変更を行います。

-l

実行中の pcheck の一覧を表示します。

-p *pcheck-pid*

pcheck の pid を指定します。

-f *pfilename*

プロセス監視の停止/再開や動的パラメーターを変更する場合、その設定を行っている pfile 名を指定します。

pfile 名は 256 文字未満で指定してください。

-c *option*

*option* には実行する動作を指定します。*option* は、あわせて 1024 文字未満で指定してください。*option* に指定する動作の規定は以下のとおりです。

start | resume

プロセス監視の再開を指定します。

stop | suspend

プロセス監視の停止を指定します。

shutdown

プロセス監視の終了を指定します。

reload

pfile の動的変更を指定します。

pcheck を終了させることなく、変更した pfile の再読み込みや、フェールオーバー時のプロセス再開を実現できます。

restart

プロセス情報を再読み込み後に、プロセス監視の再開を指定します。

dump [*dumpfile*]

pcheck が管理する共有メモリ情報のファイル出力を指定します。

*dumpfile* には、出力ファイル名を指定します。

*dumpfile* 省略時には、pfile の SHM\_DUMP\_FILE で設定しているファイルに共有メモリ情報を出力します。

unload *dumpfilename*

*dumpfilename* ファイル情報の画面表示を指定します。*dumpfilename* には、共有メモリ情報を出力したファイル名を指定してください。

change *msg\_check\_interval*|*monitor\_interval* *l\_val*

共有メモリのメッセージチェック間隔のタイマー値、または、プロセス監視間隔のタイマー値の変更を指定します。  
 l\_val には、変更後のタイマー値を 10 進数で指定します。  
 msg\_check\_interval の指定値は 1 秒～60\*60\*24 秒 (24 時間)の範囲です。

monitor\_interval の指定値は 1 秒～60\*60\*24 秒(24 時間)の範囲です。  
 msg\_check\_interval 以上の値で、msg\_check\_interval の正の整数倍の値を設定してください。

authlicense

現在登録されているコードワードを確認してライセンスを反映します。

show param

以下のようにプロセス監視の pfile 設定値を表示します。

```
MSG_CHECK_INTERVAL = 5
MONITOR_INTERVAL   = 10
MONITOR_TRY_COUNT  = 2
SHM_DUMP_FILE      = /opt/HA/PS/proc1
PFILE               = pfile
MESSAGE_BOX        = start
MONITOR_STOP_COUNT = 2
FAIL_PROC_COUNT    = 0
ALL_PROC_COUNT     = 4
```

MESSAGE\_BOX には以下のような処理中の共有メモリのメッセージが表示されます。

メッセージ	説明
start	プロセス監視の実行
resume	プロセス監視の実行 (startと同じ)
stop	プロセス監視の停止
suspend	プロセス監視の停止 (stopと同じ)
shutdown	プロセス監視の終了
change	一時的なpfile値の変更
dump	共有メモリ情報のファイル出力

MONITOR\_STOP\_COUNT には監視停止中のプロセス数が表示されます。

FAIL\_PROC\_COUNT にはリトライオーバーとなっているプロセス数が表示されます。

ALL\_PROC\_COUNT には監視中、停止中にかかわらず、すべての監視プロセス数が表示されます。

show pent

以下のように設定したプロセス単位の監視情報を表示します。

```
pname      = /opt/HA/PS/proc1
pid        = 1883
retry_count = 0
restart_count = 0
```

```

proc_sts                = AVAIL
stall_chk (xxx)         = AVAIL
retry_over_act          = continue
rerun_time              = Thu May 20 10:56:48 1999
include_strings         = AAA
min_proc_count          = 4
max_proc_count          = 6
group_name              = group01
group_sts               = AVAIL
restart_waittime        = 60
pent_id                 = 4
monitor_sts             = on

```

proc\_sts には以下のようにプロセス状態が表示されます。

プロセス監視状態	説明
INIT	初期状態
RESTARTED	プロセス再開成功
AVAIL	プロセス正常動作中(通常状態)
RESTART_FAIL	プロセス再開失敗
UNAVAIL	プロセス異常検出
RETRY_OVER	プロセス再開リトライオーバー
RESTARTING	プロセス再開開始
UNKNOWN	状態不明

stall\_chk は、pfile のオプション部にストール監視を指定している場合のみ表示されます。カッコ内の xxx 3 文字は pfile のオプション部に指定したストール監視の種別により表示が異なります。

表示項目名とストール監視種別との対応は以下のとおりです。

表示項目名	説明
stall_chk (nor)	ファイルの更新時刻によるストール監視
stall_chk (msg)	ファイルの出力メッセージによるストール監視
stall_chk (opf)	対象プロセスのオープンファイルによるストール監視
stall_chk (thd)	対象プロセスの起動スレッドによるストール監視
stall_chk (gen)	対象プロセスのプロセス状態によるストール監視初期状態

stall\_chk には以下のようにストール監視状態が表示されます。

ストール監視状態	説明
INIT	監視中
AVAIL	監視中
DETECT	ストール検出
SUSPECT *1	ストール推定
UNAVAIL	監視失敗
UNKNOWN	状態不明

\*1 対象プロセスのプロセス状態によるストール監視でのみ表示

group\_name, group\_sts は、pfile のオプション部に groupstag を

指定している場合のみ表示されます。

group\_sts の表示内容については、show group の項を参照してください。

include\_strings、min\_proc\_count、max\_proc\_count、restart\_waitime は pfile のオプション部に指定している場合のみ表示されます。

pent\_id には、pfile 中の pent の通番が1から順に表示されます。

monitor\_sts には、以下のように pent 単位の監視状態が表示されます。

pent 監視状態	説明
on	監視状態
off	監視停止状態
unknown	状態不明

show group

以下のように設定したプロセス単位の監視情報を表示します。

```
group_name          = GROUP1
group_sts           = AVAIL
restart_time        = Thu May 20 10:56:48 1999
monitor_sts        = on
```

group\_sts には以下のようにプロセス状態が表示されます。

グループ監視状態	説明
INIT	初期状態
RESTARTED	プロセス再開成功
AVAIL	プロセス正常動作中(通常状態)
RESTART_FAIL	プロセス再開失敗
UNAVAIL	プロセス異常検出
RETRY_OVER	プロセス再開トライオーバー
RESTARTING	プロセス再開開始
UNKNOWN	状態不明

monitor\_sts には、以下のようにグループ単位の監視状態が表示されます。

pent 監視状態	説明
on	監視状態
off	監視停止状態
unknown	状態不明

※ grouptag 指定されたグループの中で、監視状態のプロセスが1つでも存在する場合、monitor\_sts には、on が表示されます。

show proc

以下のように指定したプロセスの起動状況を表示します。

```
UP      = /opt/HA/PS/test_a
UP      = /opt/HA/PS/test_b
DOWN    = /opt/HA/PS/test_c
```

また、指定したプロセスのうち 1 つでも起動できていないものがある場合、proc は戻り値として 1 を返却します。  
すべて起動している場合は 0 を返却します。

show env

以下のように pcheck 実行時の環境変数の状態を表示します。

```
HAPS_CHILDPROC_KILL           = on
HAPS_SILENT_MODE               = off
HAPS_PENDING_TIME              = 60
HAPS_GETPROC_COUNT             = 1000
HAPS_GETPROC_RETRY_COUNT       = 6
HAPS_MIN_PROC_CHECK_RETRY_COUNT = 2
HAPS_MAX_PROC_CHECK_RETRY_COUNT = 1
HAPS_NPROC_CHECK_NOPPROC       = 0
```

HAPS\_CHILDPROC\_KILL が on の場合は、shutdown 実行時に pcheck の子プロセスを kill します。

HAPS\_SILENT\_MODE が on の場合は、サイレントモード on の状態です。

HAPS\_PENDING\_TIME は、pcheck 起動時の自動待ち合わせ時間を表示します。

デフォルトは、60 (秒)で、環境変数 HAPS\_PENDING\_TIME を指定することで 0 ~ 60\*60\*24 (1 日) の範囲で変更することが可能です。  
HAPS\_GETPROC\_COUNT は、監視対象選択時のプロセス情報取得量を表示します。

デフォルトは、1000 (個)で、環境変数 HAPS\_GETPROC\_COUNT を指定することで 1 ~ 100000 の範囲で変更することが可能です。

HAPS\_GETPROC\_RETRY\_COUNT は、監視対象選択時のリトライ回数を表示します。

デフォルトは、6 (回)で、環境変数 HAPS\_GETPROC\_RETRY\_COUNT を指定することで 1 ~ 100 の範囲で変更することが可能です。

HAPS\_MIN\_PROC\_CHECK\_RETRY\_COUNT は、下限監視時のリトライ回数を表示します。

デフォルトは、2 (回)で、環境変数

HAPS\_MIN\_PROC\_CHECK\_RETRY\_COUNT を指定することで 1 ~ 100 の範囲で変更することが可能です。

HAPS\_MAX\_PROC\_CHECK\_RETRY\_COUNT は、上限監視時のリトライ回数を表示します。

デフォルトは、1 (回)で、環境変数

HAPS\_MAX\_PROC\_CHECK\_RETRY\_COUNT を指定することで 1 ~ 100 の範囲で変更することが可能です。

HAPS\_NPROC\_CHECK\_NOPPROC が 1 の場合は、プロセス個数監視時に監視対象プロセス自身の消滅を障害条件から除外します。デフォルトは、0 です。

環境変数を設定する手順の詳細については、

「4.8. サイレントモードでの運用手順」、

「4.9. pcheck 終了時に子プロセスを回収する手順」、

「4.10. pcheck 起動時の自動待ち合わせ時間を変更する手順」、

「4.12. 監視対象選択時のプロセス情報取得量を変更する手順」、

「4.13. 監視対象選択時のリトライ回数を変更する手順」、

「4.14. プロセス個数監視時のリトライ回数を変更する手順」を参照してください。

## check alive

以下のように監視対象プロセスの消滅監視状態の確認結果を表示します。  
また監視状態を終了ステータスで返却します。

2021-3-30 11:32:55 Alive Check OK (0) [pfile]

2021-3-30 11:32:55 Alive Check NG (状態コード 0 以外) [pfile]

行頭には監視対象プロセスの消滅監視状態の確認実行の日時情報が表示されます。

[pfile] には監視定義ファイル名が表示されます。

() 内には状態コードが表示されます。状態コードの値は本コマンドの終了ステータスと同じです。状態コードは以下の定義となります。

状態コード	説明
0	プロセスは正常動作中(通常状態)
1	コマンドパラメーターのエラー またはpcheckが起動していない
103	プロセスの再開失敗
104	プロセスは存在しない(停止状態)
105	内部エラーが発生した
106	プロセスの再開断念(リトライオーバー)
107	プロセスの再開開始

※ 複数プロセスを監視中の場合は、すべての監視対象が正常動作しているの  
であれば終了ステータス 0 となります。

また複数プロセスを監視中、正常動作していない対象が1つでもあった  
場合は最初の問題プロセスの状態を上表で対照して終了ステータスを  
決定します。

## check stall

以下のように監視対象プロセスのストール監視状態の確認結果を表示します。  
また監視状態を終了ステータスで返却します。

2021-3-30 14:38:58 Stall Check OK (0) [pfile]

2021-3-30 14:38:58 Stall Check NG (状態コード 0 以外) [pfile]

行頭には監視対象プロセスのストール監視状態の確認実行の日時情報が  
表示されます。

[pfile] には監視定義ファイル名が表示されます。

()内には状態コードが表示されます。状態コードの値は本コマンドの終了ステータスと同じです。

状態コードは以下の定義となっています。

状態コード	説明
0	プロセスは準備できている(通常状態)
1	コマンドパラメーターのエラー pcheckが起動していない またはストール監視していない
102	プロセスストールを検出した (ユーザーが定義したストール状態となった)



103	プロセスストールと推定される (ユーザーが定義したストール状態ではないがプロセスの動きがない)
104	内部エラーが発生した

※ 複数プロセスを監視中の場合は、すべての監視対象が通常動作中であれば終了ステータス 0 となります。  
また複数プロセスを監視中、通常動作ではない対象が1つでもあった場合は最初の問題プロセスの状態を上表で対照して終了ステータスを決定します。

#### 終了ステータス

成功すると 0 を返し、失敗するとそれ以外を返します。

#### 注意事項

- ・ 本コマンドは、pcheck で設定された共有メモリ領域を参照するため、pcheck コマンド実行中でない場合は、使用できません。
- ・ 本コマンドは、pcheck の動作を予約するコマンドのため複数同時に実行はできません。
- ・ オプションを指定する場合、-c オプションは最後に指定してください。
- ・ 停止したプロセス監視の再開を行う場合には、-c reload オプションを使用して pfile の再読み込みを行ってください。

#### 使用例

pfile で監視中のプロセスについて監視の停止を行います。

```
/opt/HA/PS/bin/padmin -f pfilename -c stop
```

pfile で監視中のプロセスについて、プロセス単位に監視の停止を行います。

※以下の例では、pfile で設定しているプロセスについて、grouptag が group\_oracle のプロセスと pent\_id が 3 と 4 のプロセスについて監視の停止を行います。

```
/opt/HA/PS/bin/padmin -f /var/opt/HA/PS/conf/bin/pfile1 -c stop group_oracle 3 4
```

pfile の再読み込みを行います。

※stop オプションなどで停止した pfile の再開を行う場合には、必ず reload を行ってください。

```
/opt/HA/PS/bin/padmin -f pfilename -c reload
```

pfile で設定しているプロセスについて監視の再開を行います。

```
/opt/HA/PS/bin/padmin -f pfilename -c start
```

pfile で設定しているプロセスについて、プロセス単位に監視の停止を行います。

※以下の例では、pfile で設定しているプロセスについて、grouptag が group\_oracle のプロセスと pent\_id が 3 と 4 のプロセスについて監視の再開を行います。

```
/opt/HA/PS/bin/padmin -f /var/opt/HA/PS/conf/bin/pfile1 -c start group_oracle 3 4
```

pfile で監視中のプロセスについて設定値を表示します。

```
/opt/HA/PS/bin/padmin -f pfilename -c show param
```

#### 関連項目

pcheck

## 名称

pcheck - プロセスの監視および再開コマンド

## 構文

```
pcheck -f pfilename
pcheck -f pfilename [-t]
pcheck -f pfilename [-w wait_time]
```

## 機能説明

入力で指定された *pfile* に基いて、プロセスの監視、および障害発生時のプロセスの再開処理を行います。

- f *pfilename*      プロセスの監視、および障害発生時のプロセスの再開処理を行います。  
引数には、プロセス監視/再開について設定した *pfile* 名を指定します。  
*pfile* 名は 256 文字未満で指定してください。
  
- w *wait\_time*      プロセスの監視を始める前に一定の猶予時間を与えます。  
引数には、プロセス監視を開始するまでの待ち合わせ時間を秒単位で指定し、  
指定値は 1 秒以上が有効です。  
pcheck 起動時に、対象プロセスが起動されていないケースを考慮し、  
プロセス起動に必要な時間を指定することで無駄な消滅検出を防止できます。
  
- t                    デバッグモードでの運用を指定します。  
標準出力に実行履歴を出力します。

*pfile* の設定値については、「3.3. *pfile* ファイルについて」を参照してください。

## 終了ステータス

成功すると 0 を返し、失敗するとそれ以外を返します。

## 注意事項

- ・ 本コマンドを終了する場合、kill(1) コマンドを使用しないでください。  
本コマンドを終了する場合には、padmin コマンドを使用してください。  
/opt/HA/PS/bin/padmin -f *pfile* -c shutdown
- ・ 本コマンドは、プロセス再開時に、プロセス再起動スクリプトの終了を待ち合わせます。  
そのため、プロセス再起動スクリプト内で無限ループとなったり、実行プロセスの終了を  
待ち合わせたりすると、プロセス監視が効果的に作用しない場合があります。
- ・ 対象プロセスを再開する場合、依存関係のあるプロセス(同名プロセスなど)が存在すると、  
プロセスの再開および監視が正常に行われない場合があります。プロセス再起動スクリプトに  
おいて、対象プロセス再開時に、依存関係のあるプロセスの再設定を行ってください。
- ・ 本コマンドで監視可能な最大プロセス数は、256 です。

## 使用例

pfile に設定したプロセスの監視を行います。  
`/opt/HA/PS/bin/pcheck -f pfile &`

pfile に設定したプロセスの監視を、30 秒後に開始します。  
`/opt/HA/PS/bin/pcheck -f pfile -w 30 &`

pfile では `/home/woo/test` というプロセスの監視を行うように以下の設定をします。

<code>IPCKEY</code>	<code>0x10000001</code>
<code>MSG_CHECK_INTERVAL</code>	<code>5</code>
<code>MONITOR_INTERVAL</code>	<code>10</code>
<code>SHM_DUMP_FILE</code>	<code>/home/woo/dump</code>

`/home/woo/test:/home/woo/test.sh:600:5:continue`

## 関連項目

padmin

## 名称

putil - プロセス情報の取得と pfile のチェックコマンド

## 構文

```
putil [-c ppid] [-g gid] [-p pid] [-k keyword[,keyword1,..]] [-m]
putil [-f pfilename [-v]]
putil [-z]
```

## 機能説明

プロセス情報の取得と ProcessSaver の pfile のチェックを行います。  
プロセス情報は以下のフォーマットで出力されます。

UID	: ユーザーID
PID	: プロセス ID
PPID	: 親プロセス ID
GID	: グループ ID
process name	: プロセス名 プロセス起動時のプロセス名を表示します。 pfile には、このプロセス名を指定してください。 このとき、引き数による識別が不要な場合、引き数は省略可能です。
status	: 現在のプロセスのステータスです。

## オプション

-p *pid*

プロセス ID が *pid* のプロセス情報を出力します。  
-p 、-k オプションが同時に指定された場合-p オプションが優先されます。

-g *gid*

グループ ID が *gid* のプロセス情報を出力します。

-c *ppid*

プロセス ID が *ppid* のプロセスの子プロセスのプロセス情報を出力します。

-k *keyword1* [,*keyword2*,...]

プロセス名の中に *keyword1* が含まれるプロセスのプロセス情報を出力します。  
キーワードを複数指定すると and 条件で検索できます。  
その場合キーワードをコンマ(',')で区切り指定します。

-m

pfile の自動生成(標準出力)を行います。  
-c , -g , -p オプションまたは、-k オプションとともに使用します。  
-c , -g , -p オプションまたは、-k オプションで対象となったプロセスの  
プロセス名に対応する pfile の自動生成(標準出力)を行います。

-z

ゾンビプロセスのプロセス情報を出力します。-k オプションとともに使用できます。  
-z オプションを指定しない場合、ゾンビプロセスのプロセス情報は出力されません。

-f *pfilename*

*pfilename* で指定された pfile をチェックし、不正な箇所があった場合  
エラー情報を出力します。エラー情報は以下のフォーマットで出力されます。  
ブロック情報のチェックはサポートしていません。

---

PARAMETER : VALUE  
: Warning

PARAMETER はパラメーター名です。VALUE は pfile によって設定されている値です。  
COMMENT はエラー原因です。

-v

pfile のチェック結果の、より詳しい情報を出力します。

## 診断

正常終了:終了ステータスに0が設定されます。

異常終了:終了ステータスに0以外が設定されます。

終了ステータス 1:PID 指定以外プロセス情報取得エラー  
2:PID 指定プロセス情報取得エラーまたは pfile チェックエラー  
3: オプションエラー

## 使用例

動作中の全プロセスのプロセス情報を表示します。

# putil

[ 出力例 ]

---

UID : 0  
PID : 13189  
PPID : 13186  
GID : 3  
process name : zeus.isapi -g 3  
status : 1

UID : 0  
PID : 13191  
PPID : 13186  
GID : 0  
process name : /usr/local/zeus/web/bin/zeus.web  
status : 1

UID : 0  
PID : 13192  
PPID : 13186  
GID : 0  
process name : /usr/local/zeus/web/bin/zeus.web  
status : 1

~~~ 以下省略 ~~~

---

プロセス名に syslogd という文字列が含まれるプロセスのプロセス情報を表示します。

```
# putil -k syslogd
```

[ 出力例 ]

```
-----  
UID           : 0  
PID           : 513  
PPID          : 1  
GID           : 0  
process name  : syslogd -m 0  
status        : 1  
-----
```

プロセス名に syslogd という文字列が含まれるプロセスに対応する pfile の自動生成を行います。

```
# putil -m -k syslogd
```

[ 出力例 ]

```
-----  
# ProcessSaver configuration file  
##### PARAM #####  
IPCKEY           0x10000001  
MSG_CHECK_INTERVAL 5  
MONITOR_INTERVAL 10  
SHM_DUMP_FILE    /var/opt/HA/PS/log/pcheck_dump  
  
##### PENT #####  
## pname:restart shell:grace:retry_count_max:retry_over_action  
syslogd -m 0:-:0:0:continue  
-----
```

作成された pfile の IPCKEY, MSG\_CHECK\_INTERVAL, MONITOR\_INTERVAL, SHM\_DUMP\_FILE はデフォルト値が設定されます。  
また、restart shell、grace、retry\_count\_max、retry\_over\_action のそれぞれについても、'-'、'0'、'0'、'continue' がデフォルト値として設定されます。

pfile\_syslog の正当性をチェックします。

・IPCKEY パラメーターが不正な事例です。

```
# putil -f pfile_syslog
```

[ 出力例 ]

```
-----  
PARAM-AREA WARNING  
-----  
PARAMETER      : VALUE  
                : Warning  
-----  
IPCKEY         : 0x00000000  
                : IPCKEY is an integer value from 1 to 0x7ffffff  
  
PENT[No.001]-AREA WARNING  
-----
```

・grace\_time パラメーターが不正な事例です。

```
# putil -f pfile_syslog -v
```

[ 出力例 ]

```
-----  
*** PENT[No.001]-AREA *****  
process_name    : syslogd -m 0  
shell_path      : /etc/init.d/syslog  
grace_time      : 3000  
retry_num       : 3000  
retry_over_action : continue  
  
PENT[No.001]-AREA WARNING  
-----  
PARAMETER      : VALUE  
                : Warning  
-----  
retry_num      : 3000  
                : retry_num is an integer value from 0 to 1024  
-----
```

#### 注意事項

- ・ -m オプション指定時、同一プロセス名が存在する場合はすべての内容が出力されます。  
親子関係があるプロセスについては親プロセスのみの記述に変更してください。
- ・ -m オプション指定時、PID が 0 のプロセス の pfile は作成できません。

#### 関連項目

pcheck



## 名称 ストール監視のユーザー提供ライブラリ

### (1) 初期化関数

#### 関数名

HA\_PS\_ckstall\_init()

#### 形式

```
#include "sys/ha_ps_user.h"

extern int
HA_PS_ckstall_init(const char *file_name);
                const char *file_name; /* 監視対象ファイル名 */
```

#### 機能

本関数は、監視対象ファイルを作成します。  
既にファイルが存在するとアクセス時刻を更新します。  
対象プロセスが pcheck と連携してストール監視を開始する際には、  
本関数を最初に呼び出してください。

#### 入力パラメーター

file\_name (const char \*)  
監視対象ファイル名を指定します。省略はできません。

#### 戻り値

関数が正常に終了した場合は、HA\_PS\_SUCCESS を返却します。  
それ以外の場合は、HA\_PS\_FAILED を返却します。

- 監視対象ファイル名が指定されていない。
- 監視対象ファイルを open できない。

#### 出力パラメーター

ありません。

(2) 監視対象ファイルの更新関数

関数名

HA\_PS\_ckstall\_update ()

形式

```
#include "sys/ha_ps_user.h"

extern int
HA_PS_ckstall_update(void);
```

機能

本関数は、監視対象ファイルのアクセス時刻を更新します。  
監視対象プロセスが pcheck と連携しストール監視を実現する際には、  
本関数を定期的呼び出して下さい。  
pfile で規定された時間以内に本関数を呼び出さなければ、  
pcheck においてストール状態と認定されます。

入力パラメーター

ありません。

戻り値

関数が正常に終了した場合は、HA\_PS\_SUCCESS を返却します。  
それ以外の場合は、HA\_PS\_FAILED を返却します。

- 初期化関数が一度もコールされていない。
- 監視対象ファイルの更新に失敗した。

出力パラメーター

ありません。

(3) 監視対象ファイルの消去関数

関数名

HA\_PS\_ckstall\_end ()

形式

```
#include "sys/ha_ps_user.h"

extern int
HA_PS_ckstall_end (void);
```

機能

本関数は、監視対象ファイルを消去します。  
監視対象プロセスが pcheck と連携したストール監視を終了する際には、  
本関数を最後に呼び出してください。  
本関数が呼び出されると、pcheck はストール監視を終了します。  
ストール監視を再開する場合は、初期化関数から順に呼び出してください。

入力パラメーター

ありません。

戻り値

関数が正常に終了した場合は、HA\_PS\_SUCCESS を返却します。  
それ以外の場合は、HA\_PS\_FAILED を返却します。  
- 初期化関数が一度もコールされていない。

出力パラメーター

ありません。

CLUSTERPRO  
MC ProcessSaver 2.9 for Linux  
ユーザーズガイド

2024 年 4 月 第 12 版  
日本電気株式会社  
東京都港区芝五丁目 7 番地 1 号  
TEL (03) 3454-1111(代表)

© NEC Corporation 2024

日本電気株式会社の許可なく複製、改変などを行うことはできません。  
本書の内容に関しては将来予告なしに変更することがあります。

保護用紙